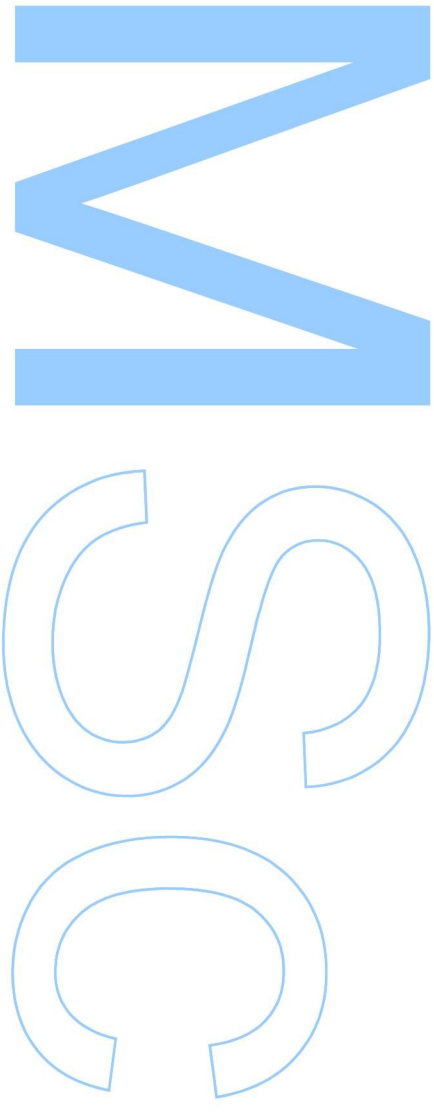
Is Multiple Kernel Learning better than other classifier methods?

José António Amorim Lage de Carvalho
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2019

Orientador
Inês de Castro Dutra, Professor Auxiliar
Faculdade de Ciências da Universidade do Porto



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Abstract

Multiple kernel learning (MKL) is a machine learning approach that uses a combination of kernels aiming for better modeling of different types of variables. It is particularly suitable to solve multimodal learning tasks and has been used in multiple fields. However very few works compare the method with other machine learning models, which hinders its actual quality and performance when compared with other simpler models. In this work, we implement a variation of an MKL algorithm and compare its quality and time performance with other machine learning models. Although having a potential to build better data models because it can use multiple kernels for subsets of variables, MKL has a very high computational cost. The algorithm combines several kernels, but it needs to find the best weight for each kernel. The main algorithm needs to handle four main problems: (1) which kernels to choose, (2) how many kernels to use, (3) how to combine them, and (4) how to find the weights. We concentrate on the fourth problem and present a novel algorithm that uses a genetic algorithm to choose the best set of weights. We start by studying various approaches to MKL and discussing their advantages and drawbacks. Contrary to most works in the literature, we compare our method with Support Vector Machines, decision trees, random forests, k-nearest neighbours, naive Bayes and neural networks, applied to multiple datasets. We use the F1-score to compare the quality performance of the models and guarantee that the datasets have well balanced classes. Results show that MKL can have a competitive quality when compared with the other models for some datasets, but its computational time is still prohibitive even for small datasets.

Resumo

Multiple kernel learning (MKL) é uma abordagem de aprendizagem automática que usa uma combinação de *kernels* visando gerar modelos de dados que levem em consideração diferentes tipos de variáveis. É particularmente interessante para a modelagem de dados multi-modais e tem sido usado em vários domínios. No entanto, muito poucos trabalhos comparam o método com outros modelos de aprendizagem automática, escondendo a real qualidade e desempenho deste método quando comparado com outros. Neste trabalho, implementamos uma variação de um algoritmo MKL e comparamos sua qualidade e desempenho de tempo com outros modelos de aprendizagem automática. Embora tenha potencial para construir melhores modelos de dados porque pode usar vários `kernels` para subconjuntos de variáveis, o MKL possui um alto custo computacional. O algoritmo combina vários `kernels`, mas precisa encontrar o melhor peso para cada `kernel`. O algoritmo precisa lidar com quatro problemas principais: (1) quais de cada `kernel` escolher, (2) quantos escolher, (3) como combiná-los e (4) como encontrar os pesos. Concentramo-nos no quarto problema e apresentamos um novo algoritmo que usa uma anordagem genética para escolher o melhor conjunto de pesos. Começamos estudando várias abordagens do MKL e discutindo suas vantagens e desvantagens. Ao contrário da maioria dos trabalhos da literatura, comparamos nosso método com máquinas de vetores de suporte, árvores de decisão, florestas aleatórias, k vizinhos mais próximos, naive Bayes e redes neurais, aplicados a vários conjuntos de dados. Usamos o escore F1 para comparar a qualidade e desempenho dos modelos, garantindo que os conjuntos de dados tenham classes bem equilibradas. Os resultados mostram que o MKL pode ter uma qualidade competitiva quando comparado com outros modelos para alguns conjuntos de dados, mas seu tempo computacional ainda é proibitivo, mesmo para pequenos conjuntos de dados.

Acknowledgements

First I would like to express my gratitude to my family for the support they provided in the last few months. I also would like to thank my advisor Inês Dutra for the guidance, help, and aid that she granted to me in the last year. Lastly, I would like to thank my friends that help me at best and the worst moments to endure the weight of this work, Thank you all.

Contents

List of Tables

List of Figures

Acronyms

MKL Multiple Kernel Learning

SVM Support Vector Machine

AI Artificial Intelligence

KNN K Neareast Neighbours

RBF Radial Basis Function

RAM Random-Access Memory

Chapter 1

Introduction

Machine learning software products are increasing trend in our world with many people using tools like the google translator or siri. The field became particularly trending thanks to google's research in neural networks for deep learning [3] which resulted in solving many applications. Despite the success of neural networks and deep learning, there are still several models that can be useful, specially for learning tasks that involve heterogeneous sources of data. A particular technique is the Multiple Kernel Learning (MKL), which has multiple supporters and has proved to produce better models than, for example, neural networks [15] or Support Vector Machines (SVM) [9] in several applications. MKL is used to solve multimodal problems where multiple works have evidence of improved performance relatively to at least the SVM. For example, in emotion recognition on speech using many features of sound [31] and in an efficient way to join multimodal data [20]. Most work on MKL classifiers compare results against SVMs, but only a few perform a fair comparison with other classification models, which hinders MKL's actual quality and performance when compared with other simpler machine learning models. MKL is not much used due to its high computational complexity. The larger the number of instances and variables, the larger is the search space, specially when tuning parameters for the underlying classifier. Moreover, the combination of kernels using a highly dimensional dataset can deeply hurt performance. In this work, we compare the performance of MKL with several other classifiers and implement a new MKL method where parameters are generated resourcing to a genetic algorithm. We test the performance (time-wise and quality-wise) using various datasets also explored in the literature. Our main contributions are:

- Implementation of a new MKL algorithm based on genetic algorithms inspired by the work of Pinar *et al.* [21].
- Comparison of MKL with other classification methods.

This work is organized as follows. Next chapter introduces the main concepts in SVMs and MKL, describing the two different ways of modelling the data: column-based and row-based. The row-based approach is the one that we follow, since SVMs use instances as support vectors to

build a model that would distinguish between classes. In that chapter we also discuss the main concepts on machine learning, emphasizing supervised classification in contrast to regression or unsupervised learning. We also discuss about SVM and MKL computational complexities and present the classification models we use in this work. In Chapter 3 we review the main works in the literature that use MKL for classification and discuss about their limitations. Next, we present our MKL modeling. We start by discussing about the various choices for implementing MKL (types of kernels, functions to combine kernels and parameter values) and introduce our MKL genetic based algorithm. We next present the datasets used in this work together with the experimental methodology. Chapter 5 presents our results and compares MKL with other classifiers. We compare time and quality on various datasets also used in the MKL literature. Finally, we conclude this work and present perspectives of future work.

Chapter 2

Background

In this chapter we present essential topics to clearly understand this work. We start by briefly discussing about machine learning techniques with emphasis on supervised learning and Support Vector Machines (SVM). We then discuss about model evaluation and validation presenting the main evaluation metrics and validation methods. Next, we present concepts related with SVMs and MKL such as kernels and positive definite functions. An overview of the SVM and MKL methods is then presented followed by a description of genetic algorithms and by a brief description of the classification methods we use in this work to compare with MKL.

2.1 Machine Learning

Machine learning is a sub field of artificial intelligence (AI) that provides a set of algorithms that allow a machine to learn patterns automatically from a set of data without human intervention. The algorithms are divided in supervised and unsupervised. We focus on supervised learning, which is the central subject of our work.

The task of supervised learning is defined as:
Given a training set x of N input-output pairs

$$(x_1, y_1), (x_2, y_2), \cdots (x_N, y_N),$$

where each y_j was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f. When the y_j is one of a finite set of values, the learning task is called classification. It is called binary classification if y_j can assume only two values. When y_j is a number, the learning task is called regression [23].

The concept is simple, but finding a suitable function h is not trivial. Issues related with missing or incorrectly collected and stored data may interfere with the process of finding a good h. When the data is perfect, the machine learning models could overfit. That happens when our approximation h learns very well the data x_j, but when it receives new data that was never seen

before, it performs poorly. Overfitting happens because the function h is excessively adjusted to the data in x.

2.1.1 Evaluation metrics

An error measure is used to assess the learned function. In fact, the same data x and labels y may fit different functions. For supervised learning methods that do classification, the evaluation is usually performed using a confusion (or contingency) matrix. The confusion matrix shows correct classified instances in the main diagonal. All other values are classification errors. Table 2.1 shows an example of a confusion matrix for a set y that contains only two distinct values (binary classification). Considering one of the class values as positive and the second one as negative, this matrix shows the counters for correctly classified positive (tp) and negative (tn) instances in the main diagonal, while errors, false negatives (fn) and false positives (fp) are shown in the secondary diagonal. The confusion matrix can be generalized to any number of class values. From the confusion matrix, we can summarize the most common evaluation metrics for classification:

- $recall = \frac{tp}{fn+tp}$
- $precision = \frac{tp}{tp+fp}$
- $F1\text{-}measure = \frac{2precision \times recall}{precision+recall}$
- $accuracy = \frac{tp+tn}{tp+tn+fn+fp}$
- $error = 1 - accuracy$

Table 2.1: confusion matrix

	positive	negative
predicted positive	tp	fp
predicted negative	fn	tn

When measuring errors, it is also important to choose the normalization norm L_1 or L_2. In some optimization MKL algorithms, a regularization (e.g L_1 or L_p) to avoid overfitting in the estimation of kernels. The regularization have a great affect in the performance of the optimization of MKL algorithms. These algorithms lead to sparse solutions for the kernel weights and their results are interpretable. The L_1-norm regularization leads to more robust classification performance, since it can exclude the weak or irrelevant basis kernels. However, due to their sparse solution for the kernel weights, the L_1-MKL algorithms will be unable to model the whole information content of the data, resulting in a poor generalization capability. To address these issues, the L_p-norm-regularized MKL algorithms are proposed. These solutions are not easily interpretable due to their dense solutions, these algorithms can model all the information content

of each basis kernel [20]. The no dense solutions are more interpretable because they have more zeros and allow to relate the values of the weights with the importance of each kernel to the model. In these algorithms we say the weights are in these norms if:

- L_1: $\Delta_1 = \{d = (d_1, .., d_M) | d_i \in \mathbb{R}^+, ||d||_1 \leq 1\}$
- L_p: $\Delta_p = \{d = (d_1, .., d_M) | d_i \in \mathbb{R}^+, ||d||_p \leq 1\}$

where: Δ_p is the set of weights in the norm regularizations p.

2.1.2 Validation methods

Validation methods are a way to give a perspective about how a certain model will perform when it takes new data that was never seen before. The simplest method is holdout. It divides the data in two subsets: training and test where, commonly, the test set is smaller than the train set. A common way to make this split is to divide the original data in 80% for training and 20% for test. Other method is the k-fold cross validation, where the data is divided randomly in k sets of rows. Training is performed k times using k-1 folds for training and the remaining for test. In this method, the test set will be different for each iteration. Yet another method is bootstrapping where the idea is to perform multiple holdouts but resampling the dataset with replacement in each iteration. When dividing the datasets for validation it is common to keep the same distribution of classes among the folds for the training an test sets. Models are assessed using a performance metric that can be any of the ones discussed in Section 2.1.1. When using cross-validation, evaluating performance can be tricky [8]. The best way of calculating the metrics is using macro-averaging, which is to aggregate the results of confusion matrices produced per fold for the test set and calculate the metrics from the aggregation of those numbers. When training models it may be necessary to choose one among several parameters for the algorithms. Therefore, an internal training-test is performed, **tuning**, which helps choosing the parameters that are best suited for the training set. A model is then built using these best parameters and tested on the test set to produce the final model performance.

2.2 Kernel

Let X be a non empty set. A function $k : X \times X \mapsto \mathbb{R}$ is called a kernel [7].

Kernels are used in the context of this work to map the data to higher dimensional spaces without explicitly calculating them. Some common utilized kernels are the Radial Basis Function (RBF) kernel, the Polynomial kernel and the linear kernel, defined as:

- RBF: $k(x, x') = e^{-\frac{||x-x'||^2}{2\sigma^2}}$ where $sigma$ is a parameter chosen at train.

- Polynomial: $k(x, x') = (x^T x' + c)^d$
- Linear $k(x, x') = < x, x' >$,where $< x, x' >$ is the inner product between the vector x and x'.

2.3 Positive definite function

A symmetric function $k : X \times X \mapsto \mathbb{R}$ is positive definite [7] if $\forall (a_1, a_2, ..., a_n) \in \mathbb{R}^n, \forall (x_1, x_2, ..., a_n) \in X$,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0$$

For the methods we present in this work, the kernel used must be positive definite.

2.4 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a model that was proposed by Vapnik [5] as a binary classifier. The main idea of SVM is to draw a hyper-plane where the distance of the support vectors of each class is maximal. Support vectors are instances in the dataset where the distance to the hyperplane is minimal as shown in Figure 2.1. That method will make use of a kernel function in its calculations.

The SVM algorithm uses mathematical optimization to train the classifier as follows:

The primal definition of the optimization problem is:

$$\begin{aligned} \underset{w,b,\zeta}{\text{minimize}} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to} \quad & \forall i \; y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i \\ & \forall i \in 1, .., n \; \zeta_i \geq 0 \end{aligned} \tag{2.1}$$

where w is a vector of weights given to each variable of the dataset, y_i is the value of the class in the row i and C is the smoothing factor which allows a certain degree of classification error.

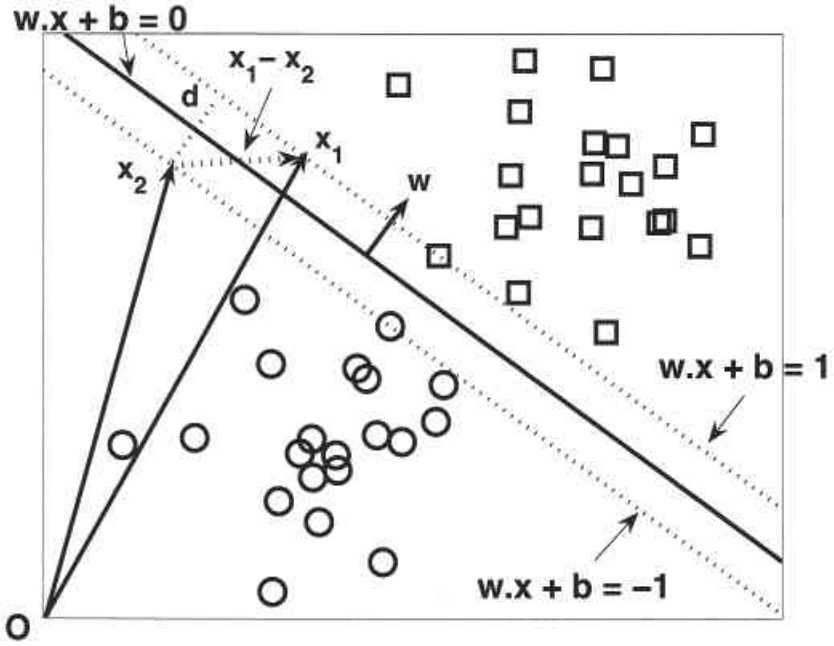


Figure 2.1: SVM illustration: in this image we are trying to distinguish the circles from the squares (two classes). The SVM tried to fit the best line (solid) that maximizes the distance between classes using the support vectors and auxiliary lines, the margins (traced lines). Where $d = \frac{1}{||w||}$ as in the primal Equation 2.1 (Steinbach's book [28])

The dual definition of the optimization problem is:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \forall i \in 1, .., n \; 0 \leq \alpha_i \leq \frac{1}{2n\lambda} = C \end{aligned} \tag{2.2}$$

where y_i is the value of the class in row i, x_i is the feature vector of the row i and α_i is the weight given to the row i of the dataset. n is the number of rows (instances) in the dataset.

The primal solution (Equation 2.1) uses weights to each feature of the dataset and calculates explicitly the high dimension $\phi(x)$. That is something that we would need if you want to solve non linear problems. As shown in Figure 2.1, the way the SVM solves the problem is creating a line that divides the classes. This does not work if the data is not linearly separable according to the classes. The way the SVM solves that is by mapping the features to a higher dimension where it is possible to linearly divide the classes with a hyperplane. But we don't want to explicitly calculate that new dimension because of its computational complexity as we would need to project each feature to that dimension and calculate the weight there. Fortunately, we can use the dual formulation where, instead of mapping each feature to another dimension, we perform all the products in their original space and only map to another dimension after performing the

products.

The dual problem (Equation 2.2) assigns a weight to each row of the dataset. As the method works with support vectors, the complexity is reduced. In order to project features to a higher dimension the "kernel trick" explained in the last paragraph is used. The "kernel trick" is the way of separating the features in a higher dimensional space without needing to calculate it, using a kernel function.

Both optimization formulations (primal and dual) can be solved using a quadratic solver.

Notice that SVM is also capable of solving multiple class problems by using the one vs one or the one vs all nethods, which involves training multiple models with subsets of the dataset. SVM has a variation to solve regression problems.

2.5 Multiple kernel learning

Multiple Kernel Learning (MKL) is a machine learning technique where multiple kernels are combined to produce a single classifier. The base learner can be any kernel-based learning algorithm. In this work we use the SVM classifier with multiple kernels. The base learner uses a kernel in its computation of the prediction (Classification/Regression). An $K\eta$ kernel is computed as a combination of other kernels as shown below:

$$K\eta(x, x') = f\eta(\{K_m(x^m, x'^m)\}_{m=1}^{P}|\eta)$$

where $f\eta$ is a function which combines the P kernels in the kernel $K\eta(\cdot, \cdot)$ with η being the vector of weights associated with each kernel [9]. This approach is similar to an assemble learning algorithm as it creates a model by using multiple simple models to learn and predict. In our case we will train and predict using a set of functions (kernels), instead of models.

MKL can be used to solve two main kinds of problems:

1. Multimodal problems: this is achieved by assigning different sets of kernels to the variables of each modality. This method could be used when our data possess different types of data, for example text and video. But this also can be used when we intend to extract new features from the data and use a different set of kernels to address that data.

2. Problems that an SVM is not capable of solving for using a single kernel: as an SVM only uses a single kernel, it is possible that for every combination of parameters in the kernels: linear, polynomial and RBF, the method is not expressive enough to solve the problem. That can be improved with MKL by using a combination of kernels but this time each kernel uses every feature in the dataset. The objective of this is to reach an optimal kernel to solve the problem we are addressing [20].

2.6 Genetic algorithm

The genetic algorithm is inspired by the natural selection in nature, and is used to optimize a given function [13]. The algorithm starts with a set of solutions to the problem, where a solution is represented as a string over a finite alphabet [23]. A fitness function measures how good an individual is to solve the problem. Greater fitness values are from better individuals. This function is normally formulated dependent of the problem we want to solve. This algorithm is by nature iterative. Each iteration is also called a generation because after an iteration, the set of solutions (population) will create a new one that will be passed to the next generation. In each generation, a set of operations is performed: selection, crossover and mutation. The selection is used to choose the parents that will give origin to new children for the next generation. This choice is typically made having in consideration the fitness of each individual. For example, choosing parents with high fitness probability. After selecting two parents, the crossover operation is applied, by choosing an index in the string that represents the individual and concatenating one of the parents's prefix with the second parent's suffix. This operation is applied such that the two parents produce two offsprings. Usually, the new generation keeps the same number of individuals in the next generation's population, by evaluating the fitness of the new population and discarding the less fit ones with a given probability. The mutation is an operation that can occur in the individuals with low probability. It typically changes the value of a character of the individual at some given index. This is made to increase the diversity of solutions which could help to achieve better fitness scores in the next generations. Algorithm 1 illustrates these main steps. In this algorithm, only one child is created from two parents.

Algorithm 1 genetic algorithm

```
1: population ← initialPopulation()
2: while Termination criterion not reached do
3:    newpop ← []
4:    fit ← fitnessCalculation(population) {Gets the fitness values from population}
5:    for  j from 1 to population size do
6:       parent1 ← Selection(population, fit) {Select first parent based on fitness}
7:       parent2 ← Selection(population, fit) {Select second parent based on fitness}
8:       child ← Crossover(parent1, parent2) {Generate a descendant from parents}
9:       if random > mutationProbability then
10:         child ← Mutation(chid) {if probability of mutation is reached mutate}
11:      end if
12:      newpop.add(child)
13:   end for
14:   population ← newpop
15: end while
16: return best(population) {Return the best individual}
```

2.7 Other Machine Learning algorithms

Other algorithms that will be used at this work are the K neareast neighbours (KNN), the decision tree, the neural network, the naive Bayes and the random forest. We will briefly explain each of them.

KNN for some given number k. Given a new example, the k training examples that have the input features closest to that example are used to predict the target value for the new example. The prediction could be the mode, average, or some interpolation between the prediction of these k training examples, perhaps weighting closer examples more than distant examples. For this method to work, a distance metric is required that measures the closeness of two examples. First, define a metric for the domain of each feature, in which the values of the features are converted to a numerical scale that is used to compare values. Suppose $X_i(e)$ is a numerical representation of the value of feature X_i for the example e. Then $(X_i(e_1) - X_i(e_2))$ is the difference between example e1 and e2 on the dimension defined by feature X_i. The Euclidean distance, the square root of the sum of the squares of the dimension differences, could be used as the distance between two examples. One important issue is the relative scales of different dimensions; increasing the scale of one dimension increases the importance of that feature. [22].

A decision tree represents a function, in a form of a tree that takes as input a vector of attribute values and returns a "decision" – a single output value. A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input features, A_i, and the branches from the node are labeled with the possible values of the attribute, $A_i = v_{ik}$. Each leaf node in the tree specifies a value to be returned by the function.[23]

Neural networks are a popular target representation for learning. These networks are inspired by 1the neurons in the brain. Neural networks have had considerable success in low-level reasoning for which there is abundant training data, such as for image interpretation, speech recognition and machine translation. There are many different types of neural networks. Let's consider feed-forward neural networks. Feed-forward networks can be seen as a hierarchy consisting of linear functions interleaved with activation functions. Neural networks can have multiple input features and multiple target features. These features are all real valued. Discrete features can be transformed into indicator variables or ordinal features. The inputs feed into layers of hidden units, which can be considered as features that are never directly observed, but are useful for prediction. [22]

Naive Bayes model is a common Bayesian network model used in machine learning and is graphically represented by a tree with only a root and the features of the dataset all linked only to the root. In this model, the "class" variable C (which is to be predicted) is the root

and the "attribute" variables X_i are the leaves. The model is "naive" because it assumes that the attributes are conditionally independent of each other, given the class. Once the model has been trained, it can be used to classify new examples for which the class variable C is unobserved. With observed attribute values $x_1, \cdots, x_n$, the probability of each class is given by $P(C|x_1, \cdots, x_n) = \alpha P(C) \prod_i P(x_i|C)$. A deterministic prediction can be obtained by choosing the most likely class [23].

The random forest algorithm works by creating many decision trees and training each one in a random set of features of the original data. In order to predict new instances, it runs each decision tree and outputs the mean (for regression tasks) or mode (for classification tasks) results of all trees. In that case, it works as an ensemble model, where it combines the output predictions of various base models. That is one of the best models used in practice as, in some cases, it is capable of avoiding overfitting.

Chapter 3

State of the art

Many works have been developed in the area of MKL. Some of them approach the problem studying how to improve complexity of the algorithms, e.g., [12]. Others, more empirical, propose different methods for learning weights for the kernels, e.g., [20]. Gonen and Alpaydin [9] propose a set of components that may vary in an MKL algorithm. These are:

- Learning method
- Functional form
- Target function
- Training method
- Base learner

For the Learning method they describe forms of training an MKL algorithm by choosing weights for the kernels or adding kernels in boosting method, which is the idea of adding kernels to a combination of kernels until that combination doesn't improve anymore. Other choice is not using weights at all as in fixed rules (e.g., summation or multiplication of the kernels).

The Functional form relates with the function that defines how the kernels used will be combined. That could, for example, be the sum of all kernels associated with a weight as in a linear combination method.

The target function is a metric that will be used to measure the performance of a kernel combination. Ideally, we want to maximize that function to achieve a better combination and, consequently, a good model.

The training method is the form which we train the MKL. It could be done in one step, where the training of weights is incorporated in the training of the SVM. This could be achieved by just adding the weights of the kernels to the mathematical formulation of SVM. It could also be done

in two steps, where, first, the weights are obtained by some method, then the resulting kernel for that process is incorporated in the SVM and then a training of its own parameters takes place.

The authors also give a list of base learners commonly used in MKL, among them, the SVM. They also did an experimental work trying some combinations of methods using three datasets.

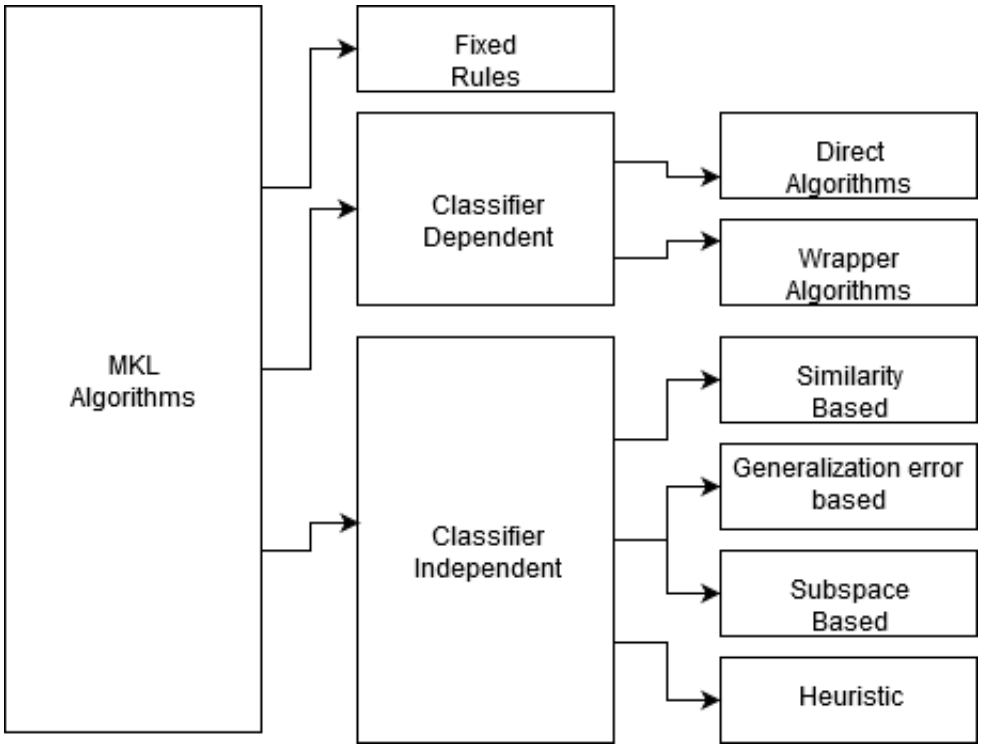


Figure 3.1: Classification of MKL algorithms done by Niazmardi *et al.*

Another work that performed an exploratory analysis of MKL is the one by Niazmardi *et al.* [20]. In their work they established a hierarchy of MKL methods shown in Figure 3.1. Essentially, classifier dependent algorithms are in the category "one step" because the kernels and the classifier are trained at the same time while the classifier independent ones use some other way to optimize the kernels weights separated from the classifier. Niazmardi *et al.* also performed an exploratory review of the norms L_1 and L_2. In the end, they used a linear combination of kernels to address a problem of image classification.

In the last few years, MKL methods have been used in many areas with success. For example, in Psichology, Beyan *et al.* [4] used an MKL method to detect emergent leaders. In the area of security, Anderson *et al.*[1] used MKL to help in the task of malware detection. Fan and Chen used the method to estimate the localization of wifi nodes[6]. Multiple works were done in images as the one by Gu *et al.* [10]. Zhao *et al.* [32] tried to identify regions used to discriminate classes in images, using MKL. In speech emotion recognition, the work by Zha *et al.*[31] uses MKL with many features, and compares with SVM. Gu *et al.*[11] uses a non linear MKL to incorporate spacial and spectral features of hyperspectral images. In 2017, Khattab *et al.*[17] used an adaptive MKL to classify hyperspectral images. Other approaches can be found as in

Huang *et al.*[15] where they used a hierarchical MKL, Pinar *et al.*[21] that uses an MKL based in fuzzy integrals to detect explosive hazards, and Hino *et al.*[12] that used a weights optimization based in entropy.

Gustavo Augusto [2] and Tiago Santos [24] bouth made works with the objective of detecting breast cancer using MKL.

Table 3.1 shows a summary of the MKL used in these papers sorted by year.

reference #	**year**	**kind of combination**	**how weights are obtained**	**comparisons**
[16]	2009	linear	Shogun	SVM only
[29]	2010	linear	Shogun	SVM only
[9]	2011	linear	many	SVM only
[33]	2011	nonlinear	gradient descent	other MKL, SVM
[10]	2012	linear	SVM optimization problem	other MKL, SVM
[12]	2012	linear	minimal conditional entropy	other MKL, i-vectors
[1]	2014	linear	not explained	only SVM
[30]	2014	many	many	other MKL, SVM
[15]	2014	nonlinear	don't use weights	Neural Networks
[2]	2014	linear	matlab implementation	SVM
[19]	2015	handles missing data	-	other MKL
[32]	2016	linear	gradient descent	SVM only
[4]	2016	nonlinear	gradient descent	RFLA and SVM
[21]	2016	linear	Genetic algorithm and fuzzy integrals formulation	other MKL
[11]	2016	nonlinear	projection based gradient descent	SVM and MKL
[31]	2016	linear	SVM optimization problem	SVM only
[6]	2016	linear	SVM optimization problem	SVM only
[18]	2017	linear	not explained	SVM, KNN
[17]	2017	linear	gradient descent	changing parameters only
[24]	2017	linear	matlab implementation	SVM
[20]	2018	linear	many	other MKL, SVM
[25]	2018	Detection of evolving concepts in non-stationary data streams	-	SVM, other works of same topic

Table 3.1: Other works involving MKL

As shown in Table 3.1, most works compare MKL with other MKL methods or with SVMs. Only one work compares MKL with a neural network. Besides, very few works discuss about execution times to run their experiments. In this work, we implement an MKL method using a genetic algorithm to learn the kernels weights. We also perform a comparison of this MKL method with various other classification models regarding quality and execution times.

Chapter 4

Evaluating MKL

In this chapter we describe our experimental methodology and datasets used. We start by discussing about some MKL solutions available and our choice of implementing our own. Next, we develop our genetic-based algorithm. We also describe the datasets used, validation method, evaluation metrics and methodology employed to run the other machine learning models used in this work.

4.1 SVM and MKL

Our base learner is an SVM as this is used in all works we know of that implement MKL. The SVM algorithm, as explained in Section 2, requires the use of a kernel to achieve its objective. The parameters that can be changed in this algorithm are C, the size of the margin, and the kernel dependent parameters: *gamma* for the radial basis function (RBF) kernel and the degree of the polynomial for the polynomial kernel.

MKL is an algorithm that frequently uses the SVM as a base learner. The difference is that, instead of using a single kernel, MKL combines multiple kernels trying to achieve better results. As explained in the work of Gonen and Alpaydin[9] there are 2 ways of combining kernels: linearly or non-linearly. In this work, we use a linear combination of kernels. The MKL process is performed using the genetic algorithm presented in Section 4.2 to obtain weights to the SVM that will solve the Equation 2.2 to create our model.

Very few software is available that implement MKL. Among them are Shogun [26] and Mklaren [27]. We decided to implement our own MKL method for two reasons: (1) Shogun does not allow to introduce new kernels or combinations other than linear for the kernels; (2) Mklaren can not run for all datasets we tested. An error message is issued by the function `alignf` implemented in the `cvxopt` python module when solving the optimization problem to find the best set of kernels weights.

4.2 Genetic algorithm

Aiming at reducing the combinatorial nature of MKL, we use a genetic algorithm to select weights for each kernel. The input to this algorithm is a set of kernels weights and the size of the population. This approach was also used by Pinar [21].

The fitness function we used was the mean of 5 holdouts of the F1 measure. For this work, the fitness calculation was performed for all individuals in parallel in order to reduce the execution time. Parents are randomly selected but individuals with higher fitness have a higher probability of producing descendants. Crossover is performed by choosing a random point on the genome and merging the first part of the first parent to the second part of the second parent to create a new individual that will become a member of the next generation. Mutation is performed by simply randomizing a weight from a number between 0 and 1. As we do not know anything about the effect of weights before doing this, we thought a random replacing of the value would be the best approach. For the mutation probability, we chose to have a mutation value that can change during the genetic algorithm. The mutation probability starts with value 0.01 to each individual. If at any point, 3 consecutive generations do have the same best fitness then the mutation probability is increased to 0.05. Otherwise, the probability returns to 0.01. That approach was made to avoid our optimization process to be stuck at a local maximum by introducing diversity in the population. For all experiments, we used the same seed in order to have comparable results. The pseudocode of the method described here is shown in Algorithm 2.

Algorithm 2 Genetic Algorithm as implemented in this work

```
best_individual ← NULL {Saves the best individual from iterations}
population ← initialPopulation() {Saves the best individual from iterations}
for i from 1 to 100 do
    if runtime > 11h then
        return best_individual {Return the best individual}
    end if
    for j from 1 to population size do
        fit ← fitnessCalculation(population[j]) {Gets the fitness values from population in parallel}
    end for
    best_individual ← BEST(best_individual, fit) {saves the best individual between the actual population and the old best individual}
    children ← []
    for j from 1 to population size do
        parent1 ← Selection(population, fit) {Select the parent based on fitness}
        parent2 ← Selection(population, fit) {Select the parent based on fitness}
        child ← Crossover(parent1, parent2) {Generate the new generation from parents}
        if random > mutationProbability then
            child ← Mutation(chid) {if probability of mutation is reached mutate}
        end if
        children ← children.add(child)
    end for
    population ← children
    if fitness is the same for 3 iterations then
        mutationProbability ← mutationProbability + 0.05
    else
        mutationProbability ← 0.01
    end if
end for
return best_individual {Return the best individual}
```

4.3 Software and tools

The code was written in python 2.7 because we started by using the Mklaren tool, which was written in that version of python. We also used the tools available in the `scikit-learn` package to create every model used in this work. We also used the `imbalanced-learn` package to rebalance the data in unbalanced datasets. Finally, we use part of the tool `mklaren` available in GitHub (https://github.com/mstrazar/mklaren) because it was successful in other works [27]. But during our experiences, as mentioned before, we found out that the process of getting

the weights for the kernels implemented with the function `alignf` returned an error from the quadratic optimizer for some of our datasets. We are not sure of the cause as all data is treated in the same way independent of the dataset. The error message appears only for some of them independent of the number of features or rows in our test scenarios. Every test was made in the same machine which has 24 cores, 800 MHz CPU, max 2100 MHz, min 800 MHz and a Random-access memory (RAM) of 128 GBytes. The code used in this work can be found in GitHub (https://github.com/JoseAALC/MKLcomparasion).

4.4 Preprocessing

We first started by removing duplicate rows in the dataset to avoid having the same data multiple times across the samples. Leaving duplicates in the dataset could bias some models and, in some cases, the time complexity increases with the number of samples which could make our model training slower with duplicates. Also, all rows that have missing data present are eliminated.

Then we take each categorical variable and create $N - 1$ variables to each one of the N possible values that indicate if that value is present for that variable(binarization). Converted all categorical variables into multiple boolean variables.

Then the data is balanced if the class with more frequency is at least 60% of all data. In this work, that value is the minimum necessary to consider the data unbalanced. We performed under-sampling as a method to rebalance the data.

Finally, the data is normalized because some models are sensitive to the range of those values, for example, K-Nearest Neighbors.

4.5 Experimental methodology

Before training the models, we preprocess the data. We keep at most 1200 instances in each dataset in order to reduce execution time. The undersampling is performed randomly. A second preprocessing is performed in one of the datasets in order to artificially introduce multimodal characteristics (this is explained later). After preprocessing, each dataset is balanced, if needed, according to the class values using undersampling, as explained before. We then split the dataset in 20% for validation and 80% for training.

This work tests neural networks, decision trees, naive Bayes, random forest, and KNN against our approach of MKL. For tuning, we use 5 fold cross-validation, optimization by F1-score and a grid search method which makes the number of experiences for each model, dependent on the number of combinations of parameters. For KNN we use 1, 2, 3, 5, 7, 9 and 11 nearest neighbours. For the decision trees, we just use the default without tuning. For the neural network, we also do not make any tuning. We just use a hidden layer with 100 nodes as default. For the naive Bayes, we vary the smoothing parameter with the values: $e^{-i}\ \forall i \in [0, 9]$. The random forest is

tuned using 100, 500 and 1000 trees. For the SVM we create 3 models in the tuning process using three kernels: polynomial, RBF and linear. We choose the best from these three as our best SVM. For the linear kernel we vary C in the range $10^i \ \forall i \in [-5, 2]$, for the RBF we also vary gamma in the range $i/\#rows \ \forall i \in [1, 6]$ and for the polynomial kernel we use the same values of C used in the RBF kernel and polynomials of degree 1, 2 and 3.

These parameters result in the number of experiences for tuning the models shown in Table 4.1. As mentioned before, decisions trees (Tree) and Neural Network were not tuned (the reason why we did not tune parameters for the decision trees and neural networks was that they could consume too much time and our time to finish this work was limited).

Table 4.1: Number of experiences during tuning for each algorithm

Algorithm	**Number of training processes during tuning**
Tree	0
KNN	35
Naive Bayes	50
Random Forest	15
SVM linear	40
SVM RBF	240
SVM Poly	120
Neural Network	0
MKL	40

The models are trained to improve the F1-score, but we report other metrics: Accuracy, area under the ROC curve (auroc), Recall and Precision. We also report execution times in order to assess each method's efficiency.

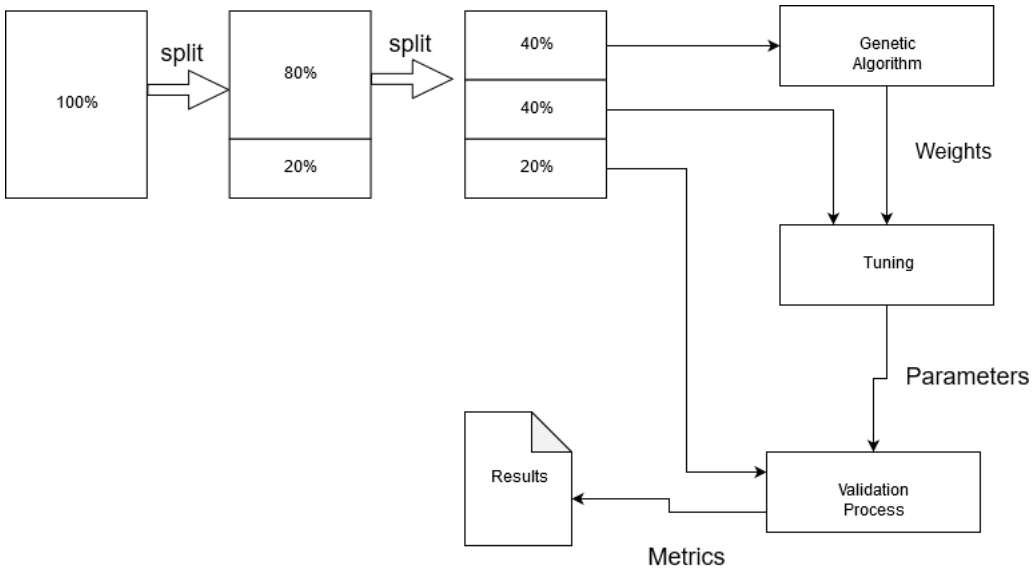


Figure 4.1: Experimental Methodology for our MKL

When we use the genetic algorithm, we split the data in a different way. First, as for the other algorithms, we split the data in 80% training and 20% testing as seen in Figure 4.1. Then we split the training set in half. The first half is used in the genetic algorithm to obtain the weights. In the next step, we use the weights and the other half to find the best C parameter for this kernel combination with these weights. Then we take our test set and our MKL trained with the weights and parameters obtained in previous steps and predict the class for the test. From there we get the metrics that we will show as results in Chapter 5.

Regarding the parameters of our genetic algorithm, we chose a population of 18 individuals and a maximum number of 100 generations. We use only 18 individuals and 100 generations in order to be able to keep the experimental time within this project's time frame. We also limit the runtime of the genetic algorithm to 11h, also because of the time constraints. Regarding the choice of kernels, we use 40 kernels in our experiments with each kernel randomly selected as linear, RBF or polynomial. If it is polynomial or RBF the parameters for those kernels are randomly selected from the same set of parameters we gave to the SVM. Notice that because of the nature of our method we do 5-holdouts for each member of the population in each iteration which conduces to a total of **9000 experiences** to conclude the genetic algorithm.

4.6 Used Datasets

We use multiple datasets as presented in Table 4.2, with varied number of features, instances and types of features. The column "#rows" corresponds to the original number of instances of each dataset. The "Final #rows" corresponds to the number of rows after preprocessing the data. Likewise, column "#features" corresponds to the initial number of features and "Final #features" corresponds to the number of features after preprocessing. Column "Type" indicates the kind of features for each dataset. Most datasets were chosen because they commonly appear in other works. Their origin is also shown in the table.

Table 4.2: datasets used

Name	**Type**	**#features**	**#rows**	**Final #features**	**Final #rows**	**Source**	**#pos**	**#neg**
heart	numeric	13	303	13	303	UCI	139	164
pendigits	numeric	16	10992	16	1200	UCI	705	495
adult	mixed	14	32561	90	636	UCI	318	318
mushrooms	categorical	22	8124	94	1200	UCI	628	572
fashion	numeric	784	70000	784	1199	Kaggle	607	592
hiragana	numeric	784	232365	784	1200	Kaggle	498	702
gisette	numeric	5000	6000	5000	1200	UCI	584	616
volcanoes	numeric	12100	7000	12100	360	Kaggle	180	180
volcanoes2	numeric	24200	7000	24200	360	This work	180	180

heart was taken from the UCI repository (https://archive.ics.uci.edu/ml/datasets/heart+Disease) the processed cleveland. Each row corresponds to a patient with 13 annotated features. The target variable indicates if the patient has a heart disease or not.

pendigits is a dataset that contains a representation of handwritten digits with 16 features that was used in the work of Gonen an Alpaydin [9] and is in the UCI repository (https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits).

The **adult** dataset was also taken from the UCI repository (https://archive.ics.uci.edu/ml/datasets/Adult). This contains demographic data where each line is a person and the classification task is to determine if that person gains over $50,000$ or not.

The **mushrooms** dataset was taken from UCI (https://archive.ics.uci.edu/ml/datasets/mushroom) and it has 22 features annotated from different types of mushrooms. The target variable is if it is poisonous or not. This dataset is very easy to model and classification metrics are usually very high. We use it for comparison, but also to validate our MKL method.

The **fashion** dataset was created by using the training dataset from MNIST fashion on kaggle (https://www.kaggle.com/zalando-research/fashionmnist). This has images of clothes represented as 28 x 28 images. We created a binary classification problem by predicting if a piece of clothing covers the upper body (classes 0,2,3,4 and 6 in the original dataset) or not.

hiragana is a dataset where we used the training set from Kuzushiji MNIST from kaggle (https://www.kaggle.com/anokas/kuzushiji), which has the objective of classifying the 49 Japanese handwritten hiragana characters. The characters are represented by 28 x 28 images. We transformed this in a binary classification problem and predict if the character belongs to the first 25 characters group or not.

gisette is a dataset containing 28 x 28 images of handwritten numbers with highly confusable digits '4' and '9'. New features were created as products of the images pixels to plunge the problem in a higher dimensional feature space (https://archive.ics.uci.edu/ml/datasets/Gisette).

The dataset **volcanoes** contains a set of 110×110 images. The target variable indicates if the image is or is not a volcano. This dataset was taken from a kaggle challenge (https://www.kaggle.com/fmena14/volcanoesvenus).

The **volcanoes2** is the same as volcanos, but we added 110×110 pixels resulting from the application of a sobel filter to the image, which is a filter that has the objective of detecting edges in an image. That was done with the intention of showing the capacity of solving multimodal problems by MKL. Sobel was used just to make the experiment simple and because we did not have time to apply a sophisticated feature extraction or use actual multimodal data because we explored the datasets common in the literature what consumed too much time and we didn't have time to add additional datasets.

Yet about **volcanoes2**, we performed an extra experiment with a different combination of kernels trying to explore multimodality capabilities of MKL. We used 20 kernels instead of 40 for

the data from volcanoes and other 20 kernels using only the features we created. This method is called $MKL20\_20$ in the next chapter.

Chapter 5

Results and Discussion

In this chapter we show the results of our experiences as described in Chapter 4 and discuss the reasons that could have influenced the behaviour of our model. We show rounded values of the metrics that we used to validate our models.

We start by comparing the performance of the algorithms in Figure 5.1. The MKL20_20 is a version of the **volcanoes2** where we use 20 kernels to the first half of the features and the other 20 to the other half.

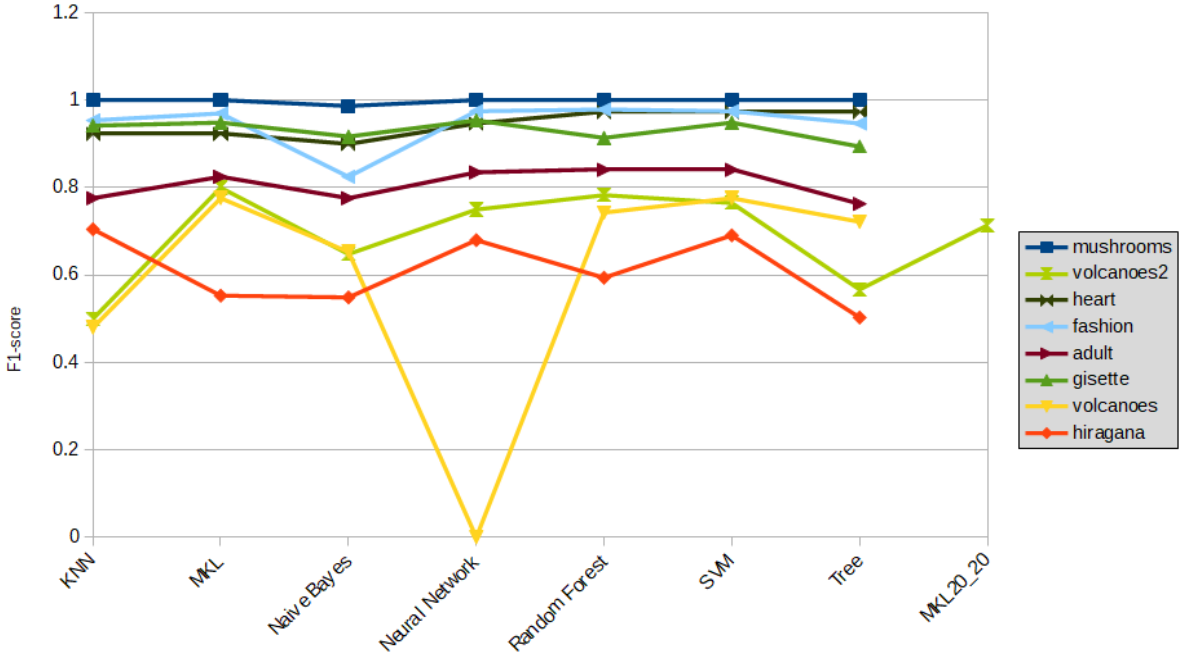


Figure 5.1: F1-scores to each algorithm in each dataset

Regarding the F1-score, in Figure 5.1, the datasets **mushrooms**, **gisette**, **heart** and **fashion** are the easiest to classify with all models. **fashion** did not perform so well with the naive Bayes model. **hiragana** seems to be the most difficult dataset. In fact, the choice of binarizing the

classification task made the discrimination of classes more difficult, because some japanese characters in the first 25-length size have similarity with the second half, and some characters in the same set are very dissimilar. This causes confusion when trying to learn patterns. It is also possible that **hiragana** had the worst performance because it is the one that suffers greater reduction in number of instances after preprocessing. We use only 1200 rows from the original dataset of 232,365 lines which is less than 1%. This reduction could have limited the capacity of the models to learn well from this dataset.

We can not consider difficulty to generalize the model as a function of the number of features since **fashion** and **hiragana** have the same number of features and achieve very different results. The kind of data also seems to have no influence because **gisette** is numeric, **mushrooms** categorical and **heart** is a numeric dataset and all had good performance. The neural network had a value of zero for the F1-score in the **volcanoes** dataset. The reason for that is that the model classifies every test example as belonging to the same class (negative).

Our MKL model was capable of, at least, keeping the same scores in relation with other models, but, as we show later, at a very high computational cost.

Next, we analyse each dataset individually.

Table 5.1: **heart** results

algorithm	**accuracy**	**F1**	**rocauc**	**precision**	**recall**	**time**
Tree	0.754	0.706	0.746	0.783	0.643	1s<
KNN	0.770	0.741	0.766	0.769	0.714	1s<
Naive Bayes	**0.836**	**0.821**	**0.835**	0.821	0.821	1s<
Random Forest	0.803	0.800	0.807	0.750	**0.857**	2s
SVM	**0.836**	**0.821**	0.835	0.821	0.821	6s
Neural Network	0.820	0.807	0.820	**0.793**	0.821	1s<
MKL	0.770	0.767	0.774	0.719	0.821	8 min 43s

In the **heart** dataset (results shown in Table 5.1), the scores of almost all models were good. The best models were the SVM and Naive Bayes. But Naive Bayes reached the highest rocauc, neural network the highest precision and Random forest provided the best recall.

Table 5.2: **pendigits** result

algorithm	**accuracy**	**F1**	**auroc**	**precision**	**recall**	**time**
Tree	0.938	0.947	0.936	0.950	0.943	1s<
KNN	**0.992**	**0.993**	**0.991**	**0.993**	**0.993**	1s
Naive Bayes	0.775	0.806	0.771	0.818	0.794	1s<
Random Forest	0.975	0.979	0.976	0.986	0.972	3s
SVM	0.975	0.979	0.974	0.979	0.979	43s
Neural Network	0.975	0.978	0.977	**0.993**	0.965	2s
MKL	0.975	0.979	0.974	0.979	0.979	1h 52 min 5s

In Table 5.2, **pendigits** obtained the best F1-score with KNN. This dataset has only 16 variables, but it is relatively easy to classify. MKL behaves as well as SVM for this dataset with little scope for improvements.

Table 5.3: **adult** results

algorithm	**accuracy**	**F1**	**auroc**	**precision**	**recall**	**time**
Tree	0.758	0.763	0.758	0.746	0.781	1s<
KNN	0.797	0.776	0.797	**0.865**	0.703	2s
Naive Bayes	0.797	0.776	0.797	**0.865**	0.703	1s<
Random Forest	**0.844**	**0.841**	**0.844**	0.855	**0.828**	3s
SVM	**0.844**	**0.841**	**0.844**	0.855	**0.828**	44s
Neural Network	0.836	0.835	0.836	0.841	**0.828**	3s
MKL	0.828	0.825	0.828	0.839	0.813	1h 44 min 34s

Starting with Table 5.3, in the **adult** dataset, SVM and Random Forest were the best models and our method of MKL was the 4th best model. That could be happening because of the random choice of kernels and parameters to each kernel, as opposite to the SVM, where the single kernel used is chosen by the tuning process. The MKL could have had bad luck and as most models already have an accuracy and F1 high, the genetic-based MKL struggled to improve. Naive Bayes and KNN are the models with best precision which means that it reduced better than any other model the number of false positives.

Table 5.4: **mushrooms** results

algorithm	**accuracy**	**F1**	**auroc**	**precision**	**recall**	**time**
Tree	1.000	1.000	1.000	1.000	1.000	1s<
KNN	1.000	1.000	1.000	1.000	1.000	8s
Naive Bayes	0.988	0.987	0.988	0.974	1.000	1s<
Random Forest	1.000	1.000	1.000	1.000	1.000	3s
SVM	1.000	1.000	1.000	1.000	1.000	1 min 58s
Neural Network	1.000	1.000	1.000	1.000	1.000	3s
MKL	1.000	1.000	1.000	1.000	1.000	4h 19 min 50s

The dataset **mushrooms** is widely used in the literature of machine learning as an easy dataset. And the scores shown in Table 5.4 demonstrate that our MKL model also is capable of achieving maximum scores for all metrics.

Table 5.5: **fashion** results

algorithm	**accuracy**	**F1**	**auroc**	**precision**	**recall**	**time**
Tree	0.946	0.947	0.946	0.936	**0.959**	1s<
KNN	0.954	0.954	0.955	0.974	0.934	55s
Naive Bayes	0.838	0.825	0.839	0.911	0.754	3s
Random Forest	**0.979**	**0.979**	**0.980**	**1.000**	**0.959**	7s
SVM	0.975	0.975	0.975	**1.000**	0.951	13 min
Neural Network	0.975	0.975	0.975	**1.000**	0.951	23s
MKL	0.971	0.970	0.971	**1.000**	0.943	11h>

The **fashion** dataset, with our formulation (top clothes versus bottom clothes) is an easy classification task, since image pixels show a very good separation between top and bottom parts of clothing. For that reason, almost all models managed to achieve a good classification, with exception of the naive Bayes. Random forest was probably better because our formulation made the prediction dependent of a smaller number of variables. Therefore a set of decision trees could easily decide the class of each image. This did not happen to the naive Bayes model because all variables are conditionally independent on each other given the class, which clearly is not the case in this dataset. MKL has a competitive quality performance being the third best model.

Table 5.6: **hiragana** results

algorithm	accuracy	F1	auroc	precision	recall	time
Tree	0.613	0.503	0.592	0.540	0.470	1s<
KNN	**0.762**	**0.705**	**0.751**	**0.731**	0.680	55s
Naive Bayes	0.521	0.549	0.546	0.452	**0.700**	3s
Random Forest	0.704	0.594	0.678	0.693	0.520	7s
SVM	0.754	0.691	0.741	0.725	0.660	15 min 57s
Neural Network	0.742	0.680	0.730	0.702	0.660	35s
MKL	0.650	0.553	0.631	0.591	0.520	11h>

According to our formulation, **hiragana** is a difficult dataset. Recall we transformed the class variable in two different values: characters that belong to the first 25 group of the 49 and characters that belong to the second half group (24 characters). This proved to be a not so good decision as results in Table 5.6 show, but served our purpose of having a binary class variable. MKL was the 4th best classifier after KNN, pure SVM and Neural Network. Trees performed very badly in this dataset for the set of fixed parameters we used (notice that we did not tune parameters for the decision trees). The MKL only was capable of train for 40 generations which could be bad for the performance quality (40 generations may not be enough to improve the F1-score). Regarding accuracy, Random Forest was capable of surpassing MKL and naive Bayes has the worst accuracy (almost random classification). On the other hand, it has the best Recall.

Table 5.7: **gisette** results

algorithm	accuracy	F1	auroc	precision	recall	time
Tree	0.900	0.894	0.899	0.927	0.863	2s
KNN	0.942	0.942	0.942	0.919	**0.966**	6 min 7s
Naive Bayes	0.925	0.917	0.923	**0.990**	0.855	20s
Random Forest	0.921	0.914	0.919	0.971	0.863	11s
SVM	0.950	0.949	0.950	0.949	0.949	1h 54 min 12s
Neural Network	**0.954**	**0.954**	**0.954**	0.942	**0.966**	55s
MKL	0.950	0.949	0.950	0.941	0.957	11h>

In **gisette**, Table 5.7, the model that achieved the best score was the neural network. Notice that our MKL model has the same F1-score as the SVM, but better Recall. This may be useful if we intend to optimize for true positives. The MKL was only capable of reaching 5 generations in the genetic algorithm (it stopped due to time exceeded). This may have prevented the method from obtaining better weights. The best precision was obtained by the naive Bayes model.

Table 5.8: **volcanoes** results

algorithm	accuracy	F1	auroc	precision	recall	time
Tree	0.722	0.722	0.722	0.722	0.722	3s
KNN	0.639	0.480	0.639	**0.857**	0.333	1min 18s
Naive Bayes	0.528	0.653	0.528	0.516	**0.889**	11s
Random Forest	0.750	0.743	0.750	0.765	0.722	11s
SVM	**0.792**	**0.776**	**0.792**	0.839	0.722	29 min 57s
Neural Network	0.500	0.000	0.500	0.000	0.000	26 min
MKL	**0.792**	**0.776**	**0.792**	0.839	0.722	11h>

Table 5.8 shows that in the **volcanoes** dataset MKL was capable of being as good as the SVM and these two are the best models. But even with SVM achieving the best accuracy and F1-score, the KNN got better precision and naive Bayes better recall. In this dataset the neural network seems to classify everything as negative as the recall and precision are 0.

Table 5.9: **volcanoes2** results

algorithm	accuracy	F1	auroc	precision	recall	time
Tree	0.597	0.567	0.597	0.613	0.528	5s
KNN	0.556	0.500	0.556	0.571	0.444	2 min 40s
Naive Bayes	0.486	0.648	0.486	0.493	**0.944**	23s
Random Forest	0.792	0.783	0.792	0.818	0.75	16s
SVM	0.778	0.765	0.778	0.813	0.722	1h 2 min 26s
Neural Network	0.750	0.750	0.750	0.750	0.75	11 min 56s
MKL	**0.806**	**0.800**	**0.806**	**0.824**	0.778	11h>
MKL20_20	0.750	0.735	0.750	0.781	0.694	11h>

volcanoes2 is a dataset that is produced by applying a sobel filter to the original volcanoes. Results in Table 5.9 show that MKL is the best model. There are two possible reasons for that: (1) the features added create confusion in the other models and (2) the weights given to the kernels improve the performance on this dataset when compared with other models and to MKL on the original volcanoes. The results of MKL were obtained with only 10 generations in the genetic algorithm which means that it is possible to reach improvements with fewer than the 100 generations we chose.

MKL20_20 is an experience where we use 20 kernels to the original volcanoes data and other 20 to the pixels extracted from the sobel filter. That experience shows worse results than freely training a combination of 40 kernels.

Table 5.10: The number of iterations of a max of 100 that the MKL did in a max of 11h time span for each dataset

dataset	**number of iterations before timeout**
heart	100
pendigits	100
adult	100
mushrooms	100
fashion	45
hiragana	40
gisette	5
volcanoes	17
volcanoes2	10
volcanoes2 with MKL20_20	19

Table 5.10 shows the number of generations the genetic algorithm was able to run in a limited time of 11h and with a limit of 100 iterations at most. Six datasets out of 10 can not finish the 100 iterations before timing out. Some of them take a very long time to run reaching only as few as 5 iterations even having a small number of features (e.g., **gisette**). The datasets with larger number of features all failed to reach the 100 iterations. Some metric values could be better if we allowed the experiments to run longer. These results play against using MKL as a classifier. For some of the easiest to learn datasets (**mushrooms**, **pendigits** and **heart**), MKL did not outperform any of the other models and took much longer to run. For another easy to learn dataset, **fashion**, it took more than 11 hours to run without outperforming any of the other models. In order to better investigate the behavior of our MKL implementation, we plotted the behavior of the genetic algorithm showing the changes in F1-score across generations for some datasets (plots are ordered from worst to best individuals of each run and not according to iterations).

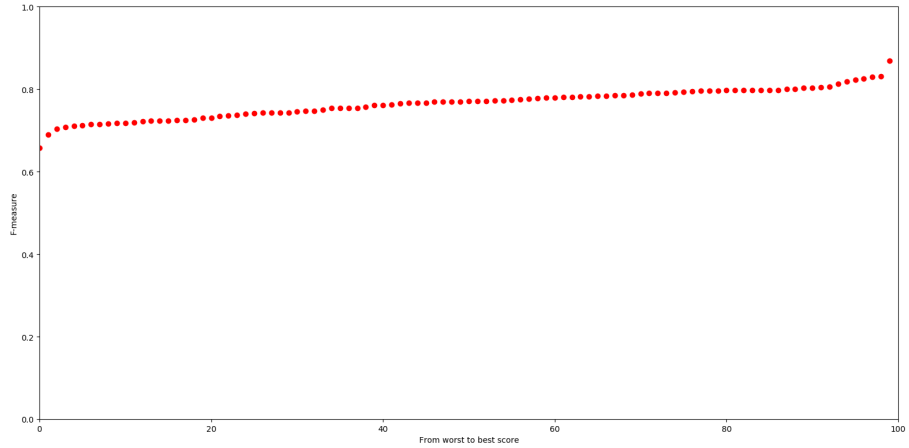


Figure 5.2: Best fitness in each generation in the **heart** dataset

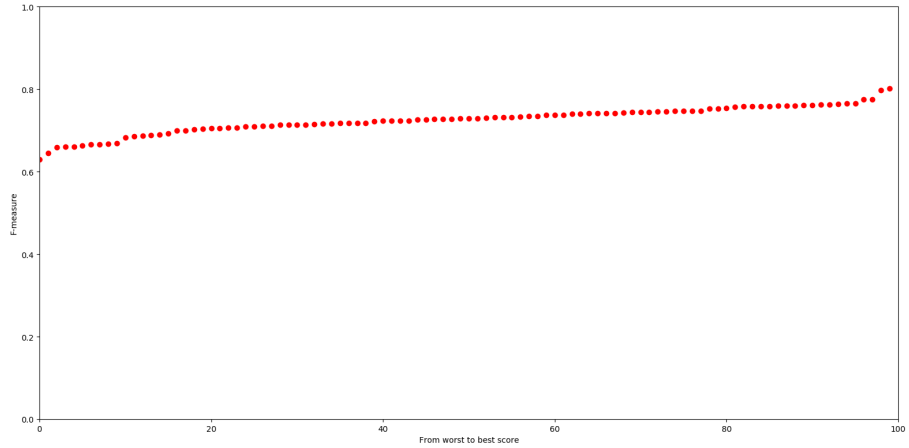


Figure 5.3: Best fitness in each generation in the **adult** dataset

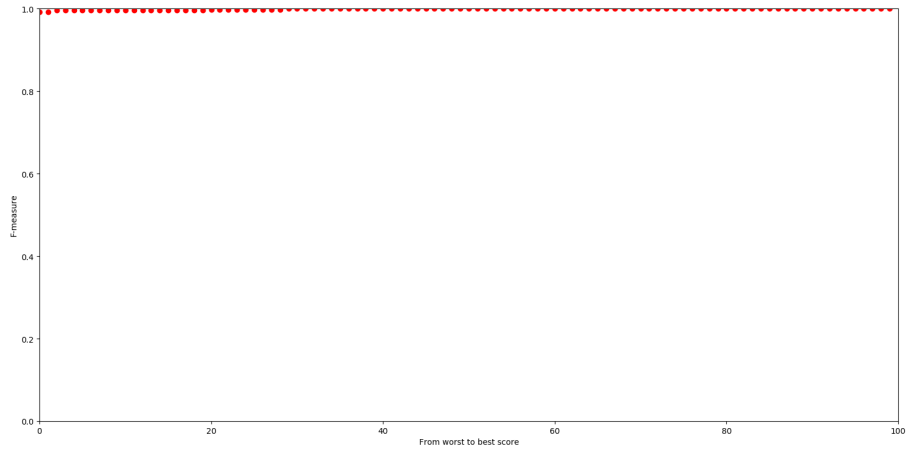


Figure 5.4: Best fitness in each generation in the **mushrooms** dataset

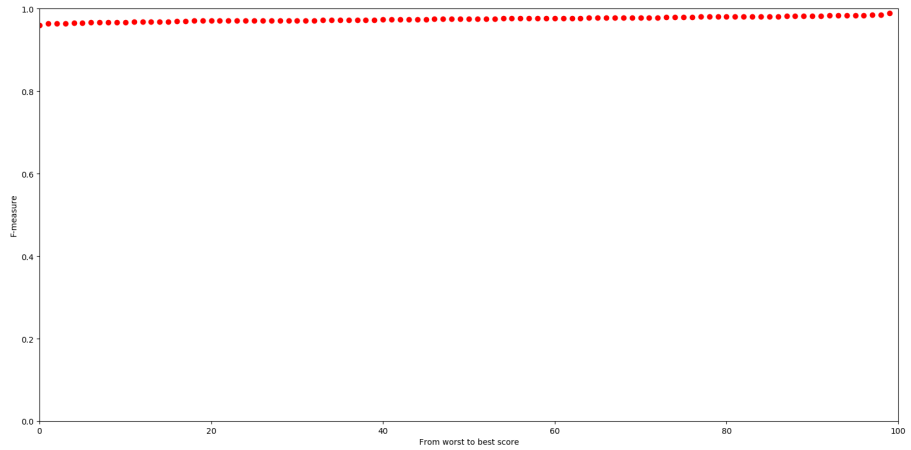


Figure 5.5: Best fitness in each generation in the **pendigits** dataset

As shown in Figures 5.2, 5.3, 5.4, and 5.5, the fitness of the genetic algorithm populations remains almost the same for all the generations in the datasets, with some slight improvement for 5.2 and 5.3. They are particularly similar for the **mushrooms** and for the **pendigits** datasets, which have similar sizes. One of the reasons for not significantly improving fitness from one generation to another is the use of a small test set when tuning for the best weights. We perform two rounds of tuning for the MKL. The first one to obtain the weights through the genetic algorithm and the second one to train and test the MKL for the best set of weights. We also need to train SVMs/MKL for each new individual of the genetic algorithm population which consumes extra time.

In this work we first intended to experiment with non linear multiple kernel learning

combinations. Because of that we started by choosing not to use Shogun. However, due to time constraints we compared only linear combinations of kernels.

For the datasets used in this work, that have high performance for the other models, it becomes very difficult for our MKL model to match performance and have some improvement. For the ones that are not so easy, our MKL can find some room for improvement, specially the ones that have some source of multimodality as in the **volcanoes2**. Our choices of parameters may not be the best. For example, randomly choosing variables to choose with a set of kernels may not be the best because some of these variables may be of different types. Nevertheless, MKL still remains a difficult classifier due to its prohibitive execution times.

In the work of Gonen and Alpaydin [9]accuracy results for the many MKL versions they tested were between 93% and 97.8%. We also can reach those levels of accuracy. Zhuang *et al.* [33] tested multiple versions of MKL with an SVM base learner and achieved maximum accuracy of 83.6%, which is above our result with MKL (77%). In the adult dataset they get a maximum accuracy of 82.1% which is similar to ours even when we cut part of the data and have a test and train set different. Notice that a direct comparison can not be made given that we had to reduce the sizes of some datasets.

Chapter 6

Conclusion

MKL is a an alternative machine learning algorithm that instead of using a single kernel, may use a combination of various kernels in order to capture differences in subsets of data. For example, one direct use of MKL is to handle multimodal data or variables of several types using a kernel for each subset. MKL has been used in various domains, but its performance has not been studied from the point of view of comparing it with other machine learning methods and studying its execution time performance. Works in the literature report benefits of using MKL, but only comparing the quality of the models produced with various MKL methods or SVM. Derived from these works, there are some MKL implementations available. The most popular and accessible are Shogun and Mklaren. Shogun only works with linear kernel combination and Mklaren fails to produce models to some datasets used in this study. Our own implementation is based on a genetic algorithm where the population is a list of weights to be used to combine kernels. We trained this MKL algorithm and compared results with other machine learning models on a variety of datasets available in the literature. We concluded that MKL, in some cases, can be competitive with other models, but at the expense of a very high computational time. Even reducing the sizes of the lager datasets, MKL takes quite a long time (more than 11 hours) to produce a model.

Some of the datasets used in this work are easy to model, therefore, for those, MKL does not have much room for improvement. For some of the most difficult datasets, MKL either failed to achieve better results than the other models or when it outperformed, it did it at a very high computational time. `volcanoes2` is the dataset where MKL has the best score. We believe that the other classifier models failed to produce a better solution because of the very low ratio between number of instances and number of features (360 rows and 24,200 features). The difficulty in obtaining good-quality model for the datasets do not seem to be related directly with the number of features. In fact, this is discussed in the literature and there is no agreed consensus for what number of features versus number of instances should be used to obtain a good classifier (cf., for example, work by Hua *et al.* [14]). For our datasets, `gisette`, `volcanoes` and `volcanoes2` are the datasets with a larger number of features but `hiragana` (with less features) is the most difficult as no algorithm was capable of reaching scores of 80% in accuracy or

F-measure. The kind of dataset also does not seem to influence the generation of good classifiers. We have datasets with features of various types and the easiest one is `mushrooms` with only categorical features.

As SVM and MKL are very time-consuming, we had to reduce the sizes of the datasets and the maximum amount of time to execute. The limitation in time forced MKL to stop before the maximum number of iterations for some datasets. This may have limited the search for better solutions. The idea of using a genetic algorithm to find the best weights introduced complexity, and, consequently, produced very high execution times. However, the method works better than randomly selecting weights to the kernels (we performed a few experiments that have shown that random selection produced bad results). We focused in combining the kernels independently of the kind of variable. One possible modification could be to group variables with similar distribution and types and use the same kernel for those. This would perhaps be more convenient to solve multimodal problems. In fact, some of the works discussed in the literature review chapter use hyper spectral images, speech, among other kinds of data. Yet another limitation of this work is that we did not explore combinations of kernels other than linear or multi-class problems.

Our motivation to use genetic algorithms was inspired by the work of Pinar *et al.* [21]. They applied a genetic-based MKL algorithm to explosive hazards detection using data from a ground penetration radar. Results of this work show that the best method used was the genetic algorithm with an L_1 normalization and NAUC of 0.504 where NAUS is the area under the normalized ROC curve. Execution times are not reported and no comparison is made with decision trees, naive bayes, KNN or any other model. Gonen and Alpaydin [9] published results about several variations of MKL on a limited number of datasets. Execution times are not reported and no comparison with other ML methods is performed. Although we share some of the same datasets, results of both papers are not directly comparable with our work because we had to reduce the number of instances and rebalance the datasets.

Summarising, our main conclusion with this work is that, when compared with other less complex methods, MKL can be competitive in terms of classification quality, but at the cost of a high computational cost. Therefore, for the datasets we used in this work, we recommend using other classifiers. Nevertheless, we believe that further work can be done, specially due to the limitations of this work. Next, we discuss about some possible paths to follow.

6.1 Future Work

This work left the following gaps that could be filled with the next works:

1. Datasets
    - study how the variation of the number of features and number of instances for the same dataset affects the quality and time performance

- study how groups of variables of the same type can affect the quality and time performance
- study multi-modal datasets
- study multi-class datasets

2. Algorithm

- profiling and optimizing bottlenecks
- parallelization
- study other choices for selecting weights
- study other choices for combining kernels and use combinations other than linear

Bibliography

[1] Blake Anderson, Curtis Storlie, Micah Yates, and Aaron McPhall. Automating reverse engineering with machine learning techniques. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 103–112. ACM, 2014.

[2] Gustavo Barbosa Augusto. Computer Aided Diagnosis for Breast Cancer Detection. Master's thesis, Department of Computer Science, Faculty of Sciences, University of Porto, Porto, Portugal, December 2014.

[3] Yoshua Bengio and Yann Lecun. *Scaling learning algorithms towards AI*. MIT Press, 2007.

[4] Cigdem Beyan, Francesca Capozzi, Cristina Becchio, and Vittorio Murino. Identification of emergent leaders in a meeting scenario using multiple kernel learning. In *Proceedings of the 2nd Workshop on Advancements in Social Signal Processing for Multimodal Interaction*, pages 3–10. ACM, 2016.

[5] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[6] Heng Fan and Zhongmin Chen. Wifi based indoor localization with multiple kernel learning. In *2016 8th IEEE InternatFional Conference on Communication Software and Networks (ICCSN)*, pages 474–477. IEEE, 2016.

[7] Gregory E Fasshauer. Positive definite kernels: past, present and future. *Dolomite Research Notes on Approximation*, 4:21–63, 2011.

[8] George Forman and Martin Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12(1):49–57, 2010.

[9] Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(Jul):2211–2268, 2011.

[10] Yanfeng Gu, Chen Wang, Di You, Yuhang Zhang, Shizhe Wang, and Ye Zhang. Representative multiple kernel learning for classification in hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 50(7):2852–2865, 2012.

[11] Yanfeng Gu, Tianzhu Liu, Xiuping Jia, Jón Atli Benediktsson, and Jocelyn Chanussot. Nonlinear multiple kernel learning with multiple-structure-element extended morphological profiles for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 54(6):3235–3247, 2016.

[12] Hideitsu Hino and Tetsuji Ogawa. An improved entropy-based multiple kernel learning. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1189–1192. IEEE, 2012.

[13] John H. Holland. Genetic algorithms and adaptation. In Oliver G. Selfridge, Edwina L. Rissland, and Michael A. Arbib, editors, *Adaptive Control of Ill-Defined Systems*, pages 317–333. Springer US, Boston, MA, 1984. ISBN: 978-1-4684-8941-5. doi:10.1007/978-1-4684-8941-5_21.

[14] Jianping Hua, Zixiang Xiong, James Lowey, Edward Suh, and Edward R. Dougherty. Optimal number of features as a function of sample size for various classification rules. *Bioinformatics*, 21(8):1509–1515, 11 2004. ISSN: 1367-4803. doi:10.1093/bioinformatics/bti171.

[15] Shian Chang Huang and Lung Fu Chang. Oil price forecasting with hierarchical multiple kernel machines. In *2014 International Symposium on Computer, Consumer and Control*, pages 260–263. IEEE, 2014.

[16] Taichi Joutou and Keiji Yanai. A food image recognition system with multiple kernel learning. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 285–288. IEEE, 2009.

[17] Noha S Khattab, Shaheera Rashwan, Hala M Ebeid, Howida A Shedeed, Walaa M Sheta, and Mohamed F Tolba. Adaptive multiple kernel self-organizing maps for hyperspectral image classification. In *Proceedings of the 8th International Conference on Computer Modeling and Simulation*, pages 119–124. ACM, 2017.

[18] Haitao Lang and Siwen Wu. Ship classification in moderate-resolution sar image by naive geometric features-combined multiple kernel learning. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1765–1769, 2017.

[19] Xinwang Liu, Lei Wang, Jianping Yin, Yong Dou, and Jian Zhang. Absent multiple kernel learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[20] Saeid Niazmardi, Begüm Demir, Lorenzo Bruzzone, Abdolreza Safari, and Saeid Homayouni. Multiple kernel learning for remote sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 56(3):1425–1443, March 2018. ISSN: 0196-2892. doi:10.1109/TGRS.2017.2762597.

[21] Anthony Pinar, Joseph Rice, Timothy Havens, Matthew Masarik, Joseph Burns, and Derek Anderson. Explosive hazard detection with feature and decision level fusion, multiple kernel learning, and fuzzy integrals. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[22] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents 2ed*. Cambridge University Press, 2010.

[23] Stuart Russell and Peter Norvig. *Artificial Intelligence-A Modern Approach (3rd internat. edn.)*. Pearson Education, 2010.

[24] Tiago André Guedes Santos. Weighted Multiple Kernel Learning for Breast Cancer Diagnosis applied to Mammograms. Master's thesis, Department of Computer Science, Faculty of Sciences, University of Porto, Porto, Portugal, December 2017.

[25] Sajjad Kamali Siahroudi, Poorya Zare Moodi, and Hamid Beigy. Detection of evolving concepts in non-stationary data streams: A multiple kernel learning approach. *Expert Systems with Applications*, 91:187–197, 2018.

[26] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7(Jul):1531–1565, 2006.

[27] Martin Stražar and Tomaž Curk. Approximate multiple kernel learning with least-angle regression. *Neurocomputing*, 340:245 – 258, 2019. ISSN: 0925-2312. doi:https://doi.org/10.1016/j.neucom.2019.02.030.

[28] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[29] Hong Wu, Hao Zhang, and Chao Li. Medical image classification with multiple kernel learning. In *Proceedings of the Second International Conference on Internet Multimedia Computing and Service*, pages 189–192. ACM, 2010.

[30] T Zare, Mohammad Sadeghi, and Hamid Abutalebi. A comparative study of multiple kernel learning approaches for svm classification. In *7'th International Symposium on Telecommunications (IST'2014)*, pages 84–89. IEEE, 2014.

[31] Cheng Zha, Ping Yang, Xinran Zhang, and Li Zhao. Spontaneous speech emotion recognition via multiple kernel learning. In *2016 eighth international conference on measuring technology and mechatronics automation (ICMTMA)*, pages 621–623. IEEE, 2016.

[32] Ji Zhao, Liantao Wang, Ricardo Cabral, and Fernando De la Torre. Feature and region selection for visual learning. *IEEE Transactions on Image Processing*, 25(3):1084–1094, 2016.

[33] Jinfeng Zhuang, Ivor W Tsang, and Steven CH Hoi. Two-layer multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 909–917, 2011.