

Master of Science in Engineering in Computer Science

Machine Learning 2020/2021

Valerio Valente | 1894954

HOMEWORK 1

Latina, Italy

**Abstract**

This report describes the building process and development phases.

In order to achieve the project goal, we have studied the classification algorithms and, since there are several algorithms to solve classification problems, we decided to focus only on Decision Tree and SVM. My job was about trying to classify and find the best accuracy over a dataset containing a set of functions.

**Introduction**

Before introducing the algorithm, we have to first understand what exactly classification means. Classification is the most common Supervised Learning task of predicting the class of given data, more precisely, it is about labelling classes into categories. Examples for classification problems can be handwritten text recognition, document classification, spam recognition and so on. The spam filter is a good example: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails. Note that some regression algorithms can be used for classification as well, and vice versa.. For solving this kind of problems, there are several classification algorithms already invented such as Support Vector Machines, Decision Trees, Neural Networks as so on. However, we have only used SVMs and Decision Trees in order to solve our classification problem. The fundamental reason for choosing Support Vector Machines is that it gives better accuracy and perform faster prediction compared to other algorithms. Additionally, they use less memory because they just implement and use only the subset of training points in the decision phase.

**Design**

**Support Vector Machine**

A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. It is one of the most popular models in Machine Learning.

SVMs are particularly well suited for classification of complex but small- or medium-sized datasets. The main objective of Support Vector Machines is to find a hyperplane in an N-dimensional space that distinctly classifies the data points belonging to our dataset.

Because of this we need to find the plane that has the maximum margin which could help to classify other data which are not labelled. For support vector machines usage of hyperplanes matter because those are the boundaries which help to classify the data points. In next paragraphs, we will be introducing our dataset and using support vector machines we will be defining number of input features after extracting them. If the number of input features is just 2, then the hyperplane is simply a line and if the number of input features is 3 then it becomes two-dimensional plane. We will be using support vector machines in order to maximize the margin of the classifier.

Chart

Description automatically generated

Figure 1 Support Vector Machine

In logistic regression, there’s a notion of sigmoid function, it is a bounded and real function that is defined for all real input values which has non-negative derivative at each point. In sigmoid function we take the range of [0, 1] where it gets value 1 when threshold value is greater than 0.5 and gets 0 when threshold value is less than 0.5. Support Vector Machines behave similarly to sigmoid function because when they get value greater than 1 we identify it in one class and if they get value -1 we identify it into another class. Thus, SVM gets the range of values [1, -1] that acts as margin.

Text, letter

Description automatically generatedAs we mentioned before, our main objective in using support vector machines is to maximize the margin between the data points and the hyperplane. Thus, there is one concept in support vector machines called ***loss function*** that helps us to maximize the margin in ***hinge loss***. When we calculate if the predicted and actual value have the same sign the cost is 0. However, if they do not hold the same sign, we have to calculate also the *loss function*.

Figure 2 Hinge Loss Function

## Decision Trees

Like SVMs, Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multioutput tasks. They are very powerful algorithms, capable of fitting complex datasets.

Decision trees are one of the types of Supervised Machine Learning which divides or splits the data continuously according to conditions which we store into nodes and leaves.

Leaves correspond to the classification function target, which is the label we want to assign to a set of data representing a particular situation. As mentioned, there are two types of decision trees which are classification and regression trees. In our project, we will be using classification trees because we need to classify our data and get as outcome the class of function to which the given instructions belong. The fundamental algorithm used for decision tree is called ***ID3*** which stands for *Iterative Dichotomiser 3*. ID3 algorithm take advantage of entropy in order to calculate the homogeneity of the sample. Moreover, ***Information Gain*** is used to check how entropy decreases after we split our dataset. In order to understand correctly decision trees, we have to go through few definitions.

**Entropy**

As we have already mentioned in previous paragraph, entropy is used to calculate the homogeneity of the sample state. In Machine Learning, it is frequently used as an impurity measure: a set’s entropy is zero when it contains instances of only one class.

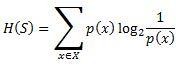


Figure 3 Entropy for Decision Trees

**Information Gain**

A reduction of entropy is often called an Information Gain.

It is used to check how entropy decreases after we split our dataset on any attribute. When we create decision trees, we have to check information gain and find the attribute that returns highest information gain.

Decision Trees modified

Figure 4 Information Gain for Decision Trees

**Implementation**

To implement the algorithm, we had to go through several steps. First of all we analyzed the provided dataset. It was a *json* file containing more than six-thousand functions belonging to four possible classes: Encryption, String Manipulation, Math and Sorting.

Text

Description automatically generated

Figure 5 The imported json dataset

Therefore, each item (row) of our dataset represents one of these functions having an attribute called “*semantic”*, in which the class is specified, and another attribute called “*lista\_asm*”, containing all the assembly instructions.

A picture containing text, receipt

Description automatically generated

Figure 6 A little sample of the json dataset; the lista\_asm is visible

Next needed step was about vectorize our dataset.

This step was necessary because programs are not capable of understanding words and sentences in the same manner as humans do. In order to make documents’ corpora more palatable for computers, they must first be converted into some numerical structure.

***Bag-of-Words*** is a very intuitive approach to this problem, the methods comprise of:

* Splitting the documents into tokens by following some sort of pattern.
* Assigning a weight to each token proportional to the frequency with which it shows up in the document and/or corpora.
* Creating a document-term matrix with each row representing a document and each column addressing a token.

The vectorizer objects provided by ***Scikit-Learn*** allow us to perform all the above steps at once efficiently, and even apply preprocessing and rules regarding the number and frequency of tokens. Even though many different versions are available, we are going to focus our analysis only on the main three encountered:

* **Count Vectorizer**: The most straightforward one, it counts the number of times a token shows up in the document and uses this value as its weight.
* **Hash Vectorizer**: This one is designed to be as memory efficient as possible. Instead of storing the tokens as strings, the vectorizer applies the hashing trick to encode them as numerical indexes. The downside of this method is that once vectorized, the features’ names can no longer be retrieved.
* **TF-IDF Vectorizer**: TF-IDF stands for “term frequency-inverse document frequency”, meaning the weight assigned to each token not only depends on its frequency in a document but also how recurrent that term is in the entire corpora.

For the aim of our project, we adopted the Count Vectorizer since it fit the most the situation we were facing.

After the data import and the vectorization, we split the dataset into a *Training Set* and a *Validation Set*. This is a fundamental step that allows you to train correctly your algorithm and avoid overfitting.

As mentioned in introduction, we have adopted Support Vector Machines and Decision Trees in order to train our algorithm. To apply and get the results for both of them we took advantage of S*klearn* python library.

As showed during the Machine Learning course, we built a *Classifier-Map* containing both the models. Hence, in order to select one precise model you can simply take it by referring to the key in the map characterized by a char (D or S).

Finally, we fit the chosen model and computed the accuracy. It is interesting to notice that SVM model got better accuracy (0.983) than DT (0.976).

For the subsequent step we measured the performances, so we implemented a confusion matrix.

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents

the instances in an actual class (or vice versa).

Chart

Description automatically generated

Confusion matrix simply takes the dataset as input and returns a table of four different combinations which are *true positive* (TP), *true negative* (TN), *false positive* (FP) and *false negative* (FN). Taking advantage of the Confusion Matrix we also computed other useful parameters: *Precision, Recall* and *F1-Score.*

*A picture containing text, receipt

Description automatically generated*

Figure 7 The other parameters computed

In figure 8 and 9 you can have a glance of the results obtained for SVM and Decision Tree approaches.

Table

Description automatically generated

Figure 8 Confusion Matrix and values for SVM approach

Table

Description automatically generated

Figure 9 Confusion Matrix and values for DT approach

In further steps, by using *matplotlib* python library we have tried to plot our data which we will be mentioning in results part.

**Evaluation**

We have already talked enough about Support Vector Machines and Decision Trees and included implementation part as well. For this section, our job is to include all the important cases we considered in order to evaluate the best accuracy. We have tried to evaluate the accuracy, precision and recall for our algorithms by using *sklearn* library.

Graphical user interface, text, application

Description automatically generated

*Fig 4.1. Implementing SVM*

As seen from the *Fig 4.1* we have achieved the accuracy of 0.86, precision of 0.83 and recall of 0.54 for the Support Vector Machines.

Graphical user interface, text, application, Teams

Description automatically generated

*Fig 4.2. Implementing Decision Trees*

Additionally, we also tried decision trees to classify our model and got the accuracy of

0.88 which was highest accuracy.

**Results**

As we have implemented our algorithm, the most important part is to check the results and see how accurate our algorithm is. As described enough above, we have implemented both support vector machines and decision trees to check the accuracy of our algorithm within our dataset. With the help of *sklearn* library we have used linear kernel and active probability in support vector machines and got the accuracy of 0.86 with our extracted dataset. Additionally, we have also implemented decision trees to check the accuracy by the help of *sklearn* library and got the accuracy of 0.88.

**Conclusion**

This paper has been created and made exhaustive research on implementing our algorithm to calculate the accuracy of the dataset after doing feature extraction and using Support Vector Machine and Decision Trees as fitting into model. Till the end of this project, we have successfully made a lot of research on these topics and tried to explore and understand how all of them works. In this report, we have presented how exactly those classification algorithms work and how we exactly implemented them. The accuracy of our algorithm is

higher than 70% by implementing 30000 samples. However, for the future work we will be trying to find ways to increase the accuracy for our algorithm.

**References**

1. Understanding Support Vector Machines
   * <https://en.wikipedia.org/wiki/Support-vector_machine>
   * [https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

[machine-example-code/](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

1. Decision Trees
   * <https://en.wikipedia.org/wiki/Decision_tree>

<https://www.sciencedirect.com/topics/computer-science/decision-trees>