SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Machine Learning

2019 Fall

Nijat Mursali | 1919669

HOMEWORK 1

Rome, Italy

Table of Contents

List of Figures

Abstract

This report describes the building process and development phases of our project. During the lifetime of our project fundamental objective that we had to complete was conducting a deep research for the classification algorithms. There are several algorithms that has been done to solve classification problems. In this report we have tried to classify and find the best accuracy by importing dataset, feature extraction, usage of learning algorithms such as SVMs and decision trees and getting results from them.

Introduction

Before introducing our algorithm, we have to first understand what exactly classification means. Classification is of the ways of predicting the class of given data points in a sense of labelling classes into categories. In classification we must be sure about the task of approximating mapping function which takes input variables and converts into discrete output variables. Examples for classification problems can be speech recognition, document classification, spam recognition and so on. In speech recognition it takes the sound waves as input and through neural network it gives the output of plain text at the end. In spam recognition, as it is binary classification, we care only if the document is spam or not. As we have mentioned we will be using dataset in our algorithms, so this classification problem is one category of supervised learning which means we take labelled dataset with the input data. For solving these kind of problems, there are several classification algorithms already invented such as Support Vector Machines, Decision Trees, Neural Networks as so on. However, in this report we will be introducing SVMs and decision trees in order to solve classification problems. 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

In machine learning, SVMs are one of the ways of supervised learning models that simply analyse data used for classification and regression analysis[1]. Thus, the main objective of Support Vector Machines is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. Because of this we need to find the plane that has the maximum margin which could help to classify the future data points in our example. 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. In our report, we will be using support vectors in order to maximize the margin of the classifier.

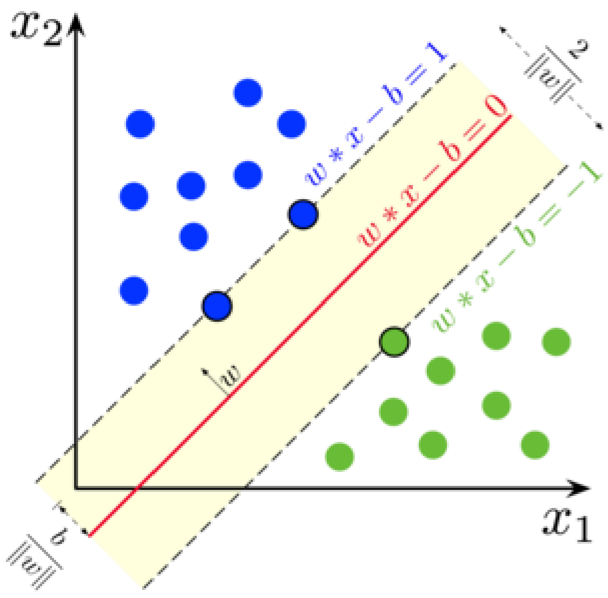


Fig 1.1. Support Vector Machine

In logistic regression, there’s a notion of sigmoid function 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. In support vector machines it is similar to sigmoid function because when it gets value greater than 1 we identify it in one class and if it gets value -1 we identify it into another class. Thus, SVM we get the range of values [1, -1] that acts as margin.

As we have mentioned in previous paragraphs, our main objective in using support vector machines is to maximize the margin between the data points and the hyperplane. Thus, there’s one concept in support vector machines called loss function that assists 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 don’t hold the same sign, in this case we have to calculate also the *loss function*.

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

Fig 1.2. Hinge Loss Function

In the implementation part of our report, we will be talking about how to use SVM in Python and how we exactly implemented it in our algorithm.

Decision Trees

Decision trees are one of the types of Supervised Machine Learning which divides or splits the data continuously according to a certain parameter where can be explained by nodes and leaves. We call them leaves because those are the decision or simply final outcomes in the tree. There are two types of decision trees which are classification and regression trees. In our report, we will be using classification trees because we need to classify our data and get the outcome of YES/NO type. The fundamental algorithm used for decision tree is called ID3 which stands for Iterative Dichotomiser 3. ID3 algorithm uses entropy in order to calculate the homogeneity of the sample. Additionally, IG which stands for information gain is used to check how entropy decrease after we split our dataset. As we will be using decision trees in our report, first 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. We first check the sample and if the sample is homogeneous then the entropy in that case should be equal to zero and if the sample can be equally dividable then we get the entropy one.

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

Fig 2.1. Entropy for Decision Trees

Information Gain

In addition to decision trees, information gain 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.

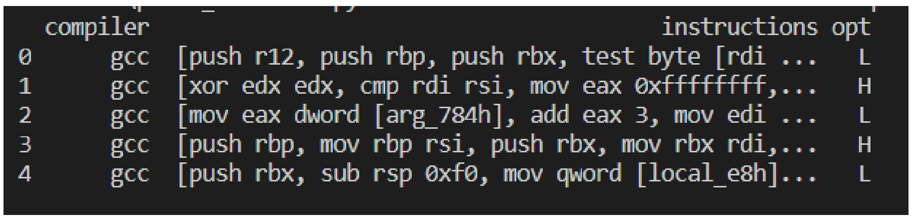
$$IG(S, A) = H(S) - H(S, A)$$

Fig 2.2. Information Gain for Decision Trees

Implementation

To implement our algorithm, we had to go through several steps. First step was to check our dataset which means what we have in our dataset and how we will be extracting it in next step. Our dataset is *jsonl* (which stands for JSON lines) contains 30 thousand functions with three different compilers. Each row of our dataset has different parameters and each row contains instructions, opts which are the ground truth labels for optimization and compilers.

Next step to implement our algorithm was to do feature extraction which helped us divide the parameters of the dataset into features. Thus, we take *train_dataset.jsonl* file in order to extract all the features into new file in order to get much better results and decrease the test time in the next steps.



```
  compiler                                         instructions opt
0      gcc  [push r12, push rbp, push rbx, test byte [rdi ...   L
1      gcc  [xor edx edx, cmp rdi rsi, mov eax 0xffffffff,...   H
2      gcc  [mov eax dword [arg_784h], add eax 3, mov edi ...   L
3      gcc  [push rbp, mov rbp rsi, push rbx, mov rbx rdi,...   H
4      gcc  [push rbx, sub rsp 0xf0, mov qword [local_e8h]...   L
```

Fig 3.1. Train Dataset

As seen from *Fig 3.1* we have three rows inside the dataset which are compilers, instructions and opts. When we extract features, we chose all the rows and extracted into new text file to use later.

For the next step we needed to measure the performance of our new dataset, so we have implemented confusion matrix. Confusion matrix simply takes the inputs and as an output it gives a table of four different combinations which are true positive, true negative, false positive and false negative. Knowing these values is helpful in order to calculate recall, precision and accuracy.

$$\textbf{Recall} = \frac{TP}{TP + FN} \qquad \textbf{Precision} = \frac{TP}{TP + FP}$$

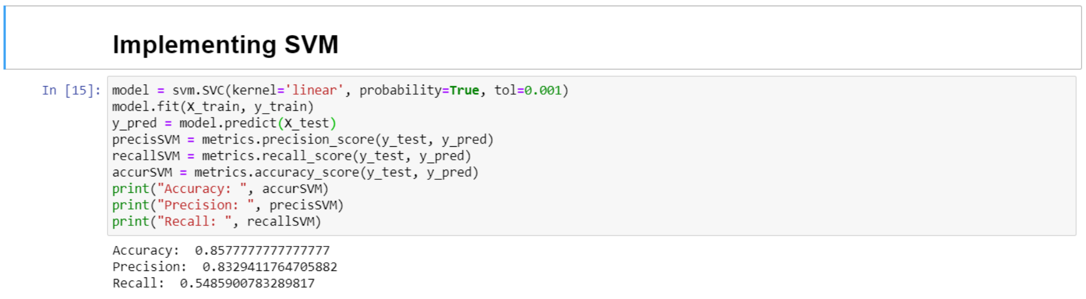
Fig 3.2. Recall and Precision

In further steps, by using *matplotlib* python library we have tried to plot our data which we will be mentioning in results part. As we have discussed in introduction part, we have used Support Vector Machines and Decision Trees in order to apply our algorithm. To apply and get the results for SVM we have used *sklearn* python library and added to model.

By using sklearn python library, implementation of decision trees became much easier because *sklearn* contains all the important functions in itself to compute the accuracy. To implement it using decision trees we have added *DecisionTreeClassifier()* and then we have tried to fit it into our train data and then find the accuracy by using *metrics*.

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.



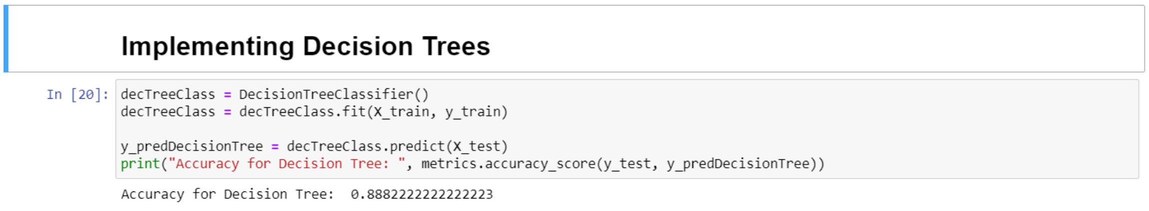
Implementing SVM

```
In [15]: model = svm.SVC(kernel='linear', probability=True, tol=0.001)
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         precisSVM = metrics.precision_score(y_test, y_pred)
         recallSVM = metrics.recall_score(y_test, y_pred)
         accurSVM = metrics.accuracy_score(y_test, y_pred)
         print("Accuracy: ", accurSVM)
         print("Precision: ", precisSVM)
         print("Recall: ", recallSVM)

         Accuracy:  0.8577777777777777
         Precision:  0.8329411764705882
         Recall:  0.5485900783289817
```

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.


Implementing Decision Trees

```
In [20]: decTreeClass = DecisionTreeClassifier()
         decTreeClass = decTreeClass.fit(X_train, y_train)

         y_predDecisionTree = decTreeClass.predict(X_test)
         print("Accuracy for Decision Tree: ", metrics.accuracy_score(y_test, y_predDecisionTree))

         Accuracy for Decision Tree:  0.8882222222222223
```

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-machine-example-code/

2. Decision Trees
   - https://en.wikipedia.org/wiki/Decision_tree
     https://www.sciencedirect.com/topics/computer-science/decision-trees