UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

MACHINE LEARNING APPLIED TO CRIME PREDICTION

A Degree Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Miquel Vaquero Barnadas

In partial fulfilment
of the requirements for the degree in
TELECOMMUNICATION SYSTEMS ENGINEERING

Advisors: Margarita Cabrera Bean and José Adrian Rodríguez Fonollosa

Barcelona, September 2016

Abstract

Machine Learning is a cornerstone when it comes to artificial intelligence and big data analysis. It provides powerful algorithms that are capable of recognizing patterns, classifying data, and, basically, learn by themselves to perform a specific task. This field has incredibly grown in popularity these days, however, it still remains unknown for the majority of people, and even for most professionals.

This project intends to provide an understandable explanation of what is it, what types are there and what it can be used for, as well as solve a real data classification problem (namely San Francisco crimes classification) using different algorithms, such as K-Nearest Neighbours, Parzen windows and Neural Networks, as an introduction to this field.

Resum

El “Machine Learning” o aprenentatge màquina és la pedra angular de la intel·ligència artificial i l’anàlisi de grans volums de dades. Proveeix algoritmes potents que són capaços de reconèixer patrons, classificar dades, i, bàsicament, aprendre per ells mateixos a fer una tasca específica. Aquest camp ha crescut en popularitat darrerament, però, tot i això, encara és un gran desconegut per la majoria de gent, inclosos molts professionals del sector.

La intenció d’aquest projecte és donar una explicació més intel·ligible de què és, quins tipus hi ha i per a què es pot fer servir, així com solucionar un problema real de classificació de dades (classificant els crims de la ciutat de San Francisco) fent servir diversos algoritmes com K-Nearest Neighbours (K veïns més propers), finestres de Parzen i Xarxes Neuronals, com a introducció a aquest camp.

Resumen

El “Machine Learning” o aprendizaje máquina es la piedra angular de la inteligencia artificial i el análisis de grandes volúmenes de datos. Provee algoritmos potentes que son capaces de reconocer patrones, clasificar datos, y, básicamente, aprender por ellos mismos a hacer una tarea específica. Este campo ha crecido en popularidad últimamente, pero, aun así, todavía es un gran desconocido para la mayoría de gente, incluidos muchos profesionales del sector.

La intención de este proyecto es dar una explicación más inteligible de qué es, qué tipos hay y para qué se puede usar, así como resolver un problema real de clasificación de datos (clasificando los crímenes de la ciudad de San Francisco) usando diversos algoritmos como K-Nearest Neighbours (K vecinos más cercanos), ventanas de Parzen y Redes Neuronales, como introducción a este campo.

Acknowledgements

I have been interested in *Machine Learning* since I first heard about it in my second year of college. Now, after some years, I have had the opportunity to research and learn a bit more about it, so I would like to thank the Communications and Signal Theory Department for accepting me as a candidate for this project. Specially, I am really grateful to my advisors for their guidance and attention during the whole project. Without their support it would not have been so bearable.

Revision history and approval record

Revision	Date	Purpose
0	08/09/2016	Document creation
1	25/09/2016	Document revision
2	27/09/2016	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Miquel Vaquero	miquelvaquero@gmail.com
Margarita Cabrera	marga.cabrera@upc.edu
Jose Adrián Fonollosa	jose.fonollosa@upc.edu

Written by:		Reviewed and approved by:	
Date	08/08/2016	Date	27/08/2016
Name	Miquel Vaquero	Name	Margarita Cabrera
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

Machine Learning (ML) is still a very mysterious field for almost everybody, it sounds complicated and it's rather difficult to explain to someone who has not any technical skills. However, it is very important nowadays and will continue to be in the next years.

This project aims to give a more understandable view of what machine learning is and, as an introduction to this field, analyse and solve a problem using different techniques and algorithms to determine which is a better approach.

The initial motivation to start this project was a Kaggle[1] competition. Kaggle is a popular data science website that regularly publishes contests about data mining and machine learning. This site was holding a challenge about crime classification in the city of San Francisco[2], and that was an interesting problem to solve as an introduction to machine learning.

ML is a pretty multidisciplinary field and it mainly involves programming and mathematics (mostly working with probabilities and density functions). Also, as it is somewhat new and rather complicated, it requires good research skills.

For the crime detection problem, a huge training database of San Francisco's crimes is provided by the organizers of the contest. This database is labelled, i.e. contains the correct category for each entry in it (e.g. burglary, assault, bribery, etc), thus it is a supervised learning problem. Having this in mind, the algorithms that have been applied to try to solve it are:

- K-Nearest Neighbours (KNN)
- Parzen windows
- Neural Networks

Along with this methods, to deal with the high amount of data that was provided and make the program executions faster, K-Means algorithm was used for clustering and reducing the database size.

This document goes from the basic explanation of what ML is and what types are there, to the more complex Neural Networks algorithm, going through the analysis of the data provided and all the other algorithms that have been tested. Whenever it has been possible, graphics and flowcharts have been provided to clarify the explanations, along with some examples. Finally, to avoid it being dense, all the code and scripts are provided in the appendix section.

2. Machine Learning (ML)

2.1. A little bit of history

It might seem that this is a pretty new technology, but, in fact, it isn't. The first ML-related work dates from 66 years ago, in 1950.

In the early days of AI, ML research used mainly symbolic data, and algorithm design was based on logic [3-7]. At about the same time, Frank Rosenblatt proposed the *Perceptron*, a statistical approach based on empirical risk minimization[8]. However, this approach remained unrecognized and undeveloped in the following decades.

The real development of statistical learning came after 1986, when David Rumelhart and James McClealland proposed the nonlinear backpropagation algorithm[9]. AI, pattern recognition, and statistics researchers became interested in this approach and nowadays is highly used in deep learning Neural Networks.

For the sake of curiosity, below there is a chronological list of the most relevant events in this field since the "starting point" in 1950 **¡Error! No se encuentra el origen de la referencia.**.

1950 – Alan Turing creates the "Turing Test". This test determined whether a computer had real intelligence or not.

1952 – Arthur Samuel writes the first computer learning program. It played checkers, and the computer was able to improve at the game the more it played, studying which moves made up to winning games.

1957 – Frank Rosenblatt designed the first neural network for computers (the perceptron), which simulates the thought processes of the human brain.

1967 – The "Nearest Neighbours" algorithm was written, allowing computers to begin using very basic pattern recognition.

1979 – Students in Stanford University invent the "Stanford Cart", which can navigate obstacles in a room on its own.

1981 – Gerald Dejong introduces the concept of Explanation Based Learning (EBL), in which a computer analyses training data and creates a general rule it can follow by discarding unimportant data.

1985 – Terry Sejnowski invents NetTalk, which learns to pronounce words the same way a baby does.

1986 – David Rumelhart and James McClealland propose the nonlinear backpropagation algorithm.

1990s – Work on Machine Learning shifts from a knowledge-driven approach to a data-driven approach. Scientists begin creating programs for computers to analyse large amounts of data (Big Data) and draw conclusions (or "learn" from the results).

1997 – IBM's Deep Blue beats the world champion at chess.

2006 – Geoffrey Hinton coins the term "deep learning" to explain new algorithms that let computers "see" and distinguish objects and text in images and video.

2011 – Google Brain is developed and its deep neural network can learn to discover and categorize objects much the way a cat does.

2012 – Google's X Lab develops a machine learning algorithm that is able to autonomously browse YouTube videos to identify the videos that contain cats.

2014 – Facebook develops *DeepFace*, a software algorithm that is able to recognize or verify individuals on photos to the same level as humans can.

2015 – Amazon launches its own machine learning platform.

2015 – Microsoft creates the Distributed Machine Learning Toolkit, which enables the efficient distribution of machine learning problems across multiple computers.

2016 – Google's *AlphaGo*, an artificial intelligence algorithm, beats a professional player at the Chinese board game Go, which is considered the world's most complex board game and is many times harder than chess. It managed to win five games out of five.

2.2. What is Machine Learning?

Machine learning has become one of the mainstays of the information technology in the past two decades and thus, an important, but hidden, part of our life. The increasing amount of data that is being generated (and stored) daily by individuals and corporations, demands a smart analysis. It is here where machine learning comes to stage as a necessary ingredient for technological progress [11].

As the word stands for, machine learning is the study of computer algorithms capable of learning to improve their performance of a task on the basis of their own previous experience. It focuses in achieving that programmable devices and "machines" learn automatically, by themselves. Basically, it is all about systems learning from data.

The field is closely related to pattern recognition and statistical inference. It works with data and processes it to discover patterns that can be later used to analyse new data. It usually relies on specific representation of data, a set of "features" that are understandable for a computer. For example, if text had to be represented, it should be through the words it contains or some other characteristics such as length of the text, number of emotional words, etc. This representation depends on the task than one is dealing with and is typically referred to as "feature extraction" [12].

2.3. Types of ML

All Machine Learning tasks can be classified in several categories; the main ones are:

2.3.1. Supervised learning

Relies on a training set where some characteristics of the data are known, typically the labels or classes (the variables to predict). For example:

A computer has to be teach to distinguish pictures of cats and dogs. Some pictures of cats and dogs might be tagged with "cat" or "dog", respectively. Labelling is usually done by human annotators to ensure high quality of data. Having this true labels of the pictures, they can be used to "supervise" the algorithm in learning the right way to classify images. Once it has learned how to classify them, it can be used on new data and predict labels ("cat" or "dog" in this case) on previously unseen images.

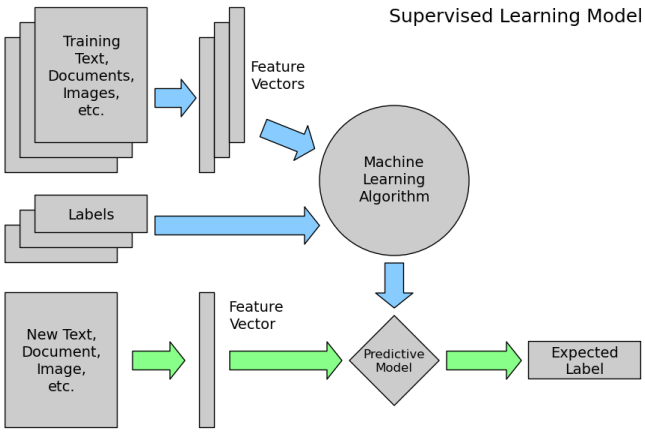

Figure 1. Supervised Learning model [13]

2.3.2. Unsupervised learning

As it can be guessed from the name, in unsupervised ML the algorithm is deprived of the labels used in the previous one. It is just provided with a large (usually huge) amount of data and characteristics of each observation (i.e. a single piece of data). Its aim is usually finding patterns among this data. For example:

Taking the last example, imagine that someone forgot to label the images of cats and dogs. However, they have to be split into two categories as well. Unsupervised ML might be used (in this case a clustering technique) to separate images in two groups based on some inherent features (characteristics) of the pictures.

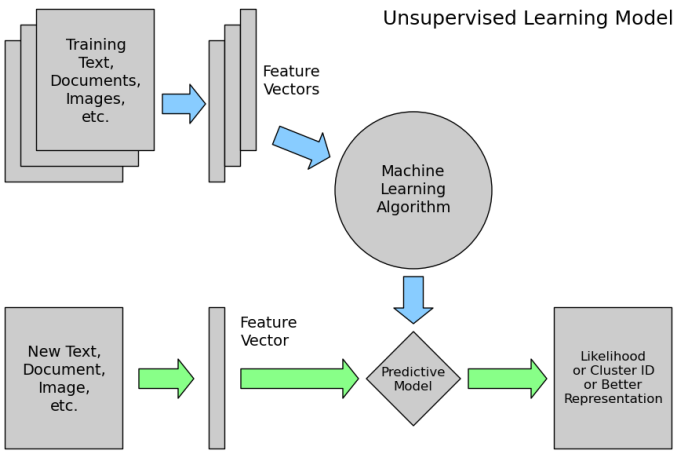

Figure 2. Unsupervised Learning model [13]

2.3.3. Reinforcement learning

The algorithm learns to react to an environment (or to some conditions) by giving positive rewards to “satisfactory” behaviours, and negative or none to “unsatisfactory” ones.

This can be easily illustrated by an example of learning to play chess. The input for the algorithm in this case is the information about whether a game played was won or lost. It does not have to have every move in the game labelled as successful or not, but only the result of the whole game. Therefore, the ML algorithm can play lots of games, and each time it gives bigger “weights” to those moves that resulted in a winning combination, and less to those leading to a lost game.

It is also common to split the methods in:

2.3.4. Lazy learning

It is a learning method in which generalization beyond the training data is delayed until a query is made to the system. They are called lazy because they wait as much as they can to create the model [14].

They learn rapidly, but they classify slowly and require large space to store the entire training dataset.

K-Nearest Neighbours is an example.

2.3.5. Eager learning

As opposed to the previous one, the system tries to construct a general, input independent target function during the system training [14].

The main advantage is that it requires much less space than a lazy learning system. Also, they deal much better with noise in the training data. However, the model creation might be slow.

Artificial Neural Networks (ANN) or Support Vector Machines (SVM) are an example.

2.4. State of the art

Internet browsers have facilitated the acquisition of large amounts of information. This technology's development has greatly surpassed that of data analysis, making large chunks of data uncontrollable and unusable. This is one reason why so many people are enthusiastic about ML.

In fields such as molecular biology and the Internet, if the concern is only about the ease of gene sorting and distributing information, there is no need of ML. However, when it comes to advanced problems such as gene functions, understanding information, and security, Machine Learning is inevitable.

As it has been seen before in section 2.1 though, ML is not a new thing and it has been continuously growing through years. Despite this fact, its foundations have remained the same. They are, basically:

- **Statistics**. ML's aim is to estimate a model from observed data, so it must use statistical measures to evaluate the model's performance and estimate it. It also needs statistics to filter noise in the data.
- **Computer science algorithm design** methodologies, for optimizing parameters and executions.

Bearing this in mind, a few of the problems that Machine Learning can solve will be explained [15], giving real world applications when possible. Of course, it does solve a wide variety of problems, but below are the most relevant.

- **Classification**:

  In this case, having an input dataset, the algorithm's goal is to, for each new unclassified data sample, be capable of assigning it to a category after performing some type of operation on it.

  A real application is email spam detection, where each element is represented as a vector of features (e.g. the number of times a specific word is repeated) and different algorithms trained with other already classified emails are applied.

- **Prediction**:

  With an already done classification, if the events to predict are similar to the previous ones, it can be determined to which class the new event belongs to.

  A real application is market trend prediction for hedge funds. For example, using certain data from tweets written in a certain country.

- **Pattern recognition**:

  Its aim is to extract repetitive structures or common characteristics between the data samples and form certain patterns.

  Facial recognition is a clear example of this application. Though faces are not always the same, they all have a generic structure (i.e. eyes, eyebrows, mouth, ears…).

- **Clustering**:

  This is an unsupervised procedure and consists in figuring out the existence of groups among the data. Thereby, objects with the same characteristics would be grouped under a same cluster or group, represented by its centre.

  Grouping a company's customers into several clusters (by age for example) so that adverts could be personalized, would be a real application.

- **Regression**:

  It is similar to classification, in this case though, the class or output variable is continuous. For example: trying to predict the housing prices under huge databases of the real-estate market.

Although it is capable of solving many problems and is a very promising field, there are also some areas where ML still performs poorly nowadays and where most of the investigation is centered. These are:

- **Transfer learning**:

  It is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [16].

  For example: a vision system that can learn to detect objects invariant to things like lighting changes, rotation, etc. by learning useful features after observing somewhat unrelated/different objects from different points of view and in different lighting conditions.

  What is interesting about this is that, even after the system has learnt, it can still improve as it learns more invariant features from different but related objects.

- **One-shot learning**:

Whereas most ML based object categorization algorithms require training on hundreds or thousands of images and very large datasets, one-shot learning aims to learn information about object categories from one, or only a few, training images [17].

This can be achieved using architectures with augmented memory capacities, such as Neural Turing Machines, which offer the ability to quickly encode and retrieve new information and avoid having to inefficiently relearn their parameters (as gradient-based networks do).

3. Crime Classification in San Francisco

As it has been said previously, this project is based on a Kaggle competition about crime classification in the city of San Francisco. In this chapter, the problem, the procedure and the algorithms will be explained along with the results obtained in each case.

3.1. Description of the problem / competition

San Francisco used to be known for housing some of the world's most notorious criminals on the island of Alcatraz. Today, it is known more for its tech scene that its criminal past. However, nowadays there is no scarcity of crime in the city.

In this competition, a training dataset with nearly 12 years of crime reports from all across San Francisco's neighbourhoods was provided. This dataset contains all crimes classified in categories, which are the different crime typologies. The main goal of the challenge is to predict this categories of the crimes that occurred.

For the algorithm evaluation, another unlabelled dataset is provided (the test one). It is used to evaluate the algorithm accuracy with new unclassified data.

To participate in the competition, results from the algorithm executions have to be submitted (uploaded) to the website. These submissions are .csv files with a specific format (explained in section 3.1.2). When uploaded, they are evaluated against the test database and they are given a score using the *Logloss* function (also explained in 3.1.2). The submission with the highest score (minimum *Logloss*) at the end, wins the contest.

3.1.1. How is the problem going to be solved?

Different algorithms will be used to come up with a good result in this contest. Each of them will be explained, tried and tested, and finally we will get to see which of them works best for this case.

Cross-validation will be used to validate the models, so the database has to be split into *test*, *train* and *validation* subsets. This split has to be stratified to ensure that the initial proportion of elements (same amount of crimes per category) is maintained in each subdivision.

The resulting train dataset is still too large (aprox. 700.000), and running the testing programs would take too long. To speed up tests and development, we will reduce the database to aprox. 8.000 records using a clustering algorithm. This algorithm will be K-Means. Then, having the

number of elements per cluster, we will be able to decide which element has more weight inside the algorithm. Technically there is no data loss.

Once the data has been treated, the following algorithms will be tried (in order of complexity):

- K-Nearest Neighbours
- Parzen windows
- Neural Networks

Each of them will be deeply explained in its chapter later on.

All the development and testing was done on a server lent by a university department. This way, executions could last all day without having to worry about them and were a little bit faster.

3.1.2. Results submission format and evaluation

Data submitted to the contest for its evaluation has to have a specific format to fulfil the requirements. To allow data to be evaluated correctly, the resulting dataset must contain the sample ID with a list of all categories and the probability of each sample to belong to each one of them. Remind that the training dataset is labelled with all sample's crime types (there are 39 different).

Submission format is:

Id,ARSON,ASSAULT,BAD CHECKS,BRIBERY,BURGLARY, … ,WARRANTS,WEAPON LAWS

0, 0.9, 0.1, 0, 0, 0, … , 0, 0

1, 0, 0, 0, 0, 0, … , 1, 0

…

etc.

Then, instead of predicting to which category may the given sample belong, the output will always be probability vectors.

Submissions are evaluated using the multi-class logarithmic loss function, which has the following formula (provided by Kaggle):

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

Where N is the number of cases in the test set, M is the number of class labels, log is the natural logarithm, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j.

Using this function, which outputs a number, results can be compared and a measure of how good the results have been can be obtained. Also, it is useful for algorithms' parameters validation before making a final submission.

3.2. Dataset Analysis

The provided dataset has different "features", each one being of a different relevance. In this chapter we will proceed to analyse this database and extract the useful information out of it.

Each record contains the following information:

- **Dates:** a timestamp of the moment that the crime occurred. It is in the following format: *Y-m-d H:i:s*. E.g.: 2015-05-13 23:53:00
- **Category:** the category of the crime. E.g.: WARRANTS
- **Descript:** a short description of the crime. E.g.: WARRANT ARREST
- **DayOfWeek:** the day of the week in which the crime occurred. E.g.: Wednesday
- **PdDistrict:** the district of the city where the crime was committed. E.g.: NORTHERN
- **Resolution:** short description of the crime resolution. E.g.: "ARREST, BOOKED"
- **Address:** the address where the crime was located. E.g.: OAK ST / LAGUNA ST
- **X:** latitude of the crime position. E.g.: -122.425891675136
- **Y:** longitude of the crime position. E.g.: 37.7745985956747

If we draw different graphics we will see more clearly how the data is distributed.

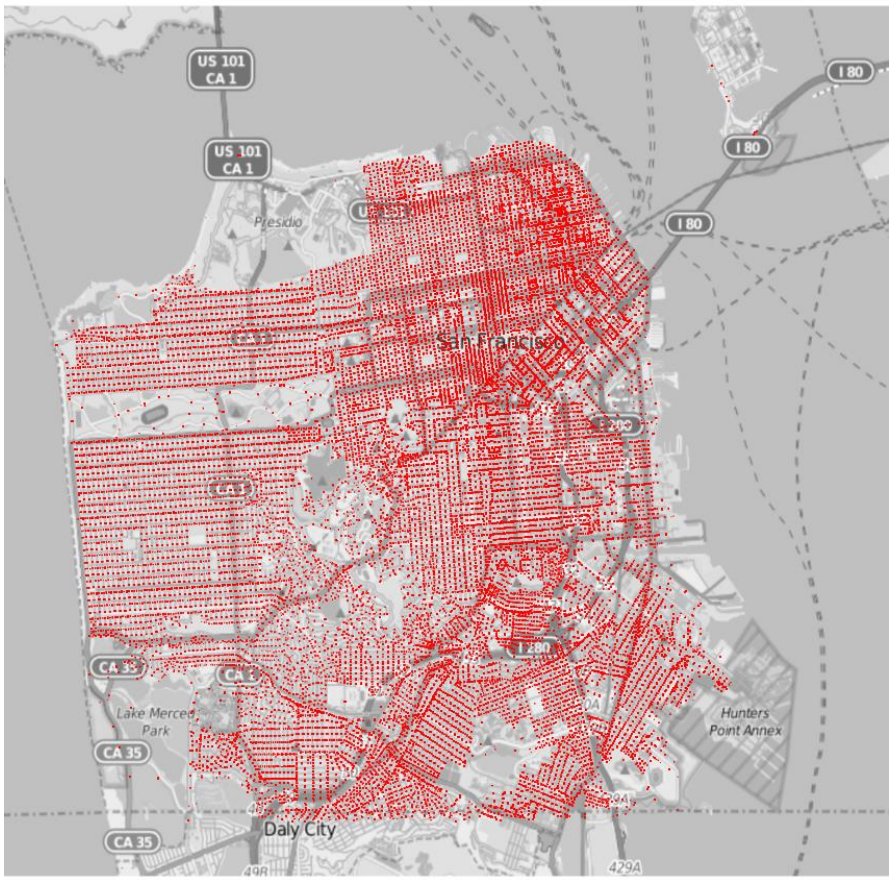

Figure 3. All crimes dot graphic

As it can be seen in Figure 3, crimes are distributed all around the city, having a hot-spot in the upper right corner.

Running a simple script which gets the number of unique categories in the dataset, we get that there are 39 different crime categories, which are: *WARRANTS, OTHER OFFENSES, LARCENY/THEFT, VEHICLE THEFT, VANDALISM, NON-CRIMINAL, ROBBERY, ASSAULT, WEAPON LAWS, BURGLARY, SUSPICIOUS OCC, DRUNKENNESS, FORGERY/COUNTERFEITING, DRUG/NARCOTIC, STOLEN PROPERTY, SECONDARY CODES, TRESPASS, MISSING PERSON, FRAUD, KIDNAPPING, RUNAWAY, DRIVING UNDER THE INFLUENCE, SEX OFFENSES FORCIBLE, PROSTITUTION, DISORDERLY CONDUCT, ARSON, FAMILY OFFENSES, LIQUOR LAWS, BRIBERY, EMBEZZLEMENT, SUICIDE, LOITERING, SEX OFFENSES NON FORCIBLE, EXTORTION, GAMBLING, BAD CHECKS, TREA, RECOVERED VEHICLE, PORNOGRAPHY/OBSCENE MAT*.

In Figure 4 it can be seen that crimes are not equally distributed among categories but there is a huge difference between them.

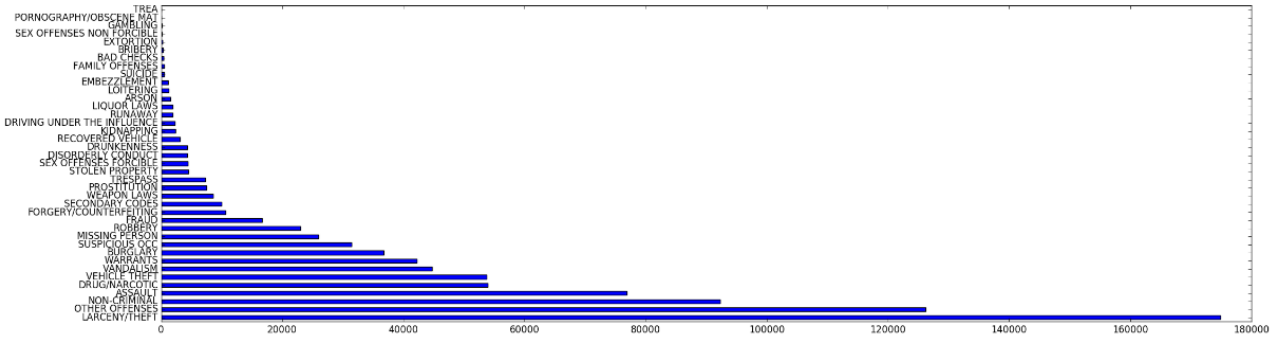

Figure 4. Crime count by category

Another interesting analysis would be counting the number of crimes that occur in each day of the week, this way we could see if this is a relevant information or not. Figure 5 shows that Friday is the most selected day by criminals, while the rest of the week is pretty equally distributed.

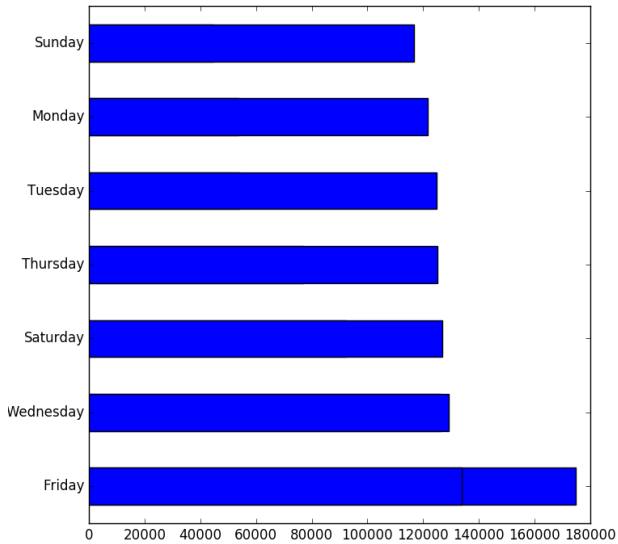

Figure 5. Number of crimes by day of the week

We can also see the crime distribution by hour of the day (Figure 6). That shows that the majority of crimes were committed between midday and midnight (including both).

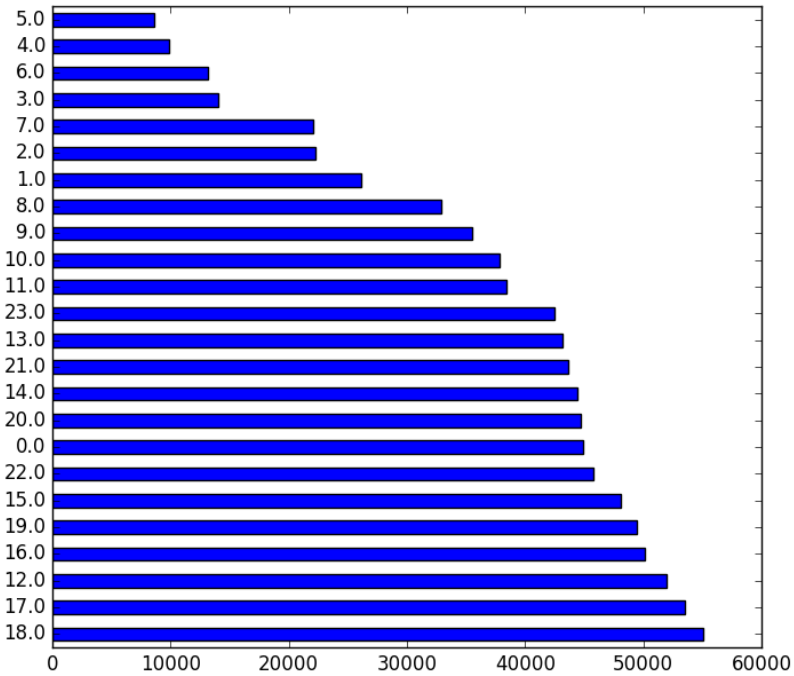

Figure 6. Crime count by hour of the day

Finally, the Figure 7 shows the crime distribution by location and category. This shows which are the historical hotspots for each type of crime category.

Figure 7. Crime density by location and category

It can be noted that most of the categories have their hotspot on the upper right corner, as the dot-graph already anticipated.

3.3. Data treatment

In the previous chapter it was explained how the dataset is distributed, and some useful information was extracted out of it. Now, having all this in mind, the useful information identification and transformation (if necessary) into a "computer-understandable" format so it can be worked with, will be carried.

Also, as cross-validation is going to be used to validate some parameters, the dataset will be split into train, test and validation subsets. Even so, the train set will still be very big (nearly 650.000 samples) and this would slow down the tests. To make them faster, a clustering algorithm will be applied to reduce the dataset size.

3.3.1. Relevant information

Remember, from the last chapter, that the following information about each record is provided: *Dates, Category, Descript, DayOfWeek, PdDistrict, Resolution, Address, X* and *Y*. In new data samples, of course, *Category* will not be there (as this is what it has to be inferred).

- Selection 1: ***X, Y*** (latitude and longitude)

A priori, it might seem that the location of a crime might be enough to distinguish from different types of them, as, probably, same types of crime are committed in the same zone. However, as it has been seen when analysing the dataset, pretty much all categories have the same "hotspot" location. So a second selection including more information is necessary.

- Selection 2: ***X, Y, Dates***

Bearing in mind the analysis, it can be seen that another distinguishing property of the crimes is the time when they happened. So, including this *Dates* feature (which also includes the time) and formatting it correctly (in the next section will see how), could be enough to discern between crimes.

- Selection 3: ***X, Y, Dates, DayOfWeek***

The last feature combination would be also including the day of the week when the crime was committed. As it has been seen in the analysis chapter, samples differ in this information, so it can be useful. It will also have to be converted to a different format, which is what will be explained in the next section.

This last feature extraction seems logic and, a priori, it seems that it would probably work. However, the first algorithm will be tested with this three different feature combination to see if the best one is really the last one. Then, all the following tests will be done with the "winner" feature selection combination.

3.3.2. Data transformation

To differentiate between samples and to be able to classify them, they must be comparable in some way. They must be in a format which allows the algorithm to put one against another and see which one fits each classification.

Not all the features provided are in a good format for this. For example, *DayOfWeek* is a string (a sequence of characters, a word), which may be comparable for a human, but for a computer it isn't. So, the following are the features that need to be converted:

- *Dates*:

  This feature comes in a string with the following format: *yyyy-mm-dd hh:mm:ss*

  This format is good for showing the date in a human-readable way, however, for a computer is not the best format to compare dates.

  To deal with this correctly, the date part (not useful) will be erased and the time will be converted to Cartesian coordinates. Parsing the previous string, the hours, minutes and seconds parts can be extracted. Then, the following operation yields the number of hours since midnight:

  $$time = hours + (minutes / 60) + (seconds / 3600)$$

  Then, the circumference is divided into 24 parts, as there are 24 hours in a day, and the angle corresponding to each part is:

  $$\varphi = (2*\pi) / 24$$

  Finally, *TimeX* and *TimeY*, the Cartesian coordinates correspond to each date:

  $$TimeX = \cos(\varphi)*time$$

  $$TimeY = \sin(\varphi)*time$$

- *DayOfWeek:*

  With the day of the week the procedure is more or less the same as the previous transformation. In this case, the circumference is divided into 7 parts, as there are 7 days in a week.

  First, each day is assigned a number:

  Monday – 0, Tuesday – 1, Wednesday – 2, Thursday – 3, Friday – 4,

  Saturday – 5, Sunday – 6

  Then, the angle corresponding to a day is got from:

  $$\varphi = (2*\pi) / 7$$

  Finally, the *d1* and *d2* day of the week Cartesian coordinates are:

  $$d1 = \cos(\varphi * dayNumber)$$

  $$d2 = \sin(\varphi * dayNumber)$$

  Being *dayNumber* the number that has been assigned to each day name at the beginning.

3.3.3. Dataset split

Results from the executions of the algorithms need to be evaluated to see which ones are better and also to see if the parameters' values used are adequate. As only one labelled dataset is provided, it has to be split it into 3 smaller parts to train, test and validate the algorithms.

For this purpose, then, the main (train) database will be split into (as said previously) train, test and validate subsets. This division will be done in the following proportions: 80% train, 10% test and 10% validate.

With this reduced training database, which is also labelled, the algorithm will be shown which are the correct results and start making it learn to classify.

With the *test* one, the accuracy of the algorithm will be tested, to see if any parameter adjustments have to be made or compare it with the other ones.

Finally, the *validate* subset allows to, rapidly, validate the value of certain parameters of the training algorithm in order to choose their correct value or the one that fits best.

3.3.4. Dataset reduction (clustering)

Remember that the initial training database had nearly 900.000 records, which is pretty high. Then, with the split, it was reduced to 80% its size (which is about 720.000 records). Still too big if the tests are meant to run fast.

So, it is required to keep reducing the training dataset until it has a manageable size. Deleting all the "extra" samples that are not wanted would end up with a messed up database that wouldn't have the same elements per category proportion than the original one. To avoid this happening, the clustering algorithm used to reduce the database size will be applied for each category. This way, it is ensured that the resulting set will maintain the same proportions from the initial one.

A clustering algorithm groups elements in a specified number of clusters. This clusters' centres are calculated and they act as the representatives of all the samples in their group. Because not all clusters will have the same number of elements, some will be more relevant than others. Their relevance is measured by their weight, which is the number of elements they contain.

The algorithm used for this purpose has been *K-means*.

K-means clustering proceeds by selecting *k* initial cluster centres and then, iteratively refining them as follows:

1. Each instance d_i is assigned to its closest cluster centre.
2. Each cluster centre C_j is updated to be the mean of its constituent instances.

The algorithm converges when there is no further change in assignment of instances of clusters, or if the maximum total number of iterations defined have been done [18].

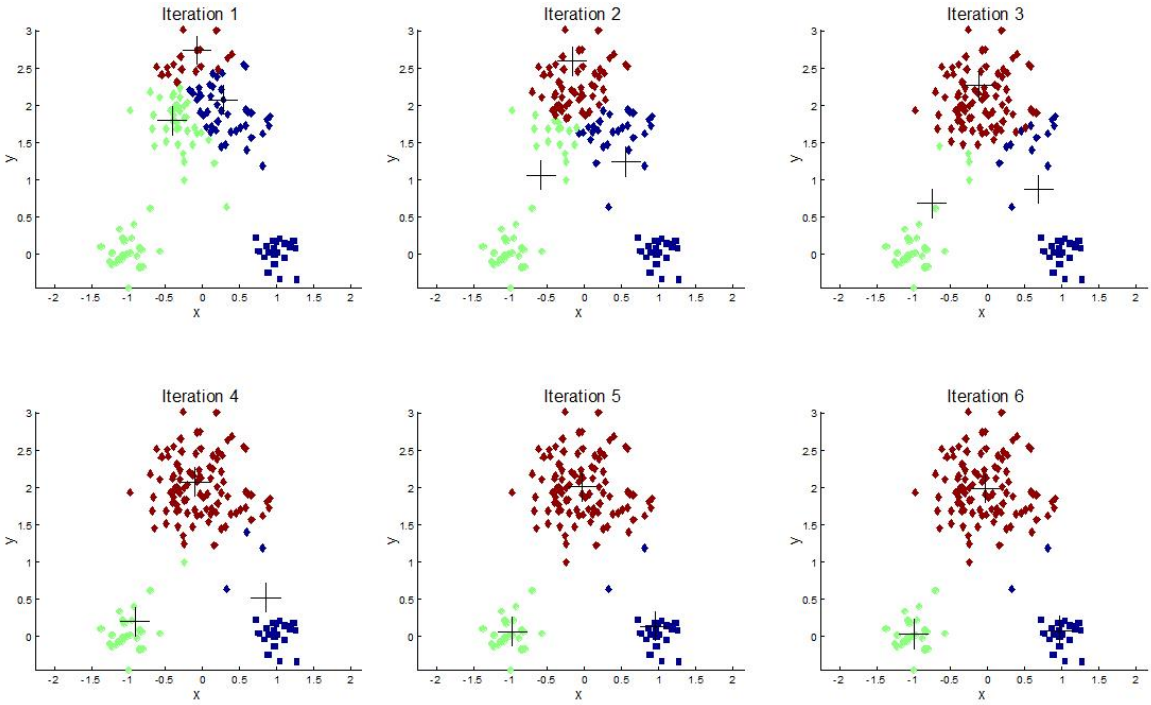

Figure 8. *K-Means* clustering iterations until convergence [19]

In this work, this algorithm has been applied for each category in the following way:

1. Get all the elements of a category
2. Obtain the number of centroids necessary to reduce the database to the number of elements that we want, bearing in mind that some classes might have very few members.
3. Apply K-means for this category.
4. Get the number of samples per cluster.
5. Add the resulting cluster centres with the number of elements they represent to the reduced database.
6. Follow up with the next category.

Note that this algorithm has to be applied after converting the values to the right format, otherwise it won't be able to calculate the cluster centre properly.

After the clustering, the final training database has approximately 8.000 records, which is a good size that allows to work faster.

3.4. Machine Learning Algorithms Used

In this section we will explain which have been the three algorithms that have been tried in order to solve this problem. Here, the theory and the implementation are discussed, whether the results and its evaluation can be found in the next section (3.5).

3.4.1. Priors classifier

It is not a machine learning algorithm, but it has been put under this section because it has been used as a reference for the rest. The results obtained with this classification were the ones to beat by the other algorithms, as this is the coarsest and simplest one and does no analysis over the data.

3.4.1.1. The algorithm

Basically, the probability of a sample to belong to a category is its prior one, given by the following formula:

$$P(x \in C_i) = \frac{n_i}{\sum_{l=1}^{k} n_l}$$

Where n_i, n_j are the number of elements in the category i and j respectively, and k is the total number of categories.

3.4.1.2. Implementation

In this case, the implementation is trivial and the only thing that has to be done is calculate the prior probability of each category and fill the output dataset with the same data for all samples.

3.4.2. K-Nearest Neighbours

3.4.2.1. The algorithm

The k-nearest neighbour classification rule is arguably the simplest and most intuitively appealing nonparametric classifier. It assigns a sample *z* to class *X* if the majority (i.e. more than ½) of the *k* values in the training dataset that are near *z* are from *X*, otherwise it assigns it to class *Y*.

This algorithm compares the given unclassified sample *z* with "all" (depending on the implementation, to speed up the execution, other comparison algorithms can be used so that it's not needed to compare it with every single value) the values in the training set and gets its *k* nearest values. Among them, it applies the previously described rule.

A simple example will make it clearer:

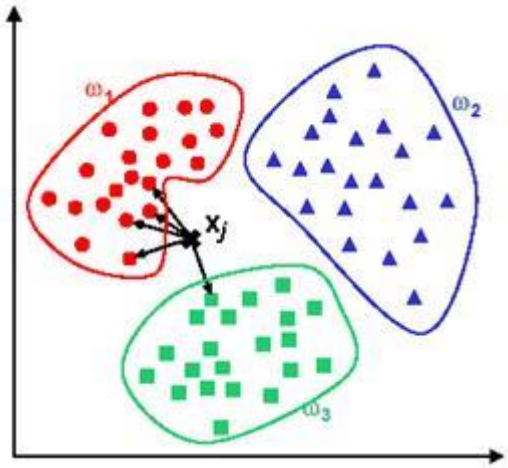

Figure 9. KNN example (k = 5)

In the previous figure (Figure 9), there are 3 categories (w_1, w_2 and w_3), and the sample to classify is x_j. The value of *k* is 5, so it has to look for the sample's nearest 5 observations. Visually, the solution is clear. The new x_j sample is assigned to w_1 class, because it is the predominant among its neighbours.

3.4.2.2. Implementation

Although it's a pretty easy algorithm to implement from scratch, making it efficient and fast it's not trivial. For this reason, Python scikit-learn library [20], which has a broad set of machine learning utilities, has been used. This library has a Nearest Neighbours package with all necessary functions. From this package, *NearestNeighbors* class has been used (concretely, *fit* and *kneighbors* methods).

Because the results must be submitted giving a probability of each sample to belong to all categories, and not just the predicted category (as said in 3.1.2), the following formula was used to calculate this probabilities:

$$P(x \in C_i) = \frac{\sum_{j \in C_i}(n_j \frac{1}{d_j})}{\sum_{l=1}^{k}(n_l \frac{1}{d_l})}$$

Where $P(x \in C_i)$ is the probability of the test sample to belong to category C_i, n_j and n_l are the number of elements represented by each train dataset sample, and d_j and d_l are the distance between test and the corresponding train sample.

With all this being said, the step-by-step implementation would be the following:

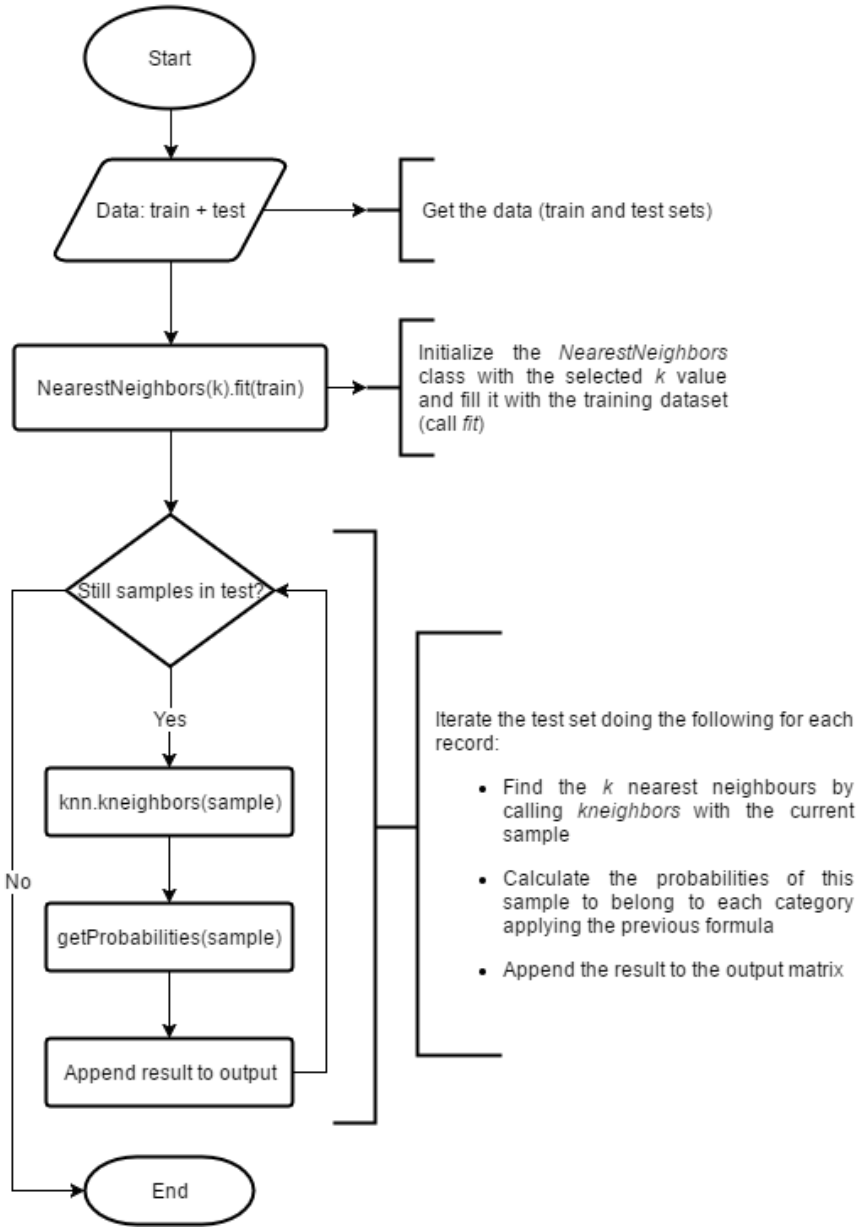

Figure 10. KNN algorithm flowchart

3.4.2.3. Parameter selection and validation

It has been talked about the *k* parameter and its importance, but no value has been selected yet nor it has been talked about why it is so important.

This parameter specifies the number of neighbours that the algorithm has to find before trying to make a prediction about which category the new data belongs to. A very large number will guarantee more accuracy, however, computational efficiency would decay. A very small number will make the execution very fast, but won't provide accurate results. Of course, large or small depends on the dataset size.

To choose a good value for this parameter, various executions varying it have been run and compared to see which has the best output. Basically, parameter validation has been performed. To run this fast, the validation and test reduced datasets have been used. Below there is a table with all the results and a graphic (for this tests only X and Y features have been used):

k	Score
100	3,43472
200	3,15277
300	3,01502
400	2,90649
500	2,83635
600	2,80531

Table 1. *k* parameter tests with output scores

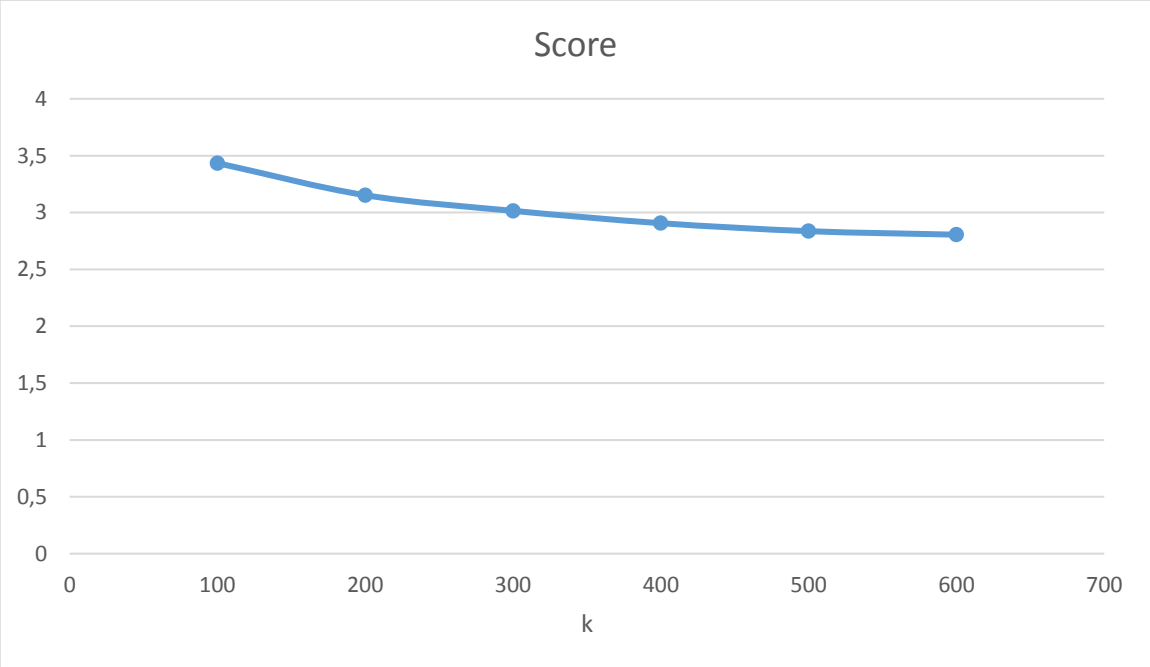

Figure 11. *k* parameter vs score graphic

As it can be noted in the graphic drawn from the different executions, the score rapidly decreases from 100 to 400, but then stops this progression and increasing by 100 the value of *k* has almost no impact in the final score. Thus, the selected value for *k* has been 600, because it has obtained the best output. Of course, one would expect that, as the value of *k* is increased, results would get better. That is true, however, the computational cost of increasing this value would not be worth the output gain.

3.4.2.4. Tests performed

As this is the simplest algorithm, different feature combinations (chapter 3.3.1) have been tested with it to see which one outputs the best results. The three performed tests were:

- Latitude and longitude (X, Y)
- X, Y, timeX, timeY
- X, Y, timeX, timeY, dayofweekX (d1), dayofweekY (d2)

The results for each one will be shown and commented later on, on the results chapter (3.5). However, to move on explaining the other algorithms, it can be said that the best combination was the last one (containing the 6 feature vector). This one is the one that has been used for all tests from this point.

3.4.3. Parzen windows

3.4.3.1. The algorithm

Parzen is a non-parametric method for estimating an unknown probability density function [21]. It relies on the fact that the probability *P* that a vector **x** will fall in a region *R* is given by:

$$P = \int_R p(\boldsymbol{x}')d\boldsymbol{x}'$$

Thus *P* is a smoothed or averaged version of the density function $p(\boldsymbol{x}')$, and this smoothed value of *p* can be estimated by estimating the probability *P*. Suppose that *n* samples $\mathbf{x_1}$, …, $\mathbf{x_n}$ are drawn independently and identically distributed according to the probability law ***p(x)***. Then, the probability that *k* of these *n* fall in *R* is given by the binomial law:

$$P_k = \binom{n}{k} P^k (1-P)^{n-k}$$

Where the expected value for *k* is: $\varepsilon[k] = nP$

If it is now assumed that $p(\boldsymbol{x})$ is continuous and that the region *R* is so small that p does not vary appreciably within it, it can be written:

$$\int_R p(\boldsymbol{x}')d\boldsymbol{x}' \cong p(\boldsymbol{x})V$$

Where **x** is a point within *R*, *n* is the total number of samples, *k* is the number of samples whose probability density function is to be estimated, and *V* is the volume enclosed by *R*. Combining the

previous equations, and supposing an unlimited number of samples is available, the n[th] estimate for *p(**x**)* is expressed in the following way:

$$p_n(\boldsymbol{x}) \cong \frac{k_n/n}{V_n}$$

Where:

$R_1, R_2, \ldots, R_n$: is a sequence of regions containing **x** where R_1 has one sample, R_2 has two, etc

$p_n(\boldsymbol{x})$: the n[th] estimate for $p(\boldsymbol{x})$

n: the total number of samples

V_n: the volume for R_n

k_n: the number of samples falling into R_n

The Parzen window approach to estimating densities can be introduced by assuming that the region R_n is a d-dimensional hypercube which encloses k samples. If h_n is the length of an edge of that hypercube, then its volume is given by:

$$V_n = h_n^d$$

The number of samples in this hypercube is given by:

$$k_n = \sum_{i=1}^{n} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h_n}\right)$$

Then, substituting these expressions into $p_n(\boldsymbol{x})$ one, the following estimation is obtained:

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h_n}\right)$$

Which can be simplified by:

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \gamma_n(\boldsymbol{x} - \boldsymbol{x_i})$$

Where $\gamma_n(\boldsymbol{x} - \boldsymbol{x_i})$ will be a Gaussian window:

$$\gamma_n(\boldsymbol{x} - \boldsymbol{x_i}) = \frac{1}{h_n^d} \frac{1}{\sqrt{(2\pi)^d |C_c|}} \exp\left(-\frac{1}{2}\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h_n}\right)^T C_c^{-1} \left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h_n}\right)\right)$$

d is the number of coordinates of each vector and $h_n = \frac{h}{\sqrt{n}}$, the parameter which determines the window width.

So, the Parzen window estimate can be considered as a sum of Gaussian functions centered at the observations (or data points). The parameter h_n determines their width.

3.4.3.2. Implementation

This algorithm's implementation is a little bit more complicated than the previous KNN one. Not because of the high complexity of the algorithm, but for the window calculation.

As the submission format is the probability of a sample to belong to each category, no prediction has to be calculated and thus only the Gaussian window calculation will be implemented.

The following is the algorithm flowchart:

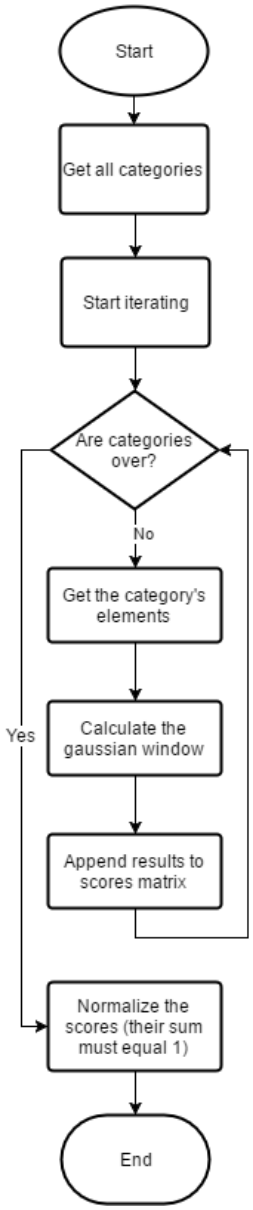

Figure 12. Parzen implementation flowchart

3.4.3.3. Parameter selection and validation

The parameter that needs to be chosen and validated with this method is the window width. A large window width will over-smooth the density and mask the structure in the data. On the other hand, a small bandwidth will yield a density estimate that is spiky and very hard to interpret.

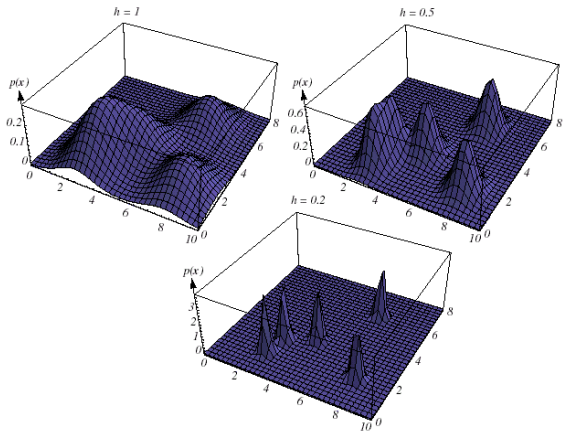

Figure 13. The effect of Parzen-window width h_n on the estimated density

The value of this parameter that minimizes the error between the estimated density and the true density has to be found.

h	Score
0,01	34,49875
0,1	32,39571
0,2	29,10708
0,5	20,24860
1	11,92964
2	3,61036
20	5,14194

Table 2. *h* parameter tests with their output scores

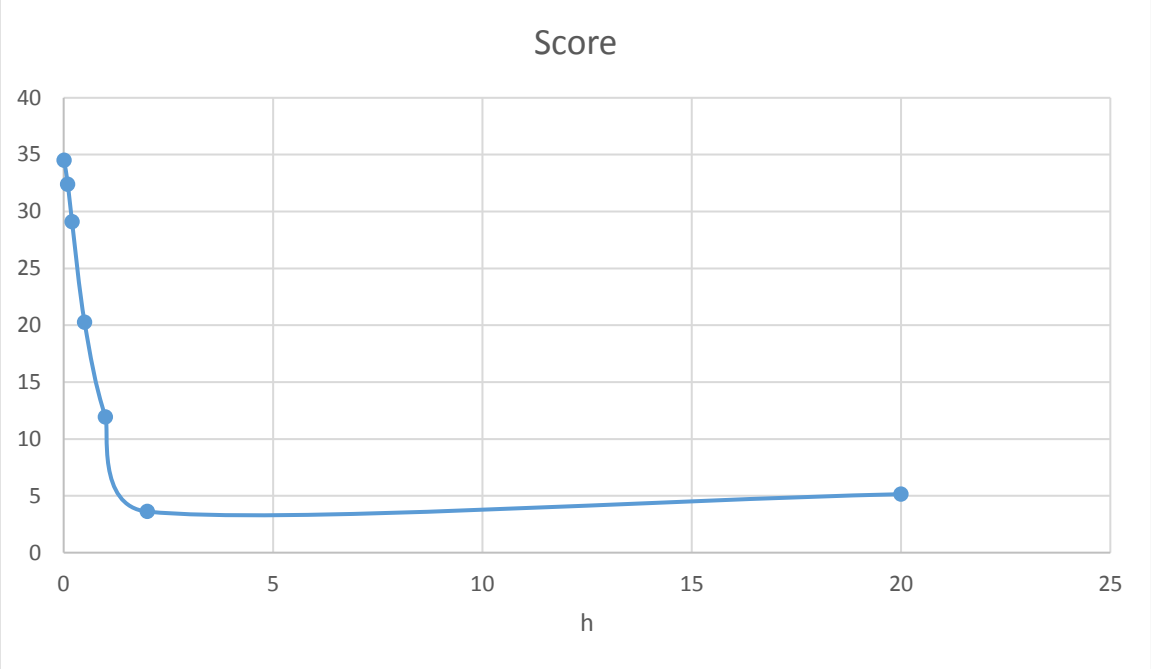

Figure 14. *h* parameter vs score

Viewing the results, it can be seen that the selected value was h = 2. Although the *Logloss* result is still high, it was the best output.

3.4.3.4. Tests performed

For this algorithm, two tests have been performed:

- *Six features (X, Y, timeX, timeY, dayX, dayY)*
- *Six features combined with prior probabilities (MAP):* this combination was weighted and it was done applying the following formula, $p = f(x|c)^{(1-\alpha)} {P_c}^{\alpha}$, being P_c the prior probability and α the weighting factor.

3.4.4. Neural Networks

3.4.4.1. The algorithm

Artificial Neural Networks (ANN) have been developed as generalizations of mathematical models of biological nervous systems. The basic processing elements of neural networks are called *artificial neurons,* or simply *neurons* or *nodes*.

In a simplified mathematical model of the neuron, the effects of the synapses are represented by connection weights that modulate the effect of the associated input signals, and the non-linear

characteristic exhibited by neurons is represented by a transfer function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm [22].

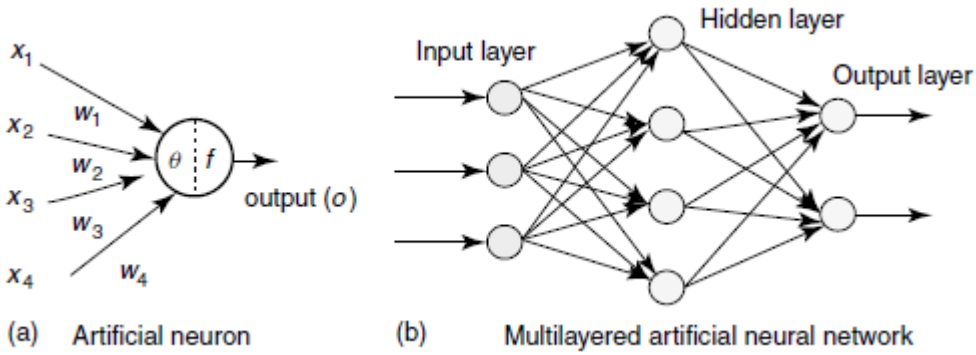

Figure 15. Artificial Neuron and Multi-layered Artificial Neural Network Architecture [22]

The basic architecture consists of three types of neuron layers: input, hidden and output layers (as it can be seen in figure 15). There are two main types of networks depending on its signal flow:

- *Feed-forward networks*: the signal flow is from input to output units and no feedback connections are present.
- *Recurrent networks*: there are feedback connections. Dynamical properties of the network are important in this case.

The data processing can extend over multiple layers of units, but the simplest case is the one above.

A neural network has to be configured such that the application of a set of inputs produces the desired set of outputs. Various methods to set the strengths of the connections exist:

- Set the weights explicitly: uses a priori knowledge.
- Train the neural network: feeding it teaching patterns and letting it change its weights according to some learning rule.

In the second method, the learning rule may be of one of the three types of machine learning that have been explained in section 2.3. Namely:

- Supervised learning: an input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done, and the errors between the desired and actual response for each node in the output layer are

found. These are then used to determine weight changes in the net according to the prevailing learning rule.

- Unsupervised learning: an output unit is trained to respond to clusters of pattern within the input. In this paradigm, the system is supposed to discover statistically salient features of the input population. There is no a priori set of categories; rather, the system must develop its own representation of the input.
- Reinforcement learning: maps situations to actions trying to maximize a numerical reward signal. The learner is not told which actions to take, instead it must discover which actions yield the most reward by trying them (trial and error).

ANN are a complicated enough algorithm to fill pages and pages explaining how they work and learn. However, it is not the scope of this project and a generic and basic view of them has already been given.

3.4.4.2. Implementation

As ANN are rather difficult to implement and understand, and they possibly are out of the scope in a beginner's course in Machine Learning; the implementation of the Artificial Neural Network used in this project has been adapted from several public scripts from other contest participants. By trying to adapt the existing work, one can better understand how ANNs work.

The code has been implemented using the Python's library *Keras* [23], which is a deep learning library.

The model used for this network is a Graph, which implements a NN graph with arbitrary layer connections, arbitrary number of inputs and arbitrary number of outputs. Then, this graph network is configured to have the following structure:

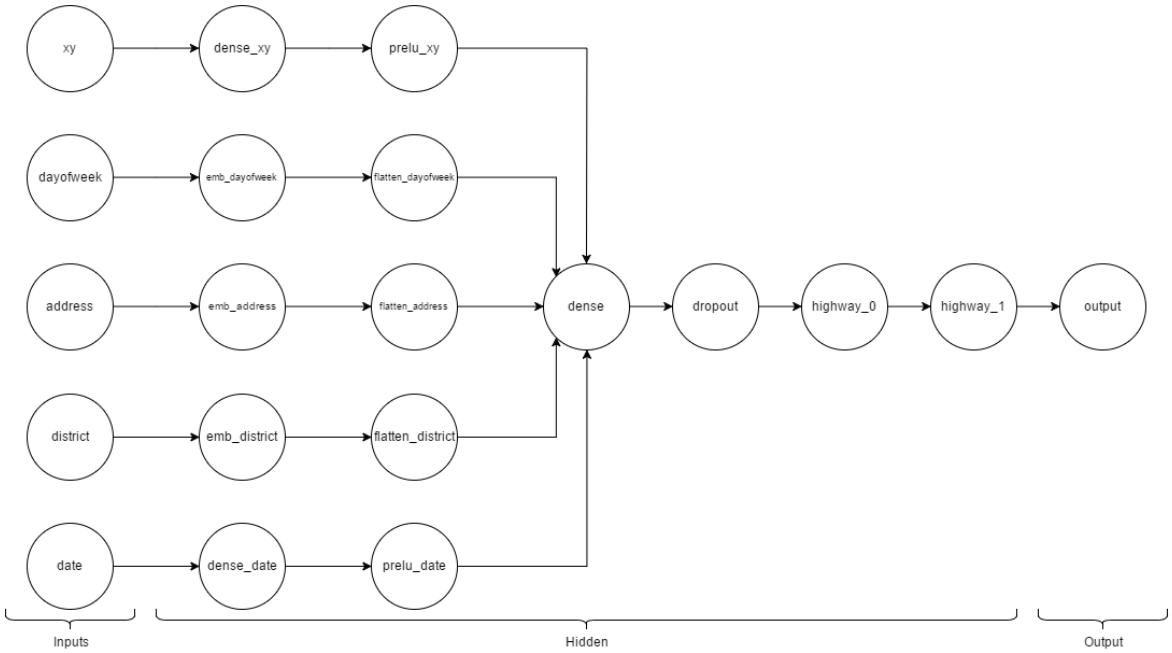

Figure 16. Implemented Artificial Neural Network

3.4.4.3. Tests performed

One test has been performed with the ANN implementation which corresponds to the previous implementation explanation (3.4.4.2).

3.5. Results

3.5.1. Priors (reference)

The following pie chart (Figure 16) shows the prior probability (in %) of a sample to belong to each category. The categories with a probability lower than 0,9% have been grouped under a new "OTHERS" category so that the graphic could be seen more clearly.

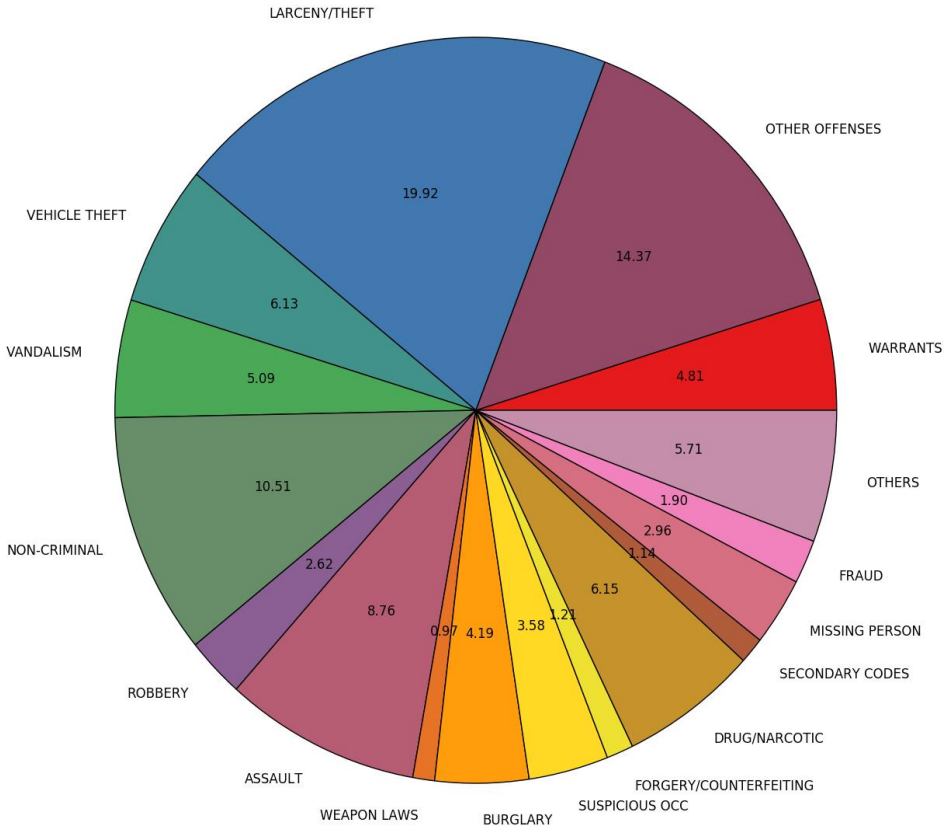

Figure 17. Prior probability of a sample to belong to each category

Using this simple classifier, a reference submission containing the previous probabilities for each sample in the test dataset was generated. The *Logloss* result on this method was the mark to be beaten by the following algorithms.

The submission *Logloss* result (score) was: **2,68080**

3.5.2. K-Nearest Neighbours

As explained in the section 3.4.2.4, several tests using different features were performed with this algorithm. This was made in order to find the best combination and use it afterwards with all the tests. This section is a walkthrough over the results in each feature combination try, giving the *Logloss* output of all of them.

In section 3.4.2.3 (parameter selection and validation), it was seen that the optimal ***k = 600*** (number of neighbours to check). This is the value that was used for all this tests.

Features	Score
X, Y	2,80531
X, Y, timeX, timeY	2,73911
X, Y, timeX, timeY, dayofweekX, dayofweekY	2,65895

Table 3. KNN simple tests results

The best result of these tests was a *Logloss* of **2,65895**, which is slightly better than the prior probabilities classifier.

3.5.3. Parzen

In section 3.4.3.3 the parameter *h* was validated, and the best result was with ***h = 2***. This has been the value used for this tests.

- *Simple test with 6 features:* **3,61036**
- *Weighted combination with prior probabilities:*

Features	Score
6 features, α = 0,75	2,79169
6 features, α = 0,8	2,74675
6 features, α = 0,9	2,69279

Table 4. Parzen-priors weighted combination tests results

The best result in this tests was the last one (**2,69279**), however it is not better than the priors and increasing the weighting factor would not make it better.

3.5.4. Neural Network

This test's execution outputted a score of **2,42982**, which is also better than the reference classifier (priors).

4. Budget

This project's object was not building a physical prototype of a product nor a software. It was centered in investigating machine learning and participate in a competition, trying to beat the previous marks in each new algorithm usage.

However, the following has to be taken into account to evaluate the cost of this project:

- Linux dedicated server per 5 months
- Junior engineer hour cost per 7 months

The research and development lasted 7 months, but the server was not available since the second month. On the other hand, the engineer in charge of the project worked on a full-time job during this time, so its dedication to the project was 4h a day (estimated). Thus, the total number of dedication hours in 7 months adds up to:

7 months * 4 weeks/month * 5 days/week * 4h/day = 560h

A large server is needed for running machine learning algorithms, since they are dealing with a lot of data and performing complicated calculations with huge matrix and vectors. The server provided for this project was a UPC self-hosted one and it was large, but for this budget estimation a smaller one (but enough for a single project development) of a popular European hosting company has been selected [25].

Bearing all this in mind, below there is a table with the project's budget breakdown:

Concept	Time	Unit Cost	Total Cost
Dedicated Server	5 months	59,99€/month	299,95 €
Junior Engineer	560h	20€/hour	11.200 €

Adding it all up, the total cost of the project would be:

299,95 € + 11.200 € = **11.499,95 €**

5. Conclusions and future development

Machine Learning is a really powerful field when it comes to AI and, if a model is done right, the level of accuracy that some algorithms can achieve can be astonishing. Certainly, the present and future of intelligent systems goes through ML and big data analysis.

With the crime classification problem, it has been seen that the most accurate algorithm was the Artificial Neural Network. Although it is true that ANN was better than KNN, it did not outperform it. This might have been a model design problem, bad feature selection, poor training or bad adjustment, or a combination of all of them. Probably, by adding more layers to the network or extending the training process, the results might have been better.

One thing that has to be noticed is that algorithms' computational cost has to always be taken into account. When dealing with huge amounts of data and complicated calculations over large matrix and vectors, executions might easily last very long. Algorithm and code optimization are also key in effective Machine Learning. In Neural Networks, this cost relies in the training process, whereas in KNN and Parzen, it relies in the decision process.

In the end, taking into account that ANN are more complicated to setup and design than KNN, one could say that both performed very similarly in this tests.

This crime classification problem might have some utility in the academic field, however it might not very useful in real life (or, at least, it is difficult to see a practical application of it).

A possible extension of this work would be to create a crime prediction system that would anticipate which zones of the city might be a crime hotspot on certain dates. This way, police could be warned and they could, for example, double the surveillance on this zone. Designing a Machine Learning algorithm (probably a Neural Network) to identify patterns among the data (unsupervised learning) would be the starting point of this future work.

Bibliography

[1]. https://www.kaggle.com [Accessed: 22/09/2016]

[2]. https://www.kaggle.com/c/sf-crime [Accessed: 22/09/2016]

[3]. R. Solomonoff. "A New Method for Discovering the Grammars of Phrase Structure Language", *Proc. Int'l Conf. Information Processing,* Unesco, 1959, pp. 285-290

[4]. E. Hunt, J. Marin, P. Stone. "Experiments in Induction", Academic Press, 1966

[5]. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers, Part II", *IBM J. Research and Development,* vol. II, no. 4, 1967, pp. 601-618

[6]. J. Quinlan. "Introduction of Decision Trees", *Machine Learning*, vol. I, Mar. 1986, pp. 81-106

[7]. Z. Pawlak. "Rough Sets: Theoretical Aspects of Reasoning about Data", *Kluwer Academic Publishers*, 1991

[8]. F. Rosenblatt. "The Perceptron: A Perceiving and Recognizing Automaton", tech. report 85-460-1, Aeronautical Lab., Cornell Univ., 1957

[9]. D. E. Rumelhart, J. L. McClelland. "Parallel Distributed Processing", MIT Press, 1986

[10]. http://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read [Accessed: 20/09/2016]

[11]. A. Smola, S.V.N. Vishwanathan. *Introduction to Machine Learning*, 1st ed. Cambridge University Press, UK, 2008

[12]. A. L. Blum, P. Langley. "Selection of relevant features and examples in machine learning", *Elsevier*, 1997

[13]. http://www.astroml.org/sklearn_tutorial/general_concepts.html [Accessed: 20/09/2016]

[14]. I. Hendrickx, A. Van den Bosch. "Hybrid algorithms with Instance-Based Classification". *Machine Learning: ECML2005*. Springer. pp. 158-169.

[15]. E. Alpaydin. "Introduction to Machine Learning", *The MIT Press*, Cambridge, Massachusetts, 2010

[16]. L. Torrey, J. Shavlik. "Transfer Learning". *University of Wisconsin* [Online] Available: ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf [Accessed: 20/09/2016]

[17]. A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillycrap. "One-Shot Learning with Memory-Augmented Neural Networks". *Google DeepMind.* [Online] Available: https://arxiv.org/pdf/1605.06065v1.pdf [Accessed: 20/09/2016]

[18]. K. Wagsta, C. Cardie, S. Rogers, S. Schroedl. "Constrained K-means Clustering with Background Knowledge". *In Proceedings of the Eighteenth International Conference of Machine Learning,* 2001, pp. 577-584.

[19]. https://apandre.wordpress.com/visible-data/cluster-analysis/ [Accessed: 20/09/2016]

[20]. http://scikit-learn.org/stable/index.html [Accessed: 20/09/2016]

[21]. http://www.byclb.com/TR/Tutorials/neural_networks/ch11_1.htm [Accessed: 21/09/2016]

[22]. A. Abraham. "Handbook of Measuring System Design", John Wiley & Sons, Ltd., 2005, pp. 901-908. ISBN: 0-470-02143-8

[23]. https://keras.io/ [Accessed: 21/09/2016]

[24]. P. Domingos. "A Few Useful Things to Know about Machine Learning". *University of Washington.* [Online] Available: https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf [Accessed: 20/09/2016]

[25]. https://www.ovh.es/servidores_dedicados/details-servers-range-HOST-id-2016-HOST-32L.xml [Accessed: 25/09/2016]

Appendices

Appendices consist of all the code scripts that have been used during this project. Below there is a reference of all of them ordered by usage, explaining what they are intended to do:

- **dataset_analysis.py:** performs an analysis of several data characteristics (i.e. hour of the day count, crimes per category count, dayofweek count) and outputs the resulting graphics.
- **density_plot.py:** builds a crime density graphic superposed to the San Francisco city map for each category.
- **sf-map.txt:** San Francisco city map information to build the previous graphic.
- **dot_plot.py:** draws a dot over the city map for each crime in the database. Outputs the all-crimes graphic and also one for each category.
- **logloss.py:** gets the submission and test files and calculates the *Logloss* score.
- **datetime_conversion.py:** performs a conversion of the dayofweek and time values according to what has been exposed in 3.3.2.
- **split_dbs.py:** splits the initial large training database into smaller train, test and validate subsets with the specified proportions.
- **kmeans.py:** applies the K-Means clustering algorithm over the split train database (80% of the original, big one).
- **priors.py:** calculates the prior probabilities of the samples to belong to each category and outputs a well formatted submission file.
- **knn.py:** applies the K-Nearest Neighbours algorithm using the train and test datasets and outputs a well formatted submission file.
- **parzen.py:** implements the Parzen windows algorithm using train and test datasets and outputs a matrix with the density function result on each field.
- **parzen_normalize.py:** as the previous script result is not normalized (not all rows sum is equal to 1), this script does it to output a well formatted submission file.
- **parzen_combine.py:** normalizes Parzen results and combines them with prior probabilities to output a well formatted submission file.
- **ann.py:** builds an Artificial Neural Network model and trains it using train dataset. After, it validates the model using test set and, finally, outputs a well formatted submission file.

The scripts provided are the ones that implement the most generic or complicated case, others have been omitted to avoid redundancy.

Glossary

ML = Machine Learning

KNN = K-Nearest Neighbours

NN = Neural Network/s

ANN = Artificial Neural Network/s