We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

4,800

Open access books available

122,000

International authors and editors

135M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

CLARIVATE ANALYTICS

BOOK CITATION INDEX

INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Selecting Representative Data Sets

Tomas Borovicka, Marcel Jirina Jr., Pavel Kordik and Marcel Jirina

Additional information is available at the end of the chapter

1. Introduction

A training set is a special set of labeled data providing known information that is used in the supervised learning to build a classification or regression model. We can imagine each training instance as a feature vector together with an appropriate output value (label, class identifier). A supervised learning algorithm deduces a classification or regression function from the given training set. The deduced classification or regression function should predict an appropriate output value for any input vector. The goal of the training phase is to estimate parameters of a model to predict output values with a good predictive performance in real use of the model.

When a model is built we need to evaluate it in order to compare it with another model or parameter settings or in order to estimate predictive performance of the model. Strategies and measures for the model evaluation are described in section 2.

For a reliable future error prediction we need to evaluate our model on a different, independent and identically distributed set that is different to the set that we have used for building the model. In absence of an independent identically distributed dataset we can split the original dataset into more subsets to simulate the effect of having more datasets. Some splitting algorithms proposed in literature are described in section 3.

During a learning process most learning algorithms use all instances from the given training set to estimate parameters of a model, but commonly lot of instances in the training set are useless. These instances can not improve predictive performance of the model or even can degrade it. There are several reasons to ignore these useless instances. The first one is a noise reduction, because many learning algorithms are noise sensitive [31] and we apply these algorithms before learning phase. The second reason is to speed up a model response by reducing computation. It is especially important for instance-based learners such as k-nearest neighbours, which classify instances by finding the most similar instances from a training set and assigning them the dominant class. These types of learners are commonly called lazy learners, memory-based learners or case-based learners [14]. Reduction of training sets can be necessary if the sets are huge. The size and structure of a training set needed to correctly estimate the parameters of a model can differ from problem to problem and a chosen instance

selection method [14]. Moreover, the chosen instance selection method is closely related to the classification and regression method. The process of instance reduction is also called instance selection in the literature. A review of instance selection methods is in section 4.

The most of learning algorithms assumes that the training sets, used to estimate the parameters of a model or to evaluate a model, have proportionally the same representation of classes. But many particular domains have classes represented by a few instances while other classes have a large number of representative instances. Methods that deal with the class imbalance problem are described in section 5.

1.1. Basic notations

In this section we set up basic notation and definitions used in the document.

A population is a set of all existing feature vectors (features). By S we denote a sample set defined as a subset of a population collected during some process in order to obtain instances that can represent the population.

According to the previous definition the term representativeness is closely related. We can define a *representative set S^** as a special subset of an original dataset S, which satisfies three main characteristics [72]:

1. It is significantly smaller in size compared to the original dataset.
2. It captures the most of information from the original dataset compared to any subset of the same size.
3. It has low redundancy among the representatives it contains.

A training set is in the idealized case a representative set of a population. Any of mentioned methods is not needed if we have representative subset of the population. But we never have itin practise. We usually have a random sample set of the population and we use various methods to make it as representative as possible. We will denote a training set by R.

In order to define a representative set we can define a *minimal consistent subset* of a training set. Given a training set R, we want to obtain a subset $R^* \subset R$ such that R^* is the smallest set of instances such that $Acc(R^*) \cong Acc(R)$, where $Acc(X)$ denotes the classification accuracy obtained using X as a training set [71].

Sets used for an evaluation of a model are the validation set V, usually used for a model selection, and the testing set T, used for model assessment.

2. Model evaluation

The model evaluation is an important but often underestimated part of model building and assessment. When we have prepared and preprocessed data we want to build a model with the ability to accurately predict future observations. We do not want a model that perfectly fits training data, but we need a model that is reliable after deployment in the real use. For this purpose we should have two phases of a model evaluation. In the first phase we evaluate a model **in order to estimate the parameters of the model** during the learning phase. This is a part of the model selection when we select the model with the best results. This phase is also

called as the validation phase. It does not necessary mean that we choose a model that best fits a particular set of data. The well learned model captures only the underlying phenomenon, not the noise. A model that captures a noise is called as *over-fitted* [47]. In the second phase we evaluate the selected model **in order to assess the real performance of the model** on new unseen data. Process steps are shown below.

1. Model selection
   (a) Model learning (Training phase)
   (b) Model validation (Validation phase)
2. Model assessment (Testing phase)

2.1. Evaluation methods

During building a model, we need to evaluate its performance in order to validate or assess it as we mention earlier. There are more methods how to check our model, but not all are usually sufficient or applicable in all situations. We should always choose the most appropriate and reliable method for our purpose. Some of common evaluation methods are [83]:

Comparison of the model with physical theory
A comparison of the model with a physical theory is the first and probably the easiest way how to check our model. For example, if our model predicts a negative quantity or parameters outside of a possible range, it points to a poorly estimated model. However, a comparison with a physical theory is not always possible nor sufficient as a quality indicator.

Comparison of model with theoretical or empirical model
Sometimes a theoretical model exists, but may be to complicated for a practical use. In this case, the theoretical model could be used for a comparison or evaluation of the accuracy of the built model.

Collect new data for evaluation
The use of data collected in an independent experiment is the best and the most preferred way for a model evaluation. It is the only way that gives us a real estimate of the model performance on new data. Only new collected data can reveal a bias in a previous sampling process. This is the easiest way if we can easily repeat the experiment and sampling process. Unfortunately, there are situations when we are not capable to collect new independent data for this purpose either due to a high cost of the experiment or another unrepeatability of the process.

Use the same data as for model building
The use the same data for evaluation and for a model building usually leads to an optimistic estimation of real performance due to a positive bias. This is not recommended method and if there is another way it could not be used for the model evaluation at all.

Reserve part of the learning data for evaluation
A reserve part of the learning data is in practise the most common way how to deal with the absence of an independent dataset for model evaluation. As the reserve part selection from the data is usually not a simple task, many methods were invented. Their usage depends on a particular domain. Splitting the data is wished to have the same effect as having two independent datasets. However, this is not true, only newly collected data can point out the bias in the training dataset.

2.1.1. Evaluation measures

For evaluating a classifier or predictor there is a large variation of performance measures. However, a measure, good for evaluating a model in a particular domain, could be inappropriate in another domain and vice versa. The choice of an evaluation measure depends on the domain of use and the given problem. Moreover, different measures are used for classification and regression problems. The measures below are shortly described basics for the model evaluation. For more details see [46, 100].

Measures for classification evaluation

The basis for analysing classifier performance is a **confusion matrix**. The confusion matrix describes how well a classifier can recognize different classes. For c classes, the confusion matrix is an $n \times n$ table, which (i, j)th entry indicates the count of instances of the class i classified as j. It means that correctly classified instances are on the main diagonal of the confusion matrix. The simplest and the most common form of the confusion matrix is a two-classes matrix as it is shown in the table 1. Given two classes, we usually use a special terminology describing members of the confusion matrix. Terms *Positive* and *Negative* refer to the classes. *True Positives* are positive instances that were correctly classified, *True Negatives* are also correctly classified instances but of the negative class. On the contrary, *False Positives* are incorrectly classified positive instances and *False Negatives* are incorrectly classified negative instances.

		Predicted	
		Positive	Negative
True	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Figure 1. Confusion matrix

The first and the most commonly used measure is the **accuracy** denoted as $Acc(X)$. The accuracy of a classifier on a given set is the percentage of correctly classified instances. We can define the accuracy as

$$Acc(X) = \frac{correctly\ classified\ instances}{all\ instances}$$

or in a two-classes case

$$Acc(X) = \frac{TP + TN}{TP + TN + FP + FN}.$$

In order of having defined the accuracy, we can define the **error rate** of a classifier as

$$Err(X) = 1 - Acc(X),$$

which is the percentage of incorrectly classified instances.

If costs of making a wrong classification are known, we can assign different cost or benefit to each correct classification. This simple method is known as **costs and benefits** or **risks and gains**. The *cost matrix* has then the structure shown in Figure 2, where λ_{ij} corresponds to the cost of classifying the instance of class i to class j. Correctly classified instances have usually a zero cost ($\lambda_{ii} = \lambda_{jj} = 0$). Given a cost matrix, we can calculate the cost of a particular

	Predicted		
	Class i	...	Class j
Class i	λ_{ii}	...	λ_{ij}
⋮	⋮	⋱	⋮
Class j	λ_{ji}	...	λ_{jj}

Figure 2. Cost matrix

learned model on a given test set by summing relevant elements of the cost matrix accordingly to the model's prediction [100]. Here, the cost matrix is used as a measure, the costs are ignored during the classification. When a cost matrix is taken into account during learning a classification model, we speak about a cost-sensitive learning, which is mentioned in section 5 in the context of class balancing.

Using the accuracy measure fails in cases, when classes are significantly imbalanced (The class imbalanced problem is discussed in section 5). Good examples could be medical data, where we can have a lot of negative instances (for example 98%) and just a few (2%) of positive instances. It gives an impressive 98% accuracy, when we simply classify all instances as negative, which is absolutely unacceptable for medical purposes. The reason for this is that the contribution of a class to the overall accuracy rate is a function of its cardinality, with the effect that rare positives have an almost insignificant impact on the performance measure [22].

Alternatives for the accuracy measure are:
Sensitivity (also called *True Positive Rate* or *Recall*) - the percentage of truly positive instances that were classified as positive,

$$sensitivity = \frac{TP}{TP + FN} \; .$$

Specificity (also called *True Negative Rate*) - the percentage of truly negative instances that were classified as negative,

$$specificity = \frac{TN}{TN + FP} \; .$$

Precision - the percentage of positively classified instances that are truly positive,

$$precision = \frac{TP}{TP + FP} \; .$$

It can be shown that the accuracy is a function of the sensitivity and specificity:

$$accuracy = sensitivity \cdot \frac{TP + FN}{TP + TN + FP + FN} + specificity \cdot \frac{TN + FP}{TP + TN + FP + FN} \; .$$

F-measure combines precision and recall. It is generally defined as

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{\beta 2 \cdot precision + \cdot recall}$$

where β specifies the relative importance of precision and recall. The F-measure can be interpreted as a weighted average of the precision and recall. A disadvantage of this

measure is that it does not take the true negative rate into account. Another measure, that overcomes disadvantages of the accuracy on imbalanced datasets is the **geometric mean** of class accuracies. For the two-classes case it is defined as

$$gm = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}} = \sqrt{sensitivity \cdot specificity}$$

The geometric mean puts all classes on an equal footing, unfortunately there is no way to overweight any class [22].

The evaluation measure should be appropriate to the domain of use. If is it possible, usually the best way to write a report is to provide the whole confusion matrix. The reader than can calculate the measure which he is most interested in.

Measures for regression evaluation

The measures described above are mainly used for classification problems rather than for regression problems. For regression problems more appropriate error measures are used. They are focused on how close is the actual model to the ideal model instead of looking if the predicted value is correct or incorrect. The difference between known value y and predicted value $f(x_i)$ is measured by so called **loss functions**. Commonly used loss functions (errors) are described bellow.

The **square loss** is one of the most common measures used for regression purposes, is it defined as

$$l(y_i, f(x_i)) = (y_i - f(x_i))^2$$

A disadvantage of this measure is its sensitivity to outliers (because squaring of the error scales the loss quadratically). Therefore, data should be filtered for outliers before using of this measure. Another measure commonly used in regression is the **absolute loss**, defined as

$$l(y_i, f(x_i)) = |y_i - f(x_i)|$$

It avoids the problem of outliers by scaling the loss linearly. Closely similar measure to the absolute loss is the ϵ-**insensitive loss**. The difference between both is that this measure does not penalize errors within some defined range ϵ. It is defined as

$$l(y_i, f(x_i)) = max(|y - f(x)| - \epsilon, 0)$$

The average of the loss over the dataset is called **generalization error** or **error rate**. On the basis of the loss functions described above we can define the **mean absolute error** and **mean squared error** as

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - f(x_i)|$$

and

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

, respectively. Often used measure is also the **root mean squared error**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$$

, which has the same scale as the quantity being estimated. As well as the squared loss the mean squared error is sensitive to outliers, while the mean absolute error is not. When a relative measure is more appropriate, we can use the **relative absolute error**

$$RAE = \frac{\sum_{i=1}^{n}|y_i - f(x_i)|}{\sum_{i=1}^{n}|y_i - \bar{y}|}$$

or the **relative squared error**

$$RSE = \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n}y_i$.

2.1.2. Bias and variance

With the most important performance measure - the mean square error, the bias, variance and bias/variance dilemma is directly related. They are described thoroughly in [41]. Due the importance of these characteristics, it is in place to describe them more in detail.

With given statistical model characterized by parameter vector θ we define estimator $\hat{\theta}$ of this model (classification or regression model in our case) as a function of n observations of x and we denote it as

$$\hat{\theta} = \hat{\theta}(x_1, \ldots, x_N)$$

The MSE is equal to the sum of the variance and the squared bias of the estimate, formally

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias(\hat{\theta})^2$$

Thus either bias or variance can contribute to poor performance of the estimator.

The **bias** of an estimator is defined as a difference between the expected value of the method and the true value of the parameter, formally

$$Bias(\hat{\theta}) = E[\hat{\theta}] - \theta = E[\hat{\theta} - \theta]$$

In another words the bias says whether the estimator is correct on average. If the bias is equal to zero, the estimator is said to be unbiased. The estimator can be biased for many reasons, but the most common source of an optimistic bias is using of the training data (or not independent data from the training data) to estimate predictive performance.

The **variance** gives us an interval within which the error appears. For an unbiased estimator the MSE is equal to the variance. It means that even though an estimator is unbiased it still may have large MSE if the variance is large.

Since the MSE can be decomposed into a sum of the bias and variance, both characteristics need to be minimized to achieve good predictive performance. It is common to trade-off some increase in the bias for a larger decrease in the variance [41].

2.2. Comparing algorithms

When we have learned more models and we need to select the best one, we usually use some of described measures to estimate the performance of the model and then we simply choose the one with the highest performance. This is often sufficient way for a model selection. Another problem is when we need to prove the improvement in the model performance, especially if we want to show that one model really outperforms another on a particular learning task. In this way we have to use a test of statistical significance and verify the hypothesis of the improved performance.

The most known and most popular in machine learning is the **paired t test** and its improved version the **k-fold cross-validated pair test**. In paired t test the originial set S is randomly devided into a training set R and a testing set T. Models M_1 and M_2 are trained on the set R and tested on the set T. This process is repeated k times(ussually 30 times [28]). If we assume that each partitioning is drawn independently, then also individual error rates can be considered as different and independent samples from a probability distribution, which follow t *distribution* with k degrees of freedom. Our null hypothesis is that the difference in mean error rates is zero. Then the $Student's\ t\ test$ is computed as follows

$$t = \frac{\sum_{i=1}^{k} \left(Err(M_1)_i - Err(M_2)_i \right)}{\sqrt{Var(M_1 - M_2)/k}}$$

Unfortunately the given assumption is less than true. Individual error rates are not independent as well as error rate differences are not independent, because the training sets and the testing sets in each iteration overlaps. The **k-fold cross-validated pair test** mentioned above is build on the same basis. The difference is in the splitting into a training and a testing set, instead of a random dividing. The original set S is splitted into k disjoint folds of the same size. In each iteration one fold is used for testing and remaining $k - 1$ folds for training the model. In this approach each test set is independent of the others, but the training sets still overlaps. For more details see [28].

The improved version, **the 5xcv paired t test**, proposed in [28] performs 5 replications of 2-fold cross-validation. In each replication, the original dataset is divided into two subsets S_1 and S_2 and each model is trained on each set and tested on the other set. This approach solves the problem of overlapping (correlated) folds, which led to poorly estimated means and large t values.

Another approaches described in literature are **McNemar's test** [33], **The test for the difference of two proportions** [82] and many others.

Methods described above consider comparison over one dataset, for comparison of classifiers over multiple data sets see [26].

2.3. Dataset comparison

In some cases we need to compare two datasets, if they have the same distributions. For example if we split the original dataset into a training and a testing set, we expect that a representative sample will be in each subset and distributions of the sets will be the same (with a specific tolerance of deviation). If we assess splitting algorithms, one of the criteria

will be the capability of the algorithm to divide the original dataset into the two identically distributed subsets.

For comparing datasets distributions we should use a statistical test under the null hypothesis that distributions of the datasets are the same. These tests are usually called **goodness-of-fit** tests and they are widely described in literature [2, 8, 59, 85]. For an univariate case we can compare distributions relatively easily using one of the numerous graphical or statistical tests e.g. histograms, PP and QQ plots, the Chi-square test for a dicrete multinominal distribution or the Kolmogorov-Smirnov non-parametric test. For more details see [87].

A multivariate case is more complicated because generalization to more dimensions is not so straightforward. Generalization of the most cited goodness-of-fit test, the Kolmogorov-Smirnov test, is in [10, 35, 54].

In the case of comparing two subsets of one set, we use a naive approach for their comparison. We suppose that two sets are approximately the same, based on comparing basic multivariate data characteristic. We believe, that for our purpose the naive approach is sufficient. Advantages of this approach are its simplicity and a low computational complexity in comparison with the goodness-of-fit tests. A description of commonly used multivariate data characteristics follows.

The first characteristic is the **mean vector**. Let $\mathbf{x}$ represent a random vector of p variables, and $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ denote the i-th instance in the sample set, the **sample mean error** is defined as

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i = \begin{pmatrix} \bar{\mathbf{x}}_1 \\ \bar{\mathbf{x}}_2 \\ \vdots \\ \bar{\mathbf{x}}_p \end{pmatrix}$$

where n is the number of observations. Thus $\bar{x}_i$ is the mean of the i-th variable on the n observations. The mean of $\mathbf{x}$ over all possible instances in the population is called **population mean vector** and is defined as a vector of expected values of each variable, formally

$$\boldsymbol{\mu} = E(\mathbf{x}) = \begin{pmatrix} E(x_1) \\ E(x_2) \\ \vdots \\ E(x_p) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{pmatrix}$$

Therefore, $\bar{\mathbf{x}}$ is an estimate of $\boldsymbol{\mu}$.

Second characteristic is the **covariance matrix**. Let $s_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$ be a sample covariance between j-th and k-th variable. We define the **sample covariance matrix** as

$$S = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,p} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p,1} & s_{p,2} & \cdots & s_{p,p} \end{pmatrix}$$

Because $s_{jk} = s_{kj}$, the covariance matrix is symmetric and there are variances s_j^2, the squares of standard deviations s_j, on the diagonal of the matrix. Therefore, the covariance matrix is

also called **variance-covariance** matrix. As for the mean, the covariance matrix over whole population is called **population covariance matrix** and is defined as

$$\Sigma = cov(\mathbf{x}) = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,p} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p,1} & \sigma_{p,2} & \cdots & \sigma_{p,p} \end{pmatrix}$$

where $\sigma_{jk} = E[(x_j - \mu_j)(x_k - \mu_k)]$.

The covariance matrix contains $p \times p$ values corresponding to all pairs of variables and their covariances. The covariance matrix could be inconvenient in some cases and therefore it can be desired to have one single number as an overall characteristic. One measure summarising the covariance matrix is called **generalized sample variance** and is defined as the determinant of the covariance matrix

$$generalized\ sample\ variance = |S|$$

The geometric interpretation of the generalized sample variance is a p-dimensional hyperellipsoid centered at $\bar{x}$.

More details about the multivariate data characteristic can be found in [77].

3. Data splitting

In the ideal situation we have collected more independent data sets or we can simply and inexpensively repeat an experiment to collect new ones. We can use independent data sets for learning, model selection and even an assessment of the prediction performance. In this situation we have not any reason to split any particular dataset. But in situation when only one dataset is available and we are not capable to collect new data, we need some strategy to perform particular tasks described earlier. In this section we review several data splitting strategies and data splitting algorithms which try to deal with the problem of absence of independent datasets.

3.1. Data splitting strategies

When only one dataset is given, several possible ways how to use available data come into consideration to perform tasks described in section 2 (training, validation, testing). We can split available data into two or more parts and use each to perform a particular task. Common practise is to split data into two or three sets:
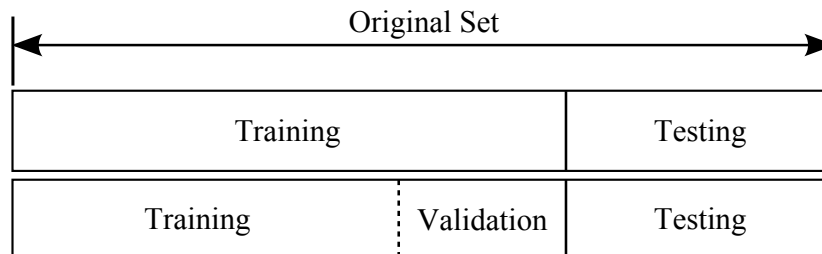
Original Set		
Training		Testing
Training	Validation	Testing

Figure 3. Two and three way splitting

Training set - a set used for learning and estimating parameters of the model.

Validation set - a set used to evaluate the model, usually for model selection.

Testing set - a set of examples used to assess the predictive performance of the model.

Let us define following data splitting strategies according to how data used in a process of model building are available.

The null strategy (*Strategy 0*) is when all available data are used for all tasks. Training, selecting and making an assessment on the same data usually leads to over-fitting of the model and to an over-optimistic estimate of the predictive accuracy. The error estimated on the same set as the model was trained is known as **re-substitution error**.

The strategy motivated by the arrival of new data (*Strategy 1*) uses one set for training and the second set, containing the first set and newly collected data, for the assessment. Merging new collected data with the old data loses the independence of model selection and assessment, which can lead to an over-optimistic estimate of the performance of the model.

The most commonly used strategy is to split data into two sets, a training set and a testing set. The training set (also called the estimation set) is used to estimate the parameters of the model and also for model selection (validation). The testing set is then used to assess the prediction performance of the model (*Strategy 2*).

Another strategy (*Strategy 3*) which splits data into two sets uses one set for learning and the second for model selection and to assess its predictive performance.

The use an independent set for each task is generally recommended. This strategy (*Strategy 4*) splits available data into three sets.

Strategy	Training	Validation	Testing
0	All data	All data	All data
1	Part 1	All data	All data
2	Part 1	Part 1	Part 2
3	Part 1	Part 2	Part 2
4	Part 1	Part 2	Part 3

Table 1. Data usage in different splitting strategies

3.2. Data splitting algorithms

Many data splitting algorithms were proposed. Quality and complexity of algorithms differ and not any approach is superior in general. Data splitting methods and algorithms and their comparison can be found in literature [15, 68, 83, 86]. Some of commonly used algorithms are described bellow.

The holdout method described in [67] is the simplest method that takes an original dataset and splits it randomly into two sets. Common practise is to use one third for testing and the rest for training or half to half. Assuming that the performance of the model increases with the count of seen instances and decreases with the count of left instances apart of the training leads to higher bias and decreases the performance. In other words, both subsets might have different distributions. Moreover, if a dataset is not large enough, and it is usually not, the

holdout method is inefficient in the use of data. For example in a classification problem one or more classes might be missing in one of the subsets, which leads to poor estimation of the model as well as to its evaluation. In deal with this some advanced versions use so called stratification. *Stratified sampling* is a probability sampling, where an original dataset is divided into non-overlapping groups called strata, and instances are selected from each strata proportionally to the appropriate probability. It ensures that each class is represented with the same frequency in both subsets. But it still does not prevent inception of the bias in training and testing sets. For better reliability of the error estimation, the methods are repeated and the resulting accuracy is calculated as an average over all iterations. It can positively reduce the bias. The **Repeated holdout** method is also known as Monte Carlo Cross-validation, Random Sub-sampling or Repeated Evaluation Sets.

The most popular resampling method is **Cross-validation**. In **k-fold cross-validation**, the original data set is splitted into k disjoint folds of the same size, where k is a parameter of the method. In each from k turns one fold is used for evaluation and the remaining $k - 1$ folds for model learning as shown in Figure 4. As in the repeated holdout method, the resulting accuracy is the average of all turns. As well as holdout method, k-fold cross-validation suffers on a pessimistic bias, when k is small. Increasing the count of folds reduces the bias, but increases the variance of the estimation [41]. Experiments have shown that good results across different domains have the k-fold cross-validation method with ten folds [40], but in general k is unfixed. The k-fold cross-validation is very similar to the repeated holdout method with advantage that all the instances of the original data set are used for learning the model and even for evaluation.
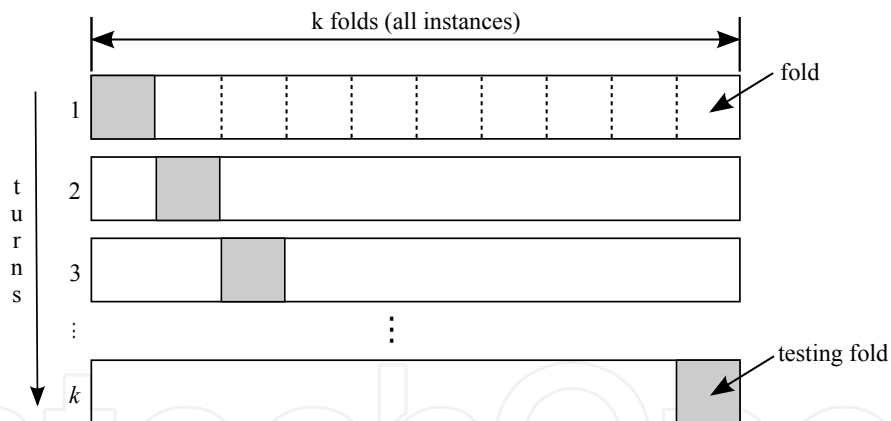

Figure 4. Cross-validation

Leave-one-out cross-validation (LOOCV) is the special case of the k-fold cross-validation in which $k = n$, where n is the size of the original dataset. All test sets have always only one instance. This method makes the best use of data and does not involve any random sub-sampling. According to this, the LOOCV gives nearly unbiased estimates of a model performance but usually with large variability. However, this method is extremely computationally expensive, that makes it often inapplicable.

The **Bootstrap** method was introduced in [89]. The main idea of the method is described as follows. Given a dataset S of size n, generate B bootstrap samples by uniform sampling (*with replacement*), n instances from the dataset. Notice that sampling with replacement allows to select the same instance more than once. After re-sampling, estimate parameters of a model

on each bootstrap sample and than estimate a prediction performance of the model on the original dataset. The overall prediction error is given by averaging these B estimates. Process is schematically shown in Figure 5.
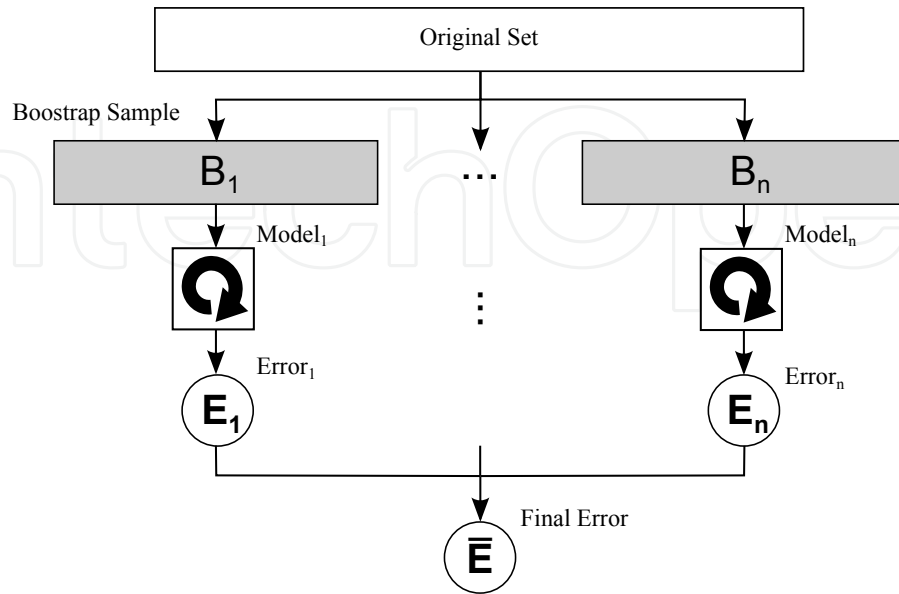

Figure 5. Bootstrap

The most known and commonly used approach is the **.632 bootstrap**. The number 0.632 in the name means the expected fraction of distinct instances of the original dataset appeared in the training set. Each instance has a probability of $1/n$ to being selected from n instances ($(1 - 1/n)$ to not being selected). It gives the probability of $(1 - 1/n)^n \approx e^{-1} \approx 0.368$ not to be selected after n samples. In other words, we expect that 63.2% instances of the original dataset will be selected for training and 36.8% remaining instances will be used for testing. The *.632 bootstrap estimate* is defined as

$$Acc(T) = \frac{1}{B} \sum_{i=1}^{B} (0.632 \cdot Acc(B_i)_{B_i'} + 0.368 \cdot Acc(B_i)_T)$$

where $Acc(B_i)_{B_i'}$ is the accuracy of the model built with bootstrap sample B_i as the training set and applied to the test set B_i' and $Acc(B_i)_T$ is the accuracy of the same model applied to the original dataset. Comparison of the bootstrap with other methods can be found in literature [5, 13, 48, 56, 89]. The results show that 0.632 bootstrap estimates have usually low variability but with a large bias in comparison with the cross-validation that gives approximately unbiased estimates, but with a high variability. It is also reported that the 0.632 bootstrap works best for small datasets. Some experiments showed that the .632 bootstrap fails in some cases, for more details see [3, 5, 11, 56].

Kennard-Stone's algorithm (CADEX) [25, 55] is used for splitting data sets into two distinct subsets which cover approximately the same region of the factor space defined by the original dataset. Instead of measuring coverage by an explicit criterion, the algorithm follows two guidelines. The first one is that no instance from one set should be too far from any instance of the other set, and the second one is that the coverage should start on the boundary of the factor space. The instances are chosen sequentially and the aim is to select the instances in each

iteration to get uniformly distributed instances over the space defined by original dataset. The algorithm works as follows. Let P be the subset of already selected instances and let Q be the dataset equal to T at the beginning. We define $Dist(p, q)$ as the distance from instance $p \in P$ to instance $q \in Q$ and $\Delta_q(P)$ will be the minimal distance from instance q over the set of already selected instances in P.

$$\Delta_q(P) = arg \min_{p \in P}(Dist(p, q))$$

The algorithm starts with adding two most distant instances from Q to P (it is not necessary to select the most distant instances, they can be any instances, but accordingly to the idea of coverage, we usually choose two most distant instances). In each iteration the algorithm selects an instance from the remaining instances in the set Q using the criterion

$$\Delta_Q(P) = arg \max_{q \in Q} \Delta_q(P)$$

In other words, for each instance remaining in the data set Q find the smallest distances to already selected instances in P and choose the one with the maximal distance among these smallest distances. The process is repeat until enough objects are selected. First iteration of the algorithm is shown in Figure 6(a) and in Figure 6(b) is final result with area covered by each set. Since the algorithm uses distances it is sensitive to the used metrics and eventual outliers. For classification purposes subsets should be selected from the individual classes [24]. Improved version of CADEX named **DUPLEX** is described in [83].
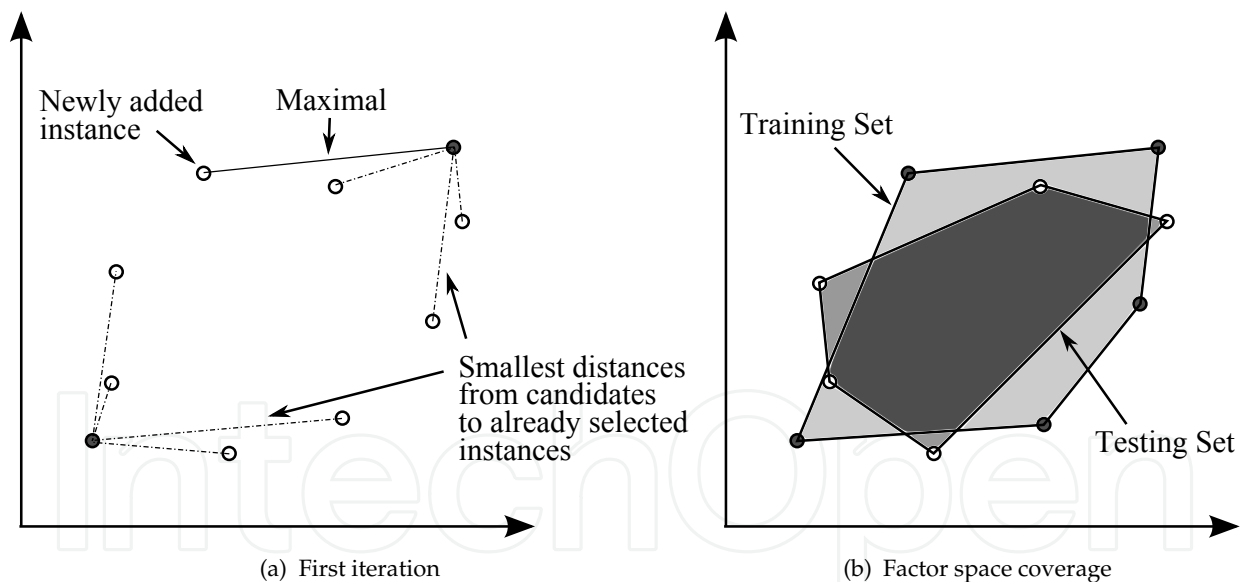

(a) First iteration   (b) Factor space coverage

Figure 6. CADEX

Other methods can be considered when we take into account the following assumption. We suppose that two sets P and Q formed by splitting the original dataset S are as similar as possible when sum of distances of all pairs (one instance from the pair is from P and the other from Q) are minimized. Formally

$$d^* = arg \min_d \sum_{\{p,q\} \in S} dist(p, q).$$

To find the optimal splitting to the two sets is computationally very expensive. Two heuristic approaches come to mind. The first is a method based on the **Nearest neighbour** rule. This simple method splits original datasets into two or more datasets by finding the nearest instance (nearest neighbour) of randomly chosen instance and putting each instance into a different subset. The second heuristics finds the **closest pair** (described in [88]) of instances in S and put one instance into P and the second instance into Q. This is repeated until the set T is empty. The result of these algorithms are two disjoint subsets of the original dataset. The question is how properly will this heuristics work in practice.

4. Instance selection

As was mentioned earlier the instance selection is a process of reducing original data set. A lot of instance selection methods have been described in the literature. In [14] it is argued that instance selection methods are problem dependent and none of them is superior over many problems then others. In this section we review several instance selection methods.

According to the strategy used for selecting instances, we can divide instance selection methods into two groups [71]:

Wrapper methods
  The selection criterion is based on the predictive performance or the error of a model (commonly, instances that do not contribute to the predictive performance are discarded from the training set).
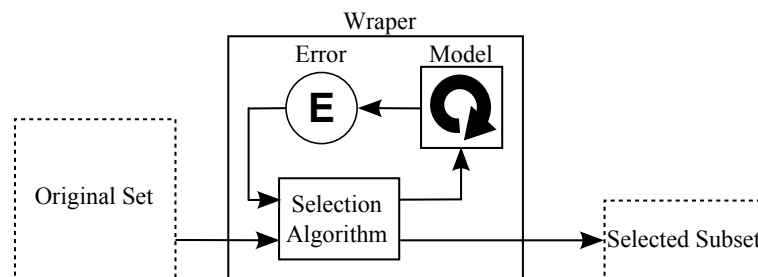

Figure 7. Wraper method

Filter methods
  The selection criterion is a function that is not based upon an algorithm used for prediction but rather on features of the instance vector.
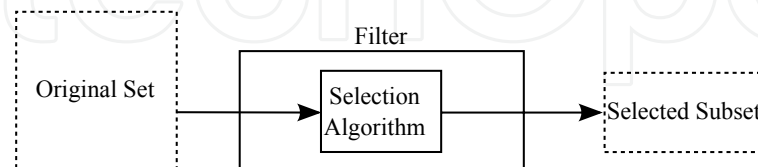

Figure 8. Filter method

Other dividing is also used in literature. Dividing of instance selection methods according to the type of application is proposed in [49]. **Noise filters** are focused on discarding useless instances while **prototype selection** is based on building a set of representatives (prototypes). How instance selection methods create final dataset offers the last presented dividing method. **Incremental methods** start with $S = \varnothing$ and take representatives from T and insert them into

S during the selection process. **Decremental methods** start with $S = T$ and remove useless instances from S during the selection process. **Mixed methods** combine previous methods during the selection process.

A good review of instance selection methods is in [65, 71]. A comparison of instance selection algorithms on several benchmark databases is presented in [50]. Some of instance selection algorithms are described bellow.

4.1. Wrapper methods

The first published instance selection algorithm is probably **Condensed Nearest Neighbour (CNN)** [23]. It is an incremental method starting with new set R which includes one instance per class chosen randomly from S. In the next step the method classifies S using R as a training set. After the classification, each wrongly classified instance from S is added to R (*absorbed*). CNN selects instances near the decision border as shown in Figure 9. Unfortunately, due to this procedure the CNN can select noise instances. Moreover, performance of the CNN is not good [43, 49].
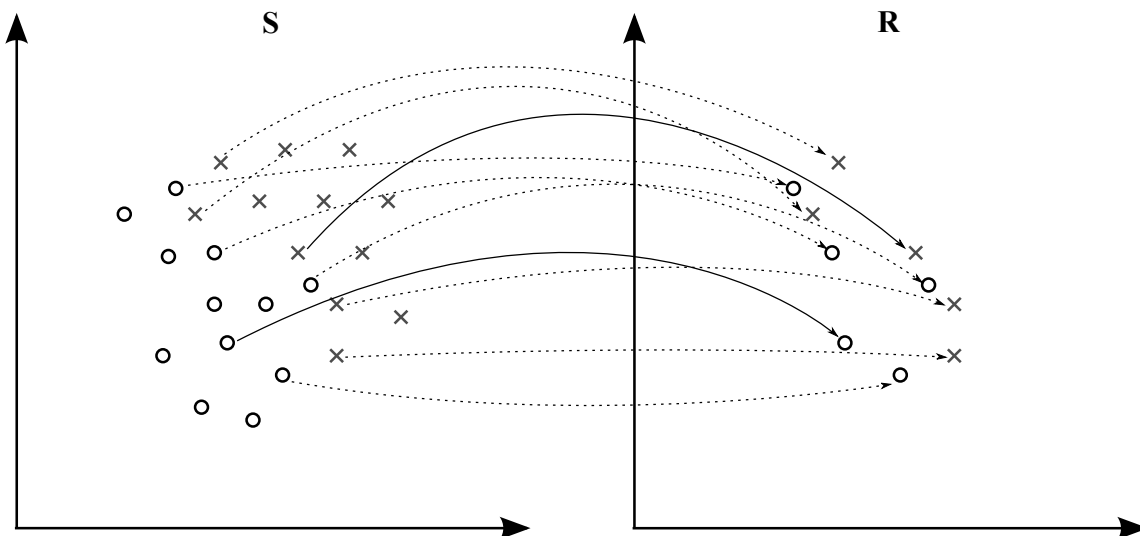

Figure 9. CNN - selected instances

Reduced Nearest Neighbour (RNN) is a modification of the CNN introduced by [39]. The RNN is a decremental method that starts with $R = S$ and removes all instances that do not decrease the predictive performance of a model trained using S.

Selective Nearest Neighbour (SNN) [79] is based on the CNN. It finds a subset $R \subset S$ satisfying that all instances are nearer to the nearest neighbour of the same class in R than to any neighbour of the other class in S.

Generalized Condensed Nearest Neighbour (GCNN)[21] is another instance selection decision rule based on the CNN. The GCNN works the same way as the CNN, but it also defines the following absorption criterion: instance x is absorbed if $\|x - q\| - \|x - p\| > \delta$, where p is the nearest neighbour of the same class as x and q is the nearest neighbour belonging to a different class than x.

Edited Nearest Neighbour (ENN) described in [98] is a decremental algorithm starting with $R = S$. The ENN removes a given instance from R if its class does not agree with the

majority class of its neighbourhoods. ENN uses k-NN rule, usually with $k = 3$, to decide about the majority class, all instances misclassified by 3-NN are discarded as shown in Figure 10. An extension that runs the ENN repeatedly until no change is made in R is known as **Repeated ENN (RENN)**. Another modification of the ENN is **All k-NN** published by [90] It is an iterative method that runs the ENN repeatedly for all $k(k = 1, 2, \ldots, l)$. In each iteration misclassified instances are discarded. Another methods based on the ENN are **Multiedit** and **Editing by Estimating Conditional Class Probabilities** described in [27] and [92], respectively.
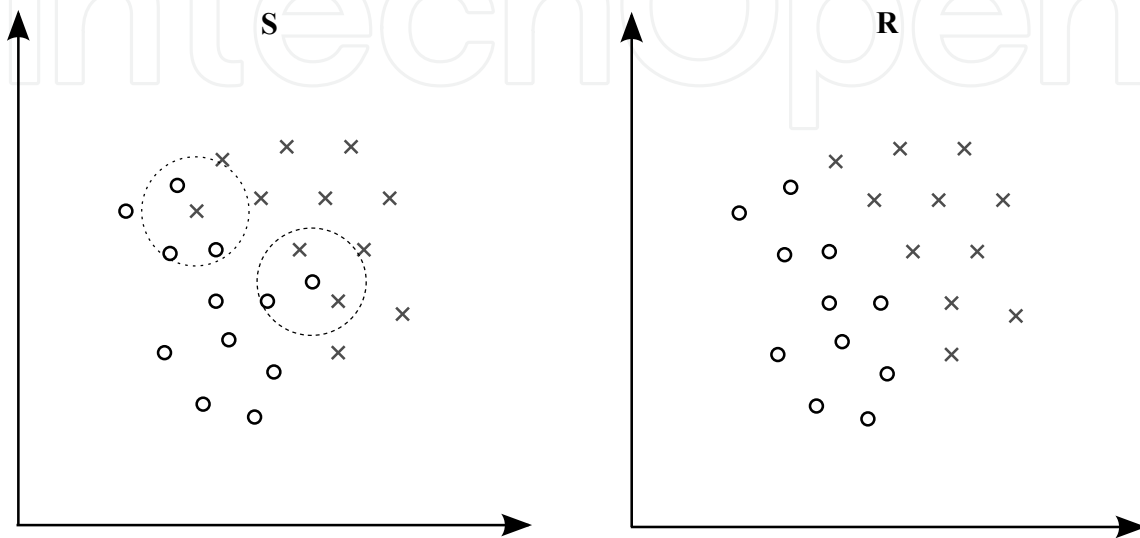

Figure 10. ENN - discarded instances (3-NN)

Instance Based (IB1-3) methods were proposed in [1]. The IB2 selects the instances misclassified by the IB1 (the IB1 is the same as the 1-NN algorithm). It is quite similar to the CNN, but the IB2 does not include one instance per class and does not repeat the process after the first pass through a training set like the CNN. The last version, the IB3, is an incremental algorithm extending the IB2. the IB3 uses a significance test and accepts an instance only if its accuracy is statistically significantly greater than the frequency of its class. Similarly, an instance is rejected if its accuracy is statistically significantly lower than the frequency of its class. Confidence intervals are used to determine the impact of the instance (0.9 to accept, 0.7 to reject).

Decremental Reduction Optimization Procedures (DROP1-5) are instance selection algorithms presented in [99]. These methods use an associate that can be defined by function $Associates(x)$ that collects all instances that have x as one of its neighbours. The DROP1 method removes instances from R that do not change a classification of its associates. The DROP2 is the same as the DROP1 but the associates are taken from the original sample set S instead of considering only instances remaining in R as the DROP1 method. The DROP3 and DROP4 methods run a noise filter first and then apply the DROP2 method. The DROP5 method is another version of the DROP2 extended of discarding the nearest opposite class instances.

Iterative Case Filtering (ICF) are described in [14]. They define $LocalSet(x)$ as a set of cases contained in the largest hypersphere centred at x such that only cases in the same class as x are contained in the hypersphere. They defined property $Adaptable(x, x')$ as $\forall x \in LocalSet(x')$.

It means that instance x can be adapted to x'. Moreover they define two properties based on the adaptable property called *reachability* and *coverage* and defined as follows.

$$Reachability(x) = x' \in S : Adaptable(x', x)$$

$$Coverage(x) = x' \in S : Adaptable(x, x')$$

The algorithm is based on these two properties. At first, the ICF uses the ENN to filter noise instances then the ICF repeatedly computes defined instance properties and in each iteration removes instances that have $|Reachability(x)| > |Coverage(x)|$. The process is repeated until no progress is observed. Another method based on the same properties, the reachability and coverage, was proposed in [104].

Many other methods were proposed in literature. Some of them are based on evolutionary algorithms (EA)[38, 64, 84, 91], other methods use the support vector machine (SVM) [9, 17, 61, 62] or tabu search (TS) [18, 42, 103].

4.2. Filter methods

The **Pattern by Ordered Projections (POP)** method [78] is a heuristic approach to find representative patterns. The main idea of the algorithm is to select only some border instances and eliminate the instances that are not on the boundaries of the regions to which they belong. It uses the function $weakness(x)$, which is defined as the number of times that example x does not represent a border in a partition for every partitions obtained from ordered projected sequences of each attribute, for discarding irrelevant instances that have weaknesses equal to the number of attributes of data set. The weakness of an instance is computed by increasing the weakness for each attribute, where the instance is not near to another instance with different class.

Another method based on finding border instances is the **Pair Opposite Class-Nearest Neighbour (POC-NN)** [75]. The POC-NN calculates the mean of all instances in each class and finds a border instance p_{b1} belonging to the class C_1 as an instance that is the nearest instance to m_2, which is the mean of class C_2. The same way it finds other border instances.

The **Maxdiff kd trees** described in [69] is a method based on *kd trees* [37]. The algorithm builds a binary tree from an original data set. All instances are in the root node and child's nodes are constructed by splitting the node by a pivot, which is a feature with the maximum difference between consecutively ordered values. The process is repeated until no node can be split. Leaves of the tree are the output condensed set.

Several methods are based on clustering. They split an original dataset into n clusters and centres of the clusters are selected as instances [9, 16, 65]. Some extensions were proposed. **The Generalized-Modified Chang algorithm (GCM)** merges the nearest clusters with the same class and uses centres of the merged clusters. **The Nearest Sub-class Classifier** method (NSB) [93] selects more instances (centres) for each class using the Maximum Variance Cluster algorithm [94]. Another method is based on clustering. The **Object Selection by Clustering (OSC)** [4] selects border instances in heterogeneous clusters and some interior instances in homogeneous clusters.

Some prototype filtering methods were proposed in the literature. The first described is **Weighting prototype (WS)**[73] method. The WS method assigns a weight to each prototype

($\forall x \in T$) and selects only those with a larger weight than a certain threshold. The WS method uses a gradient descent algorithm for computing weights of instances. Another published prototype method is **Prototype Selection by Relevance (PSR)**[70]. The PSR computes the *relevance* of each instance in T. The most similar instances in the same class are the most relevant. The PSR selects a user defined portion of relevant instances in the class and the most similar instances belonging to the different class - the border instances.

5. Class balancing

A data set is well-balanced, when all classes are represented with the same proportion, but in practise many domains of classification tasks are characterized by a small proportion of positive instances and a large proportion of negative instances, where the positive instances are usually our points of interest. This problem is commonly known as the class imbalance problem.

Although the performance of a classifier over all instances can be high, we are usually interested in classification of positive instances (true positive rate) only, where the classifier often fails, because it tends to classify all instances into the majority class. To avoid this problem some strategy should be used when a dataset is imbalanced.

Class-balancing methods can be divided into the three main groups according to the strategy of their use. Data level methods are used in preprocessing and usually utilize various ways of re-sampling. Algorithm-level methods modify a classifier or a learning process to solve the imbalance. The last strategy is based on combining various methods to increase the performance.

This chapter gives an overview of class balancing strategies and some particular methods. Two good and detailed reviews were published in [44, 57].

5.1. Data-level methods

The aim of these methods is to change distributions of classes by increasing the number of instances of the minority class (over-sampling), decreasing the number of instances of the majority class (under-sampling), by combinations of these methods or using other advanced sampling ways.

5.1.1. Under-sampling

The first and the most naive under-sampling method is **random under-sampling** [52]. The random under-sampling method balances the class distributions by discarding, at random, instances of the majority class. Because of the randomness of elimination, the method discards potentially useful instances, which can lead to a decrease of the model performance.

Several heuristic under-sampling methods have been proposed in literature, some of them are linked with instance selection metods mentioned in section 4. The first described algorithm is **Condensed nearest neighbour (CNN)** [23] and the second is **Wilson's Edited Nearest Neighbour (ENN)**[98]. Both are based on discarding noisy instances.

A method based on the ENN, the **Neighbourhood Cleaning Rule (NCL)** [63], discards instances from the minority and majority class separately. If an instance belongs to the

majority class and it is misclassified by its three nearest neighbours' instances (the nearest neighbour rule [23]), then the instance is discarded. If an instance is misclassified in the same way and belongs to the minority class, then neighbours that belongs to the majority class are discarded.

Another method based on the *Nearest Neighbour Rule* is the **One-side Sampling (OSS)** [60] method. It is based on the idea of discarding instances distant from a decision border, since these instances can be considered as useless for learning. The OSS uses 1-NN over the set *S* (initially consisting of the instances of the minority class) to classify the instances in the majority class. Each misclassified instance from the majority class is moved to *S*.

The **Tomek Links** [90] focuses on instances near a decision border. Let p,q be instances from different classes and $dist(p,q)$ is the distance between p and q. Pair p,q is called the Tomek link if there is no closer instance of an opposite class to p or q ($dist(p,x) < dist(p,q)$ or $dist(q,x) < dist(p,q)$, where x is the instance of the opposite class than p, respectively q).

5.1.2. Over-sampling

The **random over-sampling** is a naive method, that balances class distributions by replication, at random, instances of the minority class. Two disadvantages of this method were described in literature. The first one, the instance replication increases likelihood of the over-fitting [19] and the second, enlarging the training set by the over-sampling can lead to a longer learning phase and a model response [60], mainly for lazy learners.

The most known over-sampling method is **Synthetic Minority Over-sampling Technique (SMOTE)** [19]. The SMOTE does not over-sample with replacement, instead, it generates "synthetic" instances of the minority class. The minority class is over-sampled by taking each instance of the minority class and its nearest neighbour and placing the "synthetic" instance, at random, along the line joining these instances (Figure 11). This approach avoids over-fitting and causes that a classifier creates larger and less specific decision regions, rather than smaller and more specific ones. The method based on the SMOTE reported better experimental results in *TP-rate* and *F-measure* [45], the **Borderline_SMOTE**. It over-samples only the borderline instances of the minority class.

5.1.3. Advanced sampling

Some advanced re-sampling methods are based on re-sampling of results of the preliminary classification [44].

Over-sampling Algorithm Based on Preliminary Classification (OSPC) was proposed in [46]. It was reported that the OSPC can outperform under-sampling methods and the SMOTE in terms of classification performance [44].

The heuristic method proposed in [96, 97], the **Budget-sensitive progresive sampling algorithm** iteratively enlarges a training set on the basis of performance results from the previous iteration.

A combination of over-sampling and under-sampling methods to improve generalization features of learners was proposed in [45, 58, 63]. A comparison of various re-sampling strategies is presented in [7].
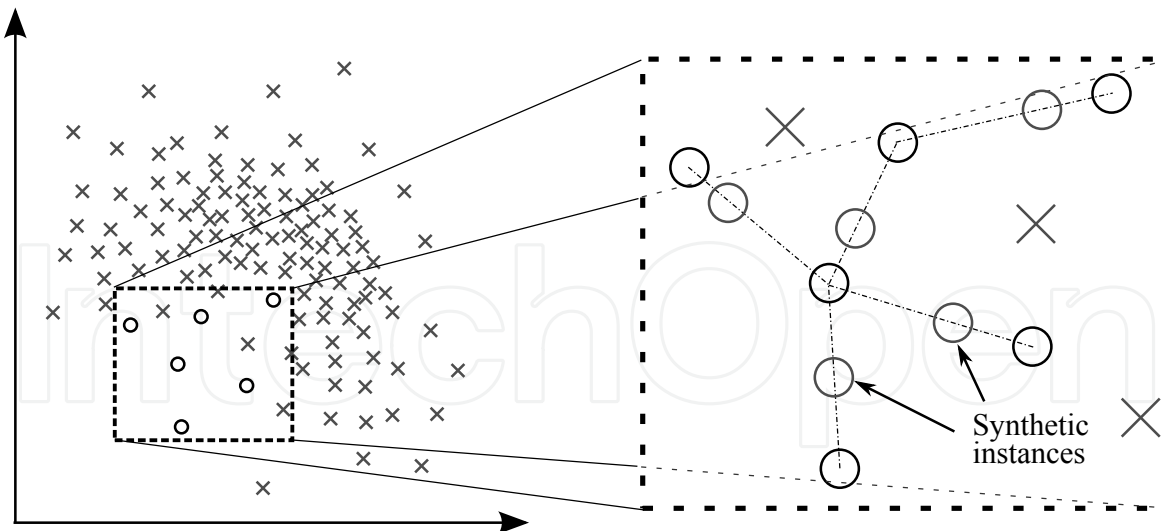
Figure 11. SMOTE - synthetic instances

5.2. Algorithm level methods

Another approach to deal with imbalanced datasets modifies a classifier or a learning process rather than changing distributions of datasets by discarding or replicating instances. These methods are mainly based on overweighting the minority class, discriminating the majority class, penalization for misclassification or biasing the learning algorithm. A short description of published methods follows.

5.2.1. Algorithm modification

Ineffectiveness of the over-sampling method when the C4.5 decision tree learner with the default settings is used was reported in [30]. It was noted that under-sampling produces a reasonable sensitivity to changes in misclassification costs and a class distribution when over-sampling produces little or no change in the performance. It was also noted that modifications of C4.5 parameters in relation to the under/over-sampling does have a strong effect on overall performance.

A method that deals with imbalanced datasets by internally biasing the discrimination procedure is proposed in [6]. This method uses a weighted distance function in a classification phase of the k-NN. Weights are assigned to classes such that the majority class has a greater weighting factor than the minority class. This weighting causes that the distance to minority class instances is lower than the distance to instances of the majority class. Instances of the minority class are then used more often when classifying a new instance.

Different approaches using the SVM biased by various ways for dealing with imbalanced datasets were published. The method proposed in [102] modifies a kernel function for this purpose. In [95] it two schemes for controlling the balance between false positives and false negatives are proposed.

5.2.2. One-class learning

A one-class learning is an alternative to discriminative approaches that deal with imbalanced datasets. In the one-class learning, a model is built using only target class instances. The

model is then learned to recognize these instances, which can be under certain conditions superior to discriminative approaches [51]. Two one-class learning algorithms were studied in literature, particularly the SVM [66, 81] and auto-encoders [51, 66]. An experimental comparison of these two methods can be found in [66]. Usefulness of the one-class learning on extremely unbalanced data sets composed of high dimensional noisy features is showed in [76].

5.2.3. Cost-sensitive learning

A cost-sensitive learning is another commonly used way in the context of imbalanced datasets. A classification model is extended with a cost model in the form of a cost matrix. Given the cost matrix as shown in Figure 2 in section 2 we can define *conditional risk* for making decision α_i about instance x as

$$R(\alpha_i|x) = \sum_j \lambda_{ij} P(j|x)$$

where $P(j|x)$ is a posterior probability of class j being true class of instance x. The goal in a cost-sensitive classification is to minimize the cost of misclassification. This means that the optimal prediction for an instance x is the class i that minimize a conditional risk. Note that the optimal decision can differ from the most probable class [32].

A method which makes classifier cost sensitive, the **MetaCost**, is proposed in [29]. The MetaCost learns an internal cost-sensitive model, then estimates class probabilities and re-labels training instances with their minimum expected cost classes. A new model is built using the relabelled dataset.

The **AdaCost** [34] method based on Adaboost [36] has been made a cost-sensitive by an over-weighting instances from the minority class, which are misclassified. Empirical experiments have shown, that the AdaCost has lower cumulative misclassification costs in comparison with the AdaBoost.

5.3. Ensemble learning methods

Ensemble methods are methods, which use a combination of methods with the aim to achieve better results. Two most known ensemble methods are *bagging* and *boosting*. The bagging (Bootstrap aggregating) proposed in [12] initially generates B bootstrap sets of the original dataset and then builds a classification or regression model using each bootstrap set. Predicted values of these models are combined to predict the final result. In classification tasks it works as follows. Each model has one vote to predict a class, the bagged classifier counts the votes and assigns the class with the most votes. For regression tasks, the predicted value is computed as the average of values predicted by each model.

The boosting, firstly described in [80], is based on the idea a powerful model is created using a set of weak models. The method is quite similar to the bagging. Like the bagging the boosting uses voting for a classification task or averaging for a regression task to predict the output value. However, the boosting is an iterative method. In each iteration a newly built model is influenced by the performance of those built previously. By assigning greater weights to the instances that were misclassified in previous iterations the model pays more attention on these instances.

Another in comparison with bagging and boosting less widely used method is *stacking* proposed in [101]. In the stacking method the original dataset is splitted into two disjoint sets, a training set and a validation set. Several base models are learned on the training set and then applied to the validation set. Using predictions from the validation set as inputs and correct values as the outputs, a higher level model is build. In comparison with the bagging and boosting, the stacking can be used to combine different types of models.

Ensemble methods such the bagging, boosting and stacking often outperform another methods. Therefore, they have been widely studied in recent years and lot of approaches have been proposed. The earlier mentioned **Adaboost** [36] and **AdaCost** [34] are other methods that use the boosting are **RareBoost** [53] or **SMOTEBoost** [20]. A method combining the bagging and stacking to identify the best combination of classifiers is used in [74]. Three agents (Naive Bayes, C4.5, 5-NN) are combined in the approach proposed in [58]. There are many other methods utilizing the mentioned approaches.

6. Conclusion

Several methods for training set re-sampling, instance selection and class balancing, published in literature, were reviewed. All of these methods are very important in processes of construction of training and testing sets. Re-sampling methods allow to split a data set into more subsets in the case of absence of an independent set for model validation or prediction performance assessment. Instance selection methods reduce a training set by removing instances useless for estimating parameters of a model, which can speed up the learning phase and response time, especially for lazy learners. Class balancing algorithms solve the problem of inequality in class distributions.

Acknowledgements

Author details

Tomas Borovicka
Faculty of Information Technology and Faculty of Biomedical Engineering at the Czech Technical University, Prague, Czech Republic

Marcel Jirina, Jr.
Faculty of Biomedical Engineering at the Czech Technical University, Prague, Czech Republic

Pavel Kordik
Department of Computer Science and Engineering, FEE, Czech Technical University, Prague, Czech Republic

Marcel Jirina
Institute of Computer Science at the Czech Academy of Sciences, Prague, Czech Republic

7. References

[1] Aha, D., Kibler, D. & Albert, M. [1991]. Instance-based learning algorithms, *Machine learning* 6(1): 37–66.

[2] Anderson, T. & Darling, D. [1954]. A test of goodness of fit, *Journal of the American Statistical Association* pp. 765–769.

[3] Andrews, D. [2000]. Inconsistency of the bootstrap when a parameter is on the boundary of the parameter space, *Econometrica* 68(2): 399–405.

[4] Arturo Olvera-López, J., Ariel Carrasco-Ochoa, J. & Francisco Martínez-Trinidad, J. [2007]. Object selection based on clustering and border objects, *Computer Recognition Systems 2* pp. 27–34.

[5] Bailey, T. & Elkan, C. [1993]. Estimating the accuracy of learned concepts.", *Proc. International Joint Conference on Artificial Intelligence*, Citeseer.

[6] Barandela, R., SÃₐnchez, J., Garcia, V. & Rangel, E. [2003]. Strategies for learning in class imbalance problems, *Pattern Recognition* 36(3): 849–851.

[7] Batista, G., Prati, R. & Monard, M. [2004]. A study of the behavior of several methods for balancing machine learning training data, *ACM SIGKDD Explorations Newsletter* 6(1): 20–29.

[8] Bentler, P. & Bonett, D. [1980]. Significance tests and goodness of fit in the analysis of covariance structures., *Psychological bulletin* 88(3): 588.

[9] Bezdek, J. & Kuncheva, L. [2001]. Nearest prototype classifier designs: An experimental study, *International Journal of Intelligent Systems* 16(12): 1445–1473.

[10] Bickel, P. [1969]. A distribution free version of the smirnov two sample test in the p-variate case, *The Annals of Mathematical Statistics* 40(1): 1–23.

[11] Bickel, P. & Freedman, D. [1981]. Some asymptotic theory for the bootstrap, *The Annals of Statistics* 9(6): 1196–1217.

[12] Breiman, L. [1996]. Bagging predictors, *Machine learning* 24(2): 123–140.

[13] Breiman, L. & Spector, P. [1992]. Submodel selection and evaluation in regression. the x-random case, *International Statistical Review/Revue Internationale de Statistique* pp. 291–319.

[14] Brighton, H. & Mellish, C. [2002]. Advances in instance selection for instance-based learning algorithms, *Data mining and knowledge discovery* 6(2): 153–172.

[15] Burman, P. [1989]. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods, *Biometrika* 76(3): 503–514.

[16] Caises, Y., González, A., Leyva, E. & Pérez, R. [2009]. Scis: combining instance selection methods to increase their effectiveness over a wide range of domains, *Proceedings of the 10th international conference on Intelligent data engineering and automated learning*, Springer-Verlag, pp. 17–24.

[17] Cano, J., Herrera, F. & Lozano, M. [2003]. Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study, *Evolutionary Computation, IEEE Transactions on* 7(6): 561–575.

[18] Cerverón, V. & Ferri, F. [2001]. Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule, *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 31(3): 408–413.

[19] Chawla, N., Bowyer, K., Hall, L. & Kegelmeyer, W. [2002]. Smote: Synthetic minority over-sampling technique, *Journal of Artificial Intelligence Research* 16: 321–357.

[20] Chawla, N., Lazarevic, A., Hall, L. & Bowyer, K. [2003]. Smoteboost: Improving prediction of the minority class in boosting, *Knowledge Discovery in Databases: PKDD 2003* pp. 107–119.

[21] Chou, C., Kuo, B. & Chang, F. [2006]. The generalized condensed nearest neighbor rule as a data reduction method, *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, Vol. 2, IEEE, pp. 556–559.

[22] Cohen, G., Hilario, M., Sax, H. & Hugonnet, S. [2003]. Data imbalance in surveillance of nosocomial infections, *Medical Data Analysis* pp. 109–117.

[23] Cover, T. & Hart, P. [1967]. Nearest neighbor pattern classification, *Information Theory, IEEE Transactions on* 13(1): 21–27.

[24] Daszykowski, M., Walczak, B. & Massart, D. [2002]. Representative subset selection, *Analytica Chimica Acta* 468(1): 91–103.

[25] de Groot, P., Postma, G., Melssen, W. & Buydens, L. [1999]. Selecting a representative training set for the classification of demolition waste using remote nir sensing, *Analytica Chimica Acta* 392(1): 67 – 75.
URL: *http://www.sciencedirect.com/science/article/pii/S0003267099001932*

[26] Demšar, J. [2006]. Statistical comparisons of classifiers over multiple data sets, *The Journal of Machine Learning Research* 7: 1–30.

[27] Devijver, P. & Kittler, J. [1980]. On the edited nearest neighbor rule, *Proc. 5th Int. Conf. on Pattern Recognition*, pp. 72–80.

[28] Dietterich, T. [1998]. Approximate statistical tests for comparing supervised classification learning algorithms, *Neural computation* 10(7): 1895–1923.

[29] Domingos, P. [1999]. Metacost: A general method for making classifiers cost-sensitive, *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 155–164.

[30] Drummond, C., Holte, R. et al. [2003]. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling, *Workshop on Learning from Imbalanced Datasets II*, Citeseer.

[31] Duda, R. & Hart, P. [1996]. *Pattern classification and scene analysis*, Wiley.

[32] Elkan, C. [2001]. The foundations of cost-sensitive learning, *International Joint Conference on Artificial Intelligence*, Vol. 17, LAWRENCE ERLBAUM ASSOCIATES LTD, pp. 973–978.

[33] Everitt, B. [1992]. *The analysis of contingency tables*, Chapman & Hall/CRC.

[34] Fan, W., Stolfo, S., Zhang, J. & Chan, P. [1999]. Adacost: misclassification cost-sensitive boosting, *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, Citeseer, pp. 97–105.

[35] Fasano, G. & Franceschini, A. [1987]. A multidimensional version of the kolmogorov-smirnov test, *Monthly Notices of the Royal Astronomical Society* 225: 155–170.

[36] Freund, Y. & Schapire, R. [1995]. A desicion-theoretic generalization of on-line learning and an application to boosting, *Computational learning theory*, Springer, pp. 23–37.

[37] Friedman, J., Bentley, J. & Finkel, R. [1977]. An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software (TOMS)* 3(3): 209–226.

[38] García, S., Cano, J. & Herrera, F. [2008]. A memetic algorithm for evolutionary prototype selection: A scaling up approach, *Pattern Recognition* 41(8): 2693–2709.

[39] Gates, G. [1972]. The reduced nearest neighbor rule (corresp.), *Information Theory, IEEE Transactions on* 18(3): 431–433.

[40] Geisser, S. [1993]. *Predictive inference: An introduction*, Vol. 55, Chapman & Hall/CRC.

[41] Geman, S., Bienenstock, E. & Doursat, R. [1992]. Neural networks and the bias/variance dilemma, *Neural computation* 4(1): 1–58.

[42] Glover, F. & McMillan, C. [1986]. The general employee scheduling problem. an integration of ms and ai, *Computers & operations research* 13(5): 563–573.

[43] Grochowski, M. & Jankowski, N. [2004]. Comparison of instance selection algorithms ii. results and comments, *Artificial Intelligence and Soft Computing-ICAISC 2004* pp. 580–585.

[44] Guo, X., Yin, Y., Dong, C., Yang, G. & Zhou, G. [2008]. On the class imbalance problem, *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, Vol. 4, IEEE, pp. 192–201.

[45] Han, H., Wang, W. & Mao, B. [2005]. Borderline-smote: A new over-sampling method in imbalanced data sets learning, *Advances in Intelligent Computing* pp. 878–887.

[46] Han, J. & Kamber, M. [2006]. *Data mining: concepts and techniques*, The Morgan Kaufmann series in data management systems, Elsevier.
URL: *http://books.google.cz/books?id=AfL0t-YzOrEC*

[47] Hawkins, D. et al. [2004]. The problem of overfitting, *Journal of chemical information and computer sciences* 44(1): 1–12.

[48] Jain, A., Dubes, R. & Chen, C. [1987]. Bootstrap techniques for error estimation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (5): 628–633.

[49] Jankowski, N. & Grochowski, M. [2004a]. Comparison of instances seletion algorithms i. algorithms survey, *Artificial Intelligence and Soft Computing-ICAISC 2004* pp. 598–603.

[50] Jankowski, N. & Grochowski, M. [2004b]. Comparison of instances seletion algorithms i. algorithms survey, *Artificial Intelligence and Soft Computing-ICAISC 2004* pp. 598–603.

[51] Japkowicz, N. [2001]. Supervised versus unsupervised binary-learning by feedforward neural networks, *Machine Learning* 42(1): 97–122.

[52] Japkowicz, N. & Stephen, S. [2002]. The class imbalance problem: A systematic study, *Intelligent data analysis* 6(5): 429–449.

[53] Joshi, M., Kumar, V. & Agarwal, R. [2001]. Evaluating boosting algorithms to classify rare classes: Comparison and improvements, *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, IEEE, pp. 257–264.

[54] Justel, A., Peña, D. & Zamar, R. [1997]. A multivariate kolmogorov-smirnov test of goodness of fit, *Statistics & Probability Letters* 35(3): 251–259.

[55] Kennard, R. & Stone, L. [1969]. Computer aided design of experiments, *Technometrics* pp. 137–148.

[56] Kohavi, R. [1995]. A study of cross-validation and bootstrap for accuracy estimation and model selection, *International joint Conference on artificial intelligence*, Vol. 14, LAWRENCE ERLBAUM ASSOCIATES LTD, pp. 1137–1145.

[57] Kotsiantis, S., Kanellopoulos, D. & Pintelas, P. [2006]. Handling imbalanced datasets: A review, *GESTS International Transactions on Computer Science and Engineering* 30(1): 25–36.

[58] Kotsiantis, S. & Pintelas, P. [2003]. Mixture of expert agents for handling imbalanced data sets, *Annals of Mathematics, Computing & Teleinformatics* 1(1): 46–55.

[59] Kruskal, J. [1964]. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis, *Psychometrika* 29(1): 1–27.

[60] Kubat, M. & Matwin, S. [1997].   Addressing the curse of imbalanced training sets: one-sided selection, *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, MORGAN KAUFMANN PUBLISHERS, INC., pp. 179–186.

[61] Kuncheva, L. [1995]. Editing for the k-nearest neighbors rule by a genetic algorithm, *Pattern Recognition Letters* 16(8): 809–814.

[62] Kuncheva, L. & Bezdek, J. [1998]. Nearest prototype classification: Clustering, genetic algorithms, or random search?, *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 28(1): 160–164.

[63] Laurikkala, J. [2001]. Improving identification of difficult small classes by balancing class distribution, *Artificial Intelligence in Medicine* pp. 63–66.

[64] Li, Y., Hu, Z., Cai, Y. & Zhang, W. [2005]. Support vector based prototype selection method for nearest neighbor rules, *Advances in Natural Computation* pp. 408–408.

[65] Liu, H. & Motoda, H. [2002]. On issues of instance selection, *Data Mining and Knowledge Discovery* 6(2): 115–130.

[66] Manevitz, L. & Yousef, M. [2002]. One-class svms for document classification, *The Journal of Machine Learning Research* 2: 139–154.

[67] McLachlan, G. & Wiley, J. [1992]. *Discriminant analysis and statistical pattern recognition*, Wiley Online Library.

[68] Molinaro, A., Simon, R. & Pfeiffer, R. [2005]. Prediction error estimation: a comparison of resampling methods, *Bioinformatics* 21(15): 3301–3307.

[69] Narayan, B., Murthy, C. & Pal, S. [2006]. Maxdiff kd-trees for data condensation, *Pattern recognition letters* 27(3): 187–200.

[70] Olvera-López, J., Carrasco-Ochoa, J. & Martínez-Trinidad, J. [2008]. Prototype selection via prototype relevance, *Progress in Pattern Recognition, Image Analysis and Applications* pp. 153–160.

[71] Olvera-López, J., Carrasco-Ochoa, J., Martínez-Trinidad, J. & Kittler, J. [2010]. A review of instance selection methods, *Artificial Intelligence Review* 34(2): 133–143.

[72] Pan, F., Wang, W., Tung, A. & Yang, J. [2005]. Finding representative set from massive data, *Data Mining, Fifth IEEE International Conference on*, IEEE, pp. 8–pp.

[73] Paredes, R. & Vidal, E. [2000]. Weighting prototypes-a new editing approach, *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, Vol. 2, IEEE, pp. 25–28.

[74] Phua, C., Alahakoon, D. & Lee, V. [2004].   Minority report in fraud detection: classification of skewed data, *ACM SIGKDD Explorations Newsletter* 6(1): 50–59.

[75] Raicharoen, T. & Lursinsap, C. [2005].   A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (poc-nn) algorithm, *Pattern recognition letters* 26(10): 1554–1567.

[76] Raskutti, B. & Kowalczyk, A. [2004]. Extreme re-balancing for svms: a case study, *ACM Sigkdd Explorations Newsletter* 6(1): 60–69.

[77] Rencher, A. [2002]. Methods of multivariate analysis.

[78] Riquelme, J., Aguilar-Ruiz, J. & Toro, M. [2003]. Finding representative patterns with ordered projections, *Pattern Recognition* 36(4): 1009–1018.

[79] Ritter, G., Woodruff, H., Lowry, S. & Isenhour, T. [1975].   An algorithm for a selective nearest neighbor decision rule (corresp.), *Information Theory, IEEE Transactions on* 21(6): 665–669.

[80] Schapire, R. [1990]. The strength of weak learnability, *Machine learning* 5(2): 197–227.

[81] Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A. & Williamson, R. [2001]. Estimating the support of a high-dimensional distribution, *Neural computation* 13(7): 1443–1471.

[82] Snedecor, G. & Cochran, W. [1967]. Statistical methods 6th ed, *IOWA State University press, USA. 450pp* .

[83] Snee, R. [1977]. Validation of regression models: methods and examples, *Technometrics* pp. 415–428.

[84] Srisawat, A., Phienthrakul, T. & Kijsirikul, B. [2006]. Sv-knnc: an algorithm for improving the efficiency of k-nearest neighbor, *PRICAI 2006: Trends in Artificial Intelligence* pp. 975–979.

[85] Stephens, M. [1974]. Edf statistics for goodness of fit and some comparisons, *Journal of the American Statistical Association* pp. 730–737.

[86] Stone, M. [1974]. Cross-validatory choice and assessment of statistical predictions, *Journal of the Royal Statistical Society. Series B (Methodological)* pp. 111–147.

[87] Thas, O. [2010]. *Comparing distributions*, Springer.

[88] Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. [2009]. *Introduction to Algorithms*, The MIT Press, London, England.

[89] Tibshirani, R. & Efron, B. [1993]. An introduction to the bootstrap, *Monographs on Statistics and Applied Probability* 57: 1–436.

[90] Tomek, I. [1976]. An experiment with the edited nearest-neighbor rule, *IEEE Transactions on Systems, Man, and Cybernetics* (6): 448–452.

[91] Vapnik, V. [2000]. *The nature of statistical learning theory*, Springer Verlag.

[92] Vázquez, F., Sánchez, J. & Pla, F. [2005]. A stochastic approach to wilsonâĂŹs editing algorithm, *Pattern Recognition and Image Analysis* pp. 35–42.

[93] Veenman, C. & Reinders, M. [2005]. The nearest sub-class classifier: a compromise between the nearest mean and nearest neighbor classifier, *Transactions on PAMI* 27(9): 1417–1429.

[94] Veenman, C., Reinders, M. & Backer, E. [2002]. A maximum variance cluster algorithm, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24(9): 1273–1280.

[95] Veropoulos, K., Campbell, C. & Cristianini, N. [1999]. Controlling the sensitivity of support vector machines, *Proceedings of the international joint conference on artificial intelligence*, Vol. 1999, Citeseer, pp. 55–60.

[96] Weiss, G. [2003]. *The effect of small disjuncts and class distribution on decision tree learning*, PhD thesis, Rutgers, The State University of New Jersey.

[97] Weiss, G. & Provost, F. [2003]. Learning when training data are costly: The effect of class distribution on tree induction, *J. Artif. Intell. Res. (JAIR)* 19: 315–354.

[98] Wilson, D. [1972]. Asymptotic properties of nearest neighbor rules using edited data, *Systems, Man and Cybernetics, IEEE Transactions on* 2(3): 408–421.

[99] Wilson, D. & Martinez, T. [2000]. Reduction techniques for instance-based learning algorithms, *Machine learning* 38(3): 257–286.

[100] Witten, I., Frank, E. & Hall, M. [2011]. *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann.

[101] Wolpert, D. [1992]. Stacked generalization*, *Neural networks* 5(2): 241–259.

[102] Wu, G. & Chang, E. [2003]. Class-boundary alignment for imbalanced dataset learning, *ICML 2003 workshop on learning from imbalanced data sets II*, pp. 49–56.

[103] Zhang, H. & Sun, G. [2002]. Optimal reference subset selection for nearest neighbor classification by tabu search, *Pattern Recognition* 35(7): 1481–1490.

[104] Zhao, K., Zhou, S., Guan, J. & Zhou, A. [2003]. C-pruner: An improved instance pruning algorithm, *Machine Learning and Cybernetics, 2003 International Conference on*, Vol. 1, IEEE, pp. 94–99.