Royal Holloway, University of London

Doctoral Thesis

Conformal and Venn Predictors for Multi-probabilistic Predictions and Their Applications

Author:

Chenzhe Zhou

Supervisor:

Alex Gammerman

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

in the

Computer Learning Research Centre

Department of Computer Science,

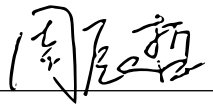
Royal Holloway, University of London

2015

Declaration of Authorship

I, CHENZHE ZHOU, declare that this thesis titled, Conformal and Venn Predictors for Multi-probabilistic Predictions and Their Applications and the work presented in it are entirely my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date:

Abstract

In machine learning, a typical algorithm gives predictions for the unknown objects based on known properties learned from the training data set. However, most algorithms can only give a single prediction (for classification, it is predicted label; for regression, it is predicted value). A demand for probabilistic prediction has risen in view of the fact that a prediction with its complementary probabilistic estimation is more informative than a single prediction. An example is the probabilistic weather forecast. We prefer hearing that tomorrow has a chance of 60% to be rainy rather than there will be rain tomorrow.

However, in most areas, single probabilistic prediction is still not enough. True probability could be either higher or lower than its estimation. If we make multiple probabilistic predictions that can hedge the true probability within an interval, we could have a better estimation. The term multi-probability is brought to mind, which means that we announce several probability distributions for the new label rather than a single one.

In this thesis, we propose several novel designs of Venn predictors and Conformal predictors that provide multi-probabilistic predictions together with single predictions. These implementations are based on k-Nearest Neighbours, Support Vector Machines and Crammer and Singer's Multi-Class Support Vector Machines. These algorithms could give high accuracy together with probabilistic predictions. Experimental testing is carried out.

We then compare these algorithms to some other algorithms for probabilistic predictions. The results demonstrate the advantages of applying these algorithms.

Acknowledgements

I am very thankful to my supervisor Alex Gammerman for his invaluable suggestions on the subjects of my research and his generous and continuing help and support during all these year I study here. I would also like to thank my advisor Zhiyuan Luo for his constructive advice and friendly guidance on every aspect from my research to life during my oversea study life.

I am very grateful to Volodya Vovk for his directions and help in theoretical research and Ilia Nouretdinov for his insightful comments and discussions regarding my work.

My sincere thanks also goes to other members in Computer Learning Research Centre, especially Mitya Adamskiy, Meng Yang, Jiaxin Kou, Valentina Fedorova, Antonis Lambrou, James Smith for offering a great environment for study and research. Also thanks to all other fellow Ph.D. students of Department of Computer Science of Royal Holloway, University of London, for the supportive and friendly environment.

Last but not the least, I would like to thank my family: my lovely parents, for bringing me to the beautiful world at the first place and supporting me spiritually and materially throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this introduction chapter, we will briefly introduce the motivation for the research in this thesis. The novelty achieved during the research will be given afterwards, along with the related publications. In the end, the structure of this thesis will be listed.

1.1 Motivation

Machine learning is a branch of Artificial Intelligence. In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed" [58].

Tom M. Mitchell provided a widely quoted, more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [40, p. 2].

Essentially, machine learning is a scientific discipline concerned with the design and development of teaching computers to make and improve predictions or behaviours based on some empirical data, such as that from sensors or data sets. It is a type of Artificial Intelligence. Another way to think about machine learning is that it is "pattern recognition"

- the act of teaching a program to react to or recognize patterns.

The process of machine learning is as follows. A learner can search through data, which can be seen as examples of the possible relations between observed variables, to capture characteristics of interest of their unknown underlying probability distribution. Then, the learner can look for patterns and use that data to recognize complex patterns and make intelligent decisions based on input data, which can also be interpreted as improving the programme's understanding.

Machine learning is becoming increasingly popular and being applied to a great variety of problems and fields. However, most machine learning algorithms do not provide any indication of how reliable their predictions are. The information on the reliability of the predictions would be very beneficial for most applications, and it is even crucial for some risk sensitive settings such as medical diagnosis and drug discovery. In this thesis, we are interested in machine learning algorithms that provide additional information about how reliable it is for each prediction.

A type of algorithm that give probabilities to complement its predictions is called *probabilistic predictors*. As its name implies, probabilistic prediction is to predict probabilities, or in other words, is to make estimations to the distribution that all the examples are drawn from.

There are lots of algorithms to make probabilistic predictions since Charles J. Stone [59] introduced a method using k-nearest neighbours in 1977. The most popular algorithms are probably approximately correct learning [62] (a.k.a. PAC learning), and hold-out estimate. However, these algorithms all have their drawbacks. For PAC learning, the shortcoming is that its estimated bound of error is usually too loose to tell us any interesting information on the data set, especially when the data set we deal with is not so large. For the hold-out estimate, the drawback is that it rigidly separates the process of learning and prediction, and the estimates rely on the hold-out examples. Furthermore, for both PAC learning and hold-out estimate, they make probabilistic estimations for a whole group of predictions rather than provide information on each prediction.

Other methods are such as Naive Bayes classifier, Logistic Regression and Platt Scaling, which can make probabilistic estimations for individual prediction. However, these algorithms are only applicable under strong assumptions.

In this thesis, we will mainly focus on a new kind of algorithm that makes probabilistic predictions. They are Conformal prediction and Venn prediction introduced in [69]. Comparing to the algorithms we mentioned above, these newly developed methods have several advantages. First, these algorithms blend the learning and prediction together. Second, the probabilistic predictions our algorithms give are much more confident and accurate than the traditional methods. Third, our algorithms provide probabilistic information on every prediction individually rather than estimate an error on all future examples as a whole.

1.2 Main Contributions

The novelty achieved during the work on this thesis is as follows.

Several new taxonomy designs based on Support Vector Machine for Venn Machine were proposed, including:

- SVM Taxonomy with Equal Length Intervals
- SVM Taxonomy with k-Means Clustering Intervals
- Taxonomy with Multi-class SVM by Crammer and Singer

During the development of these new taxonomies, we also suggested a new way to extend these algorithms from binary case to multi-class case by using a combined decision value. Based on those new taxonomies for classification, we also proposed a new taxonomy design for Regression Venn Machine.

All the above algorithms were tested on a variety of data sets, and the results showed the advantages of these algorithms in giving predicted labels and probabilistic outputs.

Besides taxonomy designs for Venn Machines, we also researched into another type of algorithm that transform predictions to probabilities. Among them are:

- Conformal Prediction with Bivariate Isotonic Regression
- Venn-ABERS Predictor with SVM

1.3 Publications

During the research covered in this thesis, the contributions were presented in two conferences and resulted in two publications. A list of publications is as follows:

1. Chenzhe Zhou, Ilia Nouretdinov, Zhiyuan Luo, Dmitry Adamskiy, Luke Randell, Nick Coldham, and Alex Gammerman, *A comparison of venn machine with platt's method in probabilistic outputs*, Artificial Intelligence Applications and Innovations (2011), 483–490

2. Chenzhe Zhou, Ilia Nouretdinov, Zhiyuan Luo, and Alex Gammerman, *Svm venn machine with k-means clustering*, Artificial Intelligence Applications and Innovations, 2014

1.4 Outline of the Thesis

This introductory chapter has given the motivation behind the research carried out in this thesis and has briefly described the main contributions.

The rest of the thesis is organised as follows.

Chapter 2 gives the background of the problem. It is devoted to known algorithms that are capable as underlying algorithms in conformal predictors and Venn predictors and other algorithms that also give probabilistic predictions.

In Chapter 3, we provide an introduction to two algorithms that belong to a new type that transform predictions into probabilities. These two algorithms are conformal

predictor with bivariate isotonic regression and Venn-ABERS predictor.

In Chapter 4, several new designs of probabilistic predictors for Classification Venn Machine are proposed and investigated. We also talk about the design of probabilistic predictors for regression that provide valid intervals as predictions instead of a single value.

In Chapter 5, we apply these algorithms mentioned in Chapter 2, Chapter 3 and Chapter 4 on several data sets and compare our algorithms to other probabilistic predictors.

Chapter 6 gives the conclusion to the thesis, outlines its main contributions and provides directions for further research.

In Appendix, the reader can find an additional introduction for data sets and the manual for our Venn predictors library.

Chapter 2

Overview of Machine Learning Algorithms

In this chapter, we will mainly introduce some background related to the problems raised in this thesis.

First, we start with the formal problem description that this thesis mainly solves together with some requisite assumptions in §2.1. Second, we give some brief introductions for the basic frameworks of Conformal prediction and Venn prediction in §2.2. Then in §2.4, several widely used traditional algorithms that apply to Conformal and Venn predictions are introduced. We also introduced some algorithms that give probabilistic predictions including Naive Bayes classifier, Logistic Regression and Platt Scaling in §2.5. In §2.6 of this chapter, Conformal and Venn predictions are compared with simple prediction and probabilistic predictions and the advantages and drawbacks are revealed in this section. The summary section §2.7 summarizes this chapter and emphasizes the key parts in the previous sections.

2.1 Problem and Assumptions

The *examples* are pairs consisting of *objects* and *labels* that carry information derived from reality.

$$(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots \tag{2.1}$$

Each example (x_i, y_i) is composed of an object x_i and its label y_i. The object x_i is a vector of attributes that are real numbers quantified from the real world measurement. In other words, $x_i \in \mathbf{R}^d$ (where d is the number of attributes), and we call this measurable space $\mathbf{X} \subset \mathbf{R}^n$ the *object space* (or the *feature space*). The label y_i is a real number, and they are elements of a measurable space $\mathbf{Y} \subset \mathbf{R}$, which is called the *label space*. We denote the pair of object and label by $z_i = (x_i, y_i)$ when we need a more compact notation. All examples are elements of a measurable space $\mathbf{Z} = \mathbf{X} \times \mathbf{Y}$ called the *example space*.

We are given a set of examples $(z_1, \ldots, z_{n-1})$ that contains $n - 1$ known examples, usually this is hereinafter referred to the *training set*. We are also given a new object x_n, which is usually referred to the *testing example*. A set of testing examples is called the *testing set*. Our goal is to predict the label $\hat{y}_n$ for the new object. Furthermore, we want the predictor to give some estimations on the likelihood of its prediction being true (i.e. $\hat{y}_n = y_n$, where y_n is the true label of object x_n).

We assume that the object space $\mathbf{X}$ is not empty and the label space $\mathbf{Y}$ has at least two different elements in it. According to the cardinality of label space, the problem can be divided into two types of tasks: classification problem and regression problem. We call a problem *classification* problem only when the cardinality $K = |\mathbf{Y}|$ of the label space is finite, which means that the label space could be represented as a set of numbers $\mathbf{Y} = \{0, 1, 2, \ldots, K - 1\}$ regardless of the original values of the label space. Classification is to assign one sort of possible categories to objects according to [40, p. 54]. This kind of problems includes medical diagnosis, handwriting recognition, spam detecting and so on. If the cardinality of the label space is infinite, usually the labels are real-valued, this sort of problems often refers to the *regression* problem. Regression problems include market

price prediction, biometrics and econometrics problems etc. In this thesis, we only took these two problems into consideration.

In order to build a probabilistic predictor, we need to make some additional assumptions. Basically, when we learn from the example space (or the environment), we always want it to be governed by some constant laws or some laws evolving slowly in the space that we could learn from. If the environment keeps changing all the time, it will be too hard for us to learn anything from it. In the traditional way to satisfy the request of a steady environment, we assume that all the examples are randomly chosen from some fixed probability distribution Q on the fixed example space $\mathbf{Z}$. In other words, all the examples share the same probability distribution Q and they are independent from each other. We say that these examples are *independent and identically distributed* (also known as *i.i.d.*). This is called the *randomness assumption* [69, p. 2].

Furthermore, we could make the randomness assumption without knowing anything about the probability distribution Q at the beginning. All we know are the example space $\mathbf{Z}$ and these examples are drawn independently from the same probability distribution Q. In this case, we say we are *learning under unconstrained randomness* [69, p. 3]. Most of the algorithms talked in this thesis are under this assumption.

However, the assumption our algorithms need is usually slightly weaker than the standard assumption. It is called the *exchangeability assumption*, and it assumes that the infinite data sequence (2.1) is drawn from a distribution Q that is *exchangeable* [69, p. 18]. This means for every positive integer n, every permutation π of $\{1, \dots, n\}$, and every measurable set $E \subset \mathbf{Z}^\infty$,

$$Q\{(z_1, z_2, \dots) \in \mathbf{Z}^\infty : (z_1, \dots, z_n) \in E\} \\ = Q\{(z_1, z_2, \dots) \in \mathbf{Z}^\infty : (z_{\pi(1)}, \dots, z_{\pi(n)}) \in E\} \quad (2.2)$$

where $\mathbf{Z}^\infty$ is the set of all infinite sequences of elements of $\mathbf{Z}$. The exchangeability assumption can be satisfied by random permutation of the data sets.

Throughout this thesis, we follow the same description of this problem and generally

we use one of the randomness assumption and the exchangeability assumption depending on which one is needed to make a stronger statement.

2.2 Conformal Prediction

In a classification or regression problem in machine learning, we simply want to predict the true label y_n for the new object x_n before we observe it from the successive sequence of the example space.

For any example sequence $(x_1, y_1), \ldots, (x_{n-1}, y_{n-1}) \in \mathbf{Z}^*$ and any new object $x_n \in \mathbf{X}$, the function we need can be represented as follows

$$D : \mathbf{Z}^* \times \mathbf{X} \to \mathbf{Y} \tag{2.3}$$

This function is called a *simple predictor*. It announces $D(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \in \mathbf{Y}$ as its prediction for y_n, the true label of x_n.

In a more complicated notion of prediction, we attempt to complement every possible label with a degree of confidence instead of choosing a single element of $\mathbf{Y}$ as our prediction. Then we try to pronounce a subset of $\mathbf{Y}$ with a predefined degree of confidence. For each possible label, if the predefined degree of confidence is greater than its complementary degree of confidence, we will include this label in our subset. By pronouncing this subset, we are confident, to a predefined degree, the true label y_n will fall in our predicted subset. The more confident we want to be, the larger the subsets should be. Since it will include more possible labels. Such predictor is called *confidence predictor* and the outcome of this predictor is called *prediction set*. It is described in [23, 69].

A confidence predictor takes an additional parameter $\epsilon \in (0, 1)$ as *significance level*. The complementary value $1 - \epsilon$ is called the *confidence level*. Given the training set, the new object and the significance level ϵ, a confidence predictor Γ is described as

$$\Gamma : \mathbf{Z}^* \times \mathbf{X} \times (0, 1) \to 2^{\mathbf{Y}} \tag{2.4}$$

where $2^{\mathbf{Y}}$ is the power set of $\mathbf{Y}$.

This predictor Γ outputs a subset

$$\Gamma^{\epsilon}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \tag{2.5}$$

The subset must satisfy

$$\Gamma^{\epsilon_1}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \subseteq \Gamma^{\epsilon_2}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \tag{2.6}$$

whenever $\epsilon_1 \geq \epsilon_2$.

With a confidence of $1 - \epsilon$, Γ announce y_n is in its prediction set

$$y_n \in \Gamma^{\epsilon}(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \tag{2.7}$$

If the prediction set contains no labels, we call this an *empty prediction*. If the prediction set contains one label (no need to be exact the true label), we call this a *certain prediction*. While the prediction set contains several labels, we say these predictions are *multiple predictions*. Only if the true label y_n is not included in the above prediction set, we say that an *erroneous prediction* is made by the confidence predictor, which means multiple predictions that include the true label are not mistakes, they are just not informative.

A confidence predictor has two important properties: *validity* and *efficiency*. Given a significance level ϵ, if the relative frequency of errors that a confidence predictor makes does not exceed ϵ for every ϵ given to it, we say a confidence predictor is *valid*. It means that the number of errors or the error rate can be controlled within an acceptable tolerance by the significance level ϵ, which is predefined by the user. The other property efficiency means the confidence predictor should give informative prediction sets. It is obvious that a prediction set with fewer labels is more informative than the prediction set with more labels. In other words, it can be understood as that an informative prediction set given by a confidence predictor should be as small as possible.

These two properties, especially the validity, can be the main advantage over other types of algorithms like simple predictors or probabilistic predictors. However, we cannot

achieve the property of validity at no cost. In contrast, the property of validity and the property of efficiency are contradictory. Given a small ϵ, the error rate can be controlled under ϵ, which means it is valid, but to make few errors or to give prediction set that includes true label at high $(1-\epsilon)$ confidence results in including more labels in the prediction set. In other words, lower significance levels result in larger prediction sets, and vice versa. Thus, we need to balance validity and efficiency, which makes confidence machine a very flexible tool [14].

A conformal predictor is such a confidence predictor. Conformal predictors refer to a set of algorithms that produce a prediction set when given a significance level. It announces that a new object will have a label that makes it similar to the known examples in some specified way [69, p. 7]. Furthermore, its estimates of reliability are not in the form of probabilities; it uses the degree to which the specified type of similarity holds among the old examples to estimate our confidence in the prediction.

The idea of a rudimentary Conformal prediction was described by Gammerman et al. [22], and the formal definitions were first described by Vovk et al. [68] and Saunders et al. [55].

In the following, we will give the definition of conformal prediction. We begin with the concept of "bag", which is also called a multi-set. A *bag* is a collection of some elements regardless of the order of the elements, which is similar to a set. The difference between a bag and a set is that repetition of the elements is allowed in a bag, which means the elements in a bag can be identical. The size n of a bag is the number of elements a bag contains. Since the elements in a bag may be repeated, we must mention what elements it contains and how many times each of them duplicated to identify a bag. We refer to a bag contains n elements as $\wr z_1, \ldots, z_n \int$, and some of the elements may be identical to each other.

Although the order of the elements in a bag is irrelevant, we mark the elements in the order in our notation so that it is convenient when we mention these elements. We write $\mathbf{Z}^{(n)}$ for the set of all bags of size n containing n elements from a measurable space $\mathbf{Z}$. We

write $\mathbf{Z}^{(*)}$ for the set of all bags of elements of $\mathbf{Z}$.

The general idea of conformal prediction is to assume the label of the new object x_n is one of the possible labels, say y. Then we measure how well the new pair of example (x_n, y) conforms with the known examples $\{z_1, \ldots, z_{n-1}\}$ through a nonconformity measure. We try every possible label in this way and calculate a nonconformity score for each possible label. To announce a possible label in our prediction set depends on whether its p-value is greater than the predefined significance level ϵ.

There are many different nonconformity measures, and each one defines a conformal predictor.

Formally, a *nonconformity measure* is a method that assesses how strange a new object is from the old examples, which maps each possible new example and each bag of old examples to a numerical score, named *nonconformity score*.

$$A : \mathbf{Z}^{(*)} \times \mathbf{Z} \to \mathbf{R} \tag{2.8}$$

It is easy to invent nonconformity measure [69, p. 24] based on the algorithms we already have. The algorithm used by a conformal predictor is called *underlying algorithms* of the conformal predictor. An introduction to some popular underlying algorithms will be given in §2.4, while how we use these underlying algorithms as nonconformity measures will have a devoted section, namely §3.3.2 in next chapter.

If we separately think of nonconformity measures dealing with different bags of examples, the set of nonconformity measures A_n for each $n = 1, 2, \ldots$ can be defined as

$$A_n : \mathbf{Z}^{(n-1)} \times \mathbf{Z} \to \mathbf{R} \tag{2.9}$$

where n is the size of the bag.

The nonconformity score for each z_i in a bag $\{z_1, \ldots, z_n\}$ is

$$\alpha_i := A_n(\{z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n\}, z_i) \tag{2.10}$$

The numerical value of α_i on its own does not give us any information about how strange the example z_i is compared to other examples. For this reason, we use *p-value*

as an indicator of strangeness that a nonconformity score α_i compared with others. It can be achieved by calculating the proportion of nonconformity scores that are greater than or equal to α_i among all scores. The definition of p-value for example z_i is given in Eq. (2.11).

$$p := \frac{|\{j = 1, \ldots, n : \alpha_j \geq \alpha_i\}|}{n} \tag{2.11}$$

In Eq. (2.11), p is the p-value for example z_i. It is a quantitative estimation of the strangeness between the example z_i and the other examples. Since α_i is compared with itself, the numerator of the p-value is at least 1. Hence, the p-value ranges from $\frac{1}{n}$ to 1, indicating that example z_i is very nonconforming (i.e. strange) if the p-value is small or it is very conforming (i.e. similar) if the p-value is large.

Here we give the formal definition of conformal prediction. Given a training set $\wr z_1, \ldots, z_{n-1} \int$, a new object x_n and a significance level ϵ, the conformal predictor Γ determined by a nonconformity measure A_n gives a prediction set as in Eq. (2.12)

$$\Gamma^\epsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) := \{y \in \mathbf{Y} : p_y > \epsilon\} \tag{2.12}$$

In Eq. (2.12), p_y is the p-value for the pair of (x_n, y) and is defined as follows

$$p_y := \frac{|\{i = 1, \ldots, n : \alpha_i \geq \alpha_n\}|}{n} \tag{2.13}$$

where

$$\alpha_i = A_n(\wr (x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y) \int, (x_i, y_i))$$
$$\alpha_n = A_n(\wr (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}) \int, (x_n, y))$$

We have a confidence of $1 - \epsilon$ that the true label y_n of x_n is included in our prediction set Γ^ϵ.

As a class of confidence predictors, conformal predictors also satisfy the criterion of validity. Algorithm 1 is a pseudo-code for a conformal predictor. In the algorithm, NCM is a function that calculates non-conformity scores based on underlying algorithms. Different implementations of NCM will be introduced in next chapter.

Algorithm 1: Conformal predictor

Data: training set $\mathbf{Z}^{(n-1)}$, testing object x_n, significance level ϵ

Result: prediction set Γ^ϵ

$\Gamma^\epsilon \leftarrow \varnothing$; `/* initial a empty prediction set */`

$K \leftarrow |\mathbf{Y}|$; `/* get the size of label space */`

for $y = 0$ **to** $K - 1$ **do** `/* try every possible label */`

  for $i = 1$ **to** n **do** `/* for each label calculate all NCM scores */`

    $\alpha_i \leftarrow$ `NCM(`$\wr(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y)\int,\ (x_i, y_i)$`);`

    `/* calculate non-conformity score for every example */`

  end

  $c \leftarrow 0$; `/* initial the counter to 0 */`

  for $i = 1$ **to** n **do** `/* count how many NCM score` $\geq a_n$ `*/`

    if $\alpha_i \geq \alpha_n$ **then**

      $c \leftarrow c + 1$;

    end

  end

  $p_y \leftarrow \frac{c}{n}$; `/* calculate p-value */`

  if $p_y > \epsilon$ **then**

    $\Gamma^\epsilon \leftarrow$ `AddToSet(`$\Gamma^\epsilon,\ y$`);`

    `/* add current label y to prediction set */`

  end

end

return Γ^ϵ

2.3 Venn Prediction

Conformal predictors give p-values as estimates of reliability for the predictions, which are not direct probabilities. Although we will introduce an algorithm to transform the outcome of conformal predictors to probabilities, making a predictor give probabilities directly will be more intuitive.

There is a variety of machine learning algorithms that can make probabilistic predictions, which is to assign a probability distribution of the label for a new object. This kind of algorithm is called *probabilistic predictors*. However, most of these predictors are based on strong statistical assumption, which is not suitable for data sets from the real world. Hence, they fail to give valid predictions when these assumptions are not satisfied.

Venn prediction allows us to give valid probability distributions under i.i.d. assumption only. It was firstly introduced in [71] and were discussed in detail in [69, Chapter 6]. Furthermore, Venn predictors produce a set of probability distributions for a predicted label, which can also be interpreted as a bounded interval of probabilities that the predicted label is true.

To introduce Venn predictors, we begin with the assumptions we make in this section. We follow most of the assumptions given in §2.1, additionally there are two more assumptions:

1. the label space $\mathbf{Y}$ is finite, which means that we only deal with classification problem.

2. the sequence of examples $z_1, z_2, \ldots$ is finite instead of continuing indefinitely. Assume the number of examples we observed is N.

Given a binary classification problem (we use problem with binary classes only for simplicity). The possible labels are denoted by $\{0, 1\}$ and a probabilistic predictor announces probability p_n for the event $y_n = 1$. Suppose we are observing examples in sequence, which means we make probabilistic prediction p_n after we observed a new object x_n and before we observe the true label for the new object x_n. This protocol is called *online set-*

ting and it is useful for demonstrating the ideas about testing calibration of probabilistic predictions.

We want know whether the predicted probability p_n tends to be too high or too low overall. This can be tested by comparing the average predicted probabilities with the overall frequency of "1" among N examples. The average probability is as follows.

$$\bar{p}_N := \frac{1}{N}\sum_{n=1}^{N} p_n \tag{2.14}$$

Also, the overall frequency of "1" is

$$\bar{y}_N := \frac{1}{N}\sum_{n=1}^{N} y_n \tag{2.15}$$

If

$$\bar{y}_N \approx \bar{p}_N \tag{2.16}$$

we say the predictions are "unbiased on average". If $\bar{p}_N$ is distinguished with $\bar{y}_N$, we say the predictions is biased on average.

In a more refined version, we test the calibration in a subset of n examples for which p_n is close to a given value p^*. We compare p^* with the frequency of "1" in this subset, say $\bar{y}_N(p^*)$. If

$$\forall p^*,\ \bar{y}_N(p^*) \approx p^* \tag{2.17}$$

then the p_n can be considered "well calibrated", in other words, have a frequentist justification. John Venn [66] was one of the first writers on frequentist probability, and Venn prediction was named after him.

However, to have a good calibration is not enough to make useful probabilistic predictions. Assume y_n equals to 1 if n is odd and 0 otherwise, hence $\bar{y}_n \approx 0.5$. We can set $p_n = 0.5$ to achieve excellent calibration, but this makes the probabilistic predictions meaningless. To overcome this, what we need in addition to good calibration is sometimes called "high resolution". This term suggests that we introduce a procedure that is used in our Venn prediction. The predictor divides objects into categories and uses the frequencies in the category that contains x_n as the probabilistic prediction p_n. The more

numerous the categories - the more information predictor takes into account in creating the categories - the greater the resolution and the more useful the probabilistic prediction should be. Back to the previous example, we can sort all examples into two categories rather than one whole category. One category is for odd numbers and the other is for even numbers. Then we predict "1" as an odd number and "0" as an even number since they fall into different categories.

Suppose we are given a binary classification problem $\wr z_1, z_2, \ldots, z_{n-1}\int$ and a new object x_n. The label space $\mathbf{Y} = \{0, 1\}$. Before we introduce Venn predictors, we give the definition of the procedure that partitions objects into categories as we mentioned. A *taxonomy* is a method that can divide objects into categories by using underlying algorithms. Practically any known machine learning algorithm can be used as an underlying algorithm, which is very similar to conformal predictors. Therefore, Venn predictors are a framework of machine learning algorithms rather than a single algorithm.

Let a taxonomy for $\wr z_1, z_2, \ldots, z_{n-1}, (x_n, y)\int$ be A_n, where $y \in \mathbf{Y}$. The category τ_i assigned to example z_i by A_n is defined in Eq. (2.18).

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_{n-1}, (x_n, y)\int, z_i) \tag{2.18}$$

Example z_i and z_j are in the same category if and only if $\tau_i = \tau_j$. The Venn predictor associated with this Venn taxonomy A_n outputs the pair (p_0, p_1) as its prediction for x_n's label, where

$$p_y := \frac{|\{(x^*, y^*) \in \wr z_1, \ldots, z_{n-1}, (x_n, y)\int : \tau^* = \tau_n \wedge y^* = y\}|}{|\{(x^*, y^*) \in \wr z_1, \ldots, z_{n-1}, (x_n, y)\int : \tau^* = \tau_n\}|} \tag{2.19}$$

for both $y \in \{0, 1\}$. The denominator is always positive, since $(x^*, y^*) = (x_n, y) \Longrightarrow \tau^* = \tau_n$. Intuitively, p_0 is the proportion of examples with label 0 to all examples of the same category of x_n. p_1 is the proportion of examples with label 1. p_0 and p_1 are the predicted probabilities that the label of x_n is 1. Hence, we call a Venn predictor *multi-probabilistic predictor*. Only when $p_0 \approx p_1$, these predictions are useful.

Venn predictors have an important property of validity: they are automatically well calibrated. The property of validity of Venn predictors allows them to produce valid

results when the examples are drawn independently from the same probability distribution without knowing what the distribution is, which seems more useful and applicable to the real-world problems. This property of validity was proofed in [69, Theorem 6.6] if we satisfy the exchangeability assumption. According to Theorem 6.6, the validity does not depend on the size of the data sequence N. Therefore, we can drop the second additional assumption we made at the beginning, which is the sequence of examples is finite. However, this known version proof is complicated, Vovk proposed a much simpler version in [70, Theorem 1] and the intuition behind it is more transparent.

The shortcoming of this procedure is that it may be too special for some specific data sets. We need to notice that more categories do not always lead to better results. So rather than speak of resolution, we will speak of efficiency. For efficiency, it requests the probabilities that a predictor gives to be more informative, which are as close to 0 or 1 as possible.

Venn predictors are suitable for multi-class problems as well. The generalized idea of Venn predictors can be described as follows: firstly we divide known objects (labels are not included here, hence we only use the information from features) into several categories by using a taxonomy. The new object is also assigned to one of these categories in the same way. Then we calculate the frequencies of labels in the category that contains the new object as the probability distribution for the new object's label. We try this several times for every possible label, which is analogous to the way we treat the new object when we use a nonconformity measure to define a conformal predictor. At last, we announce a set of probability distributions for the unknown label of the new object.

In the following, we give the explicit definition of Venn prediction. Assuming a standard machine learning classification problem: given a training set of examples $\wr z_1, z_2, \ldots, z_{n-1}\int$. Each z_i consists of a pair of object x_i and label y_i. The label space $\mathbf{Y}$ is finite. We are also given a test object x_n. Our task is to predict the label y for the new object x_n and give the estimation of the likelihood that our prediction is correct.

Suppose we have a taxonomy for n objects $x_1, \ldots, x_n$ that divides these objects into

different categories in a specific way, say A_n. Note that it is different from nonconformity measures in Conformal prediction. Nonconformity measures and taxonomies play a similar role in Conformal prediction and Venn prediction respectively. Hence we use the same symbol here.

We also consider a arbitrary label $y \in \mathbf{Y}$ for the new object x_n. A_n assigns a category τ_i to an example z_i

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\wr, z_i) \tag{2.20}$$

where n is the number of elements in the bag, $\tau_i \in \mathbf{T}$ is one of the finite categories from the category space $\mathbf{T}$.

Moreover, we assign z_i and z_j to the same category if and only if

$$A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n\wr, z_i) = A_n(\wr z_1, \ldots, z_{j-1}, z_{j+1}, \ldots, z_n\wr, z_j) \tag{2.21}$$

The category τ_n is assigned to (x_n, y). Let $\mathbf{P}_y$ be the empirical probability distribution of the labels in category τ_n. $\mathbf{P}_y$ contains K frequencies of all possible labels.

$$\mathbf{P}_y := \frac{|\{(x^*, y^*) \in \wr z_1, \ldots, z_{n-1}, (x_n, y)\wr : \tau^* = \tau_n \wedge y^* = y'\}|}{|\{(x^*, y^*) \in \wr z_1, \ldots, z_{n-1}, (x_n, y)\wr : \tau^* = \tau_n\}|} \quad y' \in \mathbf{Y} \tag{2.22}$$

$\mathbf{P}_y$ is a probability distribution on $\mathbf{Y}$.

Having tried every possible label for x_n, we get a $K \times K$ frequency matrix $\mathbf{P}$. The Venn predictor $\mathbf{P} := \{\mathbf{P}_y : y \in \mathbf{Y}\}$ is a multi-probabilistic predictor that gives K probability distributions, where K is the number of labels in this label space.

To interpret this prediction, let the *quality* of a column be the minimum entry of the column in matrix $\mathbf{P}$. Let the *best* column, which has the highest quality, be j_{best}. Then our predicted label is j_{best} and the interval of possibilities that our prediction is correct is defined as:

$$[\min_{i=0,\ldots,K-1} \mathbf{P}_{i,j_{best}} \ , \ \max_{i=0,\ldots,K-1} \mathbf{P}_{i,j_{best}}] \tag{2.23}$$

In Algorithm 2, the pseudo-code for a Venn predictor is given.

Algorithm 2: Venn predictor

Data: training set $\mathbf{Z}^{(n-1)}$, testing example x_n

Result: predicted label $\hat{y}_n$, probability interval $[l_n, u_n]$

$K \leftarrow |\mathbf{Y}|$; `/* get the size of label space */`

for $y = 0$ **to** $K - 1$ **do** `/* try every possible label */`

  for $i = 1$ **to** n **do** `/* assign a category for every example */`

    $\tau_i \leftarrow$ `Taxonomy(`$\wr(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y)\int, \ (x_i, y_i)$`);`

  end

  $\mathcal{C} \leftarrow \varnothing$; `/* initial a empty category set */`

  for $i = 1$ **to** n **do**

    if $\tau_i = \tau_n$ **then** `/* find examples in the category that` x_n `falls */`

      $\mathcal{C} \leftarrow$ `AddToSet(`$\mathcal{C}, \ y_i$`)` `/* add its label to category set */`

    end

  end

  $\mathbf{P}_y \leftarrow$ `CalcFrequency(`$\mathcal{C}$`)` ; `/* calculate the frequency of labels */`

end

$j_{best} \leftarrow$ `FindBestColumn(`$\mathbf{P}$`)` ; `/* find the column with highest quality */`

$[l_n, u_n] \leftarrow$ `FindInterval(`$j_{best}, \ \mathbf{P}$`)` ; `/* find max and min in best column */`

$\hat{y}_n \leftarrow j_{best}$;

return $\hat{y}_n, \ [l_n, u_n]$

Function CalcFrequency($\mathcal{C}$)

```
Function CalcFrequency(C)
    total ← SizeOf(C) ;                    /* get size of category set */
    count ← 0 ;                            /* initial count array to 0 */
    for i = 1 to total do                  /* for every label in the set */
        count[y_i] ← count[y_i] + 1 ;
    end
    return count
end
```

Function FindInterval($j_{best},\ \mathbf{P}$)

```
Function CalcFrequency(j_best, P)
    lower ← 1 ;                            /* initial lower to maximal value 0 */
    upper ← 0 ;                            /* initial upper to minimal value 0 */
    for i = 1 to K do                      /* for every value in column j_best */
        if lower > P_{i,j_best} then       /* find current minimal value */
            lower ← P_{i,j_best};
        end
        if upper < P_{i,j_best} then       /* find current maximal value */
            upper ← P_{i,j_best};
        end
    end
    return [lower, upper]
end
```

Function FindBestColumn($\mathbf{P}$)

```
Function FindBestColumn(P)
    max ← 0 ;                                  /* initial max to minimal value 0 */
    index ← 0 ;                                          /* initial index to 0 */
    for j = 1 to K do                          /* for every column in the matrix */
        quality ← 1 ;                      /* initial quality to maximum value 1 */
        for i = 1 to K do                        /* find the minimal in column j */
            if P_{i,j} < quality then
                quality ← P_{i,j};
            end
        end
        if quality > max then                          /* compare quality to max */
            max ← quality ;
            index ← j;
                 /* record value and index if quality greater than max */
        end
    end
    return index
end
```

2.4 Underlying Algorithms

As we mentioned before, Conformal prediction and Venn prediction are algorithmic frameworks rather than simple predictors. Generally, a Conformal predictor or Venn predictor can be built on top of almost any machine learning algorithm [69, p. 11]. These algorithms are referred to as *underlying algorithms*. In the following subsections, we will introduce some most widely-used underlying algorithms and all the underlying algorithms we used in this thesis.

2.4.1 *k*-Nearest Neighbours

Among all the methods in machine learning, k-Nearest Neighbours (k-NN) is one of the most fundamental and simple algorithms for both classification and regression.

k-Nearest Neighbours algorithm was developed to fulfil the demand of performing discriminatory analysis when reliable parametric estimations of probability densities are unknown or difficult to determine. In 1951, a non-parametric method for pattern recognition was introduced by Fix and Hodges in [18], which has become known as the k-Nearest Neighbours rule. Later in 1967, Cover and Hart worked out one of the formal properties of the k-Nearest Neighbours rule in [11]. It was shown in the paper that for an infinite example set the probability of error of the rule is bounded above by twice the Bayes probability of error when $k = 1$. These established the foundations of k-Nearest Neighbours and led to a long series of investigations.

The basic idea behind k-NN algorithm is based on the closest training examples in the feature space: an unknown object is classified the same as the most frequent label through majority voting among its k nearest neighbours. When we mentioned the closest or the nearest example, it refers to the example that has the smallest distance metric between the testing example and the specified training example. The commonly used distance metric is the geometric distance, which is also known as Euclidean distance. However, there are other choices such as Hamming distance introduced in [25] and Tangent distance

introduced in [57] in cases of different problems.

With 1-Nearest Neighbour rule, the predicted label $\hat{y}_n$ for testing example x_n is shown as follows:

$$\hat{y}_n := y_j, \quad \text{where } j = \underset{i \in \{1,\ldots,n-1\}}{\operatorname{argmin}} ||x_n - x_i|| \tag{2.24}$$

The following is a pseudo-code for k-Nearest Neighbours classification method.

Algorithm 3: k-Nearest Neighbours classifier

Data: training set $\mathbf{Z}^{(n-1)}$, new object x_n, number of neighbours k

Result: predicted label $\hat{y}_n$

for $i = 1$ **to** $n - 1$ **do** `/* calculate distances between` x_n `and` x_i `*/`

  $d_i \leftarrow$ `CalcDistance(`x_n, x_i`);`

end

neighbours $\leftarrow$ `FindKMinimal(`d, k`);`

`/* find k minimal values and record the indices */`

$y' \leftarrow$ `Vote(`*neighbours*, $\mathbf{Y}$`)` ; `/* vote for the majority label */`

$\hat{y}_n \leftarrow y'$;

return $\hat{y}_n$

2.4.2 Support Vector Machine

Support vector machines (SVMs) are a popular machine learning algorithm for classification. The related subject of support vector machines can be said to have started in late 1970s [64]. The original SVMs algorithm was invented by Vladimir N. Vapnik, which was devoted to solving linearly separable binary classification problems. The current standard version of SVMs, which is also suitable for the non-linearly separated data sets, was proposed by V. Vapnik and C. Cortes. The formal descriptions and definitions were given in [10] and [65]. After several years of research, this method has become capable to deal with multi-class classification, regression and many other machine learning tasks. It has become one of the most widely used machine learning algorithms.

The basic idea behind support vector machines is to construct a hyperplane in a high-dimensional or infinite-dimensional feature space that could separate all the examples into two categories. There are many hyperplanes that might separate the examples. Intuitively, one reasonable good separation is the largest separation represented by the hyperplane that has the largest distance to the nearest training examples of both sides. This hyperplane is called the *maximum-margin hyperplane*. Then a new object is assigned by this classifier to a category based on which side of the hyperplane it falls on.

The detailed descriptions of support vector machines are as follows: we follow the same definitions and descriptions in our previous section §2.1. We are given a training set $\mathbf{Z}^{(n-1)}$ and a new object x_n, and our goal is to predict $\hat{y}_n$ for the new object. Specifically, each $x_i \in \mathbf{X}$ is a d-dimensional vector i.e. the $\mathbf{X} \subset \mathbf{R}^d$, and the label space only has two elements i.e. $\mathbf{Y} = \{-1, +1\}$.

Suppose we have some hyperplane H_0 that separates all the examples. This hyperplane can be written as the set of points $\mathbf{x}$ satisfying

$$H_0 : \mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.25}$$

where $\mathbf{w}$ is normal to the hyperplane and $\mathbf{w} \neq 0$. We define d_+ and d_- to be the perpendicular distances from the nearest positive and negative examples to the hyperplane respectively. For the linearly separable case, we suppose all the training examples satisfy the following constraints:

$$\mathbf{w} \cdot \mathbf{x_i} + b \geq +1 \quad \text{for } y_i = +1 \tag{2.26}$$

$$\mathbf{w} \cdot \mathbf{x_i} + b \leq -1 \quad \text{for } y_i = -1 \tag{2.27}$$

Furthermore, these constraints can be written together as

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 \quad \forall i = 1, \ldots, n-1 \tag{2.28}$$

Now we have two more hyperplanes: H_1:$\mathbf{w} \cdot \mathbf{x_i} + b = +1$ and H_2:$\mathbf{w} \cdot \mathbf{x_i} + b = -1$. Note that H_1 and H_2 are parallel since they share the same normal $\mathbf{w}$ and that no training

points fall between them. Hence d_+ is the perpendicular distance between H_0 and H_1 and equals to $\frac{1}{||\mathbf{w}||}$, while d_- is the perpendicular distance between H_0 and H_2 and equals to $\frac{1}{||\mathbf{w}||}$ as well. We define the *margin* of hyperplane H_0 to be $d_+ + d_-$, simply $\frac{2}{||\mathbf{w}||}$. As we want to find the maximum-margin hyperplane, we need to maximize the margin $\frac{2}{||\mathbf{w}||}$, which is the same as to minimize $||\mathbf{w}||$, subjects to constraints (2.28). Consequently we can now transform this problem into a standard quadratic programming optimization problem as in Eq. 2.29:

$$\begin{aligned} &\frac{1}{2}||\mathbf{w}||^2 \rightarrow \min \\ &s.t.\ \forall i = 1, \ldots, n-1, \quad y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 \end{aligned} \tag{2.29}$$

By solving this quadratic programming optimization problem, we can predict $\hat{y}_n$ as $\text{sgn}(\mathbf{w} \cdot \mathbf{x_i} + b)$, where $\text{sgn}(\cdot)$ is the *sign function*. Eq. (2.30) is called the *decision function*, and the value $f(x_i)$ for x_i is called the *decision value*. Since decision function contains all the information on the solution of the optimization problem, usually $f(\cdot)$ represents an SVM classifier.

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{2.30}$$

For linearly non-separable case, non-negative slack variables ξ_i will be introduced as V. Vapnik and C. Cortes suggested in [10]. This modification allows for mislabelled examples, and still tries to choose a hyperplane that splits the examples as cleanly as possible. Because some mislabelled examples will fall on the other side of the category, which means the hyperplane will not clearly separate all the examples into two categories, this modification is called *soft margin* method. The constraints (2.28) will be replaced by the following constraints:

$$\begin{aligned} y_i(\mathbf{w} \cdot \mathbf{x_i} + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \quad \forall i = 1, \ldots, n-1 \end{aligned} \tag{2.31}$$

Thus, for a training error to occur, the corresponding ξ_i must exceed unity. Hence, $\sum_{i=1}^{n-1} \xi_i$ is an upper bound on the training errors. We need to take this extra cost for errors into account. By introducing parameter C as a trade-off between a large margin and a

small error penalty, the objective function changes from minimizing $||\mathbf{w}||$ to minimizing $||\mathbf{w}|| + C\sum_i \xi_i$. C is chosen by the user, a larger C corresponding to assigning a higher penalty to errors. The new quadratic programming optimization problem can be written as:

$$\begin{aligned} &\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n-1}\xi_i \rightarrow \min \\ s.t.\ \forall i = 1,\ldots,n-1, \quad &\xi_i \geq 0, \quad y_i(\mathbf{w}\cdot\mathbf{x_i} + b) \geq 1 - \xi_i \end{aligned} \tag{2.32}$$

Especially while $C = \infty$, all ξ_i must equal to zero, which makes the optimization problem for non-separable case (2.32) the same as the problem for linearly separable case (2.29).

To make the optimization problem easy to handle and solve, we will transform this prime form to Wolfe dual form [76] by introduce some non-negative Lagrange multipliers α_i for constraint $-y_i(\mathbf{w}\cdot\mathbf{x_i} + b) + 1 - \xi_i \leq 0$ in (2.32) and r_i for constraint $-\xi_i \leq 0$ in (2.32). In this reformulation of the problem, the new optimization problem is as follows:

$$\begin{aligned} &\sum_{i=1}^{n-1}\alpha_i - \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} y_i y_j \alpha_i \alpha_j (\mathbf{x_i}\cdot\mathbf{x_j}) \rightarrow \max \\ s.t.\ &\sum_{i=1}^{n-1} y_i\alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \forall i = 1,\ldots,n-1 \end{aligned} \tag{2.33}$$

Notice that the slack variables ξ_i and Lagrange multipliers r_i for ξ_i cancel themselves out in formula Eq. (2.33). Additionally, by applying KarushKuhnTucker conditions, which were published by and named after Harold W. Kuhn, Albert W. Tucker [36] and William Karush [33], we have some more complementarity conditions:

$$\forall i = 1,\ldots,n-1, \quad (\alpha_i - C)\xi_i = 0 \tag{2.34}$$

$$\alpha_i(-y_i(\mathbf{w}\cdot\mathbf{x_i} + b) + 1 - \xi_i) = 0 \tag{2.35}$$

The most important advantage of dual form is that the slack variables ξ_i vanish from the problem, appearing only in the constraints, while the constant C appears only as an additional constraint on the Lagrange multipliers.

Suppose $\hat{\alpha} = (\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{n-1})$ solve this optimization problem. Thus, if $0 < \hat{\alpha}_i < C$, we get $\xi_i = 0$ and $-y_i(\mathbf{w}\cdot\mathbf{x_i} + b) + 1 = 0$. We could solve b in the decision function

Eq. (2.30) as:

$$\hat{b} := \frac{1}{y_i} - \hat{\mathbf{w}} \cdot \mathbf{x_i} \tag{2.36}$$

The normal $\mathbf{w}$ in the decision function Eq. (2.30) could be obtained from the derivation of the Lagrangian for problem (2.32), which is as follows:

$$\hat{\mathbf{w}} := \frac{1}{2}\sum_{i=1}^{n-1} \alpha_i y_i \mathbf{x_i} \tag{2.37}$$

Now we have our support vector machines classifier, and we could use Eq. (2.30) to assign a label to the new object x_n.

Apparently, the sum in Eq. (2.37) contains only x_i such that $\alpha_i \neq 0$. Such x_i in training set are called *support vectors*. Furthermore, there are two types of support vectors: those with $0 < \hat{\alpha}_i < C$ and those with $\hat{\alpha}_i = C$. If $0 < \hat{\alpha}_i < C$, we get the slack variable $\hat{\xi}_i = 0$, which means the vector is on the margin. If $\hat{\alpha}_i = C$, we can have the slack variable $\hat{\xi}_i > 0$ (i.e. the positive slack variable), which means the vector is inside the margin.

Kernel Trick The optimization problem (2.33) we introduced before is a linear classifier. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to construct non-linear classifiers by applying the *kernel trick* to the problem in [6]. Kernel trick was originally proposed by A. Aizerman et al. in [2]. To use the kernel trick in the dual form of support vector machines is astonishingly straightforward: every dot product of examples $(x_i \cdot x_j)$ in (2.33) is replaced by a non-linear kernel function $\mathcal{K}(x_i, x_j)$. This maps all examples from their original feature space to a user-defined feature space (usually higher-dimensional than the original one or even infinite-dimensional). The mapping allows support vector machines to find a maximum-margin hyperplane in a transformed feature space by computing values of kernel function instead of computing the dot product of the examples in that space. This operation is often computationally cheaper than the explicit computation of the coordinates.

Some common kernel functions include:

- **Polynomial function**:

$$\mathcal{K}(\mathbf{x_i}, \mathbf{x_j}) = (\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)^d \tag{2.38}$$

  The special case for this kernel function is when $\kappa = 1$, $c = 0$ and $d = 1$, it becomes $\mathcal{K}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}$, which is exactly the dot product of examples.

- **Radial basis function**:

$$\mathcal{K}(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2) \tag{2.39}$$

  This kernel function is very popular among all kernel functions. The feature space of this kernel has an infinite number of dimensions.

- **Hyperbolic tangent function**:

$$\mathcal{K}(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c), \tag{2.40}$$

After choosing a user-specific kernel function and setting the parameter C and the parameters for kernel function, the final optimization problem for support vector machines with kernel trick can be written as:

$$\begin{aligned} &\sum_{i=1}^{n-1} \alpha_i - \frac{1}{2} \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} y_i y_j \alpha_i \alpha_j \mathcal{K}(\mathbf{x_i}, \mathbf{x_j}) \rightarrow \max \\ s.t. \ &\sum_{i=1}^{n-1} y_i \alpha_i = 0, \ 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, n-1 \end{aligned} \tag{2.41}$$

By applying the kernel function, the SVM classifier also reformulates to Eq. (2.42)

$$\hat{y} := \operatorname{sgn}(\frac{1}{2} \sum_{i=1}^{n-1} \hat{\alpha}_i y_i \mathcal{K}(\mathbf{x_i}, \mathbf{x_n}) + \hat{b}) \tag{2.42}$$

where

$$\hat{b} := \frac{1}{y_i} - \frac{1}{2} \sum_{k=1}^{n-1} \hat{\alpha}_k y_k \mathcal{K}(\mathbf{x_i}, \mathbf{x_k}) \tag{2.43}$$

The pseudo-code of the algorithm for solving this dual form optimization problem by using sequential minimal optimization (invented by John Platt in [47]) is listed in Algorithm 4.

Algorithm 4: Binary Support Vector Machines classifier

Data: label array y of all training examples

matrix $Q : Q_{ij} = y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$

stopping tolerance ϵ

Result: decision function f

$\alpha_i \leftarrow 0$; `/* initial Lagrange multipliers */`

$G_i \leftarrow -1$; `/* initial gradients */`

while *TRUE* **do**

  $(i, j) \leftarrow$ `SelectWorkingSet(`y, G, Q`);`

  `/* select two Lagrange multipliers to solve */`

  if *reach stopping tolerance* **then**

    break;

  end

  $\alpha_i \leftarrow$ `UpdateAlpha(`i`)` ; `/* update chosen alpha i */`

  $\alpha_j \leftarrow$ `UpdateAlpha(`j`)` ; `/* update chosen alpha j */`

  $G \leftarrow$ `UpdateGradient(`α_i,α_j`)` ; `/* reconstruct gradients */`

end

$\hat{b} \leftarrow$ `CalculateB()` ; `/* calculate offset b */`

return $f \leftarrow (\hat{\alpha},\ \hat{b})$

Function SelectWorkingSet(y, G, Q)

```
Function SelectWorkingSet(y, G, Q)
    i ← argmax_i {−y_i G_i};
    j ← argmin_j {−(y_j G_j − y_i G_i)^2 / (Q_ii + Q_jj − 2 y_i y_j Q_ij)};
    if y_j G_j − y_i G_i < ε then                    /* reach stopping tolerance */
        return −1
    else
        return i, j
    end
end
```

2.4.3 Multi-class SVM

Support vector machines (SVMs) is a very well developed technique and is becoming a popular algorithm in almost all fields of machine learning. However, this algorithm was originally designed for binary classification, that is, the class labels can only take two values. The lack of capability dealing with multi-class problems would hinder this algorithm being implemented in more applications. The research on how to effectively extend support vector machines for multi-class classification has never stopped.

Currently, there are two types of approaches solving multi-class SVM. One is by decomposing multi-class SVM into several binary-class SVM sub-problems while the other is by directly considering all examples in a single optimization formulation. Popular methods for using a combination of several binary SVM classifiers to solve a given multi-class problem are: one-versus-all SVM, one-versus-one SVM, error-correcting output codes [15] and directed acyclic graph SVM (DAGSVM) [49]. Common methods for constructing a single classifier for multi-class problem include: K. Crammer and Y Singer's Multi-class SVM [12] [13] and J. Weston and C. Watkins's Multi-class SVM [74], [73]. In the following, there are some descriptions of the algorithms mentioned above.

One-versus-all SVM is probably the earliest used implementation for SVM multi-class

classification [7]. This approach constructs a set of K binary classifiers, $f_0, f_1, \ldots, f_{K-1}$, where K is the number of classes. Each binary classifier is trained with examples in some class as positive examples and all the other examples as negative examples. In other words, the ith SVM classifier f_i is trained with all the examples of the ith class as positive, and all other examples as negative examples. We then combine them to get a multi-class classifier according to the largest output before applying the sign function. The predicted label $\hat{y}_n$ for new object x_n is described in Eq. (2.44):

$$\hat{y}_n := \underset{i=0,\ldots,K-1}{\operatorname{argmax}} f_i(x_n) \tag{2.44}$$

This strategy is also called the *winner-takes-all* strategy.

Another popular solution for multi-class SVM is one-versus-one SVM. The One-versus-one method was introduced in [34] and firstly used on SVM in [21] and [35]. This method constructs a set of $\frac{K(K-1)}{2}$ binary classifiers on examples from each pair of classes. For examples from the ith and the jth classes, the binary SVM classifier is referred to as f_{ij}. In addition, when we train all classifiers, we always take the examples of the former as positive examples and the latter as negative examples. Hence, f_{ij} is basically the same as f_{ji} except the labels are opposite, that is, $f_{ij}(x) = -f_{ji}(x)$. After we trained all the classifiers, we assign a label $\hat{y}_n$ to the new object x_n through the following voting strategy suggested in [21]: if the classifier f_{ij} predicts x_n is in the ith class then the vote goes for ith class. Otherwise, the vote goes for jth class. This process is repeated for all binary classifiers. Then we predict $\hat{y}_n$ the class with the majority of votes:

$$\hat{y}_n := \underset{i=0,\ldots,K-1}{\operatorname{argmax}} \sum_{j=0, j\neq i}^{K-1} \mathcal{V}(f_{ij}(x_n)), \quad \mathcal{V}(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{2.45}$$

$\mathcal{V}(x)$ is the voting function that gives "1" for voting and "0" for not voting as output. If two labels have equal votes, we take the first appeared one as our predicted label. This voting strategy is called the *max-wins-voting* strategy.

Directed acyclic graph SVM (DAGSVM) [49] is a method that shares the same idea with One-versus-one method. This method also trains a set of $\frac{K(K-1)}{2}$ classifiers on each

pair of classes. However, the difference is that DAGSVM predicts the labels of new objects through a rooted binary directed acyclic graph that uses the $\frac{K(K-1)}{2}$ classifiers as its nodes. This kind of acyclic graph starts with a rooted node, and each node points to two other nodes or two possible labels. At each node, the new object x_n will be carried out on the binary SVM classifier that the node associated with, the outcome of this SVM classifier determines which direction it goes. Thus the testing example x_n goes through a path from the rooted node to a possible label. Since one possible class will be eliminated at each node, only $(K-1)$ SVM classifiers will be carried out to predict $\hat{y}_n$ for x_n. Therefore, the time cost of this method will be reduced comparing to One-versus-one SVM, which will carry out all $\frac{K(K-1)}{2}$ classifiers to predict a label. In addition, this method also solves the problem arises in the max-wins-voting strategy, that is, there may be two possible classes have the same votes.

SVM using error-correcting output codes is also a method to decompose multi-class SVM into a series of binary SVM classifiers. Theoretically speaking, the error-correcting output codes (ECOC) is a powerful framework to deal with the multi-class problem, which can be deployed with any binary classification algorithms. It was developed by Dietterich and Bakiri in 1995 [15]. The basis of the ECOC framework consists of representing each class by a designed code. The codes encode the category information of each class for a given binary problem. Additionally, they are sequences of binary digits "0" and "1", or in SVM "-1" and "$+1$" to make it meaningful. Since each bit of the codeword is a binary digit, we concatenate the feature vector x_i and each bit of the codeword as the true label of x_i to construct several binary SVM classifiers. To classify a new object x_n, every classifier trained before is evaluated to produce a new codeword. This is then mapped to the nearest of all the possible codewords. When we say "nearest" here, we usually use the *Hamming distance* introduced by Richard Hamming in [25] as the measurement. We assume the codeword of k-bits length for each class is $W_i = (w_{i,0}, \ldots, w_{i,k-1})$, and the

predicted codeword is $B = (b_0, \ldots, b_{k-1})$. The predicted label can be given as follows:

$$\hat{y}_n := \underset{i=0,\ldots,K-1}{\operatorname{argmin}} \sum_{j=0}^{k-1} |b_j - w_{i,j}| \tag{2.46}$$

The advantage of using this method is that by encoding each label into an error-correcting code and training each bit separately, the classifier may be able to recover from the errors. If the minimal Hamming distance between any pair of known codewords is d, then this algorithm can at least correct errors in $\lfloor \frac{d-1}{2} \rfloor$ bits of the predicted codeword B. This is because the minimal Hamming distance between the predicted codeword B and any other known codewords W_i is $d - \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{d-1}{2} \rfloor + 1 > \lfloor \frac{d-1}{2} \rfloor$, which means the nearest codeword of B is still the correct one.

2.4.4 Crammer and Singer's Multi-class SVM

K. Crammer and Y. Singer [13] proposed a multi-class SVM method that casts the multi-class classification problem into a single optimization problem, rather than decomposing it into multiple binary classification problems.

We follow the same problem descriptions given in §2.1 when introducing this method. We are given $(n-1)$ training examples $(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$, which are classified into K different classes. Each x_i is a d-dimensional vector that carries d attributes. The set of possible labels $\mathbf{Y} = \{0, 1, \ldots, K-1\}$. We are also given a new object x_n, and our mission is to predict which class $\hat{y}_n$ this object belongs. In other words, we want to construct a multi-class classifier $H : \mathbf{X} \to \mathbf{Y}$ that maps an example x_i from the feature space $\mathbf{X}$ to an possible label y_i from the label space $\mathbf{Y}$. In Crammer and Singer's algorithm, it aims to construct a classifier of the form

$$H_{\mathbf{M}}(\mathbf{x}) := \underset{r=0}{\overset{K-1}{\operatorname{argmax}}} \{\mathbf{M}_r \cdot \mathbf{x}\} \tag{2.47}$$

where $\mathbf{M}$ is a matrix of size $K \times d$ and $\mathbf{M}_r$ is the rth row of $\mathbf{M}$. And we interchangeably call the value of $\mathbf{M}_r \cdot \mathbf{x}_i$ the *similarity score* for class r. In this way, we predict the label $\hat{y}_n$ the same as the index of the row in $\mathbf{M}$ that achieves the highest similarity score with

$\mathbf{x}_n$.

The starting point of building this multi-class classifier is to generalize the notion of margin to multi-class problems. We can do this by introducing the following piecewise linear bound

$$\max_r \{\mathbf{M}_r \cdot \mathbf{x}_i + 1 - \delta_{y_i,r}\} - \mathbf{M}_{y_i} \cdot \mathbf{x}_i \tag{2.48}$$

where $\delta_{p,q}$ is equal to 1 if $p = q$ and 0 otherwise. Therefore, Eq. (2.48) is zero if the similarity score $\mathbf{M}_{y_i} \cdot \mathbf{x}_i$ for the true label y_i is larger by at least one than the scores assigned to the rest of the labels. If the similarity score for the true label is larger than any other scores by a value less than one, we suffer a loss on the margin. If the similarity score for the true label is not the largest score among all scores, we then have a misclassified example. This is the generalized notion of margin for multi-class problems.

In addition, we can calculate an upper bound on the empirical loss by summing all examples in $\mathbf{Z}$ together,

$$\epsilon_{\mathbf{Z}}(\mathbf{M}) \leq \frac{1}{n-1} \sum_{i=1}^{n-1} \Big[\max_r \{\mathbf{M}_r \cdot \mathbf{x}_i + 1 - \delta_{y_i,r}\} - \mathbf{M}_{y_i} \cdot \mathbf{x}_i \Big] \tag{2.49}$$

According to the definition of linearly separable, the above loss is equal to zero for all examples, that is,

$$\forall i, \quad \max_r \{\mathbf{M}_r \cdot \mathbf{x}_i + 1 - \delta_{y_i,r}\} - \mathbf{M}_{y_i} \cdot \mathbf{x}_i = 0 \tag{2.50}$$

If a matrix $\mathbf{M}$ satisfies Eq. (2.50), it would also satisfy the following constraints,

$$\forall i, r, \quad \mathbf{M}_{y_i} \cdot \mathbf{x}_i + \delta_{y_i,r} - \mathbf{M}_r \cdot \mathbf{x}_i \geq 1 \tag{2.51}$$

If we define the norm of a matrix $\mathbf{M}$ the same as the sum of the norm of all row vectors in this matrix $||\mathbf{M}||_2^2 = ||(\mathbf{M}_0, \ldots, \mathbf{M}_{K-1})||_2^2 = \sum_{i,j} \mathbf{M}_{i,j}^2$, based on the previous work [49] and [13], we would like to find a matrix $\mathbf{M}$ of a small norm and also subjects to Eq. (2.51). Therefore, this optimization problem for linearly separable case can be represented in a standard quadratic programming optimization form,

$$\begin{aligned} &\frac{1}{2}||\mathbf{M}||_2^2 \to \min_{\mathbf{M}} \\ &s.t.\ \forall i, r \quad \mathbf{M}_{y_i} \cdot \mathbf{x}_i + \delta_{y_i,r} - \mathbf{M}_r \cdot \mathbf{x}_i \geq 1 \end{aligned} \tag{2.52}$$

In a more general case that the feature space cannot be linearly separated by this multi-class classifier, we can change the margin to a soft margin as we did in generalizing SVM from linearly separable case to linearly non-separable case. Therefore, some non-negative slack variables $\xi_i \geq 0$ are added to the constraints Eq. (2.51),

$$\forall i, r, \quad \mathbf{M}_{y_i} \cdot \mathbf{x}_i + \delta_{y_i,r} - \mathbf{M}_r \cdot \mathbf{x}_i \geq 1 - \xi_i \tag{2.53}$$

Also the optimization problem Eq. (2.52) will be replaced by

$$\begin{aligned} &\frac{1}{2}\beta||\mathbf{M}||_2^2 + \sum_{i=1}^{n-1} \xi_i \rightarrow \min_{\mathbf{M},\xi} \\ &s.t.\ \forall i, r \quad \mathbf{M}_{y_i} \cdot \mathbf{x}_i + \delta_{y_i,r} - \mathbf{M}_r \cdot \mathbf{x}_i \geq 1 - \xi_i \end{aligned} \tag{2.54}$$

To solve this prime form of the optimization problem, we transform this to its dual form by adding a set dual variables $\boldsymbol{\eta}$ and use KarushKuhnTucker theorem in [33] and [36] to convert the inequality constraints to equality constraints. To make the final dual form simple and easy to understand, we substitute some variables and omit some additive and positive multiplicative constants. Let $\mathbf{1}_i$ be the vector whose components are all zero except for the ith component that is equal to one, and let $\mathbf{1}$ be the vector whose components are all one. Then the dual form can be written as follows,

$$\begin{aligned} &-\frac{1}{2}\sum_{i,j} (\mathbf{x}_i \cdot \mathbf{x}_j)(\boldsymbol{\tau}_i \cdot \boldsymbol{\tau}_j) + \beta \sum_i \boldsymbol{\tau}_i \cdot \mathbf{1}_{y_i} \rightarrow \max_{\boldsymbol{\tau}} \\ &s.t.\ \forall i, \quad \boldsymbol{\tau}_i \leq \mathbf{1}_{y_i} \quad \text{and} \quad \boldsymbol{\tau}_i \cdot \mathbf{1} = 0 \end{aligned} \tag{2.55}$$

where $\boldsymbol{\tau}_i = \mathbf{1}_{y_i} - \boldsymbol{\eta}_i$. We can describe $\mathbf{M}$ in the form of

$$\mathbf{M}_r := \beta^{-1} \sum_i \tau_{i,r} \mathbf{x}_i \tag{2.56}$$

We substitute Eq. (2.56) in Eq. (2.47) and finally we get the classifier for predict $\hat{y}_n$ for the new object $\mathbf{x}_n$

$$\hat{y}_n := H(\mathbf{x}_n) = \underset{r=0}{\overset{K-1}{\operatorname{argmax}}} \{\mathbf{M}_r \cdot \mathbf{x}_n\} = \underset{r=0}{\overset{K-1}{\operatorname{argmax}}} \Big\{ \sum_i \tau_{i,r} (\mathbf{x}_i \cdot \mathbf{x}_n) \Big\} \tag{2.57}$$

Additionally, if we substitute $\boldsymbol{\tau}_i$ with $\mathbf{1}_{y_i} - \boldsymbol{\eta}_i$ in Eq. (2.56), we can rewrite Eq. (2.56) in terms of variable $\boldsymbol{\eta}$

$$\mathbf{M}_r := \beta^{-1}\Big[\sum_{i:y_i=r}(1-\eta_{i,r})\mathbf{x}_i + \sum_{i:y_i\neq r}(-\eta_{i,r})\mathbf{x}_i\Big] \tag{2.58}$$

Eq. (2.58) indicates that the solution of the optimization problem given by Eq. (2.55) is a linear combinations of the examples $x_1, \ldots, x_{n-1}$. In the same way in Eq. (2.37) for binary SVM classifier, we say that an example x_i is a support vector if there is a row r for which this coefficient is not zero. In Eq. (2.58), the support vectors are separated into two types through the two sums. The left part of the plus sign in Eq. (2.58) $\sum_{i:y_i=r}(1-\eta_{i,r})\mathbf{x}_i$ implies all the support vectors of class r. If $\eta_{i,r} < 1$, an example x_i is a support vector. The right part $\sum_{i:y_i\neq r}(-\eta_{i,r})\mathbf{x}_i$ implies all other support vectors from the rest classes except class r. If $\eta_{i,r} > 0$, an example x_i is a support vector.

Note in Eq. (2.55) and Eq. (2.57), an example x_i only appears in the form of inner product, which means we can replace all inner product $(\mathbf{x}_i \cdot \mathbf{x}_j)$ with the value of a kernel function $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ that satisfies Mercer's conditions mentioned in [65]. Hence, the optimization problem Eq. (2.55) can be rewritten as follows

$$\begin{aligned} -\frac{1}{2}\sum_{i,j}\mathcal{K}(\mathbf{x}_i,\mathbf{x}_j)(\boldsymbol{\tau}_i\cdot\boldsymbol{\tau}_j) + \beta\sum_i \boldsymbol{\tau}_i\cdot\mathbf{1}_{y_i} &\to \max_{\boldsymbol{\tau}} \\ s.t.\ \forall i, \quad \boldsymbol{\tau}_i \leq \mathbf{1}_{y_i} \quad &\text{and} \quad \boldsymbol{\tau}_i\cdot\mathbf{1} = 0 \end{aligned} \tag{2.59}$$

The original classifier in Eq. (2.57) can be rewritten to Eq. (2.60) as well.

$$\hat{y}_n := H(\mathbf{x}_n) = \underset{r=0}{\overset{K-1}{\operatorname{argmax}}}\Big\{\sum_i \tau_{i,r}\mathcal{K}(\mathbf{x}_i,\mathbf{x}_n)\Big\} \tag{2.60}$$

To solve the optimization problem described in Eq. (2.55) will lead to immeasurable time consuming and memory costing. However, K. Crammer and Y. Singer also gave an algorithm to solve this problem in [12] by decomposing it into small problems. The pseudo-code of this algorithm is given in Algorithm 5.

Algorithm 5: Crammer and Singer's multi-class SVM classifier

Data: training set: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n-1}, y_{n-1})\}$, and new object x_n

Result: predicted label $\hat{y}_n$

$\boldsymbol{\tau}_i \leftarrow 0$; `/* initial coefficients tau */`

$F_{i,r} \leftarrow -\beta\delta_{r,y_i}$; `/* initial matrix F */`

$A_i = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_i)$; `/* initial diagonal kernel values */`

repeat

  for $i = 1$ **to** $n-1$ **do**

    $\psi_i \leftarrow \max_r F_{i,r} - \min_{r:\tau_{i,r}<\delta_{y_i,r}} F_{i,r}$;

  end

  $p \leftarrow \operatorname{argmax}_i \psi_i$;

  for $r = 0$ **to** $K-1$ **do**

    $D_r \leftarrow \frac{F_{p,r}}{A_p} - \tau_{p,r} + \delta_{r,y_p}$;

    $\theta \leftarrow \frac{1}{K}\left(\sum_{r=0}^{K-1} D_r\right) - \frac{1}{K}$;

  end

  $\boldsymbol{\tau}_p^\star \leftarrow$ `FixedPoint(`$\mathbf{D}$, θ, $\epsilon/2$`)`;

  $\Delta\boldsymbol{\tau}_p \leftarrow \boldsymbol{\tau}_p^\star - \boldsymbol{\tau}_p$;

  for $i = 1$ **to** $n-1$ **do**

    for $r = 0$ **to** $K-1$ **do**

      $F_{i,r} \leftarrow F_{i,r} + \Delta\tau_{p,r}\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$;

    end

  end

  $\boldsymbol{\tau}_p \leftarrow \boldsymbol{\tau}_p^\star$;

until $\psi_p < \epsilon\beta$;

$\hat{y}_n \leftarrow \operatorname{argmax}_r \left\{ \sum_i \tau_{i,r}\mathcal{K}(\mathbf{x}_i, \mathbf{x}_n) \right\}$;

`/* find label with maximal similarity score */`

return $\hat{y}_n$

Function FixedPoint($\mathbf{D}$, θ, ϵ)

Function `FixedPoint(`$\mathbf{D}$, θ, ϵ`)`

initialize $l \leftarrow 0$;

repeat

$l \leftarrow l + 1$;

$\theta_{l+1} \leftarrow \frac{1}{K}\left[\sum_{r=0}^{K-1} \max\{\theta_l, D_r\}\right] - \frac{1}{K}$;

until $|\frac{\theta_l - \theta_{l+1}}{\theta_l}| \leq \epsilon$;

for $r = 0$ **to** $K - 1$ **do**

$\nu_r \leftarrow \min\{\theta_{l+1}, D_r\}$;

end

return $\boldsymbol{\tau} \leftarrow \boldsymbol{\nu} - \frac{\mathbf{B}}{A}$

end

2.5 Probabilistic Prediction Algorithms

Instead of predicting only a single label for the unknown object, probabilistic predictors are able to provide a probability distribution over a set of labels. This kind of predictors complements predicted labels with a degree of certainty. Sometimes, the information on probabilities can be more useful. T. Hastie, R. Tibshirani and J. Friedman said "The interest is often more in the class probabilities themselves, rather than in performing a class assignment." in [27].

When we obtain a set of probabilistic estimations on each class for the new object x_n, say $p_i(x_n),\ i = 0, \ldots, K-1$, we use some *optimal decision rule* [5, p. 39-40] to help us map the probability distribution to predicted label. Naturally, we classify x_n into the class that has the highest probability. This is also called the *maximum a posteriori* (MAP) estimate. Hence, the classifier of probabilistic predictors are as follows

$$\hat{y}_n := \underset{i}{\operatorname{argmax}}\, p_i(x_n) \tag{2.61}$$

In this section, we will introduce some most widely used probabilistic predictors.

Among them are: Naive Bayes Classifier, Logistic Regression and Platt Scaling. In addition, the first two algorithms are naturally probabilistic while Platt scaling turns support vector machines into probabilistic predictors.

2.5.1 Naive Bayes Classifier

Naive Bayes classifiers are a set of probabilistic predictors, which has been studied extensively since the 1950s. From its name we could easily find its two key properties, the word "Bayes" in the name derives from the fact that this algorithm is based on applying Bayes' theorem (named after T. Bayes 1702-1761) while the word "Naive" in the name stands for the fact that this algorithm is under the naive independence assumptions between the features. This assumption means that, in simple terms, the value of each feature in the feature space $\mathbf{X}$ is not related any other feature.

According to [54], Naive Bayes was introduced to the area of information retrieval in the early 1960s. After that, it has become a popular algorithm for the problem of classifying documents into different categories by using word frequencies as the features. Despite its simplicity, Naive Bayes can provide comparable results with some sophisticated methods in this area of documents classification if appropriately preprocessed [52]. Later on, it was also applied in medical diagnosis [53].

In addition to the description of problem given in §2.1, we assume that each object x_i is a d-dimensional vector, which can be represented as $x_i = (f_{i,1}, \ldots, f_{i,d})$. We refer to the collection of each feature as F_j where $j = 1, \ldots, d$. Then the probability model for a classifier can be written as

$$p(Y|F_1, \ldots, F_d) \tag{2.62}$$

This is the conditional probability of the dependent class variable Y on all feature variables $F_1, \ldots, F_d$. However, when the dimension of the feature vector is large, Eq. (2.62) becomes

infeasible to be based on. Therefore, we use Bayes' theorem to reformulate this model

$$\begin{aligned} p(Y|F_1,\ldots,F_d) &= \frac{p(Y)p(F_1,\ldots,F_d|Y)}{p(F_1,\ldots,F_d)} \\ &= \frac{p(Y)p(F_1|Y)p(F_2|Y,F_1)p(F_3,\ldots,F_d|Y,F_1,F_2)}{p(F_1,\ldots,F_d)} \\ &= \frac{p(Y)p(F_1|Y)p(F_2|Y,F_1)\ldots p(F_d|Y,F_1,\ldots,F_{d-1})}{p(F_1,\ldots,F_d)} \end{aligned} \tag{2.63}$$

Since the feature variables are given by the training set, $p(F_1,\ldots,F_d)$ is a scaling factor, which is also a constant when the training set is known. In other words, $p(Y|F_1,\ldots,F_d)$ is proportional to $p(Y)p(F_1|Y)p(F_2|Y,F_1)\ldots p(F_d|Y,F_1,\ldots,F_{d-1})$. Now we take Naive Bayes' independence assumptions between the features into account, which results in

$$\begin{aligned} p(Y|F_1,\ldots,F_d) &\propto p(Y)p(F_1|Y)p(F_2|Y,F_1)\ldots p(F_d|Y,F_1,\ldots,F_{d-1}) \\ &\propto p(Y)p(F_1|Y)p(F_2|Y)\ldots p(F_d|Y) \\ &\propto p(Y)\prod_{i=1}^{d} p(F_i|Y) \end{aligned} \tag{2.64}$$

By combining Eq. (2.64) with the optimal decision rule Eq. (2.61), we get the classifier for Naive Bayes

$$\hat{y}_n := \operatorname*{argmax}_y p(Y=y)\prod_{i=1}^{d} p(F_i=f_i|Y=y) \tag{2.65}$$

Along with the predicted label, we can also give the probabilistic estimate for the predicted label

$$p(Y=\hat{y}_n) = \frac{p(Y=\hat{y}_n)\prod_{i=1}^{d} p(F_i=f_i|Y=\hat{y}_n)}{\sum_{y=0}^{K-1} p(Y=y)\prod_{i=1}^{d} p(F_i=f_i|Y=y)} \tag{2.66}$$

To calculate this classifier, the probability $p(Y=y)$ for each label can be estimated by the relative frequency of each class in the training set. For the estimates of the probability $p(F_i|Y)$ for each feature, we need to make an additional assumption on the distribution of the feature variables or generate non-parametric models for the variables [32]. The extra assumptions are referred to as the *event models* of Naive Bayes classifiers. Among the popular event models are

- **Gaussian Naive Bayes** In this assumption, we assume the continuous values of each feature variable F_i have a Gaussian distribution. Then the probability for

feature variable F is as follows

$$p(F = f|Y = y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(f-\mu_y)^2}{2\sigma_y^2}} \tag{2.67}$$

where μ_y and σ_y^2 are normal distribution parameters, which can be calculated from the mean and variance of the values in F associated with class y.

- **Multinomial Naive Bayes** When the values of features are discrete rather than continuous, we can use multinomial naive Bayes assumption. In this assumption, we assume $(n-1)$ (the same as the number of training examples following our previous description) values of a feature variable F take on J different values, we then build a new vector $\theta = (\theta_1, \ldots, \theta_J)$ from the absolute frequency of each possible value in the original feature vector. Therefore, θ_j is the frequency of the jth value, and $\sum_{j=1}^{J} \theta_j = n-1$ At the meanwhile, we also assume the values of each feature variable have a multinomial distribution $(p_0, \ldots, p_{K-1})$, where p_i is the probability that class i occurs. Hence, the probability of observing a feature vector θ is given by

$$p(\theta|Y) = \frac{(\sum_j \theta_j)!}{\prod_j (\theta_j)!} \prod_i p_i^{\theta_i} \tag{2.68}$$

- **Bernoulli naive Bayes** When the values of features are binary numbers, we can use Bernoulli naive Bayes assumption. In this assumption, the probability of an object given a class y is given by

$$p(F_1, \ldots, F_d|Y = y) = \prod_{i=1}^{d} [F_i p(w_i|Y = y) + (1 - F_i)(1 - p(w_i|Y = y))] \tag{2.69}$$

where $p(w_i|Y = y)$ is the probability of class y generating the binary variable w_i.

2.5.2 Logistic Regression

Logistic regression is a sort of probabilistic predictors that predict probabilities for binary problems. Although it has the terminology "regression" in its name, it should be emphasized that this is a predictor for classification rather than regression [5, p. 205]. Commonly, logistic regression is also called logit regression or logit model [20, p. 128].

Logistic regression measures the relationship between the categorical label space and the feature space, of which the features are usually (but not necessarily) continuous, by using probability scores as the estimations on how likely the outcome is one of the possible labels [4]. The probability for each single testing example is obtained by substituting a linear model into a logistic function. The model of logistic regression is based on two extra assumptions. The first assumption is that the feature variables should be independent with each other. The second one is that the conditional mean $p(y|x)$ subjects to Bernoulli distribution.

Before we introduce logistic regression, we will begin with the logistic function. Logistic function was named in 1844-1845 by Pierre Franois Verhulst [67], which always takes on values between 0 and 1 [28]:

$$\hat{p}(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{2.70}$$

where $\hat{p}$ is the predicted probability and e is the natural logarithm. If we treat t as a linear combination of feature vector x, say $t = \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x}$, the logistic function can be written as:

$$\hat{p}(x) = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta} \cdot \mathbf{x})}} \tag{2.71}$$

In Eq. (2.71), β_0 and $\boldsymbol{\beta}$ are coefficients that related to the feature space. Since the range of logistic function is between 0 and 1 and logistic function is monotonically increasing, this will be interpreted as the probability that how likely the predicted label should be "1" rather than "0" in a binary case denoted by $\{0, 1\}$.

The logistic regression function Eq. (2.71) takes feature vector x as input and output $\hat{p}(x)$ as the probabilistic prediction. The value of $\beta_0 + \boldsymbol{\beta} \cdot \mathbf{x}$ can be any number from negative infinity to positive infinity, whereas the output $\hat{p}(x)$ is always confined to values between 0 and 1. Therefore, it is perfect to be interpreted as a probability.

To minimize the misclassification rate, we also use Eq. (2.61) to predict probabilities, that is to predict the label with the highest probability. In this binary case, we should predict $\hat{y} = 1$ when $\hat{p} \geq 0.5$ and $\hat{y} = 0$ when $\hat{p} < 0.5$. We notice that whenever $\beta_0 + \boldsymbol{\beta} \cdot \mathbf{x} \geq 0$

the predicted probability $\hat{p} \geq 0.5$ otherwise $\hat{p} < 0.5$. This means that logistic regression gives us a linear classifier for $\hat{y}$ as follows

$$\hat{y} = \begin{cases} 1 & \text{if } \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x} \geq 0 \\ 0 & \text{if } \beta_0 + \boldsymbol{\beta} \cdot \mathbf{x} < 0 \end{cases} \tag{2.72}$$

Therefore, logistic regression can be seen as a special case of the generalized linear model. The coefficients β_0 and $\boldsymbol{\beta}$ can be derived through maximum likelihood estimation [39].

In addition, logistic regression applies to multi-class problems by running $K-1$ independent binary logistic regression models, which is usually referred to as multinomial logistic regression. Before we apply multinomial logistic regression, we need to make another assumption that the class variable should follow the assumption of independence of irrelevant alternatives (i.e. IIA). This assumption means that the odds of one class over another do not depend on the presence or absence of other "irrelevant" alternatives. This allows the choice of K alternatives to be modelled as a set of $K-1$ independent binary choices.

As in Eq. (2.72), we write the linear classifier for multinomial logistic regression to predict the probability for kth class of example x_i as

$$f(k, i) = \boldsymbol{\beta}_k \cdot \mathbf{X}_i \tag{2.73}$$

where $\boldsymbol{\beta}_k = (\beta_0, \ldots, \beta_d)$ and $\mathbf{X}_i = (0, x_1, \ldots, x_d)$. Note that $\mathbf{X}_i$ in Eq. (2.73) is different from the feature vector that are grouped into vectors of size $d+1$ where d is the number of features in the training set.

Thus, we choose one of the outcomes, say $K-1$, as a "pivot" and then we run $K-1$ independent binary logistic regression models for the other $K-1$ labels against the pivot

label. Hence, this would proceed as follows

$$\ln \frac{p(y_i = 0)}{p(y_i = K-1)} = \boldsymbol{\beta}_0 \cdot \mathbf{X}_i$$
$$\ln \frac{p(y_i = 1)}{p(y_i = K-1)} = \boldsymbol{\beta}_1 \cdot \mathbf{X}_i$$
$$\cdots\cdots$$
$$\ln \frac{p(y_i = K-2)}{p(y_i = K-1)} = \boldsymbol{\beta}_{K-2} \cdot \mathbf{X}_i$$

If we exponentiate both sides and solve for the probabilities, we get:

$$p(y_i = 0) = p(y_i = K-1)e^{\boldsymbol{\beta}_0 \cdot \mathbf{X}_i}$$
$$p(y_i = 1) = p(y_i = K-1)e^{\boldsymbol{\beta}_1 \cdot \mathbf{X}_i}$$
$$\cdots\cdots$$
$$p(y_i = K-2) = p(y_i = K-1)e^{\boldsymbol{\beta}_{K-2} \cdot \mathbf{X}_i}$$

Using the fact that all K of the probabilities must sum to one, we find:

$$p(y_i = K-1) = \frac{1}{1 + \sum_{k=0}^{K-2} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}} \tag{2.74}$$

We can use this to find the other probabilities:

$$p(y_i = 0) = \frac{e^{\boldsymbol{\beta}_0 \cdot \mathbf{X}_i}}{1 + \sum_{k=0}^{K-2} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}}$$
$$p(y_i = 1) = \frac{e^{\boldsymbol{\beta}_1 \cdot \mathbf{X}_i}}{1 + \sum_{k=0}^{K-2} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}}$$
$$\cdots\cdots$$
$$p(y_i = K-2) = \frac{e^{\boldsymbol{\beta}_{K-2} \cdot \mathbf{X}_i}}{1 + \sum_{k=0}^{K-2} e^{\boldsymbol{\beta}_k \cdot \mathbf{X}_i}}$$

The unknown parameters β_k can then be solved as an optimization problem.

2.5.3 Platt Scaling

Unlike Naive Bayes classifier and Logistic Regression, which generate probabilities on themselves, Platt scaling is a method that transforms the outputs of a classifier into a

probability distribution over classes. The original method that applies a Platt scaling on the outputs of support vector machines was proposed by John Platt in [48]. Additionally, it has been shown to be effective for other types of classifiers as well [42].

The basic idea of Platt scaling is to fit a logistic function on the outputs of a classifier. Then the outputs of the logistic function, which are restricted to a range from 0 to 1, will be treated as the probabilistic estimates to the predicted labels. It sounds like Logistic Regression we mentioned in the previous section. However, the difference is that Logistic Regression tries to find the optimized coefficients for a logistic regression model, which takes in the linear combination of the values of features as inputs. While Platt scaling tries to find the optimized coefficients for a logistic regression model, which takes in the outputs of other classifiers as input.

Given a binary classification problem $\mathbf{Z} = \{(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ where the label space $\mathbf{Y}$ is denoted by $\{-1, +1\}$. Moreover, we are also given a classifier f trained on $\mathbf{Z}$ and the real-valued function f follows the form of

$$\hat{y}_n := \begin{cases} +1 & \text{if } f(x) \geq 0 \\ -1 & \text{if } f(x) < 0 \end{cases} \tag{2.75}$$

when predicting $\hat{y}_n$ for the new object x_n.

Therefore, the probabilistic estimates given by Platt scaling on this classifier are as follows

$$\hat{p}(y_n = +1|x_n) = \frac{1}{1 + \exp(Af(x) + B)} \tag{2.76}$$

that is, a logistic transformation of the classifier scores $f(x)$, where A and B are two scalar parameters that are learned by the algorithm.

The optimized parameter A and B can be determined by using maximum likelihood estimation from a training set (f_i, y_i). Additionally, Platt suggests to use regularized target probabilities t_i instead of $\frac{y_i+1}{2}$, which is 1 for $y_i = +1$ and 0 for $y_i = -1$, as the

new training set (f_i, t_i). t_i can be defined as:

$$t_i := \begin{cases} \frac{N_+ + 1}{N_+ + 2} & \text{if } y_i = +1 \\ \frac{1}{N_- + 2} & \text{if } y_i = -1 \end{cases} \tag{2.77}$$

where N_+ is the number of positive examples and N_- is the number of negative examples.

Then, the parameters A and B are found by minimizing the negative log likelihood of the training data, which is a cross-entropy error function.

$$-\sum_i [t_i \log(p_i) + (1 - t_i) \log(1 - p_i)] \to \min_{A,B} \tag{2.78}$$

The decision function Eq. (2.30) of SVM we introduced in §2.4.2 follows the same form we described in Eq. (2.75). When the value of $f(x)$ is large, it means that the example is far away from the separating hyperplane Eq. (2.25). We expect the predicted probability given by Eq. (2.76) to be close to 1 since we consider this example is unlikely to have the negative label "-1". In contrast, when the value of $f(x)$ is close to 0, it means that the example is near the separating hyperplane. We think the predicted probability that this example is a positive example should be much lower than the previous situation, maybe around 0.5. By applying Platt scaling, the above requirements can be completely satisfied. Note that when $B \neq 0$, the probabilistic estimates contain a correction compared to the original decision function, which means Platt scaling can calibrate the predictions based on the maximal predicted probabilities. This is useful especially when applying Platt scaling on support vector machines since the bias introduced by SVM might be calibrated in Platt scaling.

The original method of Platt scaling is only applicable to binary problems. However, by introducing pairwise coupling, Ting-Fan Wu, Chih-Jen Lin and Ruby C. Weng proposed a method to calculate predicted probabilities on multi-class problem [77]. This method can be viewed as an improved version of the coupling approach by P. Refregier and F. Vallet in [51]. The basic idea of this method is to introduce the estimated pairwise class probabilities r_{ij} first. r_{ij} is an estimate to the true probability $\mu_{ij} = P(y = i | y = i \text{ or } j, \mathbf{x})$,

which could be obtained by training a model for examples of ith and jth classes in the training set. Then we could use all r_{ij} to estimate $p_i = P(y = i|\mathbf{x})$ where $i = 0, \ldots, K-1$.

In [51], P. Refregier and F. Vallet considered that

$$\frac{r_{ij}}{r_{ji}} \approx \frac{\mu_{ij}}{\mu_{ji}} = \frac{\frac{p_i}{p_i+p_j}}{\frac{p_j}{p_i+p_j}} = \frac{p_i}{p_j} \tag{2.79}$$

Thus, choosing any $K-1$ r_{ij} and considering the condition $\sum_{i=0}^{K-1} p_i = 1$, the probabilities p_i can be obtained by solving a linear system.

In [77], a optimization formulation was proposed

$$\sum_{i=0}^{K-1} \sum_{j:j\neq i} (r_{ji}p_i - r_{ij}p_j)^2 \rightarrow \min_{\mathbf{p}} \quad s.t. \sum_{i=0}^{K-1} p_i = 1,\ p_i \geq 0,\ \forall i \tag{2.80}$$

This formulation considers all equations in Eq. (2.79) not only $K-1$ of them, which solves the problem that the results depend strongly on the selection of $K-1$ r_{ij}, pointed out previously by Price et al. [50].

In the experiments carried out in §5, this method was used for multi-class problems.

2.6 Comparisons with Simple Predictors and Probabilistic Predictors

Conformal predictors and Venn predictors represent one type of algorithm that produce predictions complemented with the information on their reliability. In this section, we compare them with other approaches. Firstly, we compare these algorithms with simple predictors. They form a large category among all machine learning algorithms. Simple predictors are a type of algorithm that output a label without any additional information. k-nearest neighbours and support vector machines are two most widely-used examples of this type of algorithm. Secondly, we compare Conformal and Venn predictors with probabilistic predictors that output a probability distribution of a new label. We will briefly describe this method that provide information on how reliable predictions are, compare them with Conformal and Venn predictors and demonstrate their limitations.

In the introduction of this thesis, we described two measurements of performance of Conformal and Venn predictors: validity and efficiency. Validity demonstrates how reliable predictions are; efficiency is concerned with how informative predictions are. For Conformal predictors, validity implies that the error rate is close to the preset significance level, and efficiency means outputs contain as few as possible multiple predictions. For Venn predictors, validity implies that observed frequencies are well-calibrated by the probability distributions it outputs. While efficiency means the probability interval output by Venn predictors is narrow and close enough to 0 or 1. For simple predictors, efficiency is guaranteed since single prediction is informative. However, there is no concept of validity for simple predictors, since simple predictors only give single predictions.

For probabilistic predictors, validity is usually guaranteed, but only under statistical assumptions that are stronger than i.i.d. (lots of probabilistic predictors assumes that examples are of a specific probability distribution). While efficiency for probabilistic predictors usually depends on the algorithms.

This is summarised in Table 2.1.

2.7 Summary

In this chapter, we introduced the frameworks of Conformal and Venn predictors alongside some underlying algorithms that used or compared in this thesis. Conformal and Venn predictors were introduced in [69] and represent a new type of algorithmic frameworks to make probabilistic predictions. They provide information on how reliable the prediction is for each new object, and more important is that this information is valid when the online mode is applied.

We demonstrated that these algorithms had distinct advantages over other probabilistic algorithms. These advantages are summarised as follows:

1. These algorithms give single predictions for the unknown objects alongside the estimations of how reliable they are, which makes them more informative than any

Table 2.1: Comparison with simple predictors and probabilistic predictors

Predictor	Output Type	Validity	Efficiency
Simple predictors	single prediction	depends on the data	depends on the algorithm
Probabilistic predictors	single probability distribution	guaranteed under strong statistical assumptions	depends on the algorithm
Conformal predictors	a set of predictions	guaranteed	depends on the nonconformity measure
Venn predictors	a set of probabilities	guaranteed	depends on the taxonomy

other simple predictors or probabilistic predictors.

2. Validity is based on a simple i.i.d. assumption (or a weaker exchangeability assumption), which can be often satisfied when data sets are randomly permuted. Valid predictions do not depend on the assumed probability distribution of examples as normal probabilistic predictors usually do.

3. These two methods are flexible frameworks rather than single algorithms. This means practically any machine learning algorithm can be used as the underlying algorithm.

Chapter 3

Transforming Predictions into Probabilities

Generally, there are two types of algorithms that give probabilistic estimations. The first type of algorithm can generate probabilities itself, e.g. Naive Bayes classifier and Logistic Regression. The other type of algorithm generates probabilities based on some outputs from other classifiers, e.g. Platt Scaling. More specifically, this kind of algorithm transforms the predictions given by other classifiers into probabilities. This concept is similar to Platt Scaling, which transforms the decision values of an SVM classifier into probabilities by fitting a sigmoid function on them.

In this chapter, we will mainly introduce two designs of algorithms that transform predictions into probabilities. This first one is called Venn-ABERS Prediction and it is introduced in §3.2. This algorithm is a different type of Venn predictor and transforms the outputs of other classifiers into probabilities by applying isotonic regression. The second one is called Conformal Prediction with Bivariate Isotonic Regression (CP-BivIR) and it is introduced in §3.3. It is an algorithm based on conformal prediction. This algorithm transforms confidences and credibilities obtained from conformal predictors into

probabilities by applying a bivariate isotonic regression to them.

3.1 Related Works

Before we introduce these two algorithms, we start with introducing some algorithms related to them. There are many algorithms that transform the predictions given by other classifiers into probabilities. Among them, Platt Scaling transforms decision values into probabilities, which was introduced in §2.5.3. Zadrozny and Elkan's SVM with Isotonic Regression was also an approach that convert SVM scores into probabilities.

3.1.1 SVM with Isotonic Regression

SVM with Isotonic Regression was proposed in [78] by Zadrozny and Elkan. The basic idea was very similar to Platt Scaling. It trains a binary SVM classifier on all training data set and calculate the decision values as well. Then it maps all decision values into an isotonic sequence of values between 0 and 1. Intuitively, the values could be output as probabilities. When a new object is coming, we calculate its decision value and use this value to locate its position in the isotonic sequence and output the average value of nearby values as the probability.

To be more explicit, suppose we are given a binary classification problem: the training set is $\wr z_1, z_2, \ldots, z_{n-1} \int$ while the label space $\mathbf{Y} = \{0, 1\}$. We are offering a new object x_n as well. Our task is to predict $\hat{y}_n$ and the probability that the label is 1. This algorithm works as follows. First, we train a binary SVM classifier on this problem with the optimal hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$. Then the decision value d_i for example z_i is defined as $d_i = \mathbf{w} \cdot \mathbf{x_i} + b$ Decision values have no imposed boundaries, ranging from $-\infty$ to ∞. We then pair decision values with the true labels of training examples together as (d_i, y_i). After we get a sequence of pairs $(d_1, y_1), (d_2, y_2), \ldots, (d_{n-1}, y_{n-1})$ and we sort this into a increasing sequence according to the decision values. Let the sorted sequence be $(d_{\pi(1)}, y_{\pi(1)}), (d_{\pi(2)}, y_{\pi(2)}), \ldots, (d_{\pi(n-1)}, y_{\pi(n-1)})$. Since possible labels are 0 and 1,

the label sequence $y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(n-1)}$ is a series of binary numbers. We then adopt a "pool-adjacent violators algorithm" (abbrev. PAVA, we will cover the details in later section) on the binary sequence and transform the sequence into a monotonically increasing sequence $y'_{\pi(1)}, y'_{\pi(2)}, \ldots, y'_{\pi(n-1)}$. For every $i < j$, there must be $y'_{\pi(i)} \leq y'_{\pi(j)}$. We predict the new object x_n by calculate its decision value d_n. We announce the probability as $y'_{\pi(i)}$ where $i = 1, 2, \ldots, n-2,\ d_{\pi(i)} \leq d_n < d_{\pi(i+1)}$.

This method is motivated by the fact that the relationship between SVM scores and the empirical probabilities appears to be monotonically increasing for many datasets. The PAV algorithm finds the stepwise-constant isotonic function that best fits the data according to a mean-squared error criterion.

3.1.2 Inductive Conformal Predictors

Inductive Conformal Predictors (abbrev. ICP) are not algorithms that transform predictions into probabilities. However, both algorithms introduced in the followings use the same concept proposed in this algorithm. It was introduced by Papadopoulos in [44].

The basic idea of Inductive Conformal Predictors is as follows: we are given a standard classification problem. Instead of taking all training examples into consideration as Transductive Conformal Predictors (abbrev. TCP) does, Inductive Conformal Predictors tries to solve the problem by dividing the original training data set into two parts. The small part extracted from the original training set is called *calibration set*, while the big part remains is called *proper training set*. A conformal predictor is firstly trained only on the proper training set and then adapted to the calibration set. Only nonconformity scores of examples in the calibration set are calculated and stored. When a new object comes, we use the conformal predictor trained previously to calculate the nonconformity score and then compare with nonconformity scores of the calibration set.

According to [44], the use of such a measure will not have an impact on the validity of the results produced by the conformal predictor. It means that Inductive Conformal Predictors still have the property of validity. Additionally, the time consuming of ICP

reduces compared to TCP while ICP still has a comparable accuracy. The only drawback of ICP is a small loss in terms of accuracy.

In the following, we give the formal definition of Inductive Conformal Predictors. Suppose we are given a classification problem: the training set is $\wr z_1, z_2, \ldots, z_{n-1} \int$ while the label space $\mathbf{Y} = \{0, 1, \ldots, K\}$. We are offering a new object x_n as well. First, we divide the training set into two parts. The proper training set $\wr z_1, z_2, \ldots, z_l \int$ contains l examples. The calibration set $\wr z_{l+1}, z_{l+2}, \ldots, z_{n-1} \int$ contains $n-l-1$ examples. The nonconformity measure A_l is trained only on the proper training set. We use A_l to calculate all nonconformity scores for examples in the calibration set.

$$\alpha_i := A_l(\wr z_1, z_2, \ldots, z_l \int, z_i), \ i = l+1, l+2, \ldots, n-1 \tag{3.1}$$

This results in a sequence $\alpha_{l+1}, \alpha_{l+2}, \ldots, \alpha_{n-1}$. For each possible $y \in \mathbf{Y}$

$$\alpha_n := A_l(\wr z_1, z_2, \ldots, z_l \int, (x_n, y)) \tag{3.2}$$

So that

$$p_y := \frac{|i = l+1, l+2, \ldots, n-1, n : \alpha_i \geq \alpha_n|}{n-l} \tag{3.3}$$

After we tried all possible labels, we can predict the classification with the largest p-value together with its confidence and credibility.

3.2 Venn-ABERS Predictor with SVM

Venn-ABERS predictor is a recently developed algorithm introduced in [70], which makes multi-probabilistic predictions for the new objects. It is modified from Zadrozny and Elkan's procedure of probability forecasting in [78]. According to [37] and [31], the method of calibrating probabilistic predictions introduced by Zadrozny-Elkan method do not always achieve its goal and sometimes leads to poorly calibrated predictions. The simple modification introduced Venn predictors into the procedure to overcome the problem of potentially weak calibration as a result of the fact that Venn predictors are always well calibrated and guaranteed to be well calibrated under the exchangeability assumption.

Suppose we are given a standard binary classification problem: a training set of examples $\wr z_1, z_2, \ldots, z_{n-1}\int$. Each z_i consists of a pair of object x_i, and label y_i. The possible labels are binary, that is, $y \in \mathbf{Y} = \{0, 1\}$. We are also given a new object x_n. Our task is to predict the label $\hat{y}_n$ for the new object x_n and give the estimation of the likelihood that our prediction is correct.

Before we discuss more generalized Venn-ABERS predictors, there are some notions to be introduced first. The first notion is the term "*scoring algorithm*". Scoring algorithm is an algorithm that trains a classifier on the training set and uses the classifier to output a prediction score $s(x_n)$ for the new object x_n and predicts the label of x_n to be "1" if and only if $s(x_n) \geq c$ (c is a fixed threshold). So $s(\cdot)$ is hereby called the *scoring function*. Many machine learning algorithms for classification are scoring algorithms. In our case, the decision function of SVM is a scoring function, since we assign a new object the positive label "+1" if and only if its decision value is greater than zero and vice versa for the negative label. The second notion is "*increasing*". We say that a function $f(\cdot)$ is increasing if its domain is an ordered set and $t_1 \leq t_2 \implies f(t_1) \leq f(t_2)$. The third notion is "*isotonic calibrator*", which is a monotonically increasing function on the set $\{(s(x_1), y_1), \ldots, (s(x_{n-1}), y_{n-1})\}$ that maximizes the likelihood

$$\prod_{i=1}^{n-1} p_i, \text{ where } p_i := \begin{cases} g(s(x_i)) & \text{if } y_i = 1 \\ 1 - g(s(x_i)) & \text{if } y_i = 0 \end{cases} \tag{3.4}$$

this function $g(\cdot)$ is unique and can be found by using the "pool-adjacent violators algorithm" (abbrev. PAVA) introduced in [3]. We call such function $g(\cdot)$ the isotonic calibrator for $((s(x_1), y_1), (s(x_2), y_2), \ldots, (s(x_{n-1}), y_{n-1}))$.

To make a prediction for new object x_n, we can find the closet $s(x_i)$ to $s(x_n)$, and announce $g(s(x_i))$ as its prediction. It is exactly what we do in SVM with Isotonic Regression. This method is likely to over-fit since the examples are used both for training and calibrating. The Venn-ABERS predictor can be defined as follows. Like we do in a standard Venn predictor, we try every possible label for the new object x_n, which is 0

and 1. s_0 is a scoring function trained on $\wr z_1, z_2, \ldots, z_{n-1}, (x_n, 0)\int$, while s_1 is a scoring function trained on $\wr z_1, z_2, \ldots, z_{n-1}, (x_n, 1)\int$ Let g_0 and g_1 be the isotonic calibrators.

$$\begin{aligned} g_0 &:= ((s_0(x_1), y_1), (s_0(x_2), y_2), \ldots, (s_0(x_{n-1}), y_{n-1}), (s_0(x), 0)) \\ g_1 &:= ((s_1(x_1), y_1), (s_1(x_2), y_2), \ldots, (s_1(x_{n-1}), y_{n-1}), (s_1(x), 1)) \end{aligned} \tag{3.5}$$

To achieve the isotonic calibrator, we do the followings according to the definition of PAVA. First we arrange the pairs $(s(x_i), y_i)$ in the increasing order based on the values of scoring function $s(x_i)$. Having obtained a binary sequence consisting of labels $\{y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(n)}\}$, we applied PAVA to find the increasing sequence of them. By adopting PAVA, we do as follows:

- **Step 1:** Start with $y_{\pi(1)}$, move to the right and stop if the pair $(y_{\pi(i)}, y_{\pi(i+1)})$ violates the monotonicity constraint, i.e. $y_{\pi(i)} > y_{\pi(i+1)}$. Pool $y_{\pi(i)}$ and the adjacent $y_{\pi(i+1)}$, by replacing them both by their average.

$$y^*_{\pi(i)} = y^*_{\pi(i+1)} := \frac{(y_{\pi(i)} + y_{\pi(i+1)})}{2} \tag{3.6}$$

- **Step 2:** Next check that $y_{\pi(i-1)} \leq y^*_{\pi(i)}$. If not, pool $\{y_{\pi(i-1)}, y_{\pi(i)}, y_{\pi(i+1)}\}$ into one average. Continue to the left until the monotonicity requirement is satisfied. Then proceed to the right until the last one.

The final isotonic calibrator g is a function mapping the increasing scores to the increasing sequence (i.e. probabilities). As the score increases, the object is more likely to be "1" in correlation with the increasing sequence. Then the multi-probability prediction output for that the predicted label being "1" is $\{p_0, p_1\}$, where $p_0 := g_0(s(x_n))$ and $p_1 := g_1(s(x_n))$. For the reason that we need to predict the probability for the prediction label being correct, we should transform the bounds $\{p_0, p_1\}$ to $\{1 - p_1, 1 - p_0\}$ when the predicted label is "0".

Venn-ABERS predictors with SVM give a probabilistic type of predictions, which is the same as regular Venn predictors. Vovk proposed and proved that VennABERS predictors are Venn predictors in [70, Proposition 1]. He reformulated a Venn-ABERS predictor

into a Venn predictor and compared it to the original Venn predictor, demonstrating that the predictions of them were identical. Therefore, Venn-ABERS predictors are Venn predictors and inherit all properties of validity from Venn predictors. According to [70], the simplified of Venn-ABERS predictor is more computational efficient and performs better. The difference between the simplified version of Venn-ABERS predictor with the original version is that in the simplified version we only train scoring function once. That is $s(\cdot) = s_0(\cdot) = s_1(\cdot)$. The pseudo-code of simplified Venn-ABERS predictor is listed as below in Algorithm 6.

3.3 Conformal Prediction with Bivariate Isotonic Regression

As we mentioned in the previous chapter, conformal predictors provide a prediction set that should contain the true label based on the given significance level and the nonconformity scores of all examples. However, the conformal predictors themselves do not give any probabilistic information except p-values. In this section, we will introduce an algorithm named Conformal Prediction with Bivariate Isotonic Regression (CP-BivIR) that transforms confidence and credibility of the prediction into probabilities by implementing the bivariate isotonic regression.

Inspired by Venn-ABERS predictor, we considered conformal predictors as a special type of scoring functions. As we described in the previous chapter, we are given a significance level ϵ. If the confidence of a possible label is greater and equal to $1 - \epsilon$, we will include this label in our prediction set. This process has a lot in common with scoring functions in Venn-ABERS predictor. Moreover, we would like to make use of both confidence and credibility. Hence, we use bivariate isotonic regression for two variables rather than isotonic regression for a single variable. Conformal Prediction with Bivariate Isotonic Regression also uses the design of Inductive Conformal Predictors, which divides the training data set into proper training set and calibration set.

Algorithm 6: Venn-ABERS predictor

Data: training set $\mathbf{Z}^{(n-1)}$ and new example x_n

Result: predicted label $\hat{y}_n$ with probabilistic interval $[p_0, p_1]$

S $\leftarrow$ `TrainSVM(`$\mathbf{Z}^{(n-1)}$`)` ; `/* train a SVM classifier */`

for $i = 1$ **to** n **do** `/* calculate all scores */`

  $s_i \leftarrow \mathrm{S}(x_i)$;

end

$s_{\pi(1)}, \ldots, s_{\pi(n)} \leftarrow$ `Sort(`$s_1, \ldots, s_n$`)` ; `/* sort all scores */`

for $i = 0$ **to** 1 **do** `/* generate isotonic calibrators` g_0 g_1 `*/`

  $y_n \leftarrow i$;

  $g_i \leftarrow$ `PAVA` $(y_{\pi(1)}, \ldots, y_{\pi(n)})$; `/* call PAVA function */`

  $p_i \leftarrow g_i(s_n)$;

end

if $p_0 > 0.5$ **then** `/* assign label based on probability */`

  $\hat{y}_n \leftarrow 1$;

  return $\hat{y}_n,\ [p_0, p_1]$

else

  $\hat{y} \leftarrow 0$;

  return $\hat{y}_n,\ [1 - p_1, 1 - p_0]$

end

The basic idea behind this algorithm is as follows. First, the training set of the problem is split into proper training set and calibration set. This is the same as Inductive Conformal Prediction suggested by Harris Papadopoulos in [44]. Second, the proper training set will be used to train a conformal predictor that will then be applied to the calibration set to make predictions for all examples in the calibration set. Third, all the predicted labels, the true labels, the confidences and credibilities of examples in the calibration set will be transformed into a two-dimensional probability matrix by implementing the

Function PAVA($y_1, \ldots, y_n$)

```
Function PAVA(y_1, ..., y_n)
    for i = 1 to n − 1 do          /* searching violator from start to end */
        j ← i ;
        while j > 0 ∧ y_j > y_{j+1} do                    /* find a violator */
            sum ← 0;
            for k = j to i + 1 do /* calculate the average of current pool */
                sum ← y_k + sum;
            end
            avg ← sum / (i+2−j);
            for k = j to i + 1 do
                y_k ← avg;
            end
                  /* assign average value to every example in the pool */
            j ← j − 1 ;                                  /* check last value */
        end
    end
    return y_1, ..., y_n
end
```

bivariate isotonic regression. Finally, when we are given a new object, we use our conformal predictor to give a predicted label for the new object and use the probability matrix to make probabilistic predictions based on its confidence and credibility.

The following subsections will be organized as follows. In §3.3.1, some details of conformal predictor related to this algorithm will be given. Then in §3.3.2, we will introduce the nonconformity measure and what we used in this algorithm. The approach of transforming confidence and credibility into probabilities will be proposed in §3.3.3.

3.3.1 Conformal Prediction

First, let us recall the definition of conformal prediction we gave in §2.2. Given a training set $\langle z_1, \ldots, z_{n-1} \rangle$, a new object x_n and a significance level ϵ, the conformal predictor Γ determined by a nonconformity measure A_n gives a prediction set as in Eq. (3.7)

$$\Gamma^\epsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) := \{y \in \mathbf{Y} : p_y > \epsilon\} \tag{3.7}$$

In Eq. (2.12), p_y is the p-value for the pair of (x_n, y) and is defined as follows

$$p_y := \frac{|\{i = 1, \ldots, n : \alpha_i \geq \alpha_n\}|}{n} \tag{3.8}$$

As we said in §2.2, a conformal predictor is also a confidence predictor, so it satisfies the criterion of validity.

Let ω be the data sequence $(x_1, y_1, x_2, y_2, \ldots)$ that a confidence predictor Γ processes. Then we can introduce a formal notation for the errors Γ makes. An error that Γ makes on the nth trial can be described as

$$\mathrm{err}_n^\epsilon(\Gamma, \omega) := \begin{cases} 1 & \text{if } y_n \notin \Gamma^\epsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n) \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

the total number of errors until the latest example x_n is

$$\mathrm{Err}_n^\epsilon(\Gamma, \omega) := \sum_{i=1}^{n} \mathrm{err}_i^\epsilon(\Gamma, \omega) \tag{3.10}$$

If all the examples are drawn from an exchangeable probability distribution P, which is one of our assumptions made in §2.1. The error at the nth trial $\mathrm{err}_n^\epsilon(\Gamma, \omega)$ is the realized value of a random variable, which may be designated as $\mathrm{err}_n^\epsilon(\Gamma, P)$. Formally, if for each ϵ

$$\mathrm{err}_1^\epsilon(\Gamma, P), \mathrm{err}_2^\epsilon(\Gamma, P), \dots \tag{3.11}$$

is a sequence of independent random variables each of which has probability ϵ of being 1 and probability $1 - \epsilon$ of being 0, we say that the confidence predictor Γ is *exactly valid*. If a confidence predictor is exactly valid, the probability of a prediction set at a confidence level $1 - \epsilon$ being in error is exactly ϵ.

However, it suggested that no confidence predictor is exactly valid according to Theorem 2.1 in [69]. The detailed proof was also given in [69]. Therefore, for confidence predictors, exact validity becomes vacuous. However, confidence predictors satisfy another criterion of validity, which is called *conservatively valid*. If a confidence predictor is conservatively valid, the probability it makes an error when it outputs a prediction set at a confidence level $1 - \epsilon$ is no greater than ϵ.

Formally, the confidence predictor Γ is conservatively valid if for any exchangeable probability distribution P on $\mathbf{Z}^\infty$ there exists a probability space with two families

$$(\xi_n^{(\epsilon)} : \epsilon \in (0,1), n = 1, 2, \dots), \ (\eta_n^{(\epsilon)} : \epsilon \in (0,1), n = 1, 2, \dots) \tag{3.12}$$

of binary-valued random variables such that:

- for a fixed ϵ, $\xi_1^{(\epsilon)}, \xi_2^{(\epsilon)}, \dots$ is a sequence of independent Bernoulli random variables with parameter ϵ;
- for all n and ϵ, $\eta_n^{(\epsilon)} \le \xi_n^{(\epsilon)}$;
- the joint distribution of $\mathrm{err}_n^\epsilon(\Gamma, P)$, $\epsilon \in (0,1)$, $n = 1, 2, \dots$, coincides with the joint distribution of $\eta_n^{(\epsilon)}$, $\epsilon \in (0,1)$, $n = 1, 2, \dots$

In other words, a confidence predictor is conservatively valid if the error $\mathrm{err}_n^\epsilon(\Gamma, P)$ is dominated in distribution by a sequence of independent Bernoulli random variables with parameter ϵ.

According to [69, Proposition 2.2], conservative validity leads to asymptotic conservativeness, which can be defined as follows: for any exchangeable probability distribution P on $\mathbf{Z}^\infty$ and any significance level ϵ,

$$\limsup_{n\to\infty} \frac{\mathrm{Err}_n^\epsilon(\Gamma, P)}{n} \leq \epsilon \tag{3.13}$$

with certainty.

When applied to a classification problem, it is very natural to give the single prediction for the new object together with its confidence and credibility as Saunders et al. suggested in [55].

A nonconformity measure A_n determines a conformal predictor that provides a prediction set with a certain confidence for the new object x_n that it should contain the object's true label y_n. We obtain this set by supposing that y_n will have a value y that makes (x_n, y) conform with the previous examples and the level of significance ϵ determines the amount of confidence that we require.

As Saunders et al. suggested in [55], it is natural to report the greatest $1-\epsilon$ for which Γ^ϵ is a single label set and predict the single label as our predicted label. However, in consideration to avoid overconfidence when the new object x_n is unusual, it is wise to report also the largest ϵ for which Γ^ϵ is empty. We call these two values *confidence* and *credibility* respectively, and the definitions were given by Vovk et al. in [69].

The confidence is defined as

$$\sup\{1-\epsilon : |\Gamma^\epsilon| \leq 1\}, \tag{3.14}$$

The credibility is defined as

$$\inf\{\epsilon : |\Gamma^\epsilon| = 0\} \tag{3.15}$$

3.3.2 Nonconformity Measure

Nonconformity measure is a function that gives quantitative estimations of the strangeness for an example compared with other examples. There are many different nonconformity

measures, and each one defines a conformal predictor. The algorithm used in the nonconformity measure is called "underlying algorithm". Most of the traditional algorithms could be used as underlying algorithms. However, we mainly focus on two of them in our nonconformity measures: one uses k-nearest neighbours and the other uses support vector machines.

We use the same assumption as in §3.3.1, and the nonconformity measure using 1 nearest neighbour algorithm can be defined as follows

$$A_n(\lbag(x_1,y_1),\ldots,(x_{n-1},y_{n-1})\rbag,(x_n,y)) := \frac{\min_{i=1,\ldots,n-1:y_i=y} d(x_n,x_i)}{\min_{i=1,\ldots,n-1:y_i\neq y} d(x_n,x_i)} \tag{3.16}$$

where $d(x_n,x_i)$ is the distance between x_n and x_i. The distance can be any distance such as Euclidean distance, Hamming distance, Manhattan distance, etc. In our algorithm, we use Euclidean distance, i.e. $d(x_n,x_i) = ||x_n - x_i||$.

The numerical value of A_n is small when the minimal distance to the example that has the same label as x_n is small by comparing to the minimal distance to the examples that have the different labels. An object is considered nonconforming if it is close to an example labelled in a different way and far from any example labelled in the same way.

The other nonconformity measure we used in the algorithm is based on support vector machines. We use the Lagrange multipliers α_i obtained from the dual form problem (3.17) as the nonconformity measures.

$$\begin{aligned} &\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathcal{K}(x_i,x_j) \to \max \\ &s.t.\ \sum_{i=1}^{n} y_i\alpha_i = 0, \quad 0 \le \alpha_i \le C,\ i = 1,\ldots,n \end{aligned} \tag{3.17}$$

The value a_i of nonconformity measure i.e. z_i equals to the Lagrange multiplier α_i. With $\alpha_i = 0$, the examples z_i are very typical, neither on the margins nor within the margin. With $\alpha_i = C$, the examples z_i are the most extreme outliers under the given choice of C. With $0 < \alpha_i < C$, the examples z_i are intermediate. This makes the Lagrange multipliers α_i of the dual form problem ideal for use as nonconformity scores. An object is considered nonconforming if it is an outlier within the margin.

3.3.3 Transforming Confidence into Probability

Isotonic Regression is an algorithm for calculating a weighted least squares regression function that is restricted to be increasing with respect to a partial order on the independent variables. The term "isotonic" means the direction of the trend of the regression curve is strictly increasing. Bivariate Isotonic Regression (BivIR) is a generalization of Isotonic Regression with respect to increasing linear ordering on both variables. This algorithm was introduced in Dykstra and Robertson in [16] by developing Isotonic Regression for a single variable with "Pool Adjacent Violators Algorithm" (PAVA).

Assuming x_{ij} is a given two-dimensional array of original values, w_{ij} is a non-negative array of weights, the algorithm is iterative in nature and uses successive one-dimensional smoothing. The algorithm produces the solution to

$$\min \sum_{i,j} (g_{ij} - x_{ij})^2 w_{ij} \tag{3.18}$$

where $\mathbf{G}(i,j) = g_{ij}$ such that $g_{ij} \leq g_{kl}$ whenever $i \leq k$ and $j \leq l$.

In order to apply this algorithm to conformal prediction, we design the framework of Conformal Prediction with BivIso as follows.

First, we divide the given training set $\{z_1, \ldots, z_{n-1}\}$ into two subsets: a proper training set $\{z_1, \ldots, z_m\}$ and a calibration set $\{z_{m+1}, \ldots, z_{n-1}\}$ where $0 < m < n-1$. In the training procedure, we only use the proper training set to train our conformal predictor.

Secondly, after we got the conformal predictor, we conduct the calibration set as a testing set and give the prediction together with its confidence and credibility for every example in the calibration set. According to the definitions of confidence and credibility we have given before in (3.14) and (3.15) and the definition of p-value in (2.11), the possible values for confidence are $\frac{0}{m+1}, \frac{1}{m+1}, \ldots, \frac{m}{m+1}$ and for credibility are $\frac{1}{m+1}, \frac{2}{m+1}, \ldots, \frac{m+1}{m+1}$. Since these numerators of all the possible values are integers, we could use these two variables as the indices of the two-dimensional array.

Let $conf_i$ be the numerator of the confidence for example z_{m+i} in the calibration set and $cred_i$ be the numerator minus one of the credibility for example z_{m+i} in the calibration set

where $i = 1, \ldots, n-m-1$. Let y_i be the true label of the example z_{m+i} in the calibration set and $\hat{y}_i$ be the predicted label of example z_{m+i} in the calibration set. We generate our two-dimensional arrays $\mathbf{X}(i,j) = x_{ij}$ and $\mathbf{W}(i,j) = w_{ij}$ as follows, $\forall i = 1, \ldots, n-m-1$,

$$\mathbf{X}(conf_i, cred_i) := \begin{cases} \mathbf{X}(conf_i, cred_i) + 1 & \text{if } y_i = \hat{y}_i \\ \mathbf{X}(conf_i, cred_i) & \text{if } y_i \neq \hat{y}_i \end{cases} \tag{3.19}$$

$$\mathbf{W}(conf_i, cred_i) := \mathbf{W}(conf_i, cred_i) + 1 \tag{3.20}$$

After we record all values in calibration set, we then use x_{ij} divided by w_{ij} as the final x_{ij}

$$\mathbf{X}(conf_i, cred_i) := \frac{\mathbf{X}(conf_i, cred_i)}{\mathbf{W}(conf_i, cred_i)}, \quad \text{if } \mathbf{W}(conf_i, cred_i) \neq 0 \tag{3.21}$$

Thirdly, we could use Bivariate Isotonic Regression to calculate the isotonic array $\mathbf{G}(i,j) = g_{ij}$. Each value in g_{ij} is equally weighted and is a real number ranging in $[0,1]$

Finally, for the new object x_n, we give the prediction with confidence and credibility by using the conformal predictor generated from the proper training set and use the confidence and credibility for the new object as indices to find the probabilities from the isotonic array generated from the calibration set.

$$p_{lower} = \mathbf{G}(n-1-m-cred_n, n-1-m-conf_n) \tag{3.22}$$

$$p_{upper} = \mathbf{G}(conf_n, cred_n) \tag{3.23}$$

The pseudo-code of Conformal predictor with bivariate isotonic regression is listed as below in Algorithm 7.

3.4 Summary

In this chapter, two implementations of algorithms that produce multi-probabilistic predictions by transforming the outputs of other algorithms into probabilities are introduced. Among them are Conformal prediction with bivariate isotonic regression and Venn-ABERS predictor with SVM.

Algorithm 7: Conformal predictor with bivariate isotonic regression

Data: proper training set $\mathbf{Z^{(m)}}$, calibration set $\mathbf{Z^{(n-1-m)}}$ and new example x_n

Result: predicted label $\hat{y}_n$ with probabilistic interval $[l_n, u_n]$

NCM $\leftarrow$ `TrainNCM(`$\mathbf{Z^{(m)}}$`)`;

for $i = m + 1$ **to** n **do** `/* loop for all examples in calibration set */`

  for $y = 0$ **to** $K - 1$ **do** `/* loop for every possible label */`

    for $j = 1$ **to** m **do** `/* for each label calculate all NCM scores */`

      $\alpha_j \leftarrow \text{NCM}(\wr(x_1, y_1), \ldots, (x_m, y_m) \int / (x_j, y_j), (x_j, y_j))$;

    end

    $\alpha_i \leftarrow \text{NCM}(\wr(x_1, y_1), \ldots, (x_m, y_m) \int, (x_i, y_i))$;

    $c \leftarrow 1$;

    for $j = 1$ **to** m **do**

      if $\alpha_j \geq \alpha_i$ **then**

        $c \leftarrow c + 1$;

      end

    end

    $p_y \leftarrow \frac{c}{n}$;

  end

  $conf_i \leftarrow$ `CalcConfidence(`$p_y, \forall y \in \mathbf{Y}$`)`;

  $cred_i \leftarrow$ `CalcCredibility(`$p_y, \forall y \in \mathbf{Y}$`)`;

end

$\mathbf{X},\ \mathbf{W} \leftarrow$ `BuildMatrix(`$conf,\ cred$`)`;

$\mathbf{G} \leftarrow$ `BivariateIsotonicRegression(`$\mathbf{X},\ \mathbf{W}$`)`;

$l_n \leftarrow \mathbf{G}(n - 1 - m - cred_n,\ n - 1 - m - conf_n)$;

$u_n \leftarrow \mathbf{G}(conf_n,\ cred_n)$;

return $\hat{y}_n,\ [l_n, u_n]$

Conformal prediction with bivariate isotonic regression can be treated as an extension of conformal prediction. As we know, conformal predictors themselves do not give direct probabilities for their predictions, while they produce p-values instead. Our algorithm uses isotonic regression to transform confidence and credibility into probabilities.

Venn ABERS prediction is a new type of Venn prediction, which transforms a sequence of scores into valid probabilities by using isotonic regression.

The advantages of these algorithms are

1. These algorithms produce multi-probabilistic predictions that give a more intuitive estimation on the true probability by hedging it within range.

2. The probabilities given by these algorithms are valid under a simple i.i.d. assumption, which is more applicable on real-world data sets than traditional probabilistic predictors that make extra strong assumptions on the data sets.

3. These algorithms are frameworks of machine learning algorithms rather than a single algorithm, and the efficiency depends on the underlying algorithms. This makes these two algorithms much more flexible than other probabilistic predictors.

Chapter 4

Designs of Taxonomy for Venn Machines

In this chapter, four new designs of taxonomy for Venn Machines will be introduced. They are SVM Venn Machines using equal intervals in §4.2, SVM Venn Machines using k-means clustering intervals in §4.3 and Multi-class SVM Venn Machines using k-means clustering intervals in §4.4. We will also introduce a taxonomy design for Regression Venn Machines in §4.5.

Venn Machine is a multi-probabilistic predictor described in [69]. The basic idea of Venn Machine is to divide every example into its corresponding category based on certain rules, then we choose all examples that share the same category with the new object and calculate the frequencies of labels in the chosen category as estimations of probabilities for the new object's label. A taxonomy is a way to divide objects into categories. The underlying algorithm is the algorithm used in the taxonomy.

As we discussed in §2.3, taxonomies are related to the properties of efficiency and validity of Venn predictors, which are very important in Venn prediction. A good taxonomy defines a good Venn predictor. Therefore, taxonomy design has become one of the

most popular research areas in Venn prediction. In §4.1, we will introduce several Venn predictors proposed in previous related works.

4.1 Previous Designs of Taxonomy for Venn Machines

Lots of taxonomy designs for Venn Machines were proposed in related works. Those taxonomies were constructed based on different underlying algorithms such as k-NN, SVM, Neural Networks, Random Forests, Logistic Regression, etc. In this section, we mainly focus on introducing some implementations of taxonomies based on k-Nearest Neighbours, Support Vector Machines and Neural Networks.

4.1.1 *k*-Nearest Neighbours Taxonomy

One of the simplest Venn taxonomies is the taxonomy based on k-Nearest Neighbours algorithm proposed in [69]. In this taxonomy, the category of an example is the same as its nearest neighbours' label. Hence, the number of categories is the same as the size of label space. Two examples are assigned to the same category if the labels of their nearest neighbours are same. If the number of an example's neighbours we examine is more than one, we will conduct a majority voting to find the majority label as the category for that example. The commonly used distance metric is the geometric distance, which is also known as Euclidean distance. We choose 1-Nearest Neighbours with Euclidean distance as the underlying algorithm for the taxonomy, therefore the category τ_i of example z_i is assigned by taxonomy A_n, which is described in Eq. (4.1).

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\wr, z_i) = y_j, \quad \text{where } j = \underset{k\in\{1,\ldots,n\},k\neq i}{\operatorname{argmin}} ||x_i - x_k|| \tag{4.1}$$

4.1.2 SVM Taxonomy with Predictions

We can design Venn taxonomies based on either outputs of SVM such as final predictions and distances to the maximum-margin hyperplane or intermediate values of SVM such

as Lagrange multipliers. The simplest design is based on the predictions given by SVM, which was introduced in [14].

Similar to k-Nearest Neighbours taxonomy, we assign the predicted label of an example given by an SVM classifier to its category. Suppose we are given a binary classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y)\int$ and the label space is $\mathbf{Y} = \{-1, +1\}$ and we need to partition this bag into categories. We train a binary SVM classifier on this bag. It is constructed with the maximum-margin hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$. For each example z_i in the bag, we calculate its predicted label $\hat{y}_i = \text{sgn}(\mathbf{w} \cdot \mathbf{x_i} + b)$ and assign this to its category. Examples belong to the same category if and only if they are predicted as of the same category by the SVM. As a result, the number of categories is the same as the size of label space that is two and the category τ_i for example z_i is one of the labels, which is $+1$ or -1. The SVM taxonomy with predictions A_n is defined in Eq. (4.2).

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\int, z_i) = \text{sgn}(\mathbf{w} \cdot \mathbf{x_i} + b) \tag{4.2}$$

Since Multi-class SVM can be decomposed into several binary SVMs or solved in a single optimization formulation as we introduced in §2.4.3. This taxonomy can easily be generalized from binary cases to multi-class cases.

4.1.3 SVM Taxonomy with Equal Size

As well as using the prediction of SVM in the design, we can also use the distances to the maximum-margin hyperplane of SVM in the design. SVM taxonomy with equal size was introduced in [14]. A Venn Machine with this taxonomy allows us to preset the number of categories to any value.

Suppose we are given a binary classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y)\int$ and the label space is $\mathbf{Y} = \{-1, +1\}$ and we need to partition this bag into T different categories. We train a binary SVM classifier on this bag with the optimal hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$. The distance d_i of an example z_i to the hyperplane is the value calculated before we apply a sign function and make a prediction, i.e. $d_i = \mathbf{w} \cdot \mathbf{x_i} + b$. Note that distances could

be negative values, which means the distances we mentioned are signed distances. The negative values of distances indicate that their examples are classified as negative label -1 by this SVM classifier. We can group examples with close values of distances together to divide this bag into categories. Two examples are assigned to the same category if they fall into the same group.

The workflow is as follows: firstly, we train a binary SVM classifier on the bag $\wr z_1, \ldots, z_{n-1}, (x_n, y)\int$ and obtain the optimal hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$. Secondly, we calculate distance $d_i = \mathbf{w} \cdot \mathbf{x_i} + b$ for every example z_i and sort the sequence $\{d_i\},\ i = 1, \ldots, n$ into a sorted sequence $\{d'_i\},\ i = 1, \ldots, n$. Thirdly, let a separating indices sequence be $\{L_i\},\ i = 0, \ldots, T$. Let L_0 be 1 and $L_1, L_2, \ldots, L_T$ be integers closest to $\frac{n}{T}, \frac{2n}{T}, \ldots, n$. Finally, we partition the sorted sequence $\{d'_i\}$ by division points $d'_{L_0}, d'_{L_1}, \ldots, d'_{L_T}$. The category τ_i of example z_i is then defined as the number of the interval formed by these division points where the corresponding distance d_i falls:

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\int, z_i) = j, \quad \text{where } j = 1, \ldots, T,\ d'_{L_{j-1}} < d_i \leq d'_{L_j} \tag{4.3}$$

4.1.4 SVM Taxonomy with Lagrange Multipliers

Besides using outputs of SVM in the designs, intermediate values will be a good idea as well. SVM taxonomy with Lagrange multipliers proposed in [14] is a taxonomy design based on Lagrange multipliers. A Venn Machine with this taxonomy also allows us to preset the number of categories to any value. Lagrange multipliers are introduced when we reformulate the optimization problem of SVM from the prime form into the dual form. The detailed description was given in §2.4.2.

Suppose we are given a binary classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y)\int$ and the label space is $\mathbf{Y} = \{-1, +1\}$ and we need to partition this bag into T categories. We train a binary SVM classifier on this bag in dual form as Eq. (2.41). The optimal hyperplane is $\mathbf{w} \cdot \mathbf{x} + b = 0$. Lagrange multipliers are calculated during the training process. For each

training example z_i, there is an optimal corresponding Lagrange multiplier α_i. According to the previous introduction to Lagrange multiplier, we know that an example z_i is not a support vector if its Lagrange multiplier $\alpha_i = 0$. If its Lagrange multiplier $0 < \alpha_i < C$ where C is the cost, the example z_i is a support vector on the margin. If its Lagrange multiplier $\alpha_i = C$, the example z_i is a support vector within the margins. Values of Lagrange multipliers reflect the strangeness of their examples, which we can use for designing Venn taxonomy by grouping examples with close values of Lagrange multipliers together.

Since values of Lagrange multipliers are non-negative, which means examples outside both margins have the same value or range. We should consider the ones on different sides of the optimal hyperplane separately. Therefore, we introduce a predicate that $\mathbf{w}\cdot\mathbf{x}+b > 0$ or not, and divide all examples into four bags roughly.

1. $\wr z_i : \alpha_i = 0 \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b > 0\int$
2. $\wr z_i : \alpha_i = 0 \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b < 0\int$
3. $\wr z_i : 0 < \alpha_i \leq C \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b > 0\int$
4. $\wr z_i : 0 < \alpha_i \leq C \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b < 0\int$

The first and second bags can be treated as two categories. The third and fourth bags can be partitioned respectively into T' categories of approximately similar size. The procedure is analogous to the one described in the previous taxonomy design. The total number T of categories equals to $2 + 2T'$. Eq. (4.4) shows the formal definition of this taxonomy.

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\int, z_i)$$
$$= \begin{cases} 0 & \text{if } \alpha_i = 0 \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b > 0 \\ T'+1 & \text{if } \alpha_i = 0 \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b < 0 \\ j & \text{if } 0 < \alpha_i \leq C \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b > 0 \quad \text{where } j = 1, \ldots, T',\ \alpha'_{L_{j-1}} < \alpha_i \leq \alpha'_{L_j} \\ -j & \text{if } 0 < \alpha_i \leq C \ \& \ \mathbf{w}\cdot\mathbf{x_i} + b < 0 \quad \text{where } j = 1, \ldots, T',\ \alpha'_{L_{j-1}} < \alpha_i \leq \alpha'_{L_j} \end{cases} \tag{4.4}$$

4.1.5 Taxonomy with One-vs-All SVM

Most of the previous SVM-based taxonomy designs are only suitable for binary cases. Lambrou et al. proposed a generalized SVM taxonomy for multi-class cases in [37] by implementing One-vs-All SVM.

To decompose a multi-class SVM problem into several binary-class SVM sub-problems is a widely-used approach for solving multi-class SVM. To use One-vs-All scheme is one of the most popular methods among them. This approach constructs K (the size of label space) binary SVM classifiers. For each subproblem, it choose one label from the label space and make all training examples with the same label as the chosen one the positive class in a binary SVM classifier. The rest training examples are grouped into one single class as the negative class in a binary SVM classifier. Then it solves this binary SVM sub-problem and calculates the decision values for testing objects. After all SVM classifiers are trained, the predicted label for each testing object is achieved by a winner-takes-all strategy. The details of this scheme was introduced in §2.4.3.

Suppose we are given a multi-class classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y)\int$ and the label space is $\mathbf{Y} = \{0, 1, \ldots, K-1\}$ and we need to partition this bag into categories. We train a One-vs-All SVM on this problem and get K binary classifiers. Let the decision functions $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ of these binary classifiers be $f_0(\cdot), f_1(\cdot), \ldots, f_{K-1}(\cdot)$. This taxonomy assigns the predicted label of an example to its category the same as the SVM taxonomy with predictions. Therefore, there are K categories and two examples are assigned to the same category if they are predicted to the same label. The taxonomy A_n is defined in Eq. (4.5).

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\int, z_i) = \underset{j=0,\ldots,K-1}{\operatorname{argmax}} f_j(x_i) \tag{4.5}$$

4.1.6 Taxonomies with Neural Networks

Neural Networks, also referred as Artificial Neural Networks, are a family of learning algorithms inspired by biological neural networks. Neural Networks simulate a system of

"neurons" that are interconnected and can communicate with each other. The connections between neurons have numerical weights that can be tuned based on experience, making Neural Networks adaptive to inputs and capable of learning. Five Venn predictors with Neural Networks were proposed by H. Papadopoulos in [45].

The basic idea of taxonomies with Neural Networks is to divide examples into categories based on the outputs of Neural Networks, which can be interpreted as probabilities for each class. To train such a Neural Networks model, we tuned the model as follows. First, the model only used a single hidden layer along with an input layer and an output layer. These three layers constituted a fully connected feedforward network. The hidden layer consisted of neurons with a hyperbolic tangent activation function. The output layer consisted of neurons with a softmax activation function, which outputs a value between 0 and 1. Furthermore, the outputs of all neurons in output layer added up to 1. Therefore, the outputs of this Neural Networks model can be interpreted as probabilities for each class.

Suppose we are given a multi-class classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y) \int$ and the label space is $\mathbf{Y} = \{0, 1, \ldots, K-1\}$ and we need to partition this bag into categories. This model was trained with the scaled conjugate gradient algorithm [41] minimizing cross-entropy error (aka. log loss):

$$CE = -\sum_{i=1}^{n} \sum_{j=0}^{K-1} t_i^j \log\left(o_i^j\right) \tag{4.6}$$

where $o_i^0, \ldots, o_i^{K-1}$ are the outputs of the network for example z_i and t_i^j equals to $[\![y_i = j]\!]$. Let $[\![\pi]\!]$ be 1 if the predicate π holds and 0 otherwise.

We denote the five taxonomies proposed in [45] as V_1, V_2, V_3, V_4, V_5. The details are given in the following.

1. The first taxonomy V_1 assigns an example a category the same as the label corresponding with its maximum output. Therefore, there are K categories, one for each possible label of the problem. Two examples belong to the same category if their maximum outputs are of the same class.

2. The second taxonomy V_2 is based on V_1 and further divides the examples in each category of V_1 into two categories according to the value of their maximum output. A high threshold θ is needed for this taxonomy to separate each original category into two halves. Hence, the number of categories is $2K$.

3. The third taxonomy V_3 is also based on V_1 and similar to V_2. V_3 divides examples in each category of V_1 into two categories according to the second largest output. A low threshold θ is needed to cut each original category into two halves. Notice that the threshold in V_3 is different with the threshold in V_2. The number of categories that V_3 produces is again $2K$.

4. The fourth taxonomy V_4 follows the same idea of V_2 and V_3. The difference is that taxonomy V_4 deals with the difference between the highest and second highest outputs. A threshold θ is needed to divide examples in each original category into two smaller categories. The number of categories that V_4 produces is still $2K$.

5. The fifth taxonomy V_5 is different with the others. V_5 assigns two examples to the same category if their outputs correspond to the same composition of labels. This can be done by applying a low threshold θ to every output of an example. If its output is above θ, the label related to this output will be added to the set. Two examples are considered in the same category if their sets are completely same. This taxonomy produces 2^K categories, however most of them are almost always empty.

4.2 SVM Taxonomy with Equal Length Intervals

In last section, we introduced some previous taxonomy designs based on k-Nearest Neighbours, Support Vector Machines and Neural Networks. In this section, we will propose an SVM taxonomy design inspired by SVM taxonomy with equal size. This taxonomy was firstly proposed in [79] and only suitable for binary cases. We will also develop a generalized version of this taxonomy for multi-class problems in this section.

In SVM taxonomy with equal size, examples are grouped into several categories with the same size based on their distances to the optimal hyperplane. As we know, examples are not evenly distributed on the whole feature space in most situations, which means that the sorted distances sequence $\{d_i'\}$ have dissimilar spacing between two adjacent examples. Take an extreme case into consideration that examples are linearly separable. The distances are either greater than $+1$ or less than -1. Hence, there exists a category consists of a bunch of values greater than $+1$ and a bunch of values less than -1. Examples in this category share very fewer similarities and give poor performance in predicting new objects falling into this category.

In our taxonomy design, we use a different approach to partition examples into categories. The workflow of our design is stated as follows. Firstly we use the training set to train an SVM and calculate the decision values ($f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$) of all examples in the training set and the new object. Secondly the whole range of decision values will be divided into several intervals of equal length. Each interval is a category and objects of which the decision values fall into the same interval are of the same category.

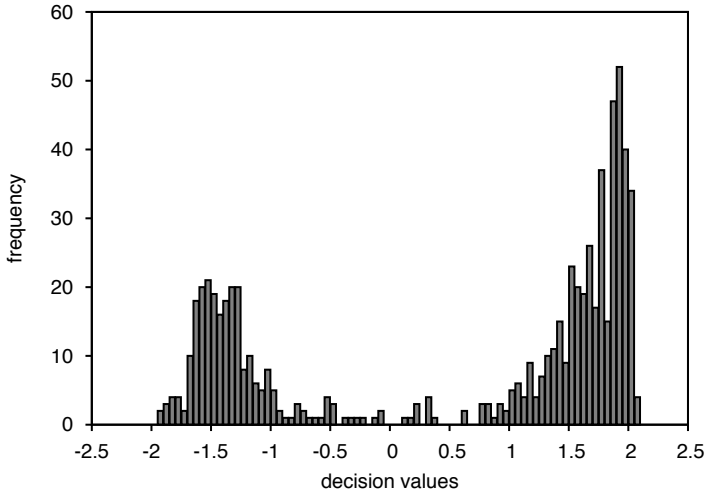
Suppose we are given a binary classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y) \int$ and the label space is $\mathbf{Y} = \{-1, +1\}$ and we need to partition this bag into T different categories. We train a binary SVM classifier on this bag with the optimal hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$. The distance d_i of an example z_i to the hyperplane is the value calculated before we apply a sign function and make a prediction, i.e. $d_i = \mathbf{w} \cdot \mathbf{x_i} + b$. All distances are calculated and among them d_{min} is the minimal value and d_{max} is the maximal value. Let a division points sequence be $\{L_i\},\ i = 0, \ldots, T$. Let L_0 be d_{min}, and $L_1, L_2, \ldots, L_T$ be $\frac{1}{T}(d_{max} - d_{min}), \frac{2}{T}(d_{max} - d_{min}), \ldots, (d_{max} - d_{min})$. The category τ_i of example z_i is then defined as the number of the interval formed by these division points where the corresponding distance d_i falls:

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y) \int, z_i) = j, \quad \text{where } j = 1, \ldots, T,\ L_{j-1} < d_i \leq L_j \tag{4.7}$$

To make it simpler, we can use preset minimal and maximal values when we calculate

the division points, since decision values of SVM usually follow a specific pattern. Fig. 4.1 shows the histogram of decision values on breast cancer data set. The figure indicates that decision values are distributed on the area around two margins (+1 and −1). For example, if we are asked to divide breast cancer examples into eight categories, we could set the minimal value to −2 and maximal value to +2. Therefore, the division points are $-\infty, -1.5, -1, \ldots, +1, +1.5, +\infty$. We use ∞ as division points to include examples outside the preset minimal and maximal values into categories.

Figure 4.1: Histogram of decision values on breast cancer data set



By using our taxonomy, examples in the same category have more in common. This is one of the advantages compared with SVM taxonomy with equal size. The other advantage is that our taxonomy reduces some computational complexity. We no longer need to sort the distances sequence and to compare the distance of the new object with division points several times to locate its category. Since we can get the category of a new object directly by calculation. However, the drawback of this design is that the size of each category is undefined. A small size of category could lead to a rough estimation of the probability. Another drawback of our design is that our taxonomy still can not deal with multi-class problems. In the next subsection, we will discuss the generalization of this taxonomy to

make it suitable for multi-class problems.

4.2.1 Combined Decision Function

In multi-class cases, if we choose to decompose the original problem into several binary sub-problems, we will need to train several binary SVM classifiers regardless of whether One-vs-One or One-vs-All approach is used. A scheme for multi-class SVM using One-vs-All approach was developed by Lambrou et al. in [37], which uses the overall predicted labels in the taxonomy. Generally, One-vs-One SVM is more efficient in accuracy than One-vs-All SVM. Therefore, we need to develop a new function to combine the outputs of all One-vs-One SVM classifiers and transform them into a single prediction score that could be used by our taxonomy. We call such function a *Combined Decision Function*.

Since we have several decision values for each example, we consider using an average value of all related decision values. Then we combine the average decision value and the predicted label together as a new combined decision value. This value can reflect the strangeness of examples. Examples with the same predicted label have close combined decision values, which will be assigned to the same category.

For a data set with K possible labels: $\{0, 1, \ldots, K-1\}$, there are $K(K-1)/2$ binary SVM if we use One-vs-One approach. For each possible label, there are $K-1$ related SVM decision functions. Then we use (4.8) to calculate the combined decision function $D(x_n)$ for the new object x_n,

$$D(x_n) := \hat{y}_n + \frac{1}{K-1} \sum_{i=0, i \neq \hat{y}_n}^{K-1} N(f_{\hat{y}_n, i}(x_n)) \tag{4.8}$$

where $\hat{y}_n$ is the overall predicted label done by max-wins voting strategy in One-vs-One SVM, $f_{\hat{y}_n, i}(x_n)$ is the decision function of SVM classifier on $\hat{y}_n$-vs-i, $N(\cdot)$ is a function that does the normalized transformation to $[0, 1]$. Another point we need to declare here is that in $f_{\hat{y}_n, i}(x_n)$ we always put $\hat{y}_n$ before i, which means we need to apply an opposite operation when $\hat{y}_n$ is greater than i. Since the examples of label $\hat{y}_n$ are treated as negative examples in i-vs-$\hat{y}_n$ classifier of a binary SVM.

This function firstly selects all $K-1$ related SVM and applies an opposite operation if $\hat{y}_n$ is not treated as the positive class in the binary SVM classifier. Then it does the normalization to transform the values into $[0,1]$. By adding all normalized values together and dividing by $K-1$, we calculate the average decision value of x_n. Finally, we output the average decision value added with predicted label $\hat{y}_n$ as the combined decision value of new example x_n. The reason for adding $\hat{y}_n$ to the arithmetic mean is that it could spread the average decision values of different classes to a range of $[0,K]$, otherwise the average decision values of all classes would have the same range of $[0,1]$. By adding the predicted label, it also keeps the same strangeness of examples.

Through the procedure we described above, we could obtain a combined decision values sequence $\{D_i\}$, where D_i is the combined decision value of example z_i calculated by substituting x_i into Eq. (4.8). Then we apply the same approach we do in SVM taxonomy with equal length intervals. The taxonomy A_n can be defined as Eq. (4.9).

$$\tau_i := A_n(\lbag z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\rbag, z_i) = j, \quad \text{where } j = 1, \ldots, T,\ L_{j-1} < D_i \leq L_j \tag{4.9}$$

In a similar way, we can use preset division points to simplify the problem. It is obviously that all combined decision values are within the range of $[0,K]$. We can set the number of categories the same as the number of possible labels (i.e. $T = K$) and let the division points $L_0, L_1, \ldots, L_K$ be $0, 1, \ldots, K$. Through this, users do not need to give the number of categories.

4.3 SVM Taxonomy with *k*-Means Clustering Intervals

In last section, we introduced a generalized SVM taxonomy with equal length intervals. When dealing with binary problems, this algorithm introduced a new parameter, i.e. number of categories. For different data sets, we need to use different settings for numbers

of categories in order to achieve best performances. While for multi-class problems, we predefine the number of categories the same as the number of possible labels according to our design. However, since the division points locate equally in the whole range regardless of the distribution of decision values, this division may yield poor results for some data sets.

In this section, we will propose an alternative method that divides examples into categories by applying a k-means clustering on the decision values. This algorithm was introduced in our previous publications [80]. The difference of this taxonomy design compared to the previous one is that it uses a different method to separate the decision values sequence instead of dividing them into equal length intervals. It can adjust the division points automatically according to the distribution of decision values. The process to acquire combined decision values is the same as SVM taxonomy with equal length intervals does.

4.3.1 Dividing Intervals by k-Means Clustering

Instead of dividing the intervals homogeneously, a new dividing scheme was then developed, which uses k-means clustering [19, 38] to divide all decision values. This idea happened when we tried to find the internal properties of a sequence of values for a better division, which is also what cluster analysis is capable of.

k-means clustering is a cluster analysis method that aims to divide n objects into k clusters in which each object belongs to the cluster with the nearest mean. Given a sequence of objects $(x_1, x_2, \ldots, x_n)$, where each object $x_i \in \mathbf{R}^d$ is a d-dimensional real vector, k-means clustering aims to partition the n objects into k ($k \leq n$) sets: $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimise the within-cluster sum of squares (WCSS):

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - m_i\|^2 \tag{4.10}$$

where m_i is the mean value of points in S_i.

The most commonly used algorithm to solve the problem uses an iterative refinement

technique. Given an initial set of k-means $m_1^{(1)}, \ldots, m_k^{(1)}$, the algorithm proceeds by alternating between the following two steps:

Assignment step: Assign each object to the cluster whose mean is closest to it.

$$S_i^{(t)} := \{x_p : ||x_p - m_i^{(t)}|| \leq ||x_p - m_j^{(t)}||, \ \forall 1 \leq j \leq k\}$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be is assigned to two or more of them. t is the index of the current iteration.

Update step: Calculate the new means to be the centroids of the objects in the new clusters.

$$m_i^{(t+1)} := \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

The algorithm has converged when the assignments no longer change.

Commonly used initialization methods are Forgy Method and Random Partition [24]. In Forgy method, k initial means are randomly chosen from the data set while each object from the data set will be randomly assigned an initial cluster in Random Partition method. The Forgy method tends to spread the initial means out, while Random Partition places all of them close to the centre of the data set. According to Hamerly et al. in [24], the Forgy method of initialization is preferable for standard k-means algorithms.

In our design, dimension d is fixed to "1", while the number of clusters is equal to the number of possible labels K. So the heuristic algorithm we used could be described as below.

1. k initial mean values are randomly generated within the data domain.
2. k clusters are created (or reassigned) by associating every object with the nearest mean value.
3. The centroid of each of the k clusters becomes the new mean value.
4. Steps 2 and 3 are repeated until the change of WCSS (4.10) between two states declines to be less than the terminal tolerance $\epsilon = 10^{-4}$.

When the heuristic algorithm finishes, we have K clusters and assign each cluster a category. Examples whose combined decision values are in the same cluster will be in the same category. The taxonomy A_n can be defined as Eq. (4.11).

$$\tau_i := A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, (x_n, y)\int, z_i) = \arg\min_j \|x_i - m_j\|, \\ \text{where } m_j = \frac{1}{|S_j|} \sum_{x_k \in S_j} x_k \tag{4.11}$$

Having applied k-means clustering, we divided the decision values into categories that could be used to calculate the matrix for new examples and make the probabilistic predictions as the standard Venn Machine does. The pseudo-code of our algorithm is listed as below in Algorithm 8.

4.4 Taxonomy with Multi-class SVM by Crammer and Singer

In §4.1 and §4.2, we introduced some SVM-based taxonomy designs that decomposes a multi-class SVM problem into several binary-class SVM sub-problems. In this section, we will introduce a new taxonomy design based on multi-class SVM that can be solved as a single optimization problem.

As we mentioned in §2.4.4, Crammer and Singer proposed a multi-class SVM in [12]. Crammer and Singer's Support Vector Machines (abbrev. MCSVM) starts from a generalized notion of the margin to multi-class problems. Using this notion they cast multi-class categorization problems as a constrained optimization problem with a quadratic objective function. Unlike most of the previous approaches that typically decompose a multi-class problem into multiple independent binary classification tasks, the Crammer and Singer's notion of margin yields a direct method for training multi-class predictors. By using the dual form of the optimization problem, this algorithm is able to incorporate kernels with a compact set of constraints and decomposes the dual form problem into multiple optimization problems of reduced size.

Algorithm 8: SVM Venn predictor with k-means clustering

Data: training set $\mathbf{Z}^{(n-1)}$, new object x_n, label space $\mathbf{Y} = \{0, 1, \ldots, K-1\}$

Result: predicted label $\hat{y}_n$ with probability interval $[l_n, u_n]$

initialization;

$D \leftarrow$ `TrainSVM(`$\mathbf{Z}^{(n-1)}$`);`

for $i = 1$ **to** n **do** `/* calculate combined decision values */`

  $d_i \leftarrow D(x_i)$;

end

$\{\tau_i\} \leftarrow$ `kMeansClustering(`$\{d_i\}$`)` ; `/* assign categories to all examples */`

for $y = 0$ **to** $K-1$ **do** `/* try every possible label */`

  $\mathcal{C} \leftarrow \varnothing$;

  for $i = 1$ **to** n **do** `/* find examples assigned to same category of` x_n `*/`

    if $\tau_i = \tau_n$ **then**

      $\mathcal{C} \leftarrow$ `AddToSet(`$\mathcal{C}$, y_i`)`

    end

  end

  $\mathbf{F}_y \leftarrow$ `CalcFrequency(`$\mathcal{C}$`);`

end

$\hat{y}_n \leftarrow$ `FindBestColumn(`$\mathbf{F}$`);`

$[l_n, u_n] \leftarrow$ `FindInterval(`$\hat{y}_n$, $\mathbf{F}$`);`

return $\hat{y}_n$, $[l_n, u_n]$

Suppose we are given a multi-class classification problem: $\wr z_1, \ldots, z_{n-1}, (x_n, y) \int$ and the label space is $\mathbf{Y} = \{0, 1, \ldots, K-1\}$ and we need to partition this bag into T different categories. Instead of decomposing a multi-class problem into multiple independent binary classification tasks, they construct a multi-class predictor H in Eq. (4.12).

$$H_{\mathbf{M}}(\mathbf{x}) := \underset{r=0}{\overset{K-1}{\operatorname{argmax}}} \{\mathbf{M}_r \cdot \mathbf{x}\} \tag{4.12}$$

Through a series of reformulations and transformations, the single optimization problem

Function kMeansClustering($\{d_i\}$)

```
Function kMeansClustering({d_i})
    for i = 1 to K do                                   /* initial k mean values */
        m_i ← i − 0.5 ;                          /* initial current centroids to i-0.5 */
        l_i ← 0 ;                                      /* initial last centroids to 0 */
    end
    while not IsConverge({m_i}, {l_i}) do                          /* hasn't converge */
        for i = 1 to K − 1 do                          /* calculate division points */
            b_i ← (m_i + m_{i+1}) / 2;
        end
        {τ_i} ← ReassignCategory({d_i}, {b_i});
        for i = 1 to K do                                /* recalculate mean values */
            l_i ← m_i ;                                        /* store current state */
            sum ← 0 ;                                 /* initial sum and count to 0 */
            count ← 0;
            for j = 1 to n do
                if τ_j = i then                    /* find examples of same category */
                    sum ← d_j + sum;
                    count ← 1 + count;
                end
            end
            m_i ← sum / count ;                                 /* calculate mean value */
        end
    end
    return τ_1, ..., τ_n
end
```

Function IsConverge($\{m_i\},\ \{l_i\}$)

```
Function IsConverge({m_i}, {l_i})
    for i = 1 to K do /* compare current centroids with last centroids */
        if |m_i − l_i| > ε then                          /* still has changes */
            return FALSE
        end
    end
    return TRUE
end
```

Function ReassignCategory($\{d_i\},\ \{b_i\}$)

```
Function ReassignCategory({d_i}, {b_i})
    for i = 1 to n do                                    /* reassign categories */
        for j = 0 to K − 1 do
            if b_j ≤ d_i < b_{j+1} then
                τ_i ← j + 1;
            end
        end
    end
    return {τ_i}
end
```

can be written as Eq. (4.13).

$$-\frac{1}{2}\sum_{i,j}\mathcal{K}(\mathbf{x}_i,\mathbf{x}_j)(\boldsymbol{\alpha}_i\cdot\boldsymbol{\alpha}_j)+\beta\sum_i\boldsymbol{\alpha}_i\cdot\mathbf{1}_{y_i}\rightarrow\max_{\boldsymbol{\alpha}}$$
$$s.t.\ \forall i,\quad \boldsymbol{\alpha}_i\leq\mathbf{1}_{y_i}\quad\text{and}\quad\boldsymbol{\alpha}_i\cdot\mathbf{1}=0 \tag{4.13}$$

The detailed derivation of this dual form problem is described in §2.4.4. By solving the optimization problem in Eq. (4.13), we can get the classifier of Crammer and Singer's multi-class SVM as Eq. (4.14).

$$\hat{y}_n:=H(\mathbf{x}_n)=\operatorname*{argmax}_{r=0}^{K-1}\Big\{\sum_i\alpha_{i,r}\mathcal{K}(\mathbf{x}_i,\mathbf{x}_n)\Big\} \tag{4.14}$$

According to our previous experience in designing taxonomies using Support Vector Machines, we can use the predictions of MCSVM, the similarity scores, which are intermediate values before adopting the argument of the maximum function, and the intermediate variables α in our taxonomy designs. Here, we use the simplest outputs, the predictions, to design our taxonomy. The taxonomy assigns the category of example z_i the same as its predicted label $\hat{y}_i$. Examples that are predicted in the same class share the same category. Therefore, the number of categories T equals the number of possible labels K. This taxonomy A_n is defined as follows.

$$\tau_i:=A_n(\wr z_1,\ldots,z_{i-1},z_{i+1},\ldots,(x_n,y)\int,z_i)=\operatorname*{argmax}_{r=0}^{K-1}\Big\{\sum_{j=1,\ldots,n}\alpha_{j,r}\mathcal{K}(\mathbf{x}_j,\mathbf{x}_i)\Big\} \tag{4.15}$$

4.5 Regression Venn Machine and Its Taxonomies

Due to the internal mechanism that Venn predictors calculate the frequency matrix of labels as estimation of the distribution, the standard Venn predictors could only deal with classification problem. A simple workaround is to discretize the real-valued labels into categorical labels before adopting a standard Venn predictor. However, the original Venn predictors give a single predicted label and lower and upper probabilistic bounds as estimations for the certainty. Therefore, we need to transform the estimation of the distribution to some meaningful real values as predictions. Inspired by Regression Conformal Prediction proposed in [46], we designed Venn predictors for regression. The predictors give an

interval with a given significance level ϵ that the true values would fall into this interval. This would sound more reasonable than a categorical label as the single prediction with probabilistic bounds in regression problems.

The basic idea of our Regression Venn Machine could be described as follows: firstly we discretize the y values of all training examples of a regression problem into several bins, and we assign a label for each bin to transform the regression problem to a classification problem. Secondly, a typical Venn Machine for classification is applied. While we obtain the frequency matrix, we also know which examples are counted in each hypothesis. Finally, we group the y values of those examples and fit a normal distribution to them before we predict an interval for the new object at different confidence levels.

The taxonomy we use in our algorithm is k-Nearest Neighbours taxonomy mentioned in §4.1.1. It is not complicated: two examples are assigned to the same category if their nearest neighbours have the same majority label. This makes the number of categories equal to the number of discretized labels.

Assume that we are given a regression problem. The training set consists of $n-1$ examples $z_1, z_2, \ldots, z_{n-1}$. We are also given a new object x_n. Each z_i is a pair of x_i and y_i, where y_i is a real number (i.e. $y_i \in \mathbf{R}$). Since the training set is finite, we have a range for y, let us say $[a, b]$. So that, $\forall 1 \leq i \leq n-1 (i = 1, 2, \ldots, n-1), a \leq y_i \leq b$. Let the number of labels be K and assign every example in training set a new label y'_i if

$$a + (b-a)\frac{y'_i}{K} < y_i \leq a + (b-a)\frac{y'_i + 1}{K}, y'_i = 0, 1, \ldots, K-1 \tag{4.16}$$

When the new object x_n comes, we make a hypothesis for each of all possible labels by assigning $y'_n = 0, 1, \ldots, K-1$ to x_n as its label. In the meanwhile, we calculate the Euclidean distance of new object x_n and example x_i as follows:

$$d(x_n, x_i) := \|x_n - x_i\| = \sqrt{\sum_{t=1}^{dim}(x_{n,t} - x_{i,t})^2} \tag{4.17}$$

where dim is the dimension of the feature space. We find all k nearest neighbours of new object x_n measured by Euclidean distance and include them into a set $\mathbf{S}_n$. We assign the

category τ_n of the new object with the majority label among all its neighbours set $\mathbf{S}_n$:

$$\begin{aligned}\tau_n :=& A_n(\wr z_1, \ldots, z_{n-1}\int, (x_n, y'_n)) \\ =& \underset{y}{\operatorname{argmax}} |\{(x^*, y^*) \in \mathbf{S}_n : y^* = y, \quad y = 0, 1, \ldots, K-1\}|\end{aligned} \tag{4.18}$$

For example z_i, we assign its category τ_i the same way.

$$\begin{aligned}\tau_i :=& A_n(\wr z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_{n-1}, (x_n, y'_n)\int, z_i) \\ =& \underset{y}{\operatorname{argmax}} |\{(x^*, y^*) \in \mathbf{S}_i : y^* = y, \quad y = 0, 1, \ldots, K-1\}|\end{aligned} \tag{4.19}$$

In each hypothesis $y'_n = i,\ i = 0, 1, \ldots, K-1$, we assign categories to all examples, find examples in the same category as (x_n, y'_n) and calculate their frequencies of labels.

After we tried every hypothesis, we could create a $K \times K$ frequency matrix $\mathbf{F}$ with each row the frequency of each label. Together with the matrix, we also know exactly which examples are associated with which frequency in the matrix. The corresponding examples are considered having something in common with each other. Hence, we could use these examples to estimate the prediction of the new object. The *quality* of a column of this matrix is its minimum entry. We call the column with the highest *quality* the *best* column. Let the *best* column be j_{best}. Then we group all the corresponding y values from the examples associated with the frequency in matrix $\mathbf{F}$ by the following approaches:

1. Choose all the examples from the i^*th row that includes the highest *quality*.

$$i^* := \underset{i}{\operatorname{argmin}}\, \mathbf{F}_{i,j_{best}} \tag{4.20}$$

2. Choose all the examples from the j^*th column that is the *best* column.

$$j^* := j_{best} \tag{4.21}$$

3. Choose only the examples from the block of the highest *quality*. A block is a set that contains all examples associated with the chosen frequency in the matrix, which is $\mathbf{F}_{i^*,j^*}$ here.

$$i^* := \underset{i}{\operatorname{argmin}}\, \mathbf{F}_{i,j_{best}},\ j^* = j_{best} \tag{4.22}$$

4. Choose all the examples from every row.

After we have the group of examples, we generate a histogram and fit a normal distribution $\mathcal{N}(\mu_n, \sigma_n^2)$ to it. We predict with $1-\epsilon$ confidence that y_n will be in the interval based on Fisher's prediction interval [17]:

$$\mu_n - t_\infty^{\frac{\epsilon}{2}} \sigma_n \sqrt{\frac{n-1}{n-2}} < y_n < \mu_n + t_\infty^{\frac{\epsilon}{2}} \sigma_n \sqrt{\frac{n-1}{n-2}} \tag{4.23}$$

where ϵ is a user-defined significance level. $t_\infty^{\frac{\epsilon}{2}}$ is the point that the t-distribution with ∞ degrees of freedom exceeds exactly $\frac{\epsilon}{2}$ of the time. We use $t_\infty^{\frac{\epsilon}{2}}$ instead of $t_{n-2}^{\frac{\epsilon}{2}}$ for the reason that we need to make sure we have enough examples in each category, which is usually larger than 100. Since $t_\infty^{\frac{\epsilon}{2}}$ is very close to $t_{n-2}^{\frac{\epsilon}{2}}$ when $n > 100$, we use $t_\infty^{\frac{\epsilon}{2}}$ instead to reduce calculation.

When given a significance level ϵ, our Regression Venn Machines will pronounce an interval

$$[\mu_n - t_\infty^{\frac{\epsilon}{2}} \sigma_n \sqrt{\frac{n-1}{n-2}}, \mu_n + t_\infty^{\frac{\epsilon}{2}} \sigma_n \sqrt{\frac{n-1}{n-2}}] \tag{4.24}$$

with $1-\epsilon$ confidence that the true value y_n falls into this interval. For Method (4), we could generate intervals for each hypothesis (i.e. row) respectively, and use the union or intersection of all intervals as the final interval.

4.5.1 Choosing Grouping Method

We tried the different grouping methods we mentioned above and compared the results. Method (2) and Method (3) achieved worse results than Method (1), of which the percentage outside predictions both dropped by 6%-7% compared with Method (1) while the width increased by 25%-60% on the contrary compared with Method (1).

The results of Method (4) are quite similar to the results of Method (1). When we looked into it, we found that whatever the number of categories is there are only two kinds of distribution. One is the distribution when the category τ_n of the new object x_n is different from the hypothesis y'_n. The other is when they are the same. This is because

the distance between these objects and the discretized labels are fixed when giving these examples and so is the category of the new object x_n. When τ_n is different from y'_n, the examples in the same category as the new object are those which had the same category before except those which have changed its category to y'_n when the new object x_n is introduced. When τ_n has the same value of y'_n, the examples in the same category as the new object are those which had the same category before together with those which have changed its category to y'_n. The difference between two kinds of distributions is not significant, which resulted in the similar results achieved by Method (1) and Method (4).

So in our algorithm, we choose Method (1) to group the examples.

4.6 Summary

In this chapter, we introduced five formerly proposed taxonomy designs. Among them are k-Nearest Neighbours taxonomy, SVM taxonomy with predictions, SVM taxonomy with equal size, SVM taxonomy with Lagrange multipliers and taxonomy with One-vs-All SVM.

We also proposed three new taxonomy designs for Venn Machines. Among them are SVM taxonomy with equal length, SVM taxonomy with k-means clustering and taxonomy with Crammer and Singer's multi-class SVM.

SVM taxonomy with equal length calculates the distances from examples to the hyperplane obtained by training an SVM classifier on the data set and sort the examples according to their distances, then the whole range of distances is separated into several intervals with equal length and the examples are divided into different categories based on the interval their distances fall into. SVM taxonomy with k-means clustering shares the same idea of calculating the distances as SVM taxonomy with equal length does. The difference is that instead of dividing them into several intervals this taxonomy divides examples through k-means clustering. Taxonomy with multi-class SVM uses the predictions given by Crammer and Singer's multi-class SVM to replace the combined decision values.

By using these taxonomies, we extend Venn Machine with SVM from binary only to multi-class applicable, which is one of the main novelty in this chapter. Also, the property of validity is not effected by the underlying algorithms used in the algorithm. Hence, the probabilities produced by applying these taxonomies into Venn Machines are valid.

A method of dealing with Regression Venn Machines was also introduced in the chapter. This method discretizes the real-valued labels of a regression problem and treats them as a classification problem. Then it solves this problem and groups the outputs into a region prediction.

Chapter 5

Experiments and Results

To have a more intuitive performance comparisons of our algorithms with other algorithms, we tested our algorithms on several benchmark data sets in both offline setting and online setting. The detailed information of the experiments will be given in the following sections of this chapter.

In §5.1, we will give the basic introductions of all the benchmark data sets we used in our comparisons. This introduction includes the source, the usage, the content and the basic dimensional characters of these data sets. Some additional information could be found in the Appendix A.

In §5.2, we will describe the ways how we evaluate these algorithms in different aspects. These measures include accuracies, probabilistic estimates, the average width of the probability bounds, Brier score and logarithmic loss.

§5.3 gives the settings for all the parameters of all these algorithms along with the reason we choose these values for the parameters. This section also includes some empirical results to support the choice of the specific values.

Then in the following three sections §5.4 to §5.6, we will give the detailed results of our comparisons between all these algorithms. The results were organized into different tables and figures by data sets. Through all these three sections, we will cover the novel

findings we acquired during interpreting the results.

At last, in §5.7, we will summarize this chapter and reiterate our findings on the results.

5.1 Data Sets

To compare all our algorithms, 12 data sets from the real world were used, which could be easily obtained from some public data sets repository like UCI Repository [61] and Delve Datasets [60] except SVMguide1 obtained from the website of LibSVM [9].

These data sets cover different fields, among them are medical diagnosis, bioinformatics, image recognition, price prediction and so on. These data sets can be separated into two type based on the type of problems raised in the data sets. One is a set of classification data sets that take categorical values as labels. The other is a set of regression data sets that take continuous values as labels. We will introduce them separately.

Some brief introductions of classification data sets are as follows.

- **WBC** is the abbreviation for Wisconsin Breast Cancer data set. This breast cancer data set was created by Dr. William H. Wolberg from the University of Wisconsin Hospitals [75]. The problem proposed in this data set is to diagnose whether a breast cancer is benign or malignant based on the clinical report. All the examples in this data set are based on clinical cases, which contains 10 features with information about breast cancer. Except the first feature, which is the ID number of the example, the other features are all discrete numbers ranging from 1 to 10. The two possible labels are benign or malignant. Note that the original data set has 699 examples including 16 instances with missing values. Hence, when we conduct experiments on this data set, we simply removed those instances with missing values to shrink the size of data set into 683.

- **SVMguide1** is a data set of an astroparticle application from Jan Conrad of Uppsala University, Sweden. This data set was used and cited in [29], however, it did not mention the source or any other details of this data set. Hence, the detailed

information of this data set remains unknown. There are 3089 examples used as training examples and 4000 used for testing. Each example consists of 4 continuous features and can be categorized into 2 classes.

- **Splice** is short for primate splice-junction gene sequences (DNA) with associated imperfect domain theory (Delve version). The original data set was donated by G. Towell, M. Noordewier and J. Shavlik [43]. The problem proposed in this data set is to recognize the boundaries between exons and introns from a given sequence of DNA. The exons are the parts of the DNA sequence retained after splicing, while introns are the parts of the DNA sequence that are spliced out. The original data set contains 3190 examples with 60 sequential DNA nucleotide positions as features, which can be represented as "A", "G", "T", "C", "D", "N", "S" ans "R". However, the incidence of "D", "N", "S" ans "R" are very tiny, only 15 among all 3190 examples. Hence, in this version of the data set, all examples with any of "D", "N", "S" ans "R" categories have been deleted. Now, each of the 60 features is always filled by one of 4 types of nucleotide. The original data set has three classes, which are "IE" (intron to exon boundary, sometimes called "donors"), "EI" (exon to intron boundary, sometimes called "acceptors") and "NE" (neither of them). In this version of the data set, the examples assigned to "IE" and "EI" are merged to make a new class while the examples with label "NE" remain the same. Now, the data set contains 3175 examples with 60 categorical features, while each example belongs to one of two possible classes.

- **USPS** is a handwritten digits recognition data set given by J. Hull in [30]. The problem proposed in this data set is to recognize the digits according to the grey-scale images. All examples in this data set are handwritten digits automatically scanned from envelopes by the U.S. Postal Service (USPS) Then these scanned images have been de-slanted and normalized into the same size, resulting in 16×16 grey-scale images. This data set includes 9298 examples, of which 7291 examples are marked

as the training set and 2007 examples are marked as the testing set. Each example contains 256 grey-scale values of all pixels in the images as features. All the examples are assigned to one of the 10 digits from "0" to "9".

- **Satimage** is the Landsat Satellite Image data set provided by Ashwin Srinivasan, University of Strathclyde. The problem proposed in this data set is to identify the object of the central pixel based on itself and all 8 pixels in its neighbourhood from a satellite image. The data set was generated from the multi-spectral scanner image in 4 different spectral bands. Hence, each example consists of 36 values of 3×3 pixels in the neighbourhood from 4 multi-spectral scanner images as features. Each of these values is numerical, ranging from 0 to 255. This data set contains 6435 examples, while 4435 examples are marked as the training set and 2000 examples are marked as the testing set. The whole data set is classified into 6 classes, each class stands for one landform and is coded as a number.

- **Segment** is the Image Segmentation data set donated by Vision Group, University of Massachusetts. The problem proposed in this data set is to identify the objects in a segmented piece from 7 outdoor images. The examples were drawn randomly from a database and hand-segmented into 3×3 pixel piece. Each instance consists of 19 continuous values carrying different kinds of information including the colour, the intensity, the contrast, the line density and so on. This data set contains 2310 examples of 7 different types.

- **DNA** is also the data set of primate splice-junction gene sequences (DNA) with associated imperfect domain theory (Statlog version). However, this data set is a refined version produced by Ross King at Strathclyde University. The source of this data set is the same as Splice data set, but the preprocessing is different. Each types of the 4 nucleotide ("A", "C", "T", "G") was replaced by 3 binary indicator. Therefore, the number of features increased to 180 from 60. At the meanwhile, this data set kept the original three classes and removed 4 ambiguous examples from the

original data set. The final data set contains 3186 examples with 180 features of binary value. Each example belongs to one of the three classes.

- **Wine** is for Wine Recognition data set donated by S. Aeberhard [1] The problem proposed in this data set is to identify three different cultivars through the results of a chemical analysis of the wines. The quantities of 13 constituents were determined in the analysis and were used as the features of all the examples. All these features are continuous. This data set contains 178 examples from 3 different classes.

- **Vehicle** is the Vehicle Silhouettes data set donated by Turing Institute [56]. The purpose of this data set is to identify the type of vehicle from a given silhouette. The silhouette images of vehicles were captured by a camera. Every example contains 18 features, which were a combination of independently-scaled features utilizing both classical measures and heuristic measures extracted from the silhouettes of 4 types of vehicles. Note that the original data set consists of 946 instances, however 100 instances are being kept by the donor. Therefore, this data set contains 846 examples with 18 features, while each example can be classified into 4 different types.

Some brief introductions of regression data sets are as follows.

- **Housing** is the Boston Housing data set created and donated by D. Harrison and D.L. Rubinfeld [26]. The purpose of this data set is to predict the median value of an owner-occupied home in suburbs of Boston. The data set contains 12 continuous and 1 binary-valued features, which cover different aspects in this area such as the crime rate, average number of rooms per dwelling, weighted distances to five Boston employment centres and so on. The labels are the median value in $1000's of a house described by the above 13 features, ranging from 5.0 to 50.0. This data set consists of 506 examples.

- **Abalone** is the Abalone data set donated by Sam Waugh [72]. The problem raised in this data set is to predict the age, which is a boring and time-consuming task,

of abalone from some physical measurements, which are easy to get. The physical measurements include some geometric measurements and some weights in different states, resulting in 7 continuous and 1 nominal features such as length, height, whole weight and so on. The labels are the number of rings in the cut shell, which can be treated as the age of abalone, ranging from 1 to 29. This data set consists of 4177 examples.

- **CompActi** is the Computer Activity data set from Delve. The purpose of this data set is to predict the portion of time that CPUs run in user mode from a large variety of computer activities. The data was collected from a computer running in a multi-user university department. The users were doing all kinds of tasks including accessing the Internet, editing files or running very CPU-bound programs. Every 5 seconds, some system measures were recorded. 12 features were extracted from the system measures to be used for predicting the portion of time. This data set consists of 8192 examples.

In our experiments, the separated training and testing sets were blended into one for every data set. Each feature was normalized to the range of $[-1, 1]$.

The classification data sets we used in offline and online comparisons could be divided into two parts based on their number of classes. The binary-only data sets are listed in the first half, while the multi-class data sets are listed in the second half. Some dimensional characteristics of these data sets including the size of data sets, the number of features and the number of classes are summarised in Table 5.1.

Some dimensional characteristics of regression data sets including the size of data sets, the number of features and the number of classes are summarised in Table 5.2.

Across all data sets, there are data sets of different types, different numbers of features in different fields. The diversity of data sets can help us assess the best algorithms on all kinds of data sets.

Table 5.1: Main dimensional characteristics of classification data sets.

Data Set	# of Examples	# of Features	# of Classes
WBC	683	10	2
SVMguide1	7089	4	2
Splice	3175	60	2
USPS	9298	256	10
Satimage	6435	36	6
Segment	2310	19	7
DNA	3186	180	3
Wine	178	13	3
Vehicle	846	18	4

Table 5.2: Main dimensional characteristics of regression data sets.

Data Set	# of Examples	# of Features	Label Range
Housing	506	13	45
Abalone	4177	8	28
CompActi	8192	12	99

5.2 Evaluations of the Algorithms

In our experiments, we used several measurements to evaluate the performance of algorithms.

The first measurement is the accuracy of predictions, which is also one of the most important measurements when evaluating an algorithm. Accuracy calculates the proportion of examples that are correctly classified by a classifier out of all testing examples. Given a trained classifier $f(\cdot)$ and a set of n testing examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, the accuracy

C_n can be calculated as

$$C_n := \frac{1}{n}\sum_{i=1}^{n} [\![\hat{y}_i = y_i]\!], \quad \hat{y}_i = f(x_i) \tag{5.1}$$

Let $[\![\pi]\!]$ be 1 if the predicate π holds and 0 otherwise. When evaluating algorithms in online settings where the testing examples are coming in succession, we calculate the accuracy of the whole data set after we achieve all predicted results. The higher the accuracy is, the better we consider the algorithm.

The second measurement is the average probabilistic outputs. The probabilistic outputs are estimates of reliability that the predictions given by a classifier are correct. Single probabilistic predictors announce single likelihood while multi-probabilistic predictors give multiple estimates of the reliability. However, to evaluate a probability for a prediction seems infeasible, since we have no clue as to the true probability in most cases. We can use the label's relative frequency to estimate its probability, when the experiments are conducted in a significant number of trials according to [66]. Hence, we use the average probabilities to estimate the accuracy.

For single probabilistic predictor, the probabilistic output is

$$\bar{p}_n := \frac{1}{n}\sum_{i=1}^{n} p_i \tag{5.2}$$

The closer to the accuracy this probability is, the better we consider the algorithm is.

For multi-probabilistic predictor, we predict lower and upper bounds, referred to as $[l_i, u_i]$, for a prediction $\hat{y}_i$, therefore our probabilistic outputs are

$$\bar{l}_n := \frac{1}{n}\sum_{i=1}^{n} l_i \tag{5.3}$$

$$\bar{u}_n := \frac{1}{n}\sum_{i=1}^{n} u_i \tag{5.4}$$

If the accuracy is hedged well by the average lower and upper bounds, we consider the algorithm is good.

When we apply algorithms on data sets with online setting, we use error rate and error probability interval, which complement the accuracy and probability interval, to

evaluate the algorithms. We refer to error probability interval as $[l'_i, u'_i]$, which equals to $[1-u_i, 1-l_i]$. The following curves are used in the figures to demonstrate the performance of the algorithms in online setting. the cumulative error curve:

$$E_n := \sum_{i=1}^{n} [\![\hat{y}_i \neq y_i]\!] \tag{5.5}$$

the cumulative lower error probability curve:

$$L_n := \sum_{i=1}^{n} l'_i \tag{5.6}$$

and the cumulative upper error probability curve:

$$U_n := \sum_{i=1}^{n} u'_i \tag{5.7}$$

We consider that the cumulative error probability bounds of a good algorithm can hedge the cumulative error curve well, which means the cumulative error curve will lay between the two cumulative bounds.

We use our third measurement to measure the narrowness of the bounds. The average width W_n measures the average width between the lower and upper bounds.

$$\bar{W}_n := \frac{1}{n}\sum_{i=1}^{n}(u_i - l_i) = \frac{1}{n}\sum_{i=1}^{n} u_i - \frac{1}{n}\sum_{i=1}^{n} l_i = \bar{u}_n - \bar{l}_n \tag{5.8}$$

If the average width of a pair of bounds is small, we consider that these probabilistic estimates are good.

We also use Brier score and logarithmic loss to evaluate the validity of the probabilistic outputs. The detailed description of these two measurements is given in the next section.

5.2.1 Brier Score

The Brier score is a proper score function that measures how accurate the probabilistic outputs are. It was proposed by Glenn W. Brier in 1950 [8].

It applies to those classifiers that give probability estimates to all possible labels like all the algorithms we described in this thesis. The number of classes can be any finite

number, which means this measurement can be applied to algorithms for both binary and multi-class data sets. In addition, the probability assigned to each class is in the range of 0 to 1, and all probabilities must sum to 1.

The basic idea of the Brier score is to measure the mean squared error between the predicted probability $\hat{p}_i$ for ith class and the actual incidence o_i, where o_i equals to 1 if the predicted label is the label for ith class otherwise it equals to 0. Hence, the more accurate the predicted probability is, which is close to 1 when predicted label is true label otherwise close to 0, the lower Brier score is. The most common formulation of the Brier score is described as

$$\text{BS} := \frac{1}{n}\sum_{i=1}^{n}(\hat{p}_i - o_i)^2 \tag{5.9}$$

However, Eq. (5.9) is only for binary case and omits half of the squared error, since $(\hat{p}_i - o_i)^2$ for one class share the same value with $(\hat{p}_i - o_i)^2$ for the other class.

In our experiments, several data sets were conducted including both binary and multi-class data sets. Therefore, we need a generalized version of Brier score as the measurement. The original definition by Brier [8] is already applicable to multi-class classification problems. Given a classifier $f(\cdot)$ and a set of n testing examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, let $\hat{p}_{ij}$ be the predicted probability for jth class in ith example and o_{ij} be the actual incidence for jth class of ith example. The original formulation of the Brier score is described as

$$\text{BS} := \frac{1}{n}\sum_{i=1}^{n}\sum_{j=0}^{K-1}(\hat{p}_{ij} - o_{ij})^2 \tag{5.10}$$

The Brier score in Eq. (5.10) takes on a value between 0 and 2.

In our experiments, we use the generalized version of Brier score in Eq. (5.10) for both binary and multi-class problems. Additionally, for algorithms that give multiple probabilistic estimates, we use the average probabilities to calculate the Brier score.

5.2.2 Logarithmic Loss

The logarithmic loss (LogLoss), which is also called the cross-entropy error function [5, p. 209], has be used as an alternative to the mean squared error. It can also be used as a

measurement for the accuracy of the predicted probabilities.

This error measure applies to those classifiers that announce probabilities that each label might be true for both binary and multi-class data sets. Again, all the algorithms we studied in this thesis can apply this measurement. The total sum of probability for each label should be 1. Meanwhile all probabilities range from 0 to 1.

The basic idea of the logarithmic loss is to calculate the entropy of the true label and the predicted probability $\hat{p}_i$. Given a classifier $f(\cdot)$ and a set of n testing examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, the formulation of the logarithmic loss for binary case is as follows

$$\text{LogLoss} := -\frac{1}{n}\sum_{i=1}^{n}\left[y_+ \log \hat{p}_i + y_- \log\left(1 - \hat{p}_i\right)\right] \tag{5.11}$$

In Eq. (5.11), the binary label y_-, y_+ is denoted by $0, 1$, while $\hat{p}_i$ is the estimated probability for label y_+ in ith example. From the formulation, we could find that the use of logarithm on the error provides extreme punishments for being both confident and wrong. Hence, the lower the logarithmic loss is, the better probabilistic estimates an algorithm gets.

In our experiments, we use a generalized version of logarithmic loss for both binary and multi-class data sets. Let $\hat{p}_{ij}$ be the predicted probability for jth class in ith example and o_{ij} be the actual incidence for jth class of ith example. o_{ij} is equal to 1 if label j is the true label otherwise it equals 0. The generalized version is described in Eq. (5.12)

$$\text{LogLoss} := -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=0}^{K-1} o_{ij} \log \hat{p}_{ij} \tag{5.12}$$

In the worst possible case, a predictor gives a predicted label with 100% probability when it turns out to be actually false will add infinite to the error score and make every other entry pointless. In our experiments, original predicted probabilities are bounded away from the extremes by using

$$\max(\min(\hat{p}, 1 - 10^{-15}), 10^{-15}) \tag{5.13}$$

to prevent the infinite in logarithm. Additionally, for algorithms that give multiple probabilistic estimates, we use the average probabilities to calculate the logarithmic loss.

5.3 Experimental Settings

In all of our experimental results, we used some abbreviations instead of the full names of our algorithms for convenience. Among them are:

- **NaiveBayes** stands for Naive Bayes classifier given in §2.5.1.
- **LogReg** stands for Logistic Regression given in §2.5.2.
- **SigSVM** stands for Platt Scaling given in §2.5.3.
- **VM-KNN** stands for Venn Machine with k-nearest neighbours introduced in [69].
- **CP-BivIR** stands for conformal predictor with bivariate isotonic regression given in §3.3.
- **VA-SVM** stands for Venn-ABERS predictor with SVM given in §3.2.
- **VM-SVM-EL** stands for SVM Venn Machine with equal length intervals given in §4.2.
- **VM-SVM-ES** stands for SVM Venn Machine with equal size intervals introduced in [14].
- **VM-SVM-OA** stands for Venn Machine with One-vs-All SVM introduced in [37].
- **VM-SVM-KM** stands for SVM Venn Machine with k-means clustering intervals given in §4.3.
- **VM-MCSVM** stands for Venn Machine with Crammer and Singer's multi-class SVM given in §4.4.

Together with all our designs of multi-probabilistic predictors, traditional or widely-used algorithms were added to the comparison as benchmarks. They are NaiveBayes for Naive Bayes classifier, LogReg for Logistic Regression, SigSVM for Platt Scaling and VM-KNN for Venn Machine with k-nearest neighbours. Hence, there are 11 algorithms in the comparisons overall.

For every data set we used in the experiments, we applied a grid search to find the best parameters of SVM. All our algorithms using SVM shared the same SVM parameters in each comparison.

In addition, the grid search for each data set was carried out on the parameter C for the cost in SVM and the parameter γ in the radial basis function (RBF) of SVM.

The searching range for C was from 2^{-5} to 2^{15} with a step of 2^2, while the searching range for γ was from 2^{-15} to 2^3 with a step of 2^2. For each trial on each pair of C and γ, we carried out a 5-fold cross-validation test. Then we compared the accuracies and recorded the best pair in Table 5.3.

Table 5.3: Parameters for underlying algorithms

Data Set	k for k-NN	Kernel function for SVM	Parameters for kernel function
WBC	1	RBF	$C = 2^9,\ \gamma = 2^{-7}$
SVMguide1	1	RBF	$C = 2^9,\ \gamma = 2^1$
Splice	1	RBF	$C = 2^1,\ \gamma = 2^{-5}$
USPS	1	RBF	$C = 2^1,\ \gamma = 2^{-5}$
Satimage	3	RBF	$C = 2^1,\ \gamma = 2^1$
Segment	1	RBF	$C = 2^7,\ \gamma = 2^{-1}$
DNA	5	RBF	$C = 2^1,\ \gamma = 2^{-7}$
Wine	3	RBF	$C = 2^1,\ \gamma = 2^{-3}$
Vehicle	3	RBF	$C = 2^7,\ \gamma = 2^{-3}$

Offline Settings for Classification

In offline setting, we conducted experiments on all 11 algorithms. However, since Venn-ABERS predictor is only applicable to binary data sets, this algorithm was only tested on WBC, SVMguide1 and Splice data sets.

Before each experiment, we merged the original training and testing sets together to create a whole data set for each experiment. Then we applied five cross-validation tests for each whole data set to avoid over-fitting on the original pair of training and testing sets. In each cross-validation test, we kept the same settings. The number of cross-validation folds was five regardless the size of the data sets. When carrying out all cross-validation tests, we randomly permuted all the examples and drew each fold through a stratified selection to make sure the hold-out fold and the whole data set shared the same class distribution. The stratified selection meant that examples of each class were grouped together and separated into five shares firstly, and then one share from examples of each class were blended into a fold for cross-validation.

After all the offline experiments, we used the measurements introduced in §5.2 to calculate the measures on each data set and grouped them into tables in §5.4.

Online Settings for Classification

While for online setting, we mainly conducted experiments on algorithms with online capability, which included VM-KNN, CP-BivIR, VA-SVM, VM-SVM-EL, VM-SVM-ES, VM-SVM-OA, VM-SVM-KM and VM-MCSVM. Since CP-BivIR has the different type of validity that could not compare with Venn predictors, we excluded CP-BivIR in our online comparisons. Moreover, VA-SVM was only applicable on binary data sets, hence it was only compared on WBC, SVMguide1 and Splice data sets.

However, we also wanted to know that how simple probabilistic predictors would perform with online settings. Hence, we add NaiveBayes, LogReg and SigSVM into comparisons on 4 data sets, which were WBC, Segment, Wine and Vehicle data sets. Due to the absence of the property of validity for simple probabilistic predictors, we only compared the performance measurements in the tables of results and listed the online performance figures in a separated subsection at the end of §5.5.

In order to satisfy our assumption of exchangeability, we also merged the training set and testing set together and randomly permuted the data sets each time before we ran an

online test. So in practice, the order of all the examples in a data set was different across all tests.

After each online test, the measurements mentioned in §5.2 were applied and the results were organized into tables in §5.5. We also gave a figure with several sub-figures, reflecting the online performance for each data set. Each subfigure contained three curves on the results of each candidate algorithm. The three curves were cumulative error curve, cumulative lower error probability curve and cumulative upper error probability curve. For the results of simple probabilistic predictors with online settings, the probabilistic output was the average probability and the average width was the difference between accuracy and average probability.

Offline Settings for Regression

In order to compare with the results of regression conformal prediction with nearest neighbours in [46], we briefly used the same number of folds and number of neighbours in our experiments. Based on their sizes, the Boston Housing, Abalone and Computer Activity data sets were split into 10, 4 and 2 folds respectively. The numbers of neighbours were 4, 16 and 8 respectively. Table 5.4 summarizes the experimental setup for each data set.

Table 5.4: Experimental setups for regression data sets.

	Housing	Abalone	CompActi
# of Folds	10	4	2
k for k-NN	4	16	8

5.3.1 Settings of Venn-ABERS Predictor

As well as the settings for the underlying algorithm, we need two more settings for Venn-ABERS prediction. One is the input parameter: the size of the calibration set, and the other is whether to calibrate the prediction or not.

Choosing the Size of Calibration Set

The original training set was split into proper training set and calibration set. The calibration set will not participate in the training of the classifier in case of over-fitting, which means we will have fewer examples for training as usual. The less training examples could lead to less accuracy while more calibration examples may have a chance in resulting in a better probabilistic outputs. We then conducted experiments on WBC and Splice data set we mentioned before by setting the size of the calibration set from 10% to 50% of the whole training set. The results are shown in Table 5.5.

Table 5.5: Comparison of results on different calibration set sizes

Data Set	Size (%)	Accuracy	Probabilistic Outputs
WBC	10%	97.88%	[75.56%, 100.00%]
	20%	97.53%	[83.41%, 97.21%]
	30%	97.17%	[85.67%, 95.97%]
	40%	97.53%	[87.72%, 96.23%]
	50%	97.17%	[88.00%, 96.53%]
Splice	10%	89.61%	[79.73%, 91.01%]
	20%	89.61%	[80.60%, 88.05%]
	30%	89.15%	[83.40%, 88.32%]
	40%	88.28%	[84.94%, 89.15%]
	50%	87.72%	[84.61%, 88.42%]

This table mainly indicates two points. The first is that the accuracy decreases as the percentage of calibration set increases. From the results of WBC data set, this is not clearly demonstrated for the reason that the size of WBC testing set is small (283 examples) and this data set is well separated judging by the high accuracy. However, the results of Splice data set strongly support our point: almost 2% decrease when 50% data

is used in calibration compared to 10%. The second point is more important and can be obviously observed from the results: as the size of calibration set increases, the bounds become narrower.

Calibrating Prediction

The other setting is whether to calibrate our predictions to the more likely label based on our probabilistic outputs or not. Since the probabilities given by Venn-ABERS predictor are calculated after the scoring function (in our case, it is an SVM classifier) makes predictions. We compare the probabilities we calculated. If probabilities are greater than 0.5 while our prediction is not $+1$, we change our prediction to -1. This is called calibrating our predictions. We also conducted WBC, Splice and scaled SVMguide1 data set we mentioned before by setting the size of the calibration set from 10% to 50% of the whole training set. The results of the original predictions from SVM and the calibrated predictions are listed in Table 5.6.

From Table 5.6 we observed a downward trend from the results. In the results of all data sets, the calibrated accuracies dropped up to 2.8% compared from the original accuracies, showing that the calibration of the predictions has no significant dependency with the better performance. When we checked the calibrated labels, we found that they were mostly located around the boundary between two classes. According to the definition of isotonic calibrators, the probability of the label being "1" depends on the distance between the object and the hyperplane. If an object in the negative half is far away from the hyperplane, the likelihood of this object being positive (i.e. "1") is almost zero, and vice versa. An outlier in the opposite half could lower or raise the average probability of adjacent objects. This gives evidence on that Venn-ABERS predictor is an algorithm that generates probabilities rather than calibrating the prediction based on the highest likelihood.

Having given careful consideration to both accuracy and narrowness of the bounds, we decided to take 30% of the whole data set as the calibration set and not to calibrate the

Table 5.6: Comparison with SVM accuracies and VA-SVM calibrated accuracies

Data Set	Size (%)	Accuracy	Calibrated Accuracy
WBC	10%	97.88%	97.17%
	20%	97.53%	95.05%
	30%	97.17%	95.05%
	40%	97.53%	94.70%
	50%	97.17%	95.41%
Splice	10%	89.61%	89.57%
	20%	89.61%	89.53%
	30%	89.15%	87.68%
	40%	88.28%	87.68%
	50%	87.72%	87.72%
SVMguide1	10%	96.13%	96.10%
(Scaled)	20%	96.00%	95.98%
	30%	95.95%	95.93%
	40%	95.94%	95.60%
	50%	95.78%	95.73%

final prediction after we obtained the likelihood.

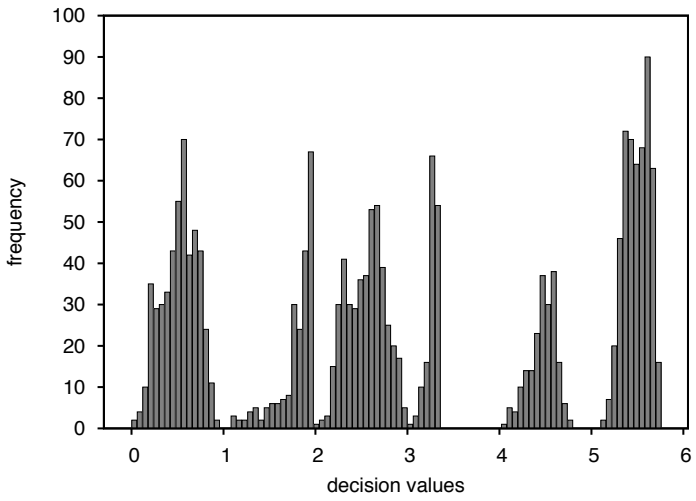
5.3.2 Settings of *k*-Means Clustering

The number of clusters k and the initial means, the two key features of k-means clustering, are often regarded as its biggest drawbacks. The number of clusters k is an input parameter: an inappropriate choice of k may yield poor results. That is why, when performing k-means clustering, it is important to run diagnostic checks for determining the number of clusters in the data set. The choice of initial means might lead the convergence to a local

minimum, which may produce counter-intuitive results. A good design of the combined decision function could make it easier to avoid these two drawbacks.

To have a more intuitive view of our combined decision function described in Eq.4.8, we applied the algorithm to Satimage data set and plotted the histogram in Fig. 5.1, roughly representing the distribution of the decision values.

Figure 5.1: Histogram of decision values on satimage data set



It can be seen obviously from the figure that there were 6 clusters in the data set, the exact number of the possible labels. Furthermore, each cluster i $(i = 1, 2, \ldots, k)$ is approximately within a range of $(i - 1, i)$, which means we could choose the initial means from each range to avoid the local minimum trap as much as possible and speed up the convergence process. We conducted k-means clustering to these decision values and calculated the 6 centroids: $0.63, 1.91, 2.64, 3.33, 4.57, 5.59$. The result seems to be a reasonable reflection of the histogram.

Then we could come to our decision that we set the number of clusters k the same as the number of possible labels K and we choose the initial means as $0.5, 1.5, \ldots, k - 0.5$ if the possible labels are $0, 1, \ldots, K - 1$.

Additionally, we need to notice that k-means clustering uses Euclidean distance as a

metric and variance as a measure of cluster scatter. It tends to produce equal length halves on both sides of a division point. Since data is split halfway between cluster means, the distances of division points to their adjacent cluster means are the same. This can lead to suboptimal splits as some objects will be attributed to the incorrect cluster, especially for unbalanced data set like Satimage data set. For example, in Fig. 5.1, the centroids of first and second clusters are 0.63 and 1.91 respectively. Hence, the division point of these two clusters 1.27. All examples falls on the left side of 1.27 would be categorized to the first cluster, however, they have a higher chance to be in cluster 2 obviously.

5.4 Classification Results with Offline Settings

In this section, we organized all the 5-fold cross-validation results into different tables by the data sets: the results of all algorithms on the same data set, including our algorithms and the benchmarking algorithms, were put together in one table. We listed five measures of the performance, each took a column in the table. Among them were accuracies, probabilistic outputs, the average width of the probabilistic bounds (if the predictor gives bounds as probabilistic estimations), the Brier score and the logarithmic loss. To make the outstanding results clear, we emphasized the best outcome among all candidate algorithms of each measure by representing it in bold type, while we also underlined the best result of each measure among all Conformal and Venn predictors for convenience.

WBC Data Set

In Table 5.7, the results of 5-fold cross-validation on breast cancer data set are listed.

Among all the algorithms, VM-SVM-KM achieved the best accuracy 97.36%. However, the improvement was not significant, ranging from 0.14% to 1.75%, compared with other algorithms. If we only compared this result with other SVM-related algorithms (i.e. the underlying algorithm is SVM or the algorithm itself is based on SVM), the improvement was even smaller, only 0.14% to 1.02%.

For the probabilistic outputs results, it indicated that almost all multi-probabilistic predictors gave probabilistic bounds that could hedge the accuracy well except CP-BivIR whose accuracy exceeded its probabilistic estimates by about 2%. For those single-probability predictors, we also found that the average predicted probabilities were not so precise as estimations for accuracies. The differences were from 0.86% to 3.40%.

The outcomes for the average width of the probabilistic bounds showed that CP-BivIR yielded the narrowest bounds, which was only 0.03 (%) while the loosest bounds were attained by VA-SVM, which was 6.90%. The results of other algorithms were of the same level, ranging from 0.36 (%) to 1.67 (%).

For the Brier scores, the results laid between 0.043 and 0.084, which was very close. The best score was from Platt scaling (SigSVM) while the worst one was from VM-KNN. The Brier scores of our algorithms were within an even smaller interval, which was 0.050 to 0.059.

While for logarithmic loss results, the best score 0.035 was also from Platt scaling, and we noticed that Logistic Regression (LogReg) also acquired very good log loss, which is 0.040. These two algorithms were incomparable in log loss compared with other algorithms. The log loss scores of our methods were around 0.100 except VA-SVM, of which the score was 0.992, about 10 times to results of other Conformal or Venn predictors.

Table 5.7: 5-fold cross-validation results on breast cancer data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	96.34%	99.74%		0.070	0.314
LogReg	96.63%	97.74%		0.051	0.040
SigSVM	97.22%	96.36%		**0.043**	**0.035**
VM-KNN	95.61%	[95.33%, 95.71%]	0.38	0.084	0.182
CP-BivIR	97.17%	[94.98%, 95.01%]	**0.03**	0.059	0.113
VA-SVM	96.78%	[91.27%, 98.17%]	6.90	0.052	0.992
VM-SVM-EL	97.22%	[96.96%, 97.72%]	0.76	0.050	0.100
VM-SVM-ES	96.34%	[95.77%, 97.22%]	1.45	0.057	0.099
VM-SVM-OA	97.22%	[97.19%, 97.56%]	0.37	0.053	0.118
VM-SVM-KM	**97.36%**	[97.10%, 97.46%]	0.36	0.056	0.116
VM-MCSVM	97.07%	[95.65%, 97.32%]	1.67	0.055	0.107

SVMguide1 Data Set

In Table 5.8, the results of 5-fold cross-validation on SVMguide1 data set are listed.

Among all the algorithms, SigSVM achieved the best accuracy, which was 97.21% while Naive Bayes yielded the worst accuracy, which was 93.19%. However, the difference was not significant compared with our algorithms, only ranging from 0.20% to 1.58%. Especially for VM-SVM-EL, VM-SVM-ES and VM-SVM-KM, their results were comparable to the best accuracy with a difference of only 0.20%

For the probabilistic outputs results, it indicated that the accuracies of almost all our multi-probability predictors except VA-SVM were slightly outside their probabilistic bounds, however, the offset (i.e. the distance between accuracy and the nearest bound in percentage) was less than 0.5%. The probabilistic bounds of VA-SVM still hedged its accuracy well. For those single probabilistic predictors, we also found that there was a difference between the estimates and the accuracies. The differences for SigSVM and for LogReg were about 0.5% and 0.12% while the offset of Naive Bayes was larger, about 3.35%.

The outcomes for the average width of the probabilistic bounds showed that VM-KNN yielded the narrowest bounds, which was only 0.04 (%) while the loosest bounds were attained by VA-SVM, which was 1.41%. The results of other algorithms were of the same level, ranging from 0.05 (%) to 0.29 (%).

For the Brier scores, the results laid between 0.046 and 0.103, which was very close. The best score was from Platt scaling (SigSVM) while the worst one was from Naive Bayes. The Brier scores of our algorithms were within an even smaller interval, which was 0.049 to 0.057, except CP-BivIR, which had a Brier score of 0.084.

While for logarithmic loss results, the best score 0.012 was from VM-SVM-KM, which was much smaller compared with others (ranging from 0.042 to 0.181). Except the best score achieved by VM-SVM-KM, the log loss scores of our other methods were around 0.100, which means the probabilistic estimates of the accuracies were precise.

Table 5.8: 5-fold cross-validation results on SVMguide1 data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	93.19%	96.54%		0.103	0.112
LogReg	95.37%	95.25%		0.071	0.054
SigSVM	**97.21%**	96.71%		**0.046**	0.042
VM-KNN	95.58%	[95.48%, 95.52%]	**0.04**	0.084	0.181
CP-BivIR	95.63%	[94.95%, 95.01%]	0.06	0.084	0.078
VA-SVM	96.95%	[95.76%, 97.17%]	1.41	0.049	0.092
VM-SVM-EL	97.01%	[97.49%, 97.56%]	0.07	0.049	0.101
VM-SVM-ES	97.00%	[97.34%, 97.63%]	0.29	0.051	0.097
VM-SVM-OA	95.99%	[96.01%, 96.06%]	0.05	0.076	0.162
VM-SVM-KM	97.01%	[97.48%, 97.55%]	0.07	0.057	**0.012**
VM-MCSVM	96.77%	[96.87%, 97.01%]	0.14	0.053	0.098

Splice Data Set

In Table 5.9, the results of 5-fold cross-validation on splice data set are listed.

Among all the algorithms, VM-SVM-KM and VM-SVM-EL both achieved the best accuracy, which was 91.69%. In addition, the improvement was significant if compared with the worst result. The results revealed that the algorithms can be divided into three according to their accuracies. The worst part included VM-KNN and CP-BivIR, which used k-NN as the underlying algorithm, only yielded an accuracy of about 75% to 77%. The best part that had an accuracy greater than 90% included all SVM-related algorithms except VM-SVM-ES. The rest of the algorithms were in the middle part, which gave an accuracy of about 85%, including Naive Bayes, Logistic Regression and VM-SVM-ES. The reason for these results was that the efficiency of Conformal and Venn predictors depends

on the underlying algorithms.

For the probabilistic outputs results, it indicated that the accuracies of our multi-probability predictors except VA-SVM and VM-MCSVM were slightly outside their probabilistic bounds. The probabilistic bounds of VA-SVM and VM-MCSVM hedged their accuracy well. For those single probabilistic predictors, we also found that there was a difference between the estimates and the accuracies. The differences for SigSVM and for LogReg were about 0.24% and 0.94% while the offset of Naive Bayes was larger, about 4.10%.

The outcomes for average width of the probabilistic bounds showed that VM-KNN and VM-SVM-OA yielded the narrowest bounds, which was only 0.08 (%) while the loosest bounds were attained by VA-SVM, which was 2.91%. The results of other algorithms were of the same level, ranging from 0.12 (%) to 0.62 (%).

For the Brier scores, the best result 0.125 was attained by SigSVM while the worst score 0.362 was attained by VM-KNN. Since Brier scores have a relationship with the accuracies, usually the worse accuracies lead to the worse Brier scores. Hence, the Brier scores of our algorithms except those algorithms using k-NN as the underlying algorithm were within a small interval, which was 0.135 to 0.195.

While for logarithmic loss results, the best score 0.091 was also from Platt scaling (SigSVM), while the best score among our algorithms was 0.229 given by VA-SVM.

USPS Data Set

In Table 5.10, the results of 5-fold cross-validation on USPS data set are listed.

We noticed that Naive Bayes achieved a bad accuracy 78.41% while it still gave an unrealistic probabilistic estimate 99.66%, which lead to the worst Brier score 0.427 and the worst log loss 2.365 among all algorithms. The Naive Bayes algorithm was not suitable for the USPS data set was probably the reason for that. Hence, we did not take Naive Bayes into account when we compared other algorithms.

Among all other the algorithms, SigSVM achieved the best accuracy, which is 98.11%.

Table 5.9: 5-fold cross-validation results on splice data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	87.28%	91.38%		0.189	0.163
LogReg	84.94%	85.88%		0.219	0.153
SigSVM	91.65%	91.41%		**0.125**	**0.091**
VM-KNN	75.27%	[74.98%, 75.06%]	**0.08**	0.362	0.546
CP-BivIR	77.43%	[77.74%, 77.86%]	0.12	0.345	0.430
VA-SVM	91.09%	[89.11%, 92.02%]	2.91	0.135	0.229
VM-SVM-EL	91.69%	[91.91%, 92.09%]	0.18	0.161	0.356
VM-SVM-ES	85.29%	[87.10%, 87.72%]	0.62	0.195	0.301
VM-SVM-OA	90.02%	[94.07%, 94.15%]	**0.08**	0.182	0.335
VM-SVM-KM	**91.69%**	[92.34%, 92.48%]	0.14	0.164	0.424
VM-MCSVM	90.29%	[90.04%, 90.60%]	0.56	0.173	0.385

However, the improvement was not significant. Comparing with the second and third best results, which were achieved by VM-SVM-EL and VM-SVM-KM respectively, the improvement was only 0.07% and 0.13%. The worst result was achieved by VM-SVM-ES, which was 90.41%

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which did not hedge the accuracy well. The accuracies were slightly outside the probability interval. For those single probabilistic predictors, the average predicted probabilities were not so precise as estimates for accuracies. The differences were 1.93% for LogReg and 1.10% for SigSVM.

The outcomes for the average width of the probabilistic bounds showed that CP-BivIR yielded the narrowest bounds 0.00 (%), which meant that the lower bound and the upper

bound were identical. However, these bounds did not hedge the accuracy, it was 2.15% less than the accuracy. The results of other algorithms were of the same level, ranging from 0.08 (%) to 0.40 (%).

Except LogReg, VM-SVM-ES and VM-SVM-OA, of which the Brier scores were 0.103, 0.139 AND 0.096 respectively, the other results laid between 0.030 and 0.058 which was very close. The best score was from Platt scaling (SigSVM) while the best score among our algorithms was from VM-SVM-EL.

While for logarithmic loss results, the best score 0.029 was also from Platt scaling. All other algorithms yielded a log loss between 0.133 and 0.190, which was at the same level, except VM-SVM-ES and VM-SVM-OA gave a log loss of 0.311 and 0.280 respectively.

Table 5.10: 5-fold cross-validation results on USPS data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	78.41%	99.66%		0.427	2.365
LogReg	93.50%	95.43%		0.103	0.133
SigSVM	**98.11%**	97.01%		**0.030**	**0.029**
VM-KNN	97.46%	[97.17%, 97.31%]	0.14	0.050	0.165
CP-BivIR	97.01%	[94.86%, 94.86%]	**0.00**	0.058	0.190
VM-SVM-EL	98.04%	[98.77%, 98.91%]	0.14	0.039	0.170
VM-SVM-ES	90.41%	[92.50%, 92.90%]	0.40	0.139	0.311
VM-SVM-OA	94.98%	[95.52%, 95.65%]	0.13	0.096	0.280
VM-SVM-KM	97.98%	[98.78%, 98.91%]	0.13	0.040	0.176
VM-MCSVM	96.62%	[97.18%, 97.26%]	0.08	0.052	0.169

Satimage Data Set

In Table 5.11, the results of 5-fold cross-validation on satimage data set are listed.

Among all the algorithms, SigSVM achieved the best accuracy 97.36%, which was very close to the second best accuracy 92.37% that obtained by VM-SVM-KM. Other Conformal and Venn predictors had comparable accuracies to the best and second best accuracies except VM-SVM-ES. The results of VM-SVM-ES, Naive Bayes and LogReg were around 80% to 85%, which were worse than other algorithms.

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which did not hedge the accuracy well. The accuracies were slightly outside the probability interval. Moreover, for those single-probability predictors, we also found that the estimate of SigSVM for the accuracy was precise, the difference between accuracy and probability estimate was only 0.13%, while other two algorithms Naive Bayes and LogReg had an offset of 1.59% and 19.02%.

The outcomes for the average width of the probabilistic bounds showed that VM-MCSVM yielded the narrowest bounds, which was only 0.07 (%) while the loosest bounds were attained by VM-SVM-ES, which was 0.35%. The results of other algorithms were of the same level, ranging from 0.11 (%) to 0.32 (%).

For the Brier scores, the results of all algorithms with an accuracy higher than 90% laid between 0.115 and 0.179, which was very close. While other four algorithms, which did not perform well on accuracy, had larger Brier scores, 0.389, 0.199, 0.293 and 0.230 for NaiveBayes, LogReg, VM-SVM-ES and VM-SVM-OA respectively. The best score was from Platt scaling (SigSVM) while the worst one was from NaiveBayes.

While for logarithmic loss results, the best score 0.035 was also from Platt scaling, and we noticed that Logistic Regression (LogReg) also acquired outstanding log loss, which is 0.159 among the other algorithms. For the rest algorithms, the range of their log loss was around 0.400 to 0.600.

Table 5.11: 5-fold cross-validation results on satimage data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	79.77%	98.79%		0.389	1.698
LogReg	85.89%	87.48%		0.199	0.159
SigSVM	**92.46%**	92.59%		**0.115**	**0.094**
VM-KNN	89.99%	[90.09%, 90.21%]	0.13	0.173	0.391
CP-BivIR	90.47%	[90.01%, 90.33%]	0.32	0.170	0.411
VM-SVM-EL	92.09%	[92.79%, 92.90%]	0.11	0.152	0.428
VM-SVM-ES	80.42%	[81.70%, 82.05%]	0.35	0.293	0.585
VM-SVM-OA	85.56%	[85.61%, 85.73%]	0.12	0.230	0.478
VM-SVM-KM	92.37%	[92.67%, 92.78%]	0.11	0.146	0.409
VM-MCSVM	91.53%	[91.51%, 91.58%]	**0.07**	0.179	0.407

Segment Data Set

In Table 5.12, the results of 5-fold cross-validation on segment data set are listed.

We noticed that Naive Bayes achieved a bad accuracy 79.91% while it still gave an unrealistic probabilistic estimate 97.55%, which lead to the worst Brier score 0.373 and the worst log loss 0.762 among all algorithms. The second worse accuracy 90.09% was achieved by VM-SVM-OA. It also had the second worse Brier score 0.161 and log loss 0.330 among all algorithms. The Naive Bayes and VM-SVM-OA were not suitable for Segment data set was probably the reason for that. Hence, we did not take Naive Bayes into account when we compared other algorithms.

Among all the algorithms, SigSVM achieved the best accuracy 97.45%, which was very close to the second best accuracy 97.32% that obtained by VM-SVM-KM. However, the improvement was not significant, ranging from 0.22% to 2.47%, compared with other

algorithms.

For the probabilistic outputs results, it indicated that only the accuracies given by VM-KNN and VM-SVM-ES were slightly outside their bounds, while the offset is less than 0.26%. Moreover, for those single-probability predictors, we also found that the average predicted probabilities were not so precise as estimates for the accuracies. The differences were from 0.82% to 3.87%.

The outcomes for average width of the probabilistic bounds showed that CP-BivIR yielded the narrowest bounds, which was only 0.11 (%) while the loosest bounds were attained by VM-MCSVM, which was 1.15(%). The results of other algorithms were of the same value 0.38(%).

For the Brier scores, the results laid between 0.047 and 0.087, which was very close. The best score was from Platt scaling (SigSVM) while the worst one was from VM-SVM-ES.

While for logarithmic loss results, the best score 0.046 was also from Platt scaling. The log loss scores of our methods were around 0.150 except VM-SVM-ES, of which the score was 0.228, about 50% higher than other results of other Conformal or Venn predictors.

DNA Data Set

In Table 5.13, the results of 5-fold cross-validation on DNA data set are listed.

Among all the algorithms, SigSVM and VM-SVM-KM achieved the best and the second best accuracy, which was 96.01% and 95.82%. The difference between these two results was only 0.19%. The results also revealed that the algorithms that used k-NN as the underlying algorithm did not perform well on this data set, only yielded an accuracy of 81.14% for VM-KNN and 83.68% for CP-BivIR.

For the probabilistic outputs results, it indicated that the accuracies of our multi-probability predictors except CP-BivIR and VM-MCSVM were slightly outside their probabilistic bounds. The probabilistic bounds of CP-BivIR and VM-MCSVM hedged their accuracy well. Moreover, for those single-probability predictors, SigSVM gave a reasonable estimate of which the difference was only 0.61%. However, the differences for NaiveBayes

Table 5.12: 5-fold cross-validation results on segment data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	79.91%	97.55%		0.373	0.762
LogReg	94.98%	95.80%		0.072	0.090
SigSVM	**97.45%**	93.68%		**0.047**	**0.046**
VM-KNN	96.36%	[96.62%, 97.00%]	0.38	0.069	0.177
CP-BivIR	96.60%	[96.55%, 96.66%]	**0.11**	0.071	0.184
VM-SVM-EL	97.23%	[97.03%, 97.41%]	0.38	0.054	0.156
VM-SVM-ES	95.50%	[95.60%, 95.98%]	0.38	0.087	0.228
VM-SVM-OA	90.09%	[89.95%, 90.33%]	0.38	0.161	0.330
VM-SVM-KM	97.32%	[96.98%, 97.36%]	0.38	0.052	0.148
VM-MCSVM	96.96%	[96.61%, 97.76%]	1.15	0.060	0.157

and for LogReg were unrealistic, they both gave their probability estimates more than 99% while their accuracies remained at about 93% and 90%.

The outcomes for average width of the probabilistic bounds showed that VM-KNN and VM-SVM-KM yielded the narrowest bounds, which was only 0.11 (%) while the loosest bounds were attained by CP-BivIR, which was 1.78%.

For the Brier scores, the best result 0.064 was attained by SigSVM while the worst score 0.302 was attained by VM-KNN. If we considered the performance of the underlying algorithms and only took SVM-related algorithms into account, the Brier scores were close, between 0.081 to 0.117.

While for logarithmic loss results, the best score 0.091 was also from Platt scaling (SigSVM), while the best score among our algorithms was 0.205 given by VM-SVM-KM.

Table 5.13: 5-fold cross-validation results on DNA data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	93.53%	99.47%		0.121	0.396
LogReg	90.46%	99.90%		0.190	1.087
SigSVM	**96.01%**	95.40%		**0.064**	**0.057**
VM-KNN	81.14%	[81.27%, 81.38%]	0.11	0.302	0.549
CP-BivIR	83.68%	[82.25%, 84.03%]	1.78	0.282	0.513
VM-SVM-EL	95.70%	[95.92%, 96.04%]	0.12	0.083	0.209
VM-SVM-ES	93.72%	[93.98%, 94.14%]	0.16	0.117	0.263
VM-SVM-OA	94.22%	[95.52%, 95.65%]	0.13	0.107	0.236
VM-SVM-KM	95.82%	[95.94%, 96.05%]	**0.11**	0.081	0.205
VM-MCSVM	95.51%	[95.33%, 96.11%]	0.78	0.094	0.272

Wine Data Set

In Table 5.14, the results of 5-fold cross-validation on wine data set are listed.

Among all the algorithms, VM-SVM-KM achieved the best accuracy, which is 98.88%. The improvement ranged from 0.57% to 2.81%, comparing with other algorithms. Since the accuracies of all algorithms were around 97%, we believed that the improvement was significant.

For the probabilistic outputs results, it indicated that almost all multi-probability predictors except VM-KNN and CP-BivIR gave probabilistic bounds, which can hedge the accuracy well. While for VM-KNN, the accuracy was slightly larger than the upper bound by 0.25% However, for CP-BivIR, the accuracy exceeded its probabilistic estimates by about 4%. Moreover, for those single-probability predictors, we also found that the average predicted probabilities were not so precise as estimates for the accuracies. The

differences were from 0.97% to 4.29%.

The outcomes for the average width of the probabilistic bounds showed that CP-BivIR yielded the narrowest bounds, which was only 0.52 (%) while the loosest bounds were attained by VM-SVM-ES, which was 7.80%. The results of other algorithms were of the same level, ranging from 2.04 (%) to 2.66 (%).

For the Brier scores, the results laid between 0.022 and 0.068, which was very close. The best score was from VM-SVM-KM while the worst one was from VM-SVM-ES.

While for logarithmic loss results, the best score 0.033 was from NaiveBayes, and we noticed that SigSVM also acquired very good log loss, which is 0.036. The log loss scores of other methods ranged from 0.064 to 0.156. Among them, the best log loss score was obtained by VM-SVM-KM.

Table 5.14: 5-fold cross-validation results on wine data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	97.75%	98.72%		0.035	**0.033**
LogReg	96.63%	98.99%		0.051	0.068
SigSVM	97.75%	93.46%		0.033	0.036
VM-KNN	96.63%	[94.34%, 96.38%]	2.04	0.067	0.156
CP-BivIR	97.19%	[92.65%, 93.17%]	**0.52**	0.061	0.150
VM-SVM-EL	98.31%	[97.54%, 99.61%]	2.07	0.034	0.099
VM-SVM-ES	96.07%	[90.15%, 97.95%]	7.80	0.068	0.156
VM-SVM-OA	97.19%	[96.62%, 98.69%]	2.07	0.057	0.156
VM-SVM-KM	**98.88%**	[97.51%, 99.58%]	2.07	**0.022**	0.064
VM-MCSVM	97.75%	[96.83%, 99.49%]	2.66	0.044	0.109

Vehicle Data Set

In Table 5.15, the results of 5-fold cross-validation on vehicle data set are listed.

In order to avoid the poor performance NaiveBayes gave on this data set, we ignored this algorithm in our following comparison. Except NaiveBayes algorithm, VM-SVM-OA performed worst. It only yielded 68.44% on accuracy and 0.457 and 0.809 on Brier score and log loss respectively.

Among all other algorithms, VM-SVM-KM achieved the best accuracy 85.82%. However, the improvement was not significant, ranging from 0.48% to 3.08%, when compared with other SVM-related algorithms. According to the accuracies of VM-KNN and CP-BivIR, this data set was also not suitable for algorithms using k-NN as underlying algorithms.

For the probabilistic outputs results, it indicated that CP-BivIR, VM-SVM-KM and VM-MCSVM gave probabilistic bounds, which can hedge the accuracy well. For other algorithms, the accuracy was slightly outside its probabilistic bounds. For those single probabilistic predictors, we also found that the average predicted probabilities were not so precise as estimates for the accuracies. The differences were around 3%.

The outcomes for the average width of the probabilistic bounds showed that CP-BivIR yielded the narrowest bounds, which was only 0.47 (%) while the loosest bounds were attained by VM-MCSVM, which was 1.35%. The results of other algorithms were of the same level, ranging from 0.59 (%) to 1.17 (%).

For the Brier scores, the results laid between 0.208 and 0.375. The best score was from Platt scaling (SigSVM) while the worst one was from VM-KNN. The best Brier score of our algorithms was achieved by VM-SVM-KM with a value of 0.245.

While for logarithmic loss results, the best score 0.148 was also from Platt scaling, and we noticed that Logistic Regression (LogReg) also acquired good log loss, which is 0.200. These two algorithms were notable in log loss compared with other algorithms. The log loss scores of our methods were around 0.500 while the best of them was 0.482 yielded by

VM-SVM-KM.

Table 5.15: 5-fold cross-validation results on vehicle data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	47.40%	77.68%		0.841	1.148
LogReg	80.85%	84.29%		0.264	0.200
SigSVM	85.34%	82.36%		**0.208**	**0.148**
VM-KNN	71.87%	[70.06%, 70.72%]	0.66	0.375	0.683
CP-BivIR	76.48%	[76.40%, 76.87%]	**0.47**	0.360	0.545
VM-SVM-EL	84.63%	[83.50%, 84.09%]	0.59	0.262	0.494
VM-SVM-ES	82.74%	[82.98%, 84.15%]	1.17	0.297	0.576
VM-SVM-OA	68.44%	[69.67%, 70.26%]	0.59	0.457	0.809
VM-SVM-KM	**85.82%**	[85.51%, 86.10%]	0.59	0.245	0.482
VM-MCSVM	84.99%	[84.29%, 85.64%]	1.35	0.282	0.556

5.5 Classification Results with Online Setting

In this section, we organized all the classification results in the online setting into different tables and figures by the data sets: the results of all algorithms, including our algorithms and the only benchmarking algorithm VM-KNN, on the same data set were put together in one table. We also exclude the results of average width for Conformal Prediction with Bivariate Isotonic Regression, due to the other kind of validity that is not comparable to Venn Machines. We listed five measures of the performance, each took a column in the table. Among them were accuracies, probabilistic outputs, the average width of the probabilistic bounds (if the predictor gives bounds as probabilistic estimations), the Brier score and the logarithmic loss. To make the best results clear, we emphasized the best outcome among all candidate algorithms of each measure by representing it in bold type. We also gave the online performance figures to show the difference of each algorithm intuitively. Each figure consists of a set of sub-figures, and each sub-figure represents the performance of one candidate algorithm. In the sub-figure, there are three lines, one solid line stands for the cumulative errors and two dotted line stands for the cumulative upper and lower bounds. The x-axis is the number of examples while the y-axis is the number of cumulative errors. In most of the figures, we set the y-axis of the sub-figures under the same scaling to compare them, except for some algorithms whose results were exceptionally worse than others.

WBC Data Set

In Table 5.16, the results of classification with online settings on breast cancer data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.2. Notice that, in Fig. 5.2b, the y-axis of VM-KNN was twice the others'.

Among all algorithms, SigSVM achieved a comparable accuracy, which was second highest, and the best average width, Brier score and log loss. NaiveBayes and LogLoss also achieved good results except that the probabilistic output for NaiveBayes was a bit

optimistic, which was 99.72% about 4% higher than its accuracy.

Among all Venn predictors, VM-SVM-OA achieved the best accuracy, which was 96.92%, while VM-KNN obtained the worst accuracy 94.87%. However, the improvement was not significant, ranging from 0.59% to 2.05%, compared with other algorithms.

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well. It is easier to see if the predictions of these predictors are well-hedged from the corresponding figure. In Fig. 5.2b, although the cumulative errors curve of VM-KNN was increased with lots of fluctuations, it was still within the bounds. The similar situation also went for all other algorithms.

The outcomes for average width of the probabilistic bounds showed that VM-SVM-OA yielded the narrowest bounds, which was 1.50 (%) while the loosest bounds were attained by VA-SVM, which was 13.55(%). The results of other algorithms were of the same level, ranging from 2.81 (%) to 4.64 (%).

For the Brier scores, the results laid between 0.057 and 0.092, which was very close. The best score was from VM-MCSVM while the worst one was from VM-KNN.

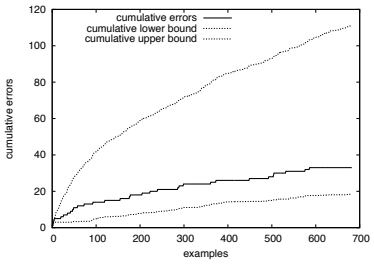
While for logarithmic loss results, the best score 0.107 was also from VM-MCSVM, while the worst score 0.193 was from VM-KNN. However, the difference of these two results was only 0.086.

SVMguide1 Data Set

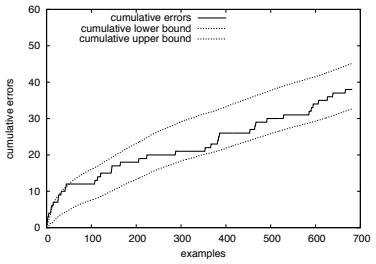
In Table 5.17, the results of classification with online settings on SVMguide1 data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.3.

Among all the algorithms, VM-MCSVM achieved the best accuracy, which was 96.39%, while VM-KNN obtained the worst accuracy 94.82%. However, the improvement was not significant, ranging from 0.07% to 59%, compared with other algorithms except VM-KNN.

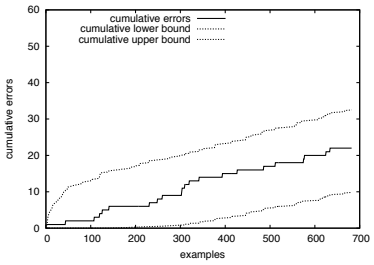
For the probabilistic outputs results, the predictions given by VM-KNN was not in the bounds according to Fig. 5.3b. It fluctuated around the lower bounds. Besides VM-KNN, the cumulative errors curve of VM-SVM-EL (Fig. 5.3c) and VM-SVM-KM (Fig. 5.3f)



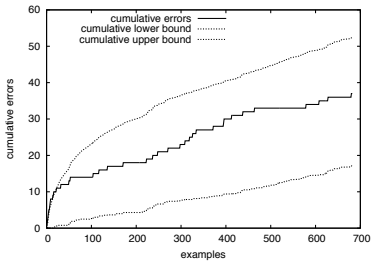
(a) VA-SVM



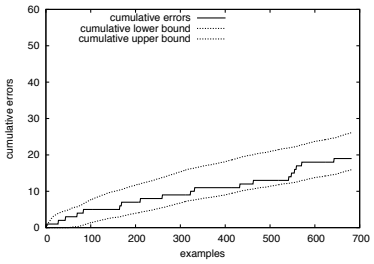
(b) VM-KNN



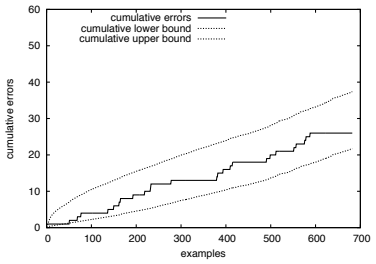
(c) VM-SVM-EL



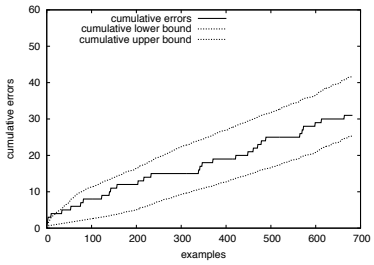
(d) VM-SVM-ES



(e) VM-SVM-OA



(f) VM-SVM-KM



(g) VM-MCSVM

Figure 5.2: Comparison of online performances on breast cancer data set.

Table 5.16: Online mode results on breast cancer data set

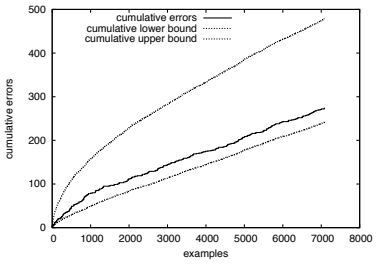
Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	95.71%	99.72%	4.01	0.083	0.375
LogReg	95.17%	97.67%	2.50	0.080	0.182
SigSVM	96.77%	96.21%	0.56	0.051	0.097
VA-SVM	95.43%	[84.48%, 98.03%]	13.55	0.077	0.145
VM-KNN	94.87%	[92.62%, 95.43%]	2.81	0.092	0.193
VM-SVM-EL	96.19%	[95.23%, 98.56%]	3.33	0.060	0.124
VM-SVM-ES	95.01%	[92.34%, 96.98%]	4.64	0.072	0.130
VM-SVM-OA	**96.92%**	[95.77%, 97.27%]	**1.50**	0.059	0.130
VM-SVM-KM	96.77%	[96.52%, 98.82%]	2.30	0.071	0.150
VM-MCSVM	96.33%	[95.53%, 98.52%]	2.99	**0.057**	**0.107**

exceeded the upper bounds at last.

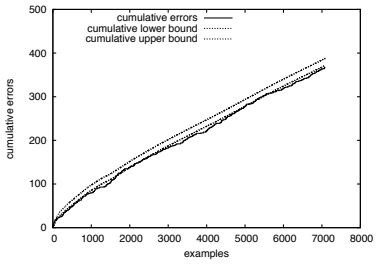
The outcomes for average width of the probabilistic bounds showed that VM-KNN yielded the narrowest bounds, which was 0.23 (%) while the loosest bounds were attained by VA-SVM, which was 3.35(%). The results of other algorithms were of the same level, ranging from 0.36 (%) to 1.35 (%).

For the Brier scores, the results laid between 0.062 and 0.098, which was very close. The best score was from VM-SVM-EL and VM-MCSVM while the worst one was from VM-KNN.

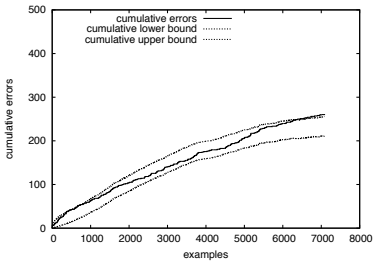
While for logarithmic loss results, the best score 0.121 was also from VM-SVM, while the worst score 0.202 was from VM-KNN. However, the difference between these two results was only 0.081.



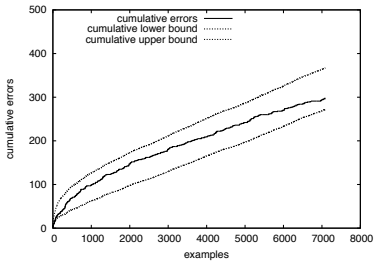
(a) VA-SVM



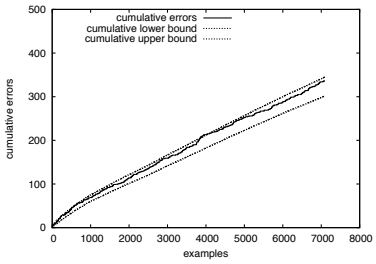
(b) VM-KNN



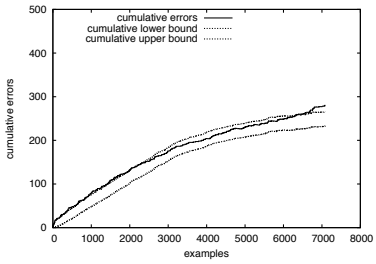
(c) VM-SVM-EL



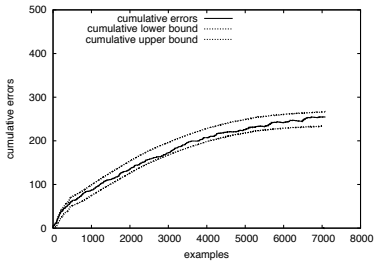
(d) VM-SVM-ES



(e) VM-SVM-OA



(f) VM-SVM-KM



(g) VM-MCSVM

Figure 5.3: Comparison of online performances on SVMguide1 data set.

Table 5.17: Online mode results on SVMguide1 data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
VA-SVM	96.18%	[93.28%, 96.63%]	3.35	0.064	**0.121**
VM-KNN	94.82%	[94.53%, 94.76%]	**0.23**	0.098	0.202
VM-SVM-EL	96.32%	[96.54%, 97.00%]	0.46	**0.062**	0.129
VM-SVM-ES	95.80%	[95.22%, 96.57%]	1.35	0.066	0.124
VM-SVM-OA	95.25%	[95.04%, 95.74%]	0.70	0.089	0.180
VM-SVM-KM	96.04%	[96.38%, 96.74%]	0.36	0.072	0.152
VM-MCSVM	**96.39%**	[96.26%, 96.71%]	0.45	0.062	0.128

Splice Data Set

In Table 5.18, the results of classification with online settings on splice data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.4. Notice that, in Fig. 5.4, the y-axis of VM-KNN and VM-SVM-ES were twice the others'.

Among all the algorithms, VM-SVM-KM achieved the best accuracy 89.54%, while VM-KNN obtained the worst accuracy 71.90%. As the results from Table 5.9, VM-KNN was not suitable for this data set, hence we excluded VM-KNN in our following comparisons. In other results, except VM-SVM-ES, which was only 81.76%, the other results remained comparable to the best accuracy ranging from 88.11% to 89.19%.

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well.

The outcomes for average width of the probabilistic bounds showed that VM-SVM-KM yielded the narrowest bounds, which was 0.47 (%) while the loosest bounds were attained by VA-SVM, which was 5.78(%). The results of other algorithms were of the same level, ranging from 0.81 (%) to 1.35 (%).

For the Brier scores, the results laid between 0.176 and 0.280. The best score was from VA-SVM while the worst one was from VM-SVM-OA. The difference between the best and worst Brier score was 0.150, about 85% of the best score. The Brier scores of other algorithms were within a small interval, which was 0.206 to 0.212.

While for logarithmic loss results, the best score 0.286 was also from VA-SVM, while the worst score 0.610 was from VM-SVM-OA.

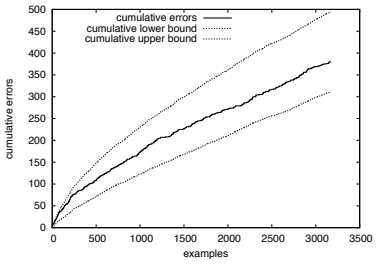
Table 5.18: Online mode results on splice data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
VA-SVM	88.11%	[84.50%, 90.28%]	5.78	**0.176**	**0.286**
VM-KNN	71.90%	[71.59%, 72.09%]	0.50	0.391	0.577
VM-SVM-EL	88.97%	[88.52%, 89.35%]	0.83	0.212	0.482
VM-SVM-ES	81.76%	[81.58%, 82.93%]	1.35	0.280	0.439
VM-SVM-OA	87.62%	[94.27%, 94.88%]	0.61	0.326	0.610
VM-SVM-KM	**89.54%**	[89.11%, 89.58%]	**0.47**	0.206	0.534
VM-MCSVM	89.19%	[88.65%, 89.46%]	0.81	0.209	0.493

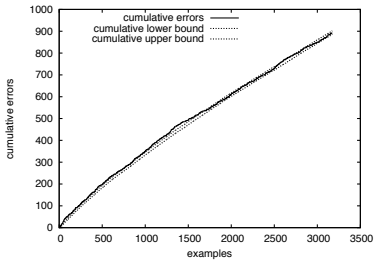
USPS Data Set

In Table 5.19, the results of classification with online settings on USPS data set were listed. The figures on the online performance were displayed in Fig. 5.5. Notice that, in Fig. 5.5, the y-axis of VM-SVM-ES and VM-SVM-OA were 1200 rather than 500, since we wanted to show the trends clearly in other figures.

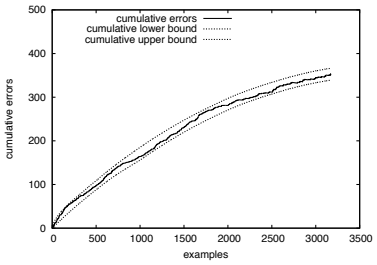
Among all the algorithms, VM-SVM-KM achieved the best accuracy 96.60%, while VM-SVM-ES obtained the worst accuracy 88.73% and VM-SVM-OA obtained the second worst accuracy 93.17%. The difference between the best and the worst accuracy was 7.87%. However, the accuracies of other algorithms were only insignificantly worse than



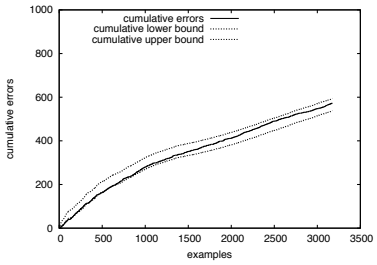
(a) VA-SVM



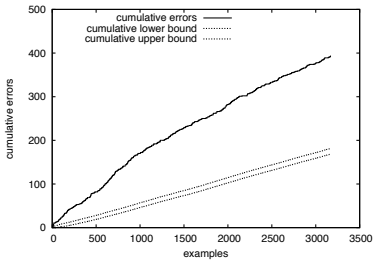
(b) VM-KNN



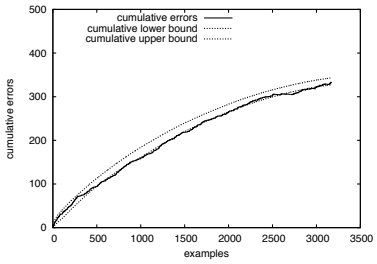
(c) VM-SVM-EL



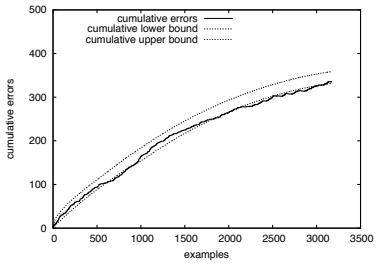
(d) VM-SVM-ES



(e) VM-SVM-OA



(f) VM-SVM-KM



(g) VM-MCSVM

Figure 5.4: Comparison of online performances on splice data set.

the best result, ranging from 95.50% to 95.68%.

For the probabilistic outputs results, all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well, according to Fig. 5.5.

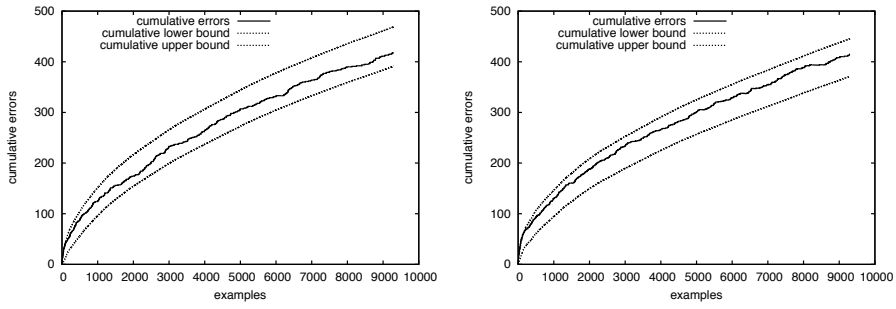
The outcomes for average width of the probabilistic bounds showed that VM-SVM-KM yielded the narrowest bounds, which was 0.67 (%) while the results of other algorithms were of the same level, ranging from 0.77 (%) to 0.87 (%).

For the Brier scores, the results laid between 0.065 and 0.188. The best score was from VM-SVM-KM while the worst one was from VM-SVM-ES. The worst score was three times the best score, the evidence also laid in Fig. 5.5c, the cumulative curve fluctuated from the upper bound to the lower bound. The Brier scores of other algorithms were within a small interval, which was 0.081 to 0.102. Among them the Brier score of VM-SVM-OA was a bit higher, which is 0.102, while others were very similar to each other.

While for logarithmic loss results, the best score 0.244 was also from VM-SVM-KM, while the worst score 0.463 was from VM-SVM-ES. Moreover, the results of other algorithms were all ranging from 0.248 to 0.294.

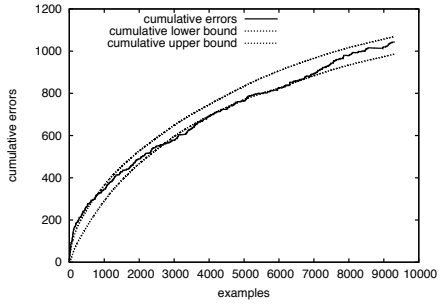
Table 5.19: Online mode results on USPS data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
VM-KNN	95.50%	[94.96%, 95.79%]	0.83	0.085	0.259
VM-SVM-EL	95.54%	[95.21%, 96.01%]	0.80	0.083	0.250
VM-SVM-ES	88.73%	[88.51%, 89.33%]	0.82	0.188	0.463
VM-SVM-OA	93.17%	[92.55%, 93.42%]	0.87	0.102	0.294
VM-SVM-KM	**96.60%**	[96.24%, 96.91%]	**0.67**	**0.065**	**0.244**
VM-MCSVM	95.68%	[95.08%, 95.85%]	0.77	0.081	0.248

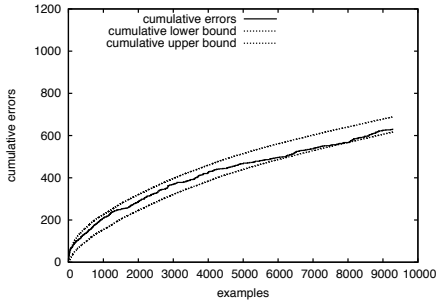


(a) VM-KNN

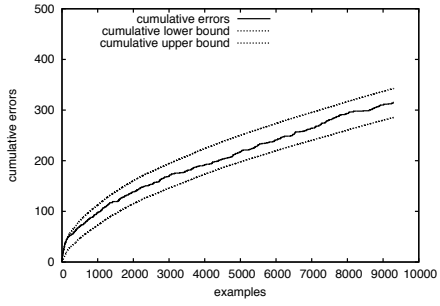
(b) VM-SVM-EL



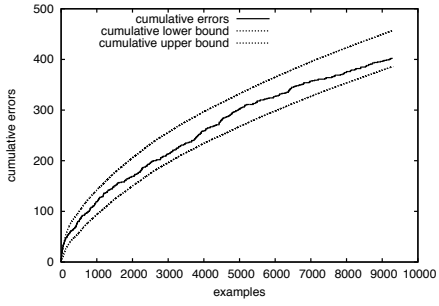
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



(f) VM-MCSVM

Figure 5.5: Comparison of online performances on USPS data set.

Satimage Data Set

In Table 5.20, the results of classification with online settings on satimage data set were listed. The figures on the online performance were displayed in Fig. 5.6. Notice that, in Fig. 5.6, the y-axis of VM-SVM-ES and VM-SVM-OA were twice the others'.

Among all the algorithms, VM-SVM-KM achieved the best accuracy 90.47%, while VM-SVM-ES obtained the worst accuracy 79.43%. The difference between the best and the worst accuracy was 11.04%. However, the accuracies of other algorithms were still comparable with the best result, ranging from 84.21% to 89.91%.

For the probabilistic outputs results, it indicated that all multi-probability predictors except VM-KNN and VM-SVM-ES gave probabilistic bounds, which can hedge the accuracy well. The cumulative errors curve of VM-KNN in Fig. 5.6a was slightly under the lower bound, while the cumulative errors curve of VM-SVM-ES in Fig. 5.6c was beyond the upper bound.

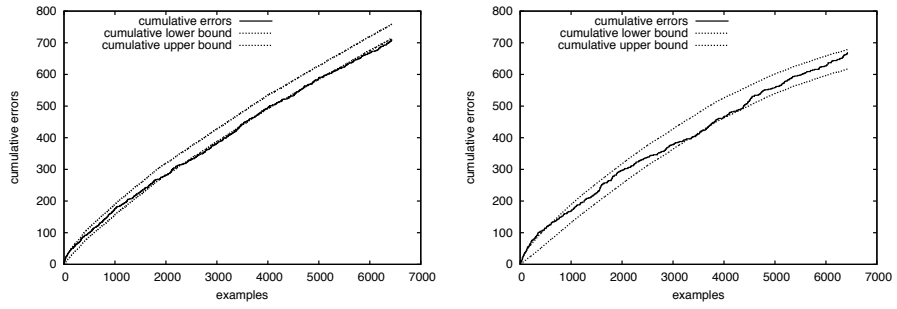
The outcomes for average width of the probabilistic bounds showed that VM-SVM-OA yielded the narrowest bounds, which was 0.058 (%) while the loosest bounds were attained by VM-SVM-ES, which was 2.82(%). The results of other algorithms were of the same level, ranging from 0.69 (%) to 1.03 (%).

For the Brier scores, the results laid between 0.189 and 0.366. The best score was from VM-KNN while the worst one was from VM-SVM-ES. The Brier scores of other algorithms were within a range from 0.219 to 0.238, which were close to each other.

While for logarithmic loss results, the best score 0.423 was also from VM-KNN, while the worst score 0.819 was still from VM-SVM-ES. The worst score was almost twice the best score. Moreover, the results of other algorithms were all around 0.600.

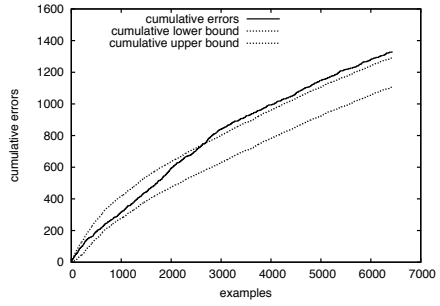
Segment Data Set

In Table 5.21, the results of classification with online settings on segment data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.7. Notice that, in Fig. 5.7, the y-axis of VM-SVM-ES and VM-SVM-OA were 1.6 times the others'.

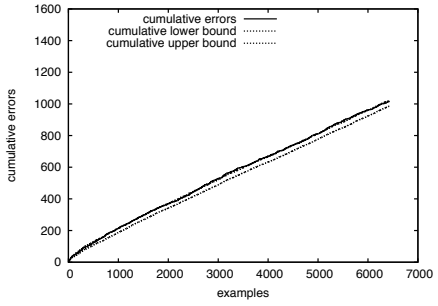


(a) VM-KNN

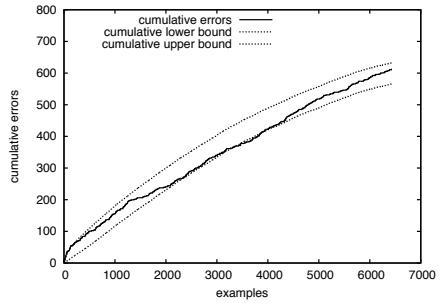
(b) VM-SVM-EL



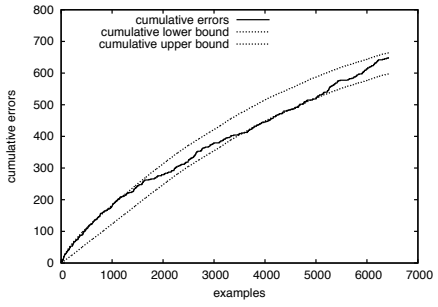
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



(f) VM-MCSVM

Figure 5.6: Comparison of online performances on satimage data set.

Table 5.20: Online mode results on satimage data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
VM-KNN	88.95%	[88.20%, 88.89%]	0.69	**0.189**	**0.423**
VM-SVM-EL	89.57%	[89.41%, 90.44%]	1.03	0.238	0.676
VM-SVM-ES	79.43%	[80.04%, 82.86%]	2.82	0.366	0.819
VM-SVM-OA	84.21%	[84.09%, 84.67%]	**0.058**	0.244	0.507
VM-SVM-KM	**90.47%**	[90.17%, 91.20%]	1.03	0.219	0.614
VM-MCSVM	89.91%	[89.68%, 90.63%]	0.95	0.232	0.656

For this data set, three simple probabilistic predictors did not obtain impressive results. Especially, NaiveBayes yielded the worst results compared to all other algorithms.

Among all Venn predictors, VM-SVM-EL achieved the best accuracy, which was 95.37%, while VM-SVM-OA obtained the worst accuracy 87.05%. The results of other algorithms ranged from 89.56% to 95.24%, which was comparable to the best accuracy.

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well. Fig. 5.7 also indicates that: all cumulative errors curves are well-grounded except the one of VM-SVM-ES and VM-SVM-OA in Fig. 5.7c, which are beyond the upper bound.

The outcomes for average width of the probabilistic bounds showed that VM-SVM-KM yielded the narrowest bounds, which was 1.64 (%) while the loosest bounds were attained by VM-MCSVM, which was 2.04 (%). The results of other algorithms were of the same level, ranging from 1.67 (%) to 2.01 (%).

For the Brier scores, the results laid between 0.085 and 0.199, which was very close. The best score was from VM-SVM-EL while the worst one was from VM-SVM-ES. The worst Brier scores from VM-SVM-OA and VM-SVM-ES were approximately twice the best score from VM-SVM-EL. The Brier scores of other algorithms were within a small

range between 0.089 to 0.098.

While for logarithmic loss results, the best score 0.231 was from VM-KNN, while the worst scores 0.437 and 0.405 were from VM-SVM-ES and VM-SVM-OA. The results of other algorithms ranged from 0.240 to 0.269, which was also close to the best score.

Table 5.21: Online mode results on segment data set

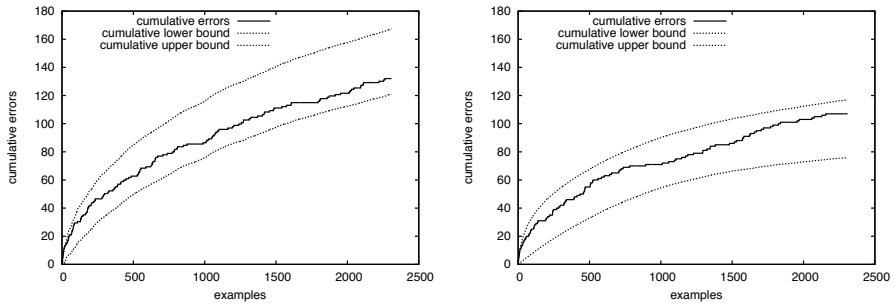
Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	80.89%	97.77%	16.88	0.362	0.889
LogReg	93.16%	97.05%	3.89	0.108	0.333
SigSVM	88.78%	84.68%	4.10	0.156	0.311
VM-KNN	94.33%	[92.71%, 94.72%]	2.01	0.098	**0.231**
VM-SVM-EL	**95.37%**	[94.93%, 96.65%]	1.72	**0.085**	0.240
VM-SVM-ES	89.56%	[89.78%, 91.48%]	1.70	0.182	0.437
VM-SVM-OA	87.05%	[87.29%, 88.96%]	1.67	0.199	0.405
VM-SVM-KM	95.24%	[94.88%, 96.52%]	**1.64**	0.089	0.245
VM-MCSVM	94.93%	[94.44%, 96.48%]	2.04	0.094	0.269

DNA Data Set

In Table 5.22, the results of classification with online settings on DNA data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.8. Notice that, in Fig. 5.8, the y-axis of VM-KNN was 1.4 times the others'.

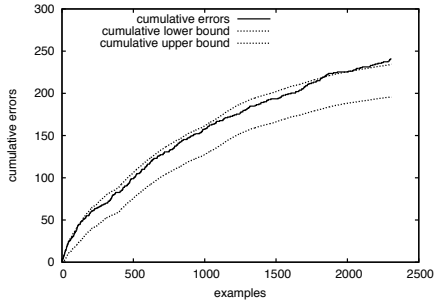
Among all the algorithms, VM-SVM-KM achieved the best accuracy 94.02%, while VM-KNN obtained the worst accuracy 79.25%. Because of the performance of the underlying algorithm k-NN on this data set, we excluded VM-KNN in our following comparisons.

For the probabilistic outputs results, it indicated that all multi-probability predictors except VM-KNN and VM-SVM-OA gave probabilistic bounds that could hedge the accu-

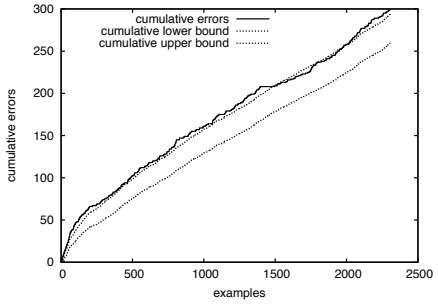


(a) VM-KNN

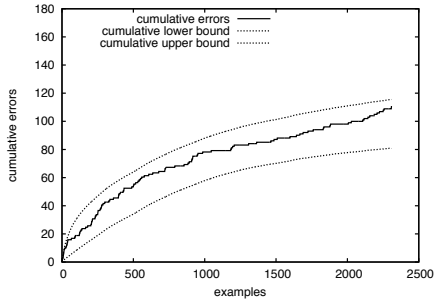
(b) VM-SVM-EL



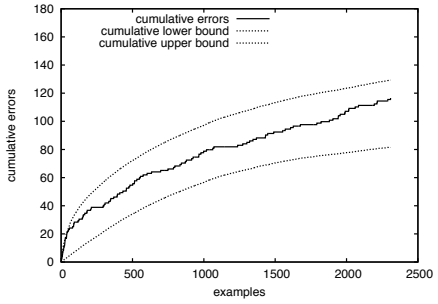
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



(f) VM-MCSVM

Figure 5.7: Comparison of online performances on segment data set.

racy well. However, the probabilistic bounds still gave the right estimates for the trends of cumulative errors curse for VM-KNN in Fig. 5.8a. The cumulative errors curse went far beyond the probabilistic bounds for VM-SVM-OA.

The outcomes for average width of the probabilistic bounds showed that VM-SVM-KM yielded the narrowest bounds, which was 0.47 (%) while the loosest bounds were attained by VM-MCSVM, which was 1.47 (%). The results of other algorithms were of the same level, ranging from 0.55 (%) to 1.38 (%).

While for Brier scores, the best score 0.146 was from VM-SVM-OA, while the worst score 0.215 was from VM-SVM-ES. However, the difference between these two results was only 0.069.

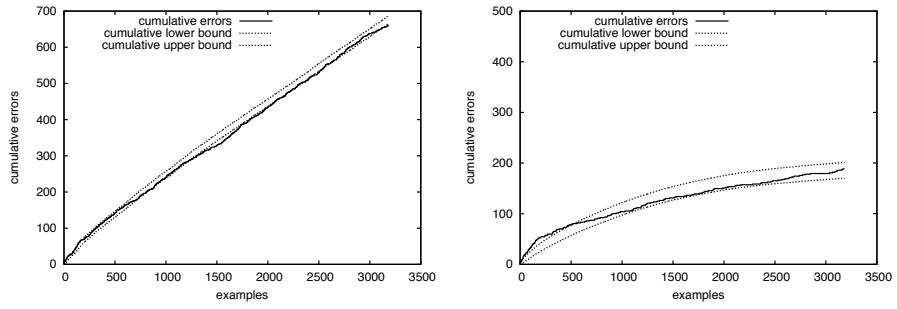
For the log loss results, the results laid between 0.310 and 0.457, which was very close. The best score was from VM-SVM-OA while the worst one was from VM-SVM-ES.

Table 5.22: Online mode results on DNA data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
VM-KNN	79.25%	[78.42%, 79.10%]	0.68	0.324	0.581
VM-SVM-EL	93.89%	[93.63%, 94.66%]	1.03	0.172	0.424
VM-SVM-ES	91.07%	[89.60%, 90.98%]	1.38	0.215	0.457
VM-SVM-OA	91.65%	[93.95%, 94.50%]	0.55	**0.146**	**0.310**
VM-SVM-KM	**94.02%**	[93.79%, 94.26%]	**0.47**	0.170	0.425
VM-MCSVM	93.36%	[93.71%, 95.18%]	1.47	0.176	0.444

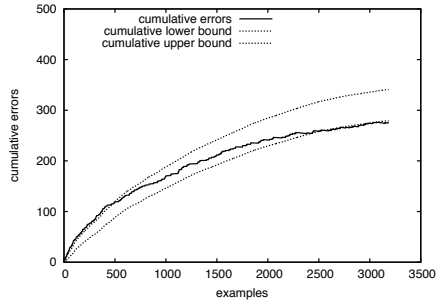
Wine Data Set

In Table 5.23, the results of classification with online settings on wine data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.9. Notice that, in Fig. 5.9, the y-axis of VM-SVM-ES was twice the others'.

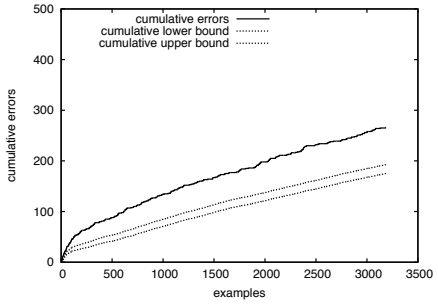


(a) VM-KNN

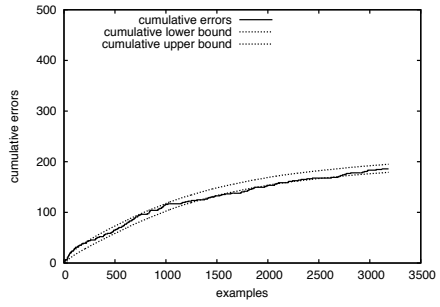
(b) VM-SVM-EL



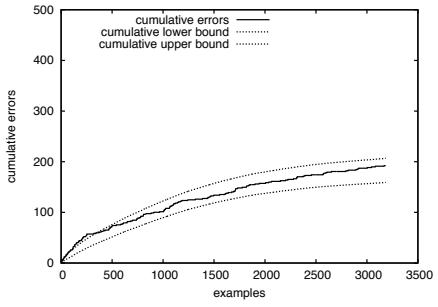
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



(f) VM-MCSVM

Figure 5.8: Comparison of online performances on DNA data set.

Compared to all other algorithms, the results produced by simple probabilistic predictors were within the average range. Among all Venn predictors, VM-SVM-KM achieved the best accuracy 95.48%, while VM-SVM-ES obtained the worst accuracy 88.70%. The other results ranged from 92.66% to 94.92%.

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well.

The outcomes for average width of the probabilistic bounds showed that VM-MCSVM yielded the narrowest bounds, which was 6.07 (%) while the loosest bounds were attained by VA-SVM, which was 16.20(%). The results of other algorithms were of the same level, ranging from 6.23 (%) to 6.45 (%).

For the Brier scores, the results laid between 0.065 and 0.161. The best score was from VM-SVM-KM while the worst one was from VM-SVM-ES.

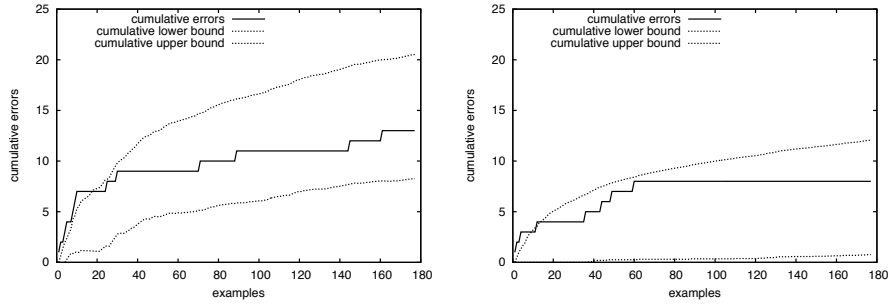
While for logarithmic loss results, the best score 0.134 was also from VM-SVM-KM, while the worst score 0.295 was from VM-SVM-ES. The worst result was more than twice the best score, which also indicated that VM-SVM-ES was not performed well here. The results of other algorithms were of the same level, ranging from 0.205 to 0.237.

Vehicle Data Set

In Table 5.24, the results of classification with online settings on vehicle data set were listed. Moreover, the figures on the online performance were displayed in Fig. 5.10.

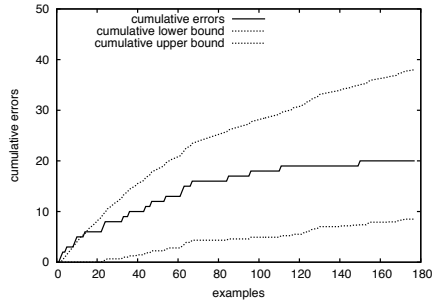
SigSVM and NaiveBayes achieved poor results on this data set, especially for NaiveBayes. The accuracy of LogReg was comparable with other Venn predictors, but its probabilistic output was 12.34% higher than the accuracy, which was too optimistic.

Among all Venn predictors, VM-SVM-KM achieved the best accuracy 79.76%, while VM-KNN and VM-SVM-OA obtained the worst accuracies 66.04% and 60.71% respectively. This is also a data set that not suitable for algorithms using k-NN or One-vs-All SVM as the underlying algorithm. The improvement ranged from 0.71% to 4.38%, comparing with other algorithms.

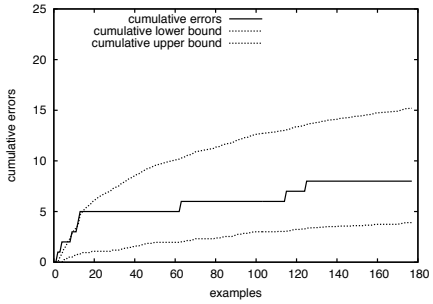


(a) VM-KNN

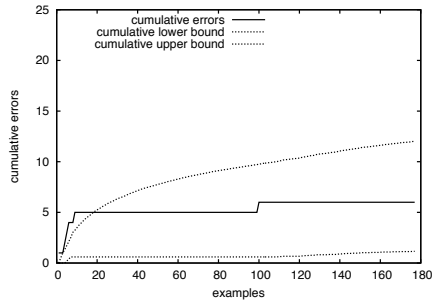
(b) VM-SVM-EL



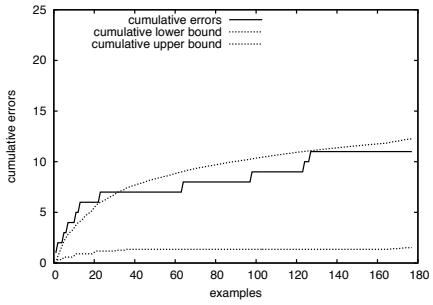
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



(f) VM-MCSVM

Figure 5.9: Comparison of online performances on wine data set.

Table 5.23: Online mode results on wine data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	94.74%	98.27%	3.53	0.085	0.252
LogReg	94.94%	98.77%	3.83	0.073	0.320
SigSVM	92.66%	87.68%	4.98	0.095	0.188
VM-KNN	92.66%	[88.66%, 95.98%]	6.32	0.114	0.237
VM-SVM-EL	94.92%	[93.46%, 99.91%]	6.45	0.090	0.206
VM-SVM-ES	88.70%	[78.22%, 94.42%]	16.20	0.161	0.295
VM-SVM-OA	95.48%	[91.41%, 97.79%]	6.38	0.069	0.135
VM-SVM-KM	**95.48%**	[93.59%, 99.82%]	6.23	**0.065**	**0.134**
VM-MCSVM	93.79%	[93.07%, 99.14%]	**6.07**	0.097	0.205

For the probabilistic outputs results, it indicated that all multi-probability predictors gave probabilistic bounds, which can hedge the accuracy well. From Fig. 5.10c, we also learned that the cumulative errors curve of VM-SVM-ES fluctuated around the upper bound, although the curve was almost always within the bounds, it exceeded the upper bound at last.

The outcomes for average width of the probabilistic bounds showed that VM-SVM-KM yielded the narrowest bounds, which was 2.21 (%) while the loosest bounds were attained by VM-SVM-ES, which was 4.26(%). The results of other algorithms were of the same level, ranging from 2.34 (%) to 3.08 (%).

For the Brier scores, the results laid between 0.354 and 0.380 if we excluded VM-KNN, which is not suitable for this data set. The best score was from VM-SVM-KN while the worst one was from VM-SVM-ES.

While for logarithmic loss results, the best score 0.711 was also from VM-SVM-KM, while the worst score 0.773 was from VM-MCSVM. However, the difference of these two

results was only 0.062.

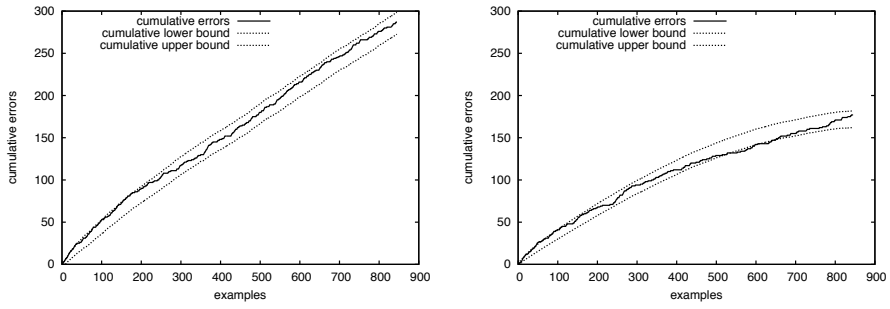
Table 5.24: Online mode results on vehicle data set

Algorithm	Accuracy	Probabilistic Outputs	Avg. Width	Brier Score	Log Loss
NaiveBayes	45.31%	80.55%	35.24	0.864	1.399
LogReg	76.71%	89.05%	12.34	0.351	0.752
SigSVM	61.78%	59.71%	2.07	0.453	0.782
VM-KNN	66.04%	[64.68%, 67.76%]	3.08	0.451	0.835
VM-SVM-EL	79.05%	[78.57%, 80.91%]	2.34	0.366	0.727
VM-SVM-ES	75.38%	[76.14%, 80.40%]	4.26	0.380	0.733
VM-SVM-OA	60.71%	[62.03%, 64.06%]	2.03	0.536	0.965
VM-SVM-KM	**79.76%**	[79.12%, 81.33%]	**2.21**	**0.354**	**0.711**
VM-MCSVM	78.82%	[78.67%, 81.05%]	2.38	0.377	0.773

5.5.1 Online Performances of Simple Probabilistic Predictors

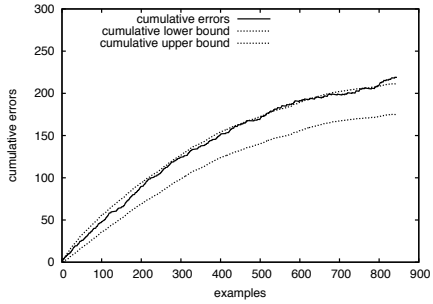
We compared the performance measurements of simple probabilistic predictors on some data sets in our previous subsection. In this subsection, we will show the online performance figures containing cumulative curves of these three simple probabilistic predictors. Since these predictors only gave single probabilistic outputs, there were only two cumulative curves. One of the curves was cumulative errors curve, the other was the cumulative probabilities curve.

Fig. 5.11 to Fig. 5.14 shown the performance curves on WBC, Segment, Wine and Vehicle data sets respectively. From the figures, we learned that for NaiveBayes and LogReg, the probabilistic estimations were very optimistic, which means these two algorithms usually gave higher estimations on accuracies. The differences between estimations and accuracies (aka. average width in previous tables) were exceptionally large compared to

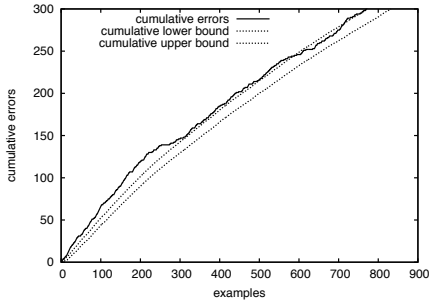


(a) VM-KNN

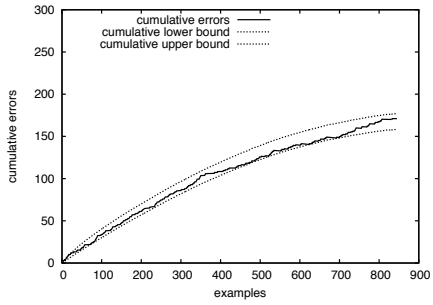
(b) VM-SVM-EL



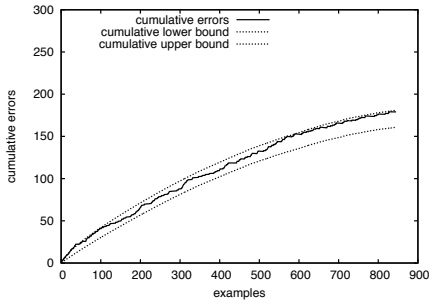
(c) VM-SVM-ES



(d) VM-SVM-OA



(e) VM-SVM-KM



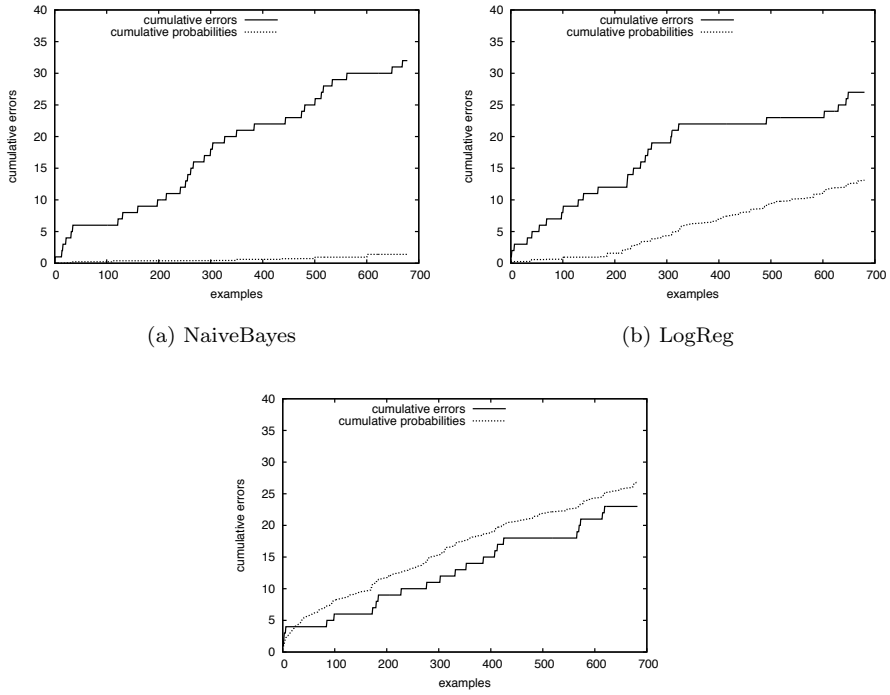
(f) VM-MCSVM

Figure 5.10: Comparison of online performances on vehicle data set.

other algorithms. Both accuracies and probabilistic outputs showed no signs that the results were improved as the training sets getting bigger.

For SigSVM, these figures showed that it produced a pessimistic estimation on the accuracy. In most cases, the cumulative probabilities went over the cumulative errors. The differences were better than the other two simple probabilistic predictors but not comparable with Venn predictors. Moreover, two cumulative errors shown a trend to increase in a slower speed as the training sets getting bigger. It indicated that the model of SigSVM kept improving itself during the online process.

To sum up, these results shown that the property of validity for simple probabilistic predictors is only guaranteed under strong statistical assumptions, which depends on the data set. This is the main drawback compared to Venn predictors, whose validity was guaranteed under an exchangeability assumption. The other drawback is that the probabilistic output was a single probability distribution for each object, while Venn predictors gave multiple estimations.

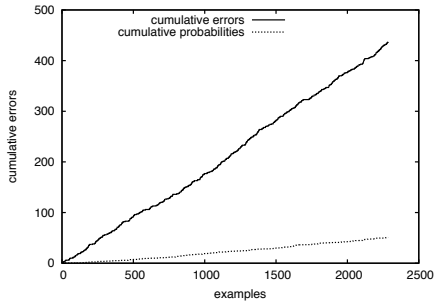


(a) NaiveBayes

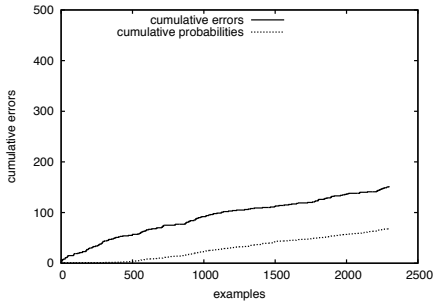
(b) LogReg

(c) SigSVM

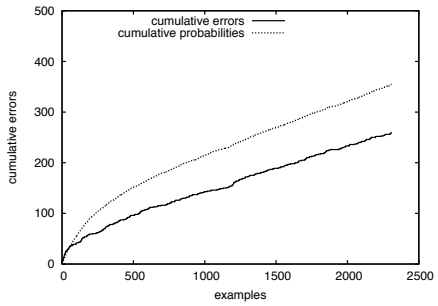
Figure 5.11: Comparison of online performances of simple probabilistic predictors on WBC data set.



(a) NaiveBayes

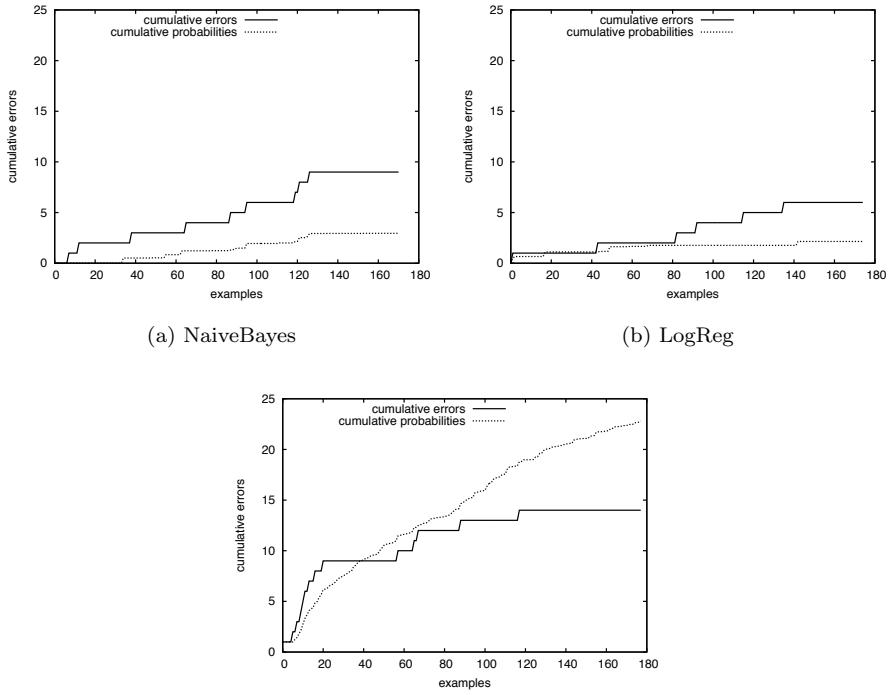


(b) LogReg



(c) SigSVM

Figure 5.12: Comparison of online performances of simple probabilistic predictors on Segment data set.

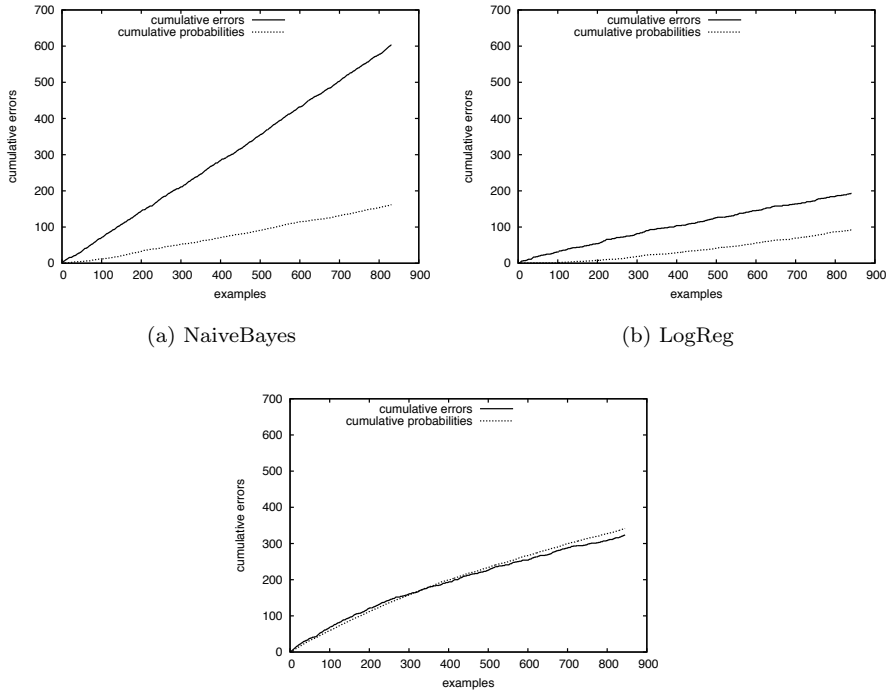


(a) NaiveBayes

(b) LogReg

(c) SigSVM

Figure 5.13: Comparison of online performances of simple probabilistic predictors on Wine data set.



(a) NaiveBayes

(b) LogReg

(c) SigSVM

Figure 5.14: Comparison of online performances of simple probabilistic predictors on Vehicle data set.

5.6 Regression Results with Offline Setting

Since our methods output predictive regions instead of point predictions, the main aim of our experiments was to check the narrowness of these regions. The first, third and the last parts of Table 5.25-5.27 give mean, median and inter-decile mean width of the regions produced for every data set by each setting of the k-NN RVM for the 90%, 95% and 99% confidence levels. In the second part of Table 5.25-5.27 we check the reliability of the obtained predictive regions for each data set. This is done by reporting the percentage of examples for which the true label is not inside the region output by the corresponding method. We also give the comparisons with the results of "TCP (8)" and "ICP (8)" from Papadopoulos's paper [46], in which the term "(8)" means the methods are using Eq. (8) in [46] as the nonconformity measure. The "#C" in these three tables means the number of categories.

The second part checks the validity of the predictive regions empirically. The above tables indicated that the percentages reported here are usually less than the required significance levels for 90% and 95% confidence. However, they are a bit larger than those for 99% confidence.

A transformation of the width values reported in Tables 5.25-5.27 to the percentage of the range of possible labels they represent shows that in general the predictive regions produced by all our methods are relatively tight. The mean width percentages of all data sets are between 25.0% and 76.6% for the 99% confidence level, between 19.1% and 58.2% for the 95% confidence level and between 16.0% and 48.9% for the 90% confidence level.

By comparing these results to [46], we can see that the widths of our method are regularly larger (15%-70%) than the results in [46] for the 90% and 95% confidence levels, but a bit smaller or almost same (10%-40%) for the 99% confidence level.

When the number of categories increases, Table 5.26 and 5.27 shows that the widths and the error rates both get improved. Especially in Table 5.27, the mean width shortened by 21% and the median width shortened by 30% while the number of categories increased

Table 5.25: Comparison of narrowness and reliability results on the Boston Housing data set.

Algorithm	Avg. Width			% Outside Predictions		
	90%	95%	99%	90%	95%	99%
#C=5	20.78	24.75	32.53	4.55%	2.17%	1.58%
#C=10	22.00	26.21	34.45	4.74%	3.56%	2.37%
TCP (8)				10.24%	5.06%	0.97%
ICP (8)				9.47%	4.88%	0.79%
Algorithm	Median Width			Inter-decile Avg. Width		
	90%	95%	99%	90%	95%	99%
#C=5	18.65	22.22	29.20	20.31	24.20	31.80
#C=10	20.52	24.45	32.13	21.37	25.46	33.46
TCP (8)	12.14	17.84	33.21	12.05	17.87	33.57
ICP (8)	13.71	19.44	38.81	13.69	19.42	41.58

from 5 to 10. However, the results went to the opposite when dealing with the Boston Housing data set. This may be caused by the small size of the data set. While conducting a 10-fold 10-category cross-validation to Boston Housing data set, the number of examples in the same category as the testing objects is quite small.

Since the k-NNR CPs mentioned in [46] are using k-Nearest Neighbours Regression as the nonconformity measure, it is different from our underlying algorithm, which is k-Nearest Neighbours for classification. For Boston Housing data set, we tried different numbers of neighbours and found the width went small when the neighbours decreased.

Table 5.28 shows the results for $k = 1$ when the number of category is 5, where k stands for the number of neighbours using in k-Nearest Neighbours. By comparing this to the results for $k = 4$, the mean width and median width decreased up to 27% for 90%,

Table 5.26: Comparison of narrowness and reliability results on the Abalone data set.

Algorithm	Avg. Width			% Outside Predictions		
	90%	95%	99%	90%	95%	99%
#C=5	9.12	10.86	14.27	8.28%	5.03%	2.42%
#C=10	8.24	9.82	12.90	7.61%	4.69%	2.39%
TCP (8)				9.94%	4.94%	0.95%
ICP (8)				10.32%	5.09%	1.01%
Algorithm	Median Width			Inter-decile Avg. Width		
	90%	95%	99%	90%	95%	99%
#C=5	9.21	10.98	14.43	9.33	11.12	14.61
#C=10	7.28	8.67	11.40	8.40	10.01	13.16
TCP (8)	6.69	9.27	16.11	6.71	9.26	16.12
ICP (8)	6.71	9.49	16.63	6.67	9.39	16.58

95% and 99%.

In Table 5.29 we report the processing times of our method. This table indicates the huge computational efficiency of our method could achieve.

5.7 Summary

In this chapter, we gave the details of all our experiments conduct on different data sets. These details included the brief description of each data set, the dimensional characteristics of all data sets, the measures we used to evaluate algorithms, the settings of all related parameters and the detailed results of all experiments grouped in tables and figures.

From our results, we gave our findings as follows:

1. Although Venn-ABERS predictor had the loosest probabilistic intervals, the pre-

Table 5.27: Comparison of narrowness and reliability results on the Computer Activity data set.

Algorithm	Avg. Width			% Outside Predictions		
	90%	95%	99%	90%	95%	99%
#C=5	19.94	23.75	31.22	6.29%	3.52%	1.10%
#C=10	15.83	18.86	24.79	4.82%	2.93%	0.89%
TCP (8)				9.98%	4.99%	0.97%
ICP (8)				9.71%	4.79%	0.95%
Algorithm	Median Width			Inter-decile Avg. Width		
	90%	95%	99%	90%	95%	99%
#C=5	19.53	23.26	30.58	22.57	26.90	34.34
#C=10	13.67	16.29	21.41	17.26	20.57	27.03
TCP (8)	10.01	13.16	21.73	10.01	13.11	21.68
ICP (8)	10.15	13.59	22.71	10.25	13.47	22.58

dicted labels it gave were hedged well within its probabilistic intervals. The Venn-ABERS predictor based on SVM gave a comparable accuracy. However, the absence of a capability to deal with multi-class problems puts a limit on its potential.

2. Conformal Prediction with bivariate isotonic regression and Venn predictor with k-Nearest Neighbours did not perform well on some data sets compared with other Venn predictors using SVM as the underlying algorithm. The main reason was these two algorithms used k-nearest neighbours rather than SVM. As we mentioned before, the efficiency of Conformal predictors and Venn predictors depends on their underlying algorithms. k-Nearest Neighbours can not perform well on some specific data sets compared with SVM, especially when the boundaries between classes are

Table 5.28: Comparison of narrowness and reliability results on the Boston Housing data set when the number of categories is 5.

Algorithm	Avg. Width			% Outside Predictions		
	90%	95%	99%	90%	95%	99%
$k = 1$	17.78	21.18	27.84	7.11%	3.56%	1.19%
$k = 4$	20.78	24.75	32.53	4.55%	2.17%	1.58%
TCP (8)				10.24%	5.06%	0.97%
ICP (8)				9.47%	4.88%	0.79%
Algorithm	Median Width			Inter-decile Avg. Width		
	90%	95%	99%	90%	95%	99%
$k = 1$	13.56	16.16	21.24	20.32	24.21	31.81
$k = 4$	18.65	22.22	29.20	20.31	24.20	31.80
TCP (8)	12.14	17.84	33.21	12.05	17.87	33.57
ICP (8)	13.71	19.44	38.81	13.69	19.42	41.58

not clear. Although the results were not as good as others, these two methods still showed their capability on predicting narrow and valid probabilistic intervals.

3. Among the remaining four SVM related Venn predictors, the SVM Venn predictor with equal size gave the worst performance. The best results were achieved by the SVM Venn predictor with k-means clustering. The improvement was not significant, but it is enough to distinguish this algorithm from the others. Another advantage of using this taxonomy is that the number of categories and the number of clusters among the objects are the same and they will be set to the number of possible labels which is given together with the classification problem. It means we are not necessary to find an optimal value for another parameter. However, the cumulative

Table 5.29: Processing time of k-NN RVM.

Algorithms	Housing	Abalone	CompActi
RVM	1.0 sec	40 sec	1.5 min
TCP	20 sec	52 min	2.6 hrs
ICP	0.6 sec	8 sec	17 sec

error curves in online setting slightly outranged or fluctuated around one boundary for some data sets. The reason for this may be that the introducing of combined decision function resulted in a bit over-fitting. The results of Venn predictor with One-vs-All SVM was also not very good, and this algorithm was not suitable for some of the data sets. However, it still showed the potential on achieving narrow bounds, which leads to an informative probabilist predictions.

4. We have also studied possible connections of probabilistic and reliable algorithms with the problem of regression. We compared approaches for regression based on Conformal Prediction and Venn Machines. Our Regression Venn predictor with k-nearest neighbours was exceeded in prediction interval width by other regression algorithms like transductive conformal predictor and inductive conformal predictor. As the results revealed, this regression Venn predictor violated the validity. However, it was less time consumable according to the comparison between transductive conformal predictor and inductive conformal predictor. Currently, the algorithms of Regression Conformal Prediction gives better performance in all the experiments, so the right design for Regression Venn Machines is a challenge for the future work.

5. If we look into the Appendix A, we will find that WBC and DNA data sets are imbalanced. However, most algorithms performed well on these data sets both offline and online mode. Only k-Nearest Neighbours based algorithms were not comparable to other algorithms.

Chapter 6

Conclusions and Future Works

This chapter sums up the previous chapters, draws the conclusions of this thesis and provides some possible directions for future work.

6.1 Conclusions

This thesis was devoted to algorithm designs based on Conformal prediction and Venn prediction. Conformal and Venn predictors allow us to measure the reliability of the single prediction given by the underlying algorithms. Moreover, these estimations have a guarantee on the overall outcome. The majority of the thesis focuses on taxonomy designs for Venn Machines.

The following new developments and results were presented in this thesis.

- A new modification of standard Conformal prediction was proposed: Conformal prediction with bivariate isotonic regression. This method transforms confidence and credibility values that are output by Conformal prediction into multi-probabilities by applying an isotonic regression.

- An implementation of a new type of Venn predictors was given: Venn-ABERS pre-

dictors. This method transforms scores from other classifiers into valid probabilities

- A taxonomy using support vector machines with equal length intervals for Venn Machines was designed. Also, this taxonomy was upgraded to support multi-class problems from originally the binary only version. The method uses combined decision values to represent all decision values derived from decomposed binary classifiers.

- Based on the combined decision values, a new taxonomy using support vector machines with k-means clustering intervals was proposed. This taxonomy divides combined decision values into different categories by implementing k-means clustering instead of separate them into equal-length intervals.

- A new taxonomy was designed, which is using multi-class support vector machines proposed by Crammer and Singer in [13]. This algorithm transforms multi-class support vector machines into a single optimization problem instead of decomposing them into several binary support vector machines like one-vs-one and one-vs-rest support vector machines. Then the prediction scores from the single classifier were used the same way as the combined decision value in the taxonomy.

- Venn prediction for regression was also implemented. However, the results were not comparable with another implementation of regression conformal prediction.

- Experimental testing of these designed taxonomies and algorithms was carried out. A comparison among all these algorithms was shown in Chapter 5.

- In the experiments, SVM-based algorithms showed their flexibility compared with other algorithms such as k-Nearest Neighbours and NaiveBayes. Almost in all experiments, SVM-based algorithms performed better than other algorithms based designs. Among all Venn taxonomy designs, taxonomies only with predictions in their designs had the worst performance. Once their underlying algorithms did not perform well, they could only yield poor results as well. Taxonomies that use intermediate values of their underlying algorithm usually performed adequately.

- The online performances of simple probabilistic predictors were examined as well. The results showed that the property of validity for simple probabilistic predictors is only guaranteed under strong statistical assumptions, which depends on the data set. This is the main drawback compared to Venn predictors. The other drawback is that the probabilistic output was a single probability distribution for each object, while Venn predictors gave multiple estimations.

6.2 Future Works

This research has left some questions openly and raised some new questions. Here we describe some possible directions for further research. Among them are:

- The experimental results has shown that the efficiency of a Conformal predictor or a Venn predictor is strongly correlated with the efficiency of its underlying algorithm. This means that we have two ways to improve the results on some specific data sets: the one is to find some more effective strangeness measures on the current underlying algorithm to build better predictors; the other one is to design Venn taxonomies or non-conformal measures on some other underlying algorithm that has better efficiency on the data sets instead of current algorithm. In the future, we can try to implement more machine learning algorithms such as random forests into Conformal and Venn Machines' frameworks. Although using Lagrange multipliers of support vector machines as taxonomy for Venn Machines has already been proposed in [14], future works are still possible to extend the capability of this taxonomy from binary-class data sets to multi-class data sets.

- Comparing to simple predictors and other probabilistic predictors, Conformal and Venn predictors have many advantages over them: they give more informative predictions, the validity of their predictions is guaranteed, they are flexible frameworks that can use most of machine learning algorithms as underlying algorithms. However, the drawback of Conformal and Venn predictors is their computational inefficiency,

especially when dealing with large data sets. Therefore, to develop some variants of Conformal or Venn predictions that can reduce the inefficiency in terms of computational time while having comparable results is also a potential direction of further research.

- As another sort of Venn predictors, Venn-ABERS prediction is less computational inefficient than Venn machines while it still holds comparable accuracies and validity. Considering the potential of this algorithm, we think the problems related to Venn-ABERS prediction are worth researching in the future. These include extending Venn-ABERS prediction to support multi-class problem, adding more selections on scoring functions.

- By introducing SVM taxonomy with k-means clustering, we see a potential researching direction that implement clustering algorithms into Venn machines.

Appendix A

Details of Data Sets

WBC Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29`).

The distribution of classes is shown in Table A.1.

Table A.1: Distribution of classes in WBC

Class	# of Examples	% of Examples
1	444	65.01%
2	239	34.99%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^9$, $\gamma = 2^{-7}$. The best $rate = 97.22\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^{3}]$.

SVMguide1 Data Set

This data set can be obtained from (`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#svmguide1`).

The distribution of classes is shown in Table A.2.

Table A.2: Distribution of classes in SVMguide1

Class	# of Examples	% of Examples
1	3089	43.57%
2	4000	56.43%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^9$, $\gamma = 2^1$. The best $rate = 97.15\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Splice Data Set

This data set can be obtained from (`http://www.cs.toronto.edu/~delve/data/splice/desc.html`).

The distribution of classes is shown in Table A.3.

Table A.3: Distribution of classes in Splice

Class	# of Examples	% of Examples
1	1527	48.09%
2	1648	51.91%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^1$, $\gamma = 2^{-5}$. The best $rate = 91.62\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

USPS Data Set

This data set can be obtained from (`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#usps`).

The distribution of classes is shown in Table A.4.

Table A.4: Distribution of classes in USPS

Class	# of Examples	% of Examples
1	1553	16.70%
2	1269	13.65%
3	929	9.99%
4	824	8.86%
5	852	9.16%
6	716	7.70%
7	834	8.97%
8	792	8.52%
9	708	7.61%
10	821	8.83%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^1$, $\gamma = 2^{-5}$. The best $rate = 98.04\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Satimage Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29`).

The distribution of classes is shown in Table A.5.

Table A.5: Distribution of classes in Satimage

Class	# of Examples	% of Examples
1	1533	23.82%
2	703	10.92%
3	1358	21.10%
4	626	9.73%
5	707	10.99%
6	1508	23.43%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^1$, $\gamma = 2^1$. The best $rate = 92.29\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Segment Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Image+Segmentation`).

The distribution of classes is shown in Table A.6.

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^7$, $\gamma = 2^{-1}$. The best $rate = 97.32\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

DNA Data Set

This data set can be obtained from (`http://www.sgi.com/tech/mlc/db/DNA.all`).

The distribution of classes is shown in Table A.7.

When conducting SVM on this data set, the best pair of parameters for RBF kernel is

Table A.6: Distribution of classes in Segment

Class	# of Examples	% of Examples
1	330	14.29%
2	330	14.29%
3	330	14.29%
4	330	14.29%
5	330	14.29%
6	330	14.29%
7	330	14.29%

Table A.7: Distribution of classes in DNA

Class	# of Examples	% of Examples
1	767	24.07%
2	765	24.01%
3	1654	51.91%

$C = 2^1$, $\gamma = 2^{-7}$. The best $rate = 95.95\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Wine Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Wine`).

The distribution of classes is shown in Table A.8.

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^1$, $\gamma = 2^{-3}$. The best $rate = 98.31\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Table A.8: Distribution of classes in Wine

Class	# of Examples	% of Examples
1	59	33.15%
2	71	39.89%
3	48	26.97%

Vehicle Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes%29`).

The distribution of classes is shown in Table A.9.

Table A.9: Distribution of classes in Vehicle

Class	# of Examples	% of Examples
1	212	25.06%
2	217	25.65%
3	218	25.77%
4	199	23.52%

When conducting SVM on this data set, the best pair of parameters for RBF kernel is $C = 2^7$, $\gamma = 2^{-3}$. The best $rate = 85.22\%$ is determined by a grid search on a range of $[2^{-5} \dots 2^{15},\ 2^{-15} \dots 2^3]$.

Housing Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Housing`).

The distribution of labels is shown in Table A.10.

Table A.10: Distribution of labels in Housing

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
5.00	17.02	21.20	22.53	25.00	50.00

Abalone Data Set

This data set can be obtained from (`http://archive.ics.uci.edu/ml/datasets/Abalone`).

The distribution of labels is shown in Table A.11.

Table A.11: Distribution of labels in Abalone

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.00	8.00	9.00	9.93	11.00	29.00

CompActi Data Set

This data set can be obtained from (`http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html`).

The distribution of labels is shown in Table A.12.

Table A.12: Distribution of labels in CompActi

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.00	81.00	89.00	83.97	94.00	99.00

Appendix B

Manual for the LibVM

LibVM is a simple, easy-to-use, and efficient software for Venn Machine on classification, which gives label prediction together with its probabilistic estimations. This library solves Venn prediction in both online and offline mode with k-nearest neighbours or support vector machines as the underlying algorithms. LibVM is written by myself for the purpose of research. This chapter explains the usage of LibVM.

Installation and Data Format

On Unix systems, type `make` to build the `vm-offline`, `vm-online` and `vm-cv` programs. Run them without arguments to show the usage of them.

The format of training and testing data file is:

```
<label> <index1>:<value1> <index2>:<value2> ...
...
...
...
```

Each line contains an instance and is ended by a '`\n`' character (Unix line ending).

For classification, `<label>` is an integer indicating the class label (multi-class is supported). For regression, `<label>` is the target value that can be any real number. The pair `<index>:<value>` gives a feature (attribute) value: `<index>` is an integer starting from 1 and `<value>` is the value of the attribute, which could be an integer number or real number. Indices must be in **ASCENDING** order. Labels in the testing file are only used to calculate accuracies and errors. If they are unknown, just fill the first column with any numbers.

A sample classification data set included in this package is `iris_scale` for training and `iris_scale_t` for testing.

Type `vm-offline iris_scale iris_scale_t`, and the program will read the training data and testing data and then output the result into `iris_scale_t_output` file by default. The model file `iris_scale_model` will not be saved by default, however, adding `-s model_file_name` to `[option]` will save the model to `model_file_name`. The output file contains the predicted labels and the lower and upper bounds of probabilities for each predicted label.

"vm-offline" Usage

```
Usage: vm-offline [options] train_file test_file [output_file]
options:
  -t taxonomy_type : set type of taxonomy (default 0)
    0 -- k-nearest neighbours (KNN)
    1 -- support vector machine with equal length (SVM_EL)
    2 -- support vector machine with equal size (SVM_ES)
    3 -- support vector machine with k-means clustering (SVM_KM)
  -k num_neighbours : set number of neighbours in kNN (default 1)
  -c num_categories : set number of categories for Venn predictor
      (default 4)
```

```
-s model_file_name : save model
-l model_file_name : load model
-p : prefix of options to set parameters for SVM
  -ps svm_type : set type of SVM (default 0)
    0 -- C-SVC    (multi-class classification)
    1 -- nu-SVC   (multi-class classification)
  -pt kernel_type : set type of kernel function (default 2)
    0 -- linear: u'*v
    1 -- polynomial: (gamma*u'*v + coef0)^degree
    2 -- radial basis function: exp(-gamma*|u-v|^2)
    3 -- sigmoid: tanh(gamma*u'*v + coef0)
    4 -- precomputed kernel (kernel values in training_set_file
       )
  -pd degree : set degree in kernel function (default 3)
  -pg gamma : set gamma in kernel function (default 1/
     num_features)
  -pr coef0 : set coef0 in kernel function (default 0)
  -pc cost : set the parameter C of C-SVC (default 1)
  -pn nu : set the parameter nu of nu-SVC (default 0.5)
  -pm cachesize : set cache memory size in MB (default 100)
  -pe epsilon : set tolerance of termination criterion (default
      0.001)
  -ph shrinking : whether to use the shrinking heuristics, 0 or
      1 (default 1)
  -pwi weights : set the parameter C of class i to weight*C,
     for C-SVC (default 1)
  -pq : quiet mode (no outputs)
```

`train_file` is the data you want to train with. `test_file` is the data you want to predict. vm-offline will produce outputs in the `output_file` by default. option `-v` randomly splits the data into n parts and calculates cross validation accuracy/mean squared error on them.

"vm-online" Usage

```
Usage: vm-online [options] data_file [output_file]
options:
  -t taxonomy_type : set type of taxonomy (default 0)
    0 -- k-nearest neighbours (KNN)
    1 -- support vector machine with equal length (SVM_EL)
    2 -- support vector machine with equal size (SVM_ES)
    3 -- support vector machine with k-means clustering (SVM_KM)
  -k num_neighbours : set number of neighbours in kNN (default 1)
  -c num_categories : set number of categories for Venn predictor
      (default 4)
  -p : prefix of options to set parameters for SVM
    -ps svm_type : set type of SVM (default 0)
      0 -- C-SVC    (multi-class classification)
      1 -- nu-SVC   (multi-class classification)
    -pt kernel_type : set type of kernel function (default 2)
      0 -- linear: u'*v
      1 -- polynomial: (gamma*u'*v + coef0)^degree
      2 -- radial basis function: exp(-gamma*|u-v|^2)
      3 -- sigmoid: tanh(gamma*u'*v + coef0)
      4 -- precomputed kernel (kernel values in training_set_file
          )
```

```
    -pd degree : set degree in kernel function (default 3)
    -pg gamma : set gamma in kernel function (default 1/
        num_features)
    -pr coef0 : set coef0 in kernel function (default 0)
    -pc cost : set the parameter C of C-SVC (default 1)
    -pn nu : set the parameter nu of nu-SVC (default 0.5)
    -pm cachesize : set cache memory size in MB (default 100)
    -pe epsilon : set tolerance of termination criterion (default
         0.001)
    -ph shrinking : whether to use the shrinking heuristics, 0 or
         1 (default 1)
    -pwi weights : set the parameter C of class i to weight*C,
        for C-SVC (default 1)
    -pq : quiet mode (no outputs)
```

`data_file` is the data you want to run the online prediction on. vm-online will produce outputs in the `output_file` by default.

"vm-cv" Usage

```
Usage: vm-cv [options] data_file [output_file]
options:
  -t taxonomy_type : set type of taxonomy (default 0)
    0 -- k-nearest neighbours (KNN)
    1 -- support vector machine with equal length (SVM_EL)
    2 -- support vector machine with equal size (SVM_ES)
    3 -- support vector machine with k-means clustering (SVM_KM)
  -k num_neighbours : set number of neighbours in kNN (default 1)
```

```
-c num_categories : set number of categories for Venn predictor
    (default 4)
-v num_folds : set number of folders in cross validation (
    default 5)
-p : prefix of options to set parameters for SVM
  -ps svm_type : set type of SVM (default 0)
    0 -- C-SVC    (multi-class classification)
    1 -- nu-SVC   (multi-class classification)
  -pt kernel_type : set type of kernel function (default 2)
    0 -- linear: u'*v
    1 -- polynomial: (gamma*u'*v + coef0)^degree
    2 -- radial basis function: exp(-gamma*|u-v|^2)
    3 -- sigmoid: tanh(gamma*u'*v + coef0)
    4 -- precomputed kernel (kernel values in training_set_file
        )
  -pd degree : set degree in kernel function (default 3)
  -pg gamma : set gamma in kernel function (default 1/
      num_features)
  -pr coef0 : set coef0 in kernel function (default 0)
  -pc cost : set the parameter C of C-SVC (default 1)
  -pn nu : set the parameter nu of nu-SVC (default 0.5)
  -pm cachesize : set cache memory size in MB (default 100)
  -pe epsilon : set tolerance of termination criterion (default
      0.001)
  -ph shrinking : whether to use the shrinking heuristics, 0 or
      1 (default 1)
  -pwi weights : set the parameter C of class i to weight*C,
      for C-SVC (default 1)
```

```
-pq : quiet mode (no outputs)
```

`data_file` is the data you want to run the cross validation on. vm-cv will produce outputs in the `output_file` by default.

Tips on Practical Use

- Scale your data. For example, scale each attribute to $[0, 1]$ or $[-1, +1]$.
- Try different taxonomies. Some data sets will not achieve good results on some data sets.
- Change parameters for better results especially when you are using SVM related taxonomies.

Examples

```
> vm-offline -k 3 train_file test_file output_file
```

Train a Venn predictor with 3-nearest neighbours as its underlying algorithm on the data set reading from `train_file`. Then conduct this classifier to `test_file` and output the results to `output_file`.

```
> vm-offline -t 1 -s model_file train_file test_file
```

Train a Venn predictor using support vector machines with equal length intervals as taxonomy from `train_file`. Then conduct this classifier to `test_file` and output the results to the default output file, also the model will be saved to file `model_file`.

```
> vm-online -t 2 data_file
```

Train an online Venn predictor classifier using support vector machine with equal size intervals as taxonomy from `data_file`. Then output the results to the default output file.

```
> vm-cv -t 3 -v 10 data_file
```

Do a 10-fold cross validation Venn predictor using support vector machine with k-means clustering intervals as taxonomy from `data_file`. Then output the results to the default output file.

Library Usage

All functions and structures are declared in different header files. There are 5 parts in this library, which are utilities, knn, svm, vm and the other driver programs.

`utilities.h` and `utilities.cpp`

The structure Problem for storing the data sets (including the structure Node for storing the attributes pair of index and value) and all the constant variables are declared in `utilities.h`.

In this file, some utilizable function templates or functions are also declared.

```
T FindMostFrequent(T *array, int size)
```

This function is used to find the most frequent category in kNN taxonomy.

```
static inline void clone(T *&dest, S *src, int size)
```

This static function is used to clone an array from `src` to `dest`.

```
void QuickSortIndex(T array[], size_t index[], size_t left,
    size_t right)
```

This function is used to quick-sort an array and preserve the original indices.

```
Problem *ReadProblem(const char *file_name)
```

This function is used to read in a data set from a file named `file_name`.

```
void FreeProblem(struct Problem *problem)
```

This function is used to free a problem stored in the memory.

```
void GroupClasses(const Problem *prob, int *num_classes_ret, int
    **labels_ret, int **start_ret, int **count_ret, int *perm)
```

This function is used in Cross Validation and other predictions using SVM related taxonomies. This function will group the examples with same label together. The last 5 parameters are using to return corresponding values. `num_classes_ret` is used to store the number of classes in the problem. `labels_ret` is an array used to store the actual label in the order of appearance. `start_ret` is an array used to store the starting index of each group of examples. `count_ret` is an array used to store the count number of each group of examples. `perm` is an array used to store the permutation of the permuted index of the problem.

`knn.h` and `knn.cpp`

The structure `KNNParameter` for storing the kNN related parameters and the structure `KNNModel` for storing the kNN related model are declared in `knn.h`.

In this file, some utilizable function templates or functions are also declared.

```
static inline void InsertLabel(T *labels, T label, int
    num_neighbours, int index)
```

This static function will insert `label` into the index-th location of the array labels of which the size is `num_neighbours`.

```
KNNModel *TrainKNN(const struct Problem *prob, const struct
    KNNParameter *param)
```

This function is used to train a kNN model from a problem `prob` and the parameter `param`, it will return a model of the structure `KNNModel`.

```
double PredictKNN(struct Problem *train, struct Node *x, const
    int num_neighbours)
```

This function is used to predict the label for object x using kNN classifier.

```
double CalcDist(const struct Node *x1, const struct Node *x2)
```

This function is used to calculate the distance between two objects x1 and x2, which will be used in kNN.

```
int CompareDist(double *neighbours, double dist, int
    num_neighbours)
```

This function is used to compare a distance `dist` with the nearest neighbours' distances stored in an array neighbours, it will return the position of `dist`, if it is greater than all the distances in neighbours, it gives `num_neighbours`.

```
int SaveKNNModel(std::ofstream &model_file, const struct KNNModel
    *model)
KNNModel *LoadKNNModel(std::ifstream &model_file)
void FreeKNNModel(struct KNNModel *model)
```

These three functions are used to manipulate the kNN model file, including "save to file", "load from file" and "free the model".

```
void FreeKNNParam(struct KNNParameter *param)
void InitKNNParam(struct KNNParameter *param)
const char *CheckKNNParameter(const struct KNNParameter *param)
```

These three functions are used to manipulate the kNN parameter file, including "free the param", "initial the param" and "check the param".

`svm.h` and `svm.cpp`

The structure SVMParameter for storing the SVM related parameters and the structure SVMModel for storing the SVM related model are declared in `svm.h`.

In this file, some utilizable function templates or functions are also declared.

```
SVMModel *TrainSVM(const struct Problem *prob, const struct
    SVMParameter *param)
```

This function is used to train a SVM model from a problem `prob` and the parameter `param`, it will return a model of the structure `SVMModel`.

```
double PredictValues(const struct SVMModel *model, const struct
    Node *x, double* decision_values)
```

This function is used to predict the label for object x using SVM classifier.

```
double PredictSVM(const struct SVMModel *model, const struct Node
    *x)
```

This function is an interface for `PredictValues()` to predict label.

```
double PredictDecisionValues(const struct SVMModel *model, const
    struct Node *x, double **decision_values)
```

This function is an interface for `PredictValues()` that wraps the main prediction function with decision values to get both `label` and `decision_values`.

```
int SaveSVMModel(std::ofstream &model_file, const struct SVMModel
    *model)
SVMModel *LoadSVMModel(std::ifstream &model_file)
void FreeSVMModel(struct SVMModel **model)
```

These three functions are used to manipulate the SVM model file, including "save to file", "load from file" and "free the model".

```
void FreeSVMParam(struct SVMParameter *param)
void InitSVMParam(struct SVMParameter *param)
const char *CheckSVMParameter(const struct SVMParameter *param)
```

These three functions are used to manipulate the SVM parameter file, including "free the param", "initial the param" and "check the param".

```
void SetPrintNull()
void SetPrintCout()
```

These two functions are used to set the output destination. `SetPrintNull()` will print the output to nowhere (except the warning and error messages and the final results). `SetPrintCout()` will print the output to the standard output stream.

`vm.h` and `vm.cpp`

The structure Parameter for storing the Venn Machine related parameters and the structure Model for storing the Venn Machine related model are declared in `vm.h`. You need to `#include ''vm.h''` in your C/C++ source files and link your program with `vm.cpp`. You can see `vm-offline.cpp`, `vm-online.cpp` and `vm-cv.cpp` for examples showing how to use them.

In this file, some utilizable function templates or functions are also declared.

```
Model *TrainVM(const struct Problem *train, const struct
    Parameter *param)
```

This function is used to train a Venn predictor from the problem train and the parameter `param`.

```
double PredictVM(const struct Problem *train, const struct Model
    *model, const struct Node *x, double &lower, double &upper,
    double **avg_prob)
```

This function is used to predict a new object x from the problem train and the model. It will return the predicted label, lower for lower bound of the probability, upper for upper bound and `avg_prob` for calculate performance measures are also returned.

```
void CrossValidation(const struct Problem *prob, const struct
    Parameter *param, double *predict_labels, double *
    lower_bounds, double *upper_bounds, double *brier, double *
    logloss)
```

This function is used to do a cross validation on the problem `prob` and the parameter `param`. The other 5 parameters are used to return the corresponding values.

```
void OnlinePredict(const struct Problem *prob, const struct
    Parameter *param, double *predict_labels, int *indices,
    double *lower_bounds, double *upper_bounds, double *brier,
    double *logloss)
```

This function is used to do a online prediction on the problem `prob` and the parameter `param`. The other 6 parameters are used to return the corresponding values.

```
int SaveModel(const char *model_file_name, const struct Model *
    model)
Model *LoadModel(const char *model_file_name)
void FreeModel(struct Model *model)
```

These three functions are used to manipulate the model file, including "save to file", "load from file" and "free the model".

```
void FreeParam(struct Parameter *param)
const char *CheckParameter(const struct Parameter *param)
```

These two functions are used to manipulate the parameter file, including "free the param" and "check the param".

`vm-offline.cpp`, `vm-online.cpp` and `vm-cv.cpp`

These three files are the driver programs for LibVM. `vm-offline.cpp` is for training and testing data sets in offline setting. `vm-online.cpp` is for doing online prediction on data sets. `vm-cv.cpp` is for doing cross validation on data sets.

The structure of these files are similar. In these programs, the command-line inputs will be parsed, the data sets will be read into the memory, the train and predict process will be called, the performance measure process will be carried out and finally the memories it claimed will be cleaned up. It includes the following functions.

```
void ExitWithHelp()
```

This function is used to print out the usage of the executable file.

```
void ParseCommandLine(int argc, char *argv[], char *file_name)
```

This function is used to parse the options from the command-line input, and return the values like file names to the other parameters which is represented by `file_name`.

Additional Information

This library can be obtained from https://github.com/fated/libvm. For any questions and comments, please email c.zhou@cs.rhul.ac.uk.

Acknowledgements

Special thanks to Chih-Chung Chang and Chih-Jen Lin, which are the authors of LibSVM.

Bibliography

[1] S Aeberhard, D Coomans, and O De Vel, *Comparison of classifiers in high dimensional settings*, Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep (1992), no. 92-02.

[2] A Aizerman, Emmanuel M Braverman, and L I Rozoner, *Theoretical foundations of the potential function method in pattern recognition learning*, Automation and remote control **25** (1964), 821–837.

[3] Miriam Ayer, HD Brunk, and GM Ewing, *An empirical distribution function for sampling with incomplete information*, The annals of mathematical statistics **26** (1955), no. 4, 641–647.

[4] Mohit Bhandari and Anders Joensson, *Clinical research for surgeons*, Thieme, 2011.

[5] Christopher M Bishop, *Pattern recognition and machine learning*, vol. 1, Springer, New York, 2006.

[6] E Boser, N Vapnik, Isabelle M Guyon, T Bell Laboratories, Bernhard E Boser, and Vladimir N Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on computational learning theory, 1992, pp. 144–152.

[7] Lon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Lawrence D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard,

et al., *Comparison of classifier methods: a case study in handwritten digit recognition*, International Conference on Pattern Recognition, 1994, p. 77.

[8] GW Brier, *Verification of forecasts expressed in terms of probability*, Monthly weather review **27** (1950), no. 693, 594–5.

[9] Chih-chung Chang and Chih-jen Lin, *Libsvm: a library for support vector machines*, ACM Transactions on Intelligent Systems and Technology (TIST) **2** (2011), no. 3, 27.

[10] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[11] T Cover and P Hart, *Nearest neighbor pattern classification*, IEEE Transactions on Information Theory **13** (1967), no. 1, 21–27.

[12] K Crammer and Y Singer, *On the learnability and design of output codes for multiclass problems*, Machine Learning (2002), 201–233.

[13] Koby Crammer and Y Singer, *On the algorithmic implementation of multiclass kernel-based vector machines*, The Journal of Machine Learning Research **2** (2002), 265–292.

[14] Dmitry Devetyarov, *Confidence and venn machines and their applications to proteomics*, (2010).

[15] TG Dietterich and G Bakiri, *Solving multiclass learning problems via error-correcting output codes*, arXiv preprint cs/9501101 (1995), 263–286.

[16] Richard L Dykstra and Tim Robertson, *An algorithm for isotonic regression for two or more independent variables*, The Annals of Statistics **10** (1982), no. 3, 708–716.

[17] Ronald A Fisher, *The fiducial argument in statistical inference*, Annals of Eugenics **6** (1935), no. 4, 391–398.

[18] Evelyn Fix and Joseph L Hodges Jr, *Discriminatory analysis-nonparametric discrimination: consistency properties*, Tech. Report May, DTIC Document, 1951.

[19] EW Forgy, *Cluster analysis of multivariate data: efficiency versus interpretability of classifications*, Biometrics **21** (1965), 768–769.

[20] David Freedman, *Statistical models: theory and practice*, Cambridge University Press, 2009.

[21] Jerome Friedman, *Another approach to polychotomous classification*, Tech. report, Technical report, Department of Statistics, Stanford University, 1996.

[22] A Gammerman, V Vovk, and V Vapnik, *Learning by transduction*, Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (1998), 148–155.

[23] Alexander Gammerman and Vladimir Vovk, *Hedging predictions in machine learning: The second computer journal lecture*, The Computer Journal **50** (2006), no. 2, 151–163.

[24] Greg Hamerly and Charles Elkan, *Alternatives to the k-means algorithm that find better clusterings*, Proceedings of the eleventh international conference on Information and knowledge management (New York, NY, USA), CIKM '02, ACM, 2002, pp. 600–607.

[25] Richard W Hamming, *Error detecting and error correcting codes*, Bell System Technical Journal **29** (1950), no. 2, 147–160.

[26] David Harrison and Daniel L Rubinfeld, *Hedonic housing prices and the demand for clean air*, Journal of Environmental Economics and Management **5** (1978), no. 1, 81–102.

[27] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani, *The elements of statistical learning*, vol. 2, Springer, 2009.

[28] David W Hosmer Jr and Stanley Lemeshow, *Applied logistic regression*, John Wiley & Sons, 2004.

[29] CW Hsu, CC Chang, and CJ Lin, *A practical guide to support vector classification*, **1** (2003), no. 1, 1–16.

[30] Jonathan J Hull, *A database for handwritten text recognition research*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **16** (1994), no. 5, 550–554.

[31] Xiaoqian Jiang, Melanie Osl, Jihoon Kim, and Lucila Ohno-Machado, *Smooth isotonic regression: A new method to calibrate predictive models*, AMIA Summits on Translational Science Proceedings **2011** (2011), 16–20.

[32] George H John and Pat Langley, *Estimating continuous distributions in bayesian classifiers*, Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, 1995, pp. 338–345.

[33] William Karush, *Minima of functions of several variables with inequalities as side constraints*, Ph.D. thesis, Masters thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[34] Stefan Knerr, Lon Personnaz, and Grard Dreyfus, *Single-layer learning revisited: a stepwise procedure for building and training a neural network*, Neurocomputing, Springer, 1990, pp. 41–50.

[35] Ulrich H-G Kreel, *Pairwise classification and support vector machines*, Advances in kernel methods, 1999, pp. 255–268.

[36] H W Kuhn and A W Tucker, *Nonlinear programming*, 1951, pp. 481–492.

[37] Antonis Lambrou and Harris Papadopoulos, *Reliable probability estimates based on support vector machines for large multiclass datasets*, Artificial Intelligence (2012), 182–191.

[38] S Lloyd, *Least squares quantization in pcm*, IEEE Transactions on Information Theory **28** (1982), 129–137.

[39] Scott Menard, *Applied logistic regression analysis*, vol. 106, Sage, 2002.

[40] Tom Mitchell, *Machine learning*, McGraw Hill, Burr Ridge, IL, 1997.

[41] Martin Fodslette Mller, *A scaled conjugate gradient algorithm for fast supervised learning*, Neural Networks **6** (1993), no. 4, 525–533.

[42] Alexandru Niculescu-Mizil and Rich Caruana, *Predicting good probabilities with supervised learning*, no. 1999, 2005, pp. 625–632.

[43] Michiel O Noordewier, Geoffrey G Towell, and Jude W Shavlik, *Training knowledge-based neural networks to recognize genes*, Advances in neural information processing systems, 1991, pp. 530–536.

[44] Harris Papadopoulos, *Inductive conformal prediction: Theory and application to neural networks*, Tools in artificial intelligence (2008).

[45] ———, *Reliable probabilistic classification with neural networks*, Neurocomputing **107** (2013), 59–68.

[46] Harris Papadopoulos, V Vovk, and A Gammerman, *Regression conformal prediction with nearest neighbours*, Journal of Artificial Intelligence **40** (2011), 815–840.

[47] J Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*, Advances in kernel methods support vector learning (1998), 1–21.

[48] ———, *Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods*, Advances in large margin classifiers **10** (1999), no. 3, 61–74.

[49] J Platt, N Cristianini, and J Shawe-Taylor, *Large margin dags for multiclass classification*, Advances in Neural Information Processing Systems 12, 1999, pp. 547–553.

[50] David Price, Stefan Knerr, Lon Personnaz, Grard Dreyfus, et al., *Pairwise neural network classifiers with probabilistic outputs*, Neural Information Processing Systems, vol. 7, 1994, pp. 1109–1116.

[51] Ph. Refregier and F. Vallet, *Probabilistic approach for multiclass classification with neural networks*, International Conference on Artificial Networks, 1991, pp. 1003–1007.

[52] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al., *Tackling the poor assumptions of naive bayes text classifiers*, ICML, vol. 3, 2003, pp. 616–623.

[53] Irina Rish, *An empirical study of the naive bayes classifier*, IJCAI 2001 workshop on empirical methods in artificial intelligence, vol. 3, 2001, pp. 41–46.

[54] Stuart Russell, Peter Norvig, and Artificial Intelligence, *A modern approach*, Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs **25** (1995).

[55] C. Saunders, A. Gammerman, and V. Vovk, *Transduction with confidence and credibility*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99) **2** (1999), 722–726.

[56] J Paul Siebert, *Vehicle recognition using rule based methods*, (1987).

[57] PY Simard, YA LeCun, JS Denker, and B Victorri, *Transformation invariance in pattern recognition - tangent distance and tangent propagation*, Neural Networks: Tricks of the Trade **1524** (1998), 239–274.

[58] Phil Simon, *Too big to ignore: The business case for big data*, John Wiley & Sons, 2013.

[59] CJ Stone, *Consistent nonparametric regression*, The annals of statistics **5** (1977), no. 4, 595–620.

[60] The University of Toronto, *Delve datasets*, 1995.

[61] Irvine University of California, *Uc irvine machine learning repository*, 1987.

[62] LG Valiant, *A theory of the learnable*, Communications of the ACM **27** (1984), no. 11, 1134–1142.

[63] V Vapnik, *The nature of statistical learning theory*, Springer Science & Business Media, 2000.

[64] Vladimir Vapnik, *Estimation of dependences based on empirical data [in russian]*, Nauka, Moscow, 1979.

[65] Vladimir Naumovich Vapnik and Vlamimir Vapnik, *Statistical learning theory*, vol. 2, Wiley New York, 1998.

[66] John Venn, *The logic of chance: An essay on the foundations and province of the theory of probability, with especial reference to its application to moral and social science*, Macmillan, 1866.

[67] P F Verhulst, *Mathematical researches into the law of population growth increase*, Nouveaux Mmoires de l'Academie Royale des Sciences et Belles-Lettres de Bruxelles **18** (1845), 1–42.

[68] V Vovk, *Derandomizing stochastic prediction strategies*, Machine Learning **35** (1999), no. 3, 247–282.

[69] V Vovk, A Gammerman, and G Shafer, *Algorithmic learning in a random world*, Springer Science & Business Media, 2005.

[70] Vladimir Vovk, *Venn predictors and isotonic regression*, arXiv preprint arXiv:1211.0025 (2012), 1–4.

[71] Vladimir Vovk, Glenn Shafer, and Ilia Nouretdinov, *Self-calibrating probability forecasting*, Advances in Neural Information Processing Systems, 2003, p. None.

[72] Sam Waugh, *Extending and benchmarking cascade-correlation*, Ph.D. thesis, University of Tasmania, 1995.

[73] J Weston and C Watkins, *Support vector machines for multi-class pattern recognition.*, ESANN (1999), 5–10.

[74] Jason Weston and Chris Watkins, *Multi-class support vector machines*, Tech. report, Citeseer, 1998.

[75] William H WH Wolberg and Olvi L Mangasarian, *Multisurface method of pattern separation for medical diagnosis applied to breast cytology.*, Proceedings of the national academy of sciences **87** (1990), no. 23, 9193–9196.

[76] Philip Wolfe, *A duality theorem for nonlinear programming*, Quarterly of applied mathematics **19** (1961), no. 3, 239–244.

[77] TF Wu, CJ Lin, and RC Weng, *Probability estimates for multi-class classification by pairwise coupling*, The Journal of Machine Learning Research **5** (2004), 975–1005.

[78] Bianca Zadrozny and Charles Elkan, *Transforming classifier scores into accurate multiclass probability estimates*, Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02 (New York, New York, USA), ACM Press, 7 2002, p. 694.

[79] Chenzhe Zhou, Ilia Nouretdinov, Zhiyuan Luo, Dmitry Adamskiy, Luke Randell, Nick Coldham, and Alex Gammerman, *A comparison of venn machine with platt's method in probabilistic outputs*, Artificial Intelligence Applications and Innovations (2011), 483–490.

[80] Chenzhe Zhou, Ilia Nouretdinov, Zhiyuan Luo, and Alex Gammerman, *Svm venn machine with k-means clustering*, Artificial Intelligence Applications and Innovations, 2014.