

Project 3: Open Kaggle Competition

DS310: Machine Learning for Data Analytics

Team name: PA3-6

Team members: Ismael Diaz, Mike Ruff

Participation weights: Ismael (100), Mike (100)

Date: April 27th, 2020

Description of the competition

As students in DS 310, our primary responsibility for this competition is to maximize profits using the machine learning concept we learned in DS 310. The competition consists of roughly 27 teams who each receive the same training and testing datasets. Both datasets contain similar attributes for evaluating stock performance such as revenue, gross profit, and operating expenses. The only major difference is that the testing data does not have the column labeled “class” since it is our responsibility to develop a classification algorithm to predict whether the trader should buy the stock or not. The number 1 in the class column identifies stocks that the trader should buy, while a 0 represents stocks to not buy.

Team’s final rank in the leaderboard with the screenshot

Public Leaderboard

















Private Leaderboard

This leaderboard is calculated with approximately 30% of the test data.

The final results will be based on the other 70%, so the final standings may be different.

Raw Data

Refresh

#	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	PA3-1		<div>     </div>	0.52631	19	6h
2	PA3-60		<div>    </div>	0.51567	11	2d
3	PA3-20		<div>    </div>	0.51393	31	9h
4	PA3_13		<div>    </div>	0.49707	9	1h
5	PA3-11		<div>  </div>	0.46370	10	19h
6	PA3-6		<div>   </div>	0.45901	7	7h

Your Best Entry ↑

Your submission scored 0.45901, which is an improvement of your previous score of 0.40816. Great job!

Tweet this!

Team's solution in details

Our team decided to construct a decision tree classification algorithm for many reasons. From previous projects that we conducted in DS 200, we were able to reference some of the code and combine it with key concepts learned in DS 310. Amongst other classification algorithms, decision trees can be simple to code, visualize, and interpret. Choosing to construct a decision tree was also ideal since it's capable of handling problems with multiple features, meanwhile finding important insights.

Pre-processing

Our group decided to pursue the pre-processing approach that was given in the sample submission. We found this approach useful since it was able to impute the data with any missing values. This method used imputation and mean value from the sklearn module. Rather than just replacing nan values with a zero this method allowed for the mean of the value's column to be used. This method is more useful in a prediction model because it gives a more accurate classification when the data is inputted into a model. The mean replacement value is more representative of the data than a zero value.

Feature engineering

We were able to transform the raw data into features to better understand the model with a few commands. The `pd.DataFrame` allowed us to construct a data frame from a dictionary that was inputted. This ultimately allowed us to clearly label our X and Y variables prior to using sklearn to split the data into the training set and testing set. An important part of the feature engineering was to store the feature names in a separate list. This list will then be used in the decision tree model as part of the node creation. We decided to use all 221 features in the data to achieve the best outcome in our model. This seems like it may be a large amount of data to be used for the model, but we can control this in the construction of the model by adjusting `max_depth`.

Model building and comparison

In order to build the model, we imported the sklearn and `train_test_split` model, which has four parameters. X is the first parameter which is all the imputed data. Y is the second parameter and is the model's target output data, Class. The third parameter "`test_size`" represents a percentage of input and output data. Lastly, the fourth parameter "`random_state`" is used to randomly split the model's input and output data for the training set and testing set.

In addition, we also imported the "`DecisionTreeClassifier`" to train the decision tree. We were able to input/output training data then specify various parameters to construct the decision tree such as `criterion`, `max_depth`, and `min_samples_leaf`. We found `max_depth` to be the most interesting since it impacted the overall f1 measures.

To calculate the metrics we used the `sklearn.metrics.classification_report`. This function outputs various metrics about training and testing data. Specifically, it outputs the average F1 score, precision, and recall. We found that when the F1 score for the 0 (don't buy) class and 1 (buy) class have a smaller difference the results are better for the competition. This paired with a high weighted average F1 score led to the best results for our model and data.

Hyperparameter setting and tuning

As mentioned previously there were many parameters that we had to establish prior to running the model. After doing some research we decided to set the parameter “test_size” to 0.2 since we wanted to do an 80/20 split. We also decided to initiate random_state to 100, to ensure the consistency of the model.

As previously mentioned the max_depth parameter impacted the F-1 score and other performance metrics. We achieved the best performance for our model when max_depth was set to 7. This depth reduced underfitting and overfitting affecting our model. Specifically, at max_depth of 7 the F1 score for our testing data was .70. Most depths that we used were .67 or below for the F1 score. (Include information about metrics.confusion)

		precision	recall	f1-score	support
t					
	0	0.79	0.79	0.79	51
2	1	0.41	0.42	0.42	18
0					
	micro avg	0.70	0.70	0.70	69
2	macro avg	0.60	0.60	0.60	69
2					
	weighted avg	0.70	0.70	0.70	69
2					

```
: metrics.confusion_matrix(y_test, prediction_testing1)
```

(F1 score of .7 at max_depth =7)

At a max_depth = 3, we achieved an average F1 score of 0.63. At a max_depth of 20 we achieved an average F1 score of .68.

Lessons learned

There were many lessons learned throughout our project. One very important lesson that we learned is the importance of choosing and adjusting the max_depth. Once we constructed our model we adjusted the max_depth to achieve an F1 score that was acceptable. Max_depth has a large influence on the metrics produced by the model. In the future, we plan to implement k-fold into our algorithm and conduct partitions for the feature selection, to help increase our F-1 score.

Required meeting attendance log

We met 6 times to work on the project, and both members were present for each meeting.

