Lab 3: D7054E

Umuthan Ercan (umuerc-2)
Sergio Serrano Hernandez (serser-1)
Group 34

March 21, 2023

Abstract

This lab report provides information on how we fulfilled the requirements of part A from Lab 3, which was mainly focused on the implementation of a Machine Learning model of our choice, between Naive Bayes, Decision Trees, and K-Nearest Neighbors. We chose to implement decision trees on the provided "Diabetes"[2] data set, and followed the instructions.

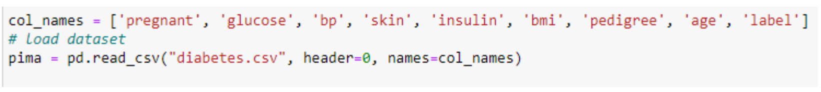
1 Introduction

Part A required us to choose between Naive Bayes, Decision Trees, and K-Nearest Neighbors. We were already familiar with KNN, and we both wanted to practice the implementation of Decision Trees in Python, so we chose to go with it. On the website[1] provided, we read and followed the instructions. We started with theoretical knowledge, the formulas were not so clear at first sight but after reaching code implementation, the topic made better sense.

2 Methodology

The methodology that we followed to implement this lab was pretty much the same as the one discussed in the tutorial [1], but we are going to walk the reader of this report through the process in any case:

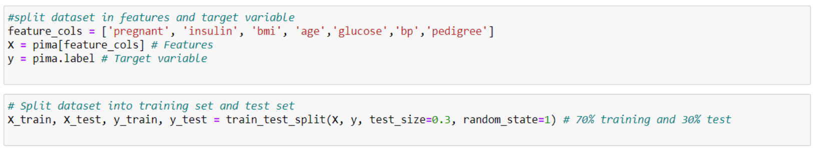
1. First and foremost, we import the necessary libraries to implement the decision tree, in this case pandas and sklearn.
2. The following step is to read the dataset with pandas, here we implement a change in regards to the shown in the tutorial, w change the parameter "header" from None, to 0. This change is due to a malfunctioning of our program in further blocks when setting "header" to None instead of 0, we believe that this is because in the tutorial they were using an older version of pandas in which the correct way of deprecating the given headers is with "header = None".



```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=0, names=col_names)
```

Figure 1: The change of the parameter "header"

3. We split the dataset in features and target variables. We split the previously splitted features and targets into training and testing datasets.

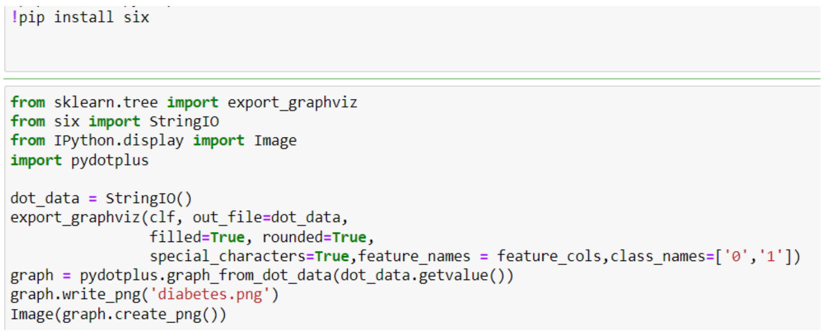


```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

Figure 2: The change of the parameter "header"

4. We instance the decision tree classifier object, train it, and test its accuracy, resulting in an accuracy of 67%.
5. After we have trained our tree, we would like to see how it looks like, hence we import the necessary libraries for visualizations and plot the decision tree as stated in the tutorial. Python could not find the module sklearn.externals.six, so we had to install the package six, and import StringIO from six.



```
!pip install six
```

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Figure 3: The code for plotting the tree

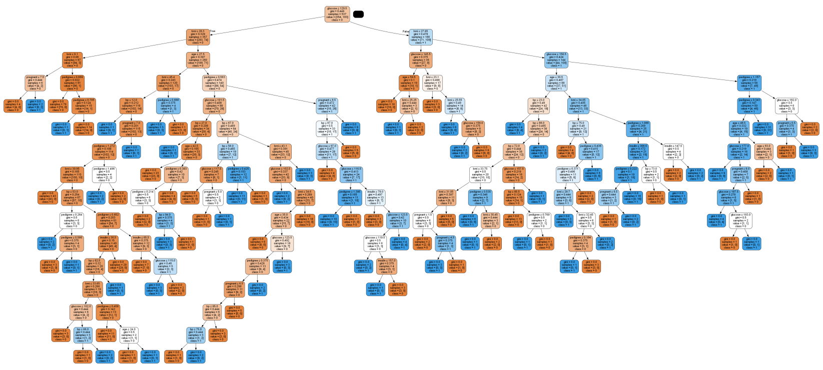


Figure 4: The tree plotted

6. Now that we have created and tested our very first decision tree, we tune it as it says in the tutorial changing the criterion form "gini" (this is the default one) to "entropy", and setting the maximum depth of the tree to 3. This increases the accuracy to a whopping 77%.

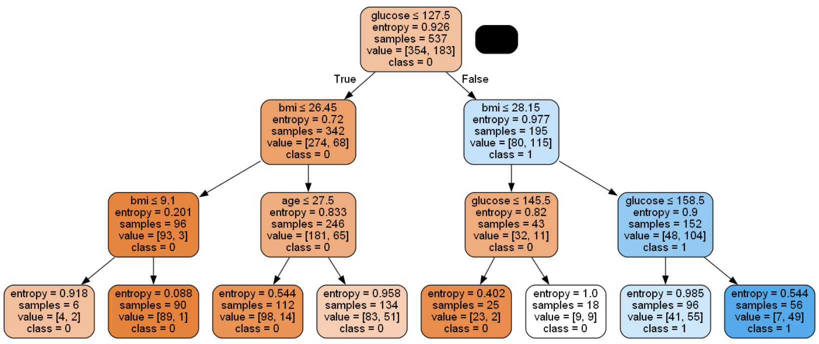


Figure 5: The tuned tree plotted

3 Results

The results that we can observe in this lab are the performance of the decision trees that we work with, for the first one with all the default parameters, we can see an already good accuracy of 67%, while in the second one just by tuning it slightly we can see an increase of the accuracy up to 77%.
Looking at the first decision tree plotted we can see how many branches and small leaves it has in the end, so reducing the maximum depth of the tree is very beneficial in order to not overfit to the training data. This kind of optimization that we are using is called pre-pruning, and it consists of stopping the learning process early before it produces leaves with very small samples.

4 Discussion

In the field of machine learning, decision trees are a popular method for classifying whether an instance belongs to one class or to other one. This is done by splitting the data into smaller subsets depending on the values of input features, until a certain classification is made. These splits will eventually decrease entropy and maximize the information gain.

A problem that arises when training such algorithms is that many splits are done to perfectly classify our training set, which will not cause any benefit when we try to classify new unseen data, namely the test dataset. There are several ways to work on this so that overfitting can be minimized, and in this lab we focus on pre-pruning, or stopping the learning process early in order not to have too many leaves.

In our experiment we can observe that pre-pruning increases the accuracy of our tree considerably, as well as it increases the explainability of the tree. We did not have to keep trying and just with one trial we could realize how beneficial pre-pruning is for decision trees in order to avoid overfitting. This is because there may be noisy samples in our training data that create leaf nodes which are unnecessary and lead to a bad performance.

Overall our results seem very promising in the favor of pre-pruning, and it would be exciting to see what a more intense pre-pruning can achieve in not only decision trees, but other bigger models such as random forests.

5 Conclusion

In conclusion, we have successfully implemented a decision tree model using the "Diabetes" [2] dataset and followed the instructions provided in the lab tutorial. We chose decision trees over Naive Bayes and K-Nearest Neighbors to practice implementing the model in Python. We split the dataset into training and testing data, trained and tested the decision tree, and visualized the tree using the necessary libraries.

We were able to achieve an accuracy of 67% with the default parameters, but we found that pre-pruning can significantly improve accuracy. By changing the criterion to "entropy" and setting the maximum depth of the tree to 3, we were able to increase the accuracy to 77%. We observed that pre-pruning is beneficial in order to not overfit to the training data, and it increases the explainability of the tree.

Overall, we gained valuable experience in implementing decision trees in Python, and we learned the importance of pre-pruning to avoid overfitting and improve accuracy. This lab was also very informative and entertaining. Even though we were required to implement just one machine learning method in this assignment, we decided to take a look at the Naive Bayes and KNN implementations on our own as well.

References

[1] "decision tree classification in python tutorial". https://www.datacamp.com/tutorial/decision-tree-classification-python#comments. Accessed: 21.03.2022.

[2] "pima indians diabetes database". https://www.kaggle.com/uciml/pima-indians-diabetes-database. Accessed: 21.03.2022.