SAPIENZA
UNIVERSITÀ DI ROMA

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 objective of the homework, we have studied the classification algorithms and decided to focus on Decision Trees and SVMs. The following report is 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 examples of 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 available 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 had the maximum margin which could help to classify other data which are not labelled. For Support Vector Machines, the usage of hyperplanes matters, because those are the boundaries which help to classify the data points. 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. Our goal is to maximize the margin of the classifier.

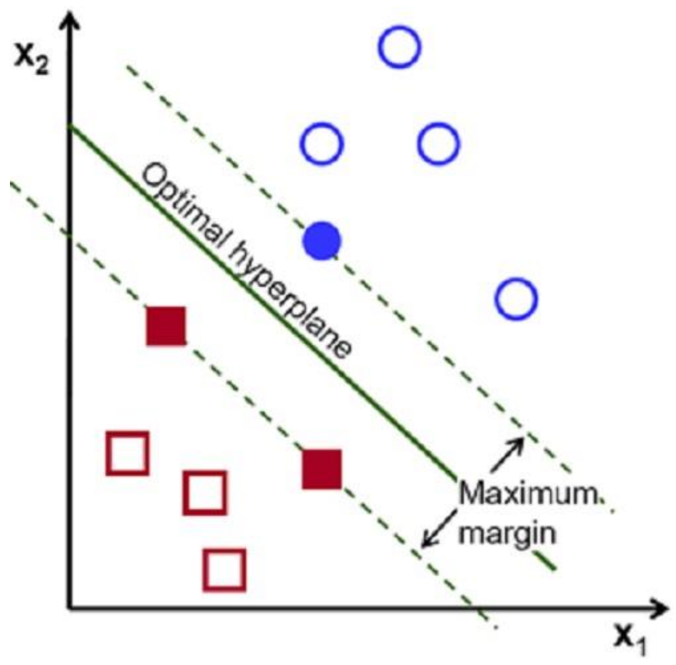


Figure 1 Support Vector Machine

In logistic regression, there's the 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.

As 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 an important value that we must consider: the *Loss Function,* that helps us to maximize the margin in *Hinge Loss*. When we calculate whether 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.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

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 takes 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.

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Figure 3 Entropy for Decision Trees

Information Gain

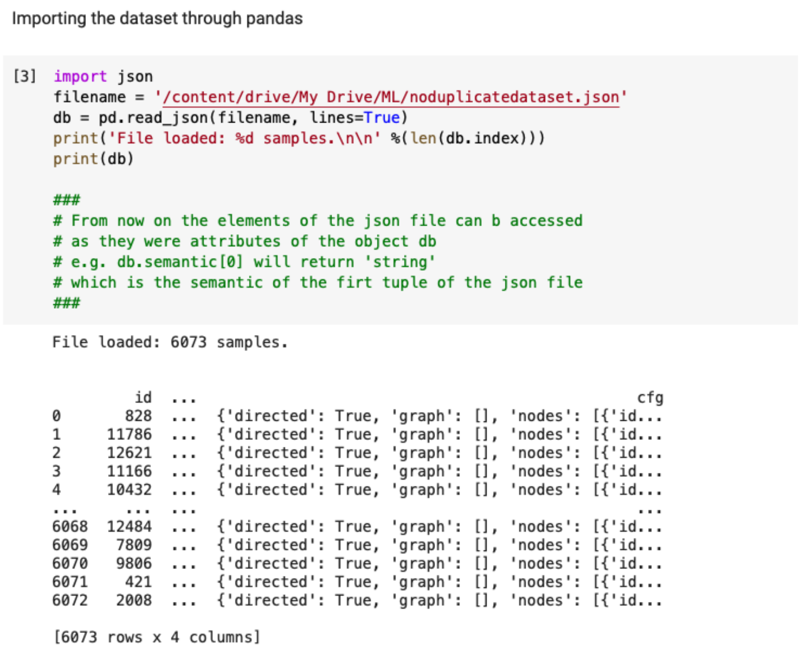
A reduction of entropy is often called an Information Gain.
It is used to check how entropy decreases after we split our dataset on any attribute. When we create decision trees, we have to check information gain and find the attribute that returns highest information gain.

$$IG(S, A) = H(S) - H(S, A)$$

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.


Importing the dataset through pandas

```
[3] import json
    filename = '/content/drive/My Drive/ML/noduplicatedataset.json'
    db = pd.read_json(filename, lines=True)
    print('File loaded: %d samples.\n\n' %(len(db.index)))
    print(db)

    ###
    # From now on the elements of the json file can b accessed
    # as they were attributes of the object db
    # e.g. db.semantic[0] will return 'string'
    # which is the semantic of the firt tuple of the json file
    ###
```

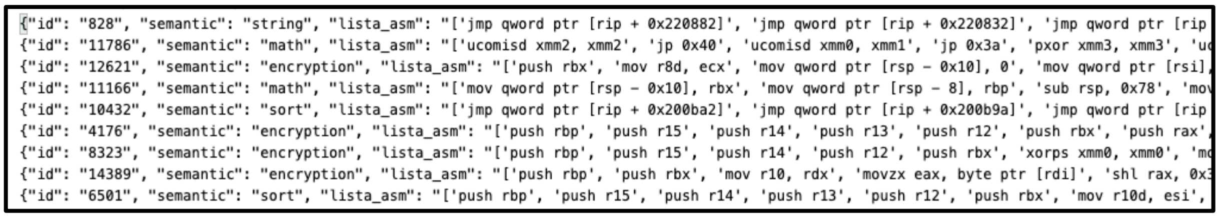
```
File loaded: 6073 samples.


         id  ...                                                cfg
0       828  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
1     11786  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
2     12621  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
3     11166  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
4     10432  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
...     ...  ...                                                ...
6068  12484  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
6069   7809  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
6070   9806  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
6071    421  ...  {'directed': True, 'graph': [], 'nodes': [{'id...
6072   2008  ...  {'directed': True, 'graph': [], 'nodes': [{'id...

[6073 rows x 4 columns]
```

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.



```
{"id": "828", "semantic": "string", "lista_asm": "['jmp qword ptr [rip + 0x220882]', 'jmp qword ptr [rip + 0x220832]', 'jmp qword ptr [rip
{"id": "11786", "semantic": "math", "lista_asm": "['ucomisd xmm2, xmm2', 'jp 0x40', 'ucomisd xmm0, xmm1', 'jp 0x3a', 'pxor xmm3, xmm3', 'uc
{"id": "12621", "semantic": "encryption", "lista_asm": "['push rbx', 'mov r8d, ecx', 'mov qword ptr [rsp - 0x10], 0', 'mov qword ptr [rsi],
{"id": "11166", "semantic": "math", "lista_asm": "['mov qword ptr [rsp - 0x10], rbx', 'mov qword ptr [rsp - 8], rbp', 'sub rsp, 0x78', 'mov
{"id": "10432", "semantic": "sort", "lista_asm": "['jmp qword ptr [rip + 0x200ba2]', 'jmp qword ptr [rip + 0x200b9a]', 'jmp qword ptr [rip
{"id": "4176", "semantic": "encryption", "lista_asm": "['push rbp', 'push r15', 'push r14', 'push r13', 'push r12', 'push rbx', 'push rax',
{"id": "8323", "semantic": "encryption", "lista_asm": "['push rbp', 'push r15', 'push r14', 'push r12', 'push rbx', 'xorps xmm0, xmm0', 'mo
{"id": "14389", "semantic": "encryption", "lista_asm": "['push rbp', 'push rbx', 'mov r10, rdx', 'movzx eax, byte ptr [rdi]', 'shl rax, 0x3
{"id": "6501", "semantic": "sort", "lista_asm": "['push rbp', 'push r15', 'push r14', 'push r13', 'push r12', 'push rbx', 'mov r10d, esi',
```

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 Sklearn 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).

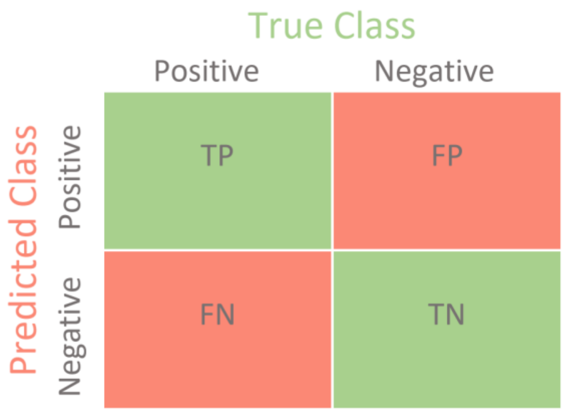


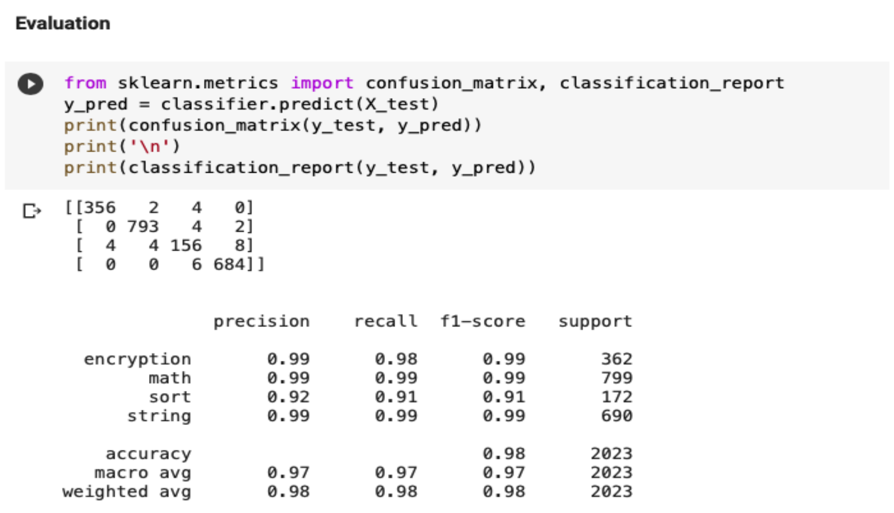
Figure 7 Confusion Matrix

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.

$$\begin{aligned} precision &= \frac{TP}{TP + FP} \\ recall &= \frac{TP}{TP + FN} \\ F1 &= \frac{2 \times precision \times recall}{precision + recall} \end{aligned}$$

Figure 8 The other parameters computed

In figure 9 and 10 you can have a glance of the results obtained for SVM and Decision Tree approaches.



Evaluation

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print('\n')
print(classification_report(y_test, y_pred))
```

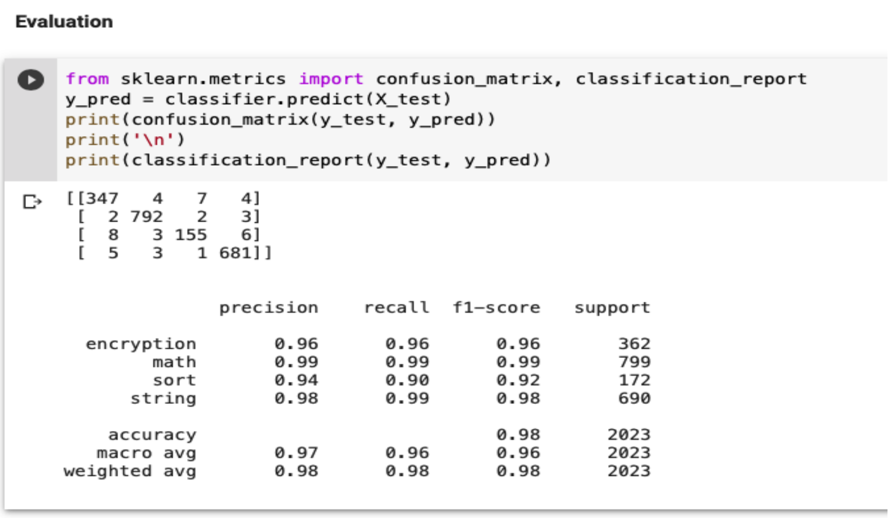
```
[[356   2   4   0]
 [  0 793   4   2]
 [  4   4 156   8]
 [  0   0   6 684]]

              precision    recall  f1-score   support

  encryption       0.99      0.98      0.99       362
        math       0.99      0.99      0.99       799
        sort       0.92      0.91      0.91       172
      string       0.99      0.99      0.99       690

    accuracy                           0.98      2023
   macro avg       0.97      0.97      0.97      2023
weighted avg       0.98      0.98      0.98      2023
```

Figure 9 Confusion Matrix and values for SVM approach



Evaluation

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print('\n')
print(classification_report(y_test, y_pred))
```

```
[[347   4   7   4]
 [  2 792   2   3]
 [  8   3 155   6]
 [  5   3   1 681]]

              precision    recall  f1-score   support

  encryption       0.96      0.96      0.96       362
        math       0.99      0.99      0.99       799
        sort       0.94      0.90      0.92       172
      string       0.98      0.99      0.98       690

    accuracy                           0.98      2023
   macro avg       0.97      0.96      0.96      2023
weighted avg       0.98      0.98      0.98      2023
```

Figure 10 Confusion Matrix and values for DT approach

Conclusion

We obtained a better accuracy with SVM, hence we decided to use this model to predict the blind set. In fact, beside the normal dataset, another set was given without labels. In order to verify whether our model was well fitted or not, we tested it over such set. The output (printed onto a txt file) is good. We can observe that there is a good proportion of all the possible functions (none is dominating the scene).