DMA Spring 19: Lab 5 (Kaggle Competition)

Team Name

OG Lytics

Team Members

Mention your team members' names here (maximum 4). Please mention them in the format *LastName, FirstName*.
For example: Potter, Harry.

1. Vicente Valdez, Pedro Junior
2. Patel, Arjun
3. Namgung, June

Data Preprocessing

Describe your view of the data. How did you explore the data? What did you find interesting? Did you have to clean the data? If yes, how did you?
Feel free to include any visualizations you used for data exploration.

Support Vector Machine Model:
After doing a basic exploratory data analysis (EDA) and looking at the ranges and type of values in the dataset a lot of decisions were taken regarding how to represent the data for the SVM model. From theory SVM model works better with normalize systems. Also since dataset was not distributed equally among all product categories a stratified train test split was taken to best represent the data. (See Figure 1).

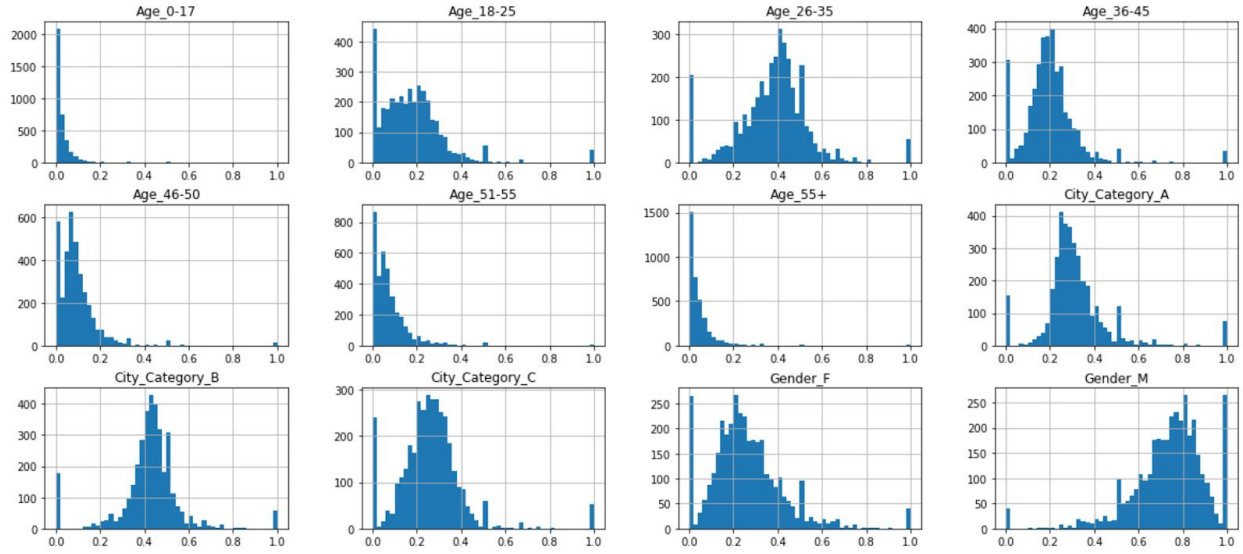


Figure 1. Histograms of some of the features. It is clear that may of the attributes are skewed meaning that in order to properly model it we need stratified splitting.

Since Gender, Age and City Category represent categorical data, one hot encoded was utilized in those columns. The columns 'Occupation', 'Stay In Current City Years' , and 'Purchase' were normalized using Min-Max normalization. After performing all the preprocessing and data cleaning on the whole dataset we group by Product_ID by aggregate the number of Users for a specific Product ID and took the mean of all the other columns.

Decision Tree:

Doing preliminary data pre-processing allowed us to see higher correlations of accuracy with different features. Because decision tree are practical in the way they choose the different splits in the data, we were able to find that the most meaningful column such as ‘Purchase’ or

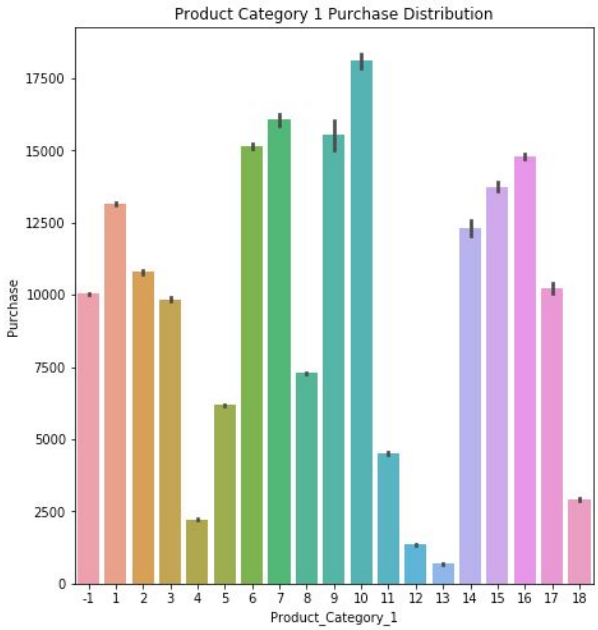
‘Occupation’ resulted in meaningful predictions. Unlike neural networks, decision trees were more fitting to add to our ensemble method because of the understandable differences under the hood.

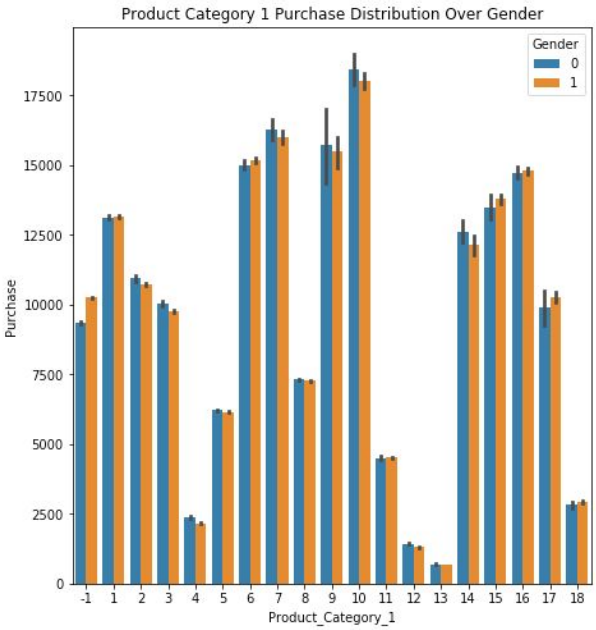
Looking at the way decision trees work, we understood that using pd.get_dummies on various meaningful columns would be ideal. Thus, we converted them as such and added them to the train split and then made predictions on them. We cleaned some of the columns that had many null values, and then did further preprocessing, grouping the dataset per `Product_ID` and aggregating based on the most meaningful and effective methods. This included getting the average and the first element of each group. After getting a low score on the validation split, we tried adding additional columns and using different aggregation functions for each for further diversity and better splits..

Random Forest:

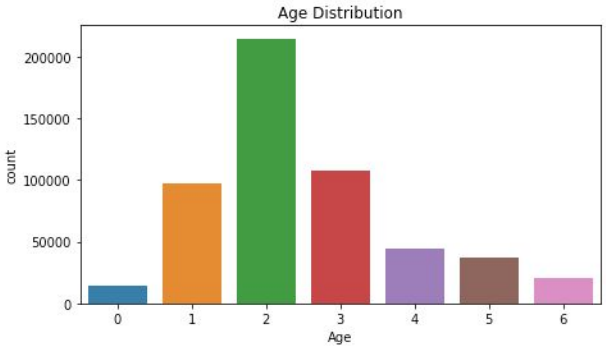
There were total 13 different columns in the original dataset. Besides the first index column called ‘unnamed’ and our target column ‘Product_Category_1’, there were 11 different columns that we can utilize.

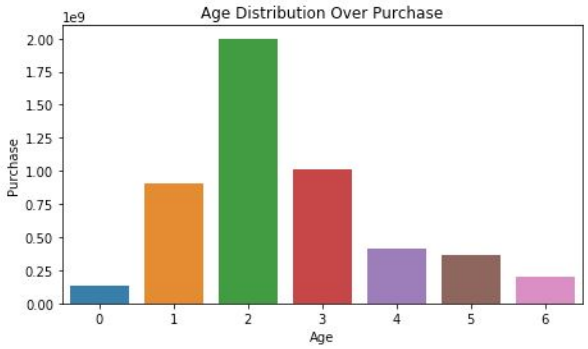
The original data has 75.41% of male, and 24.59% of female. So I thought the distribution of ‘Product_Category_1’ maybe differ over gender, but it was not.

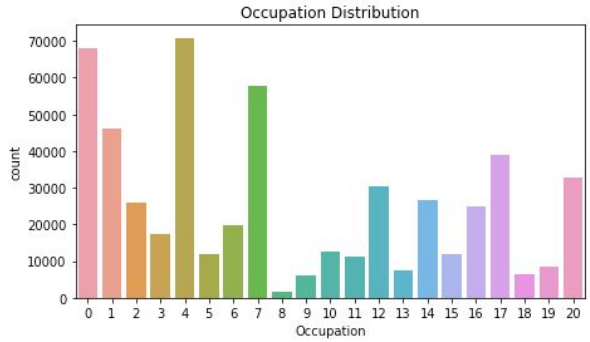


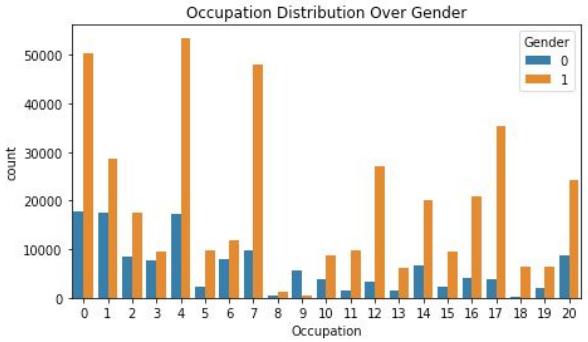


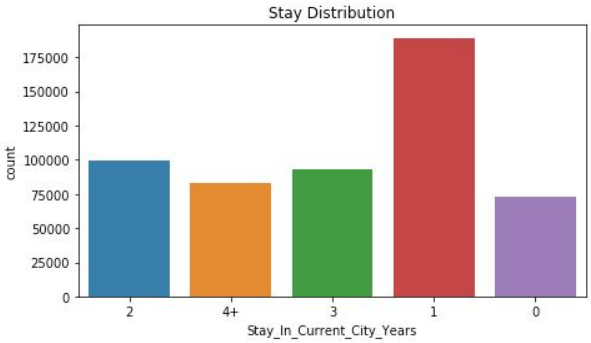
The plots below are relationship between some columns such as Age/Purchase, Gender/Occupation and City/Stay_In_Current_City_Years.

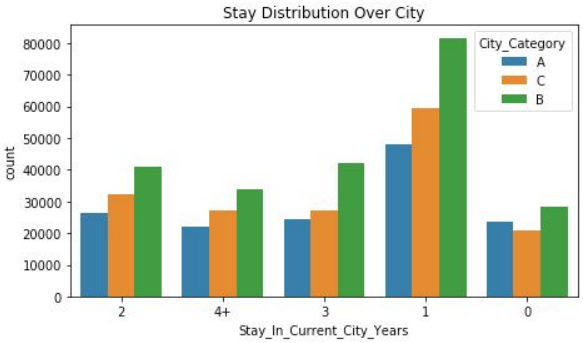












We wanted to see the correlation between all the columns, so we used LabelEncoder's fit_transform for preprocessing string data.

Feature Engineering

What features did you decide to use? Why? How did you extract those features?

Support Vector Machine Model:

As mentioned previously in the preprocessing one hot encoded was used in all categorical features. It was found that training the support vector machine using the features as they are resulted in an accuracy of the model was low at around 30%. After researching the literature

and recalling what was learned in class, Machine Learning models work best in scaled data (normalized data). Since we had numerical attributes with big numerical ranges, the categorical data that was one hot encoded (0's and 1's) was given negligible weights (not being used by the model) because of their small values. Therefore, in order to effectively take advantage of the categorical data we had to scale the other values to same range therefore the decision to normalize the data using a min max scaler. This boosted the accuracy up to ~70%.

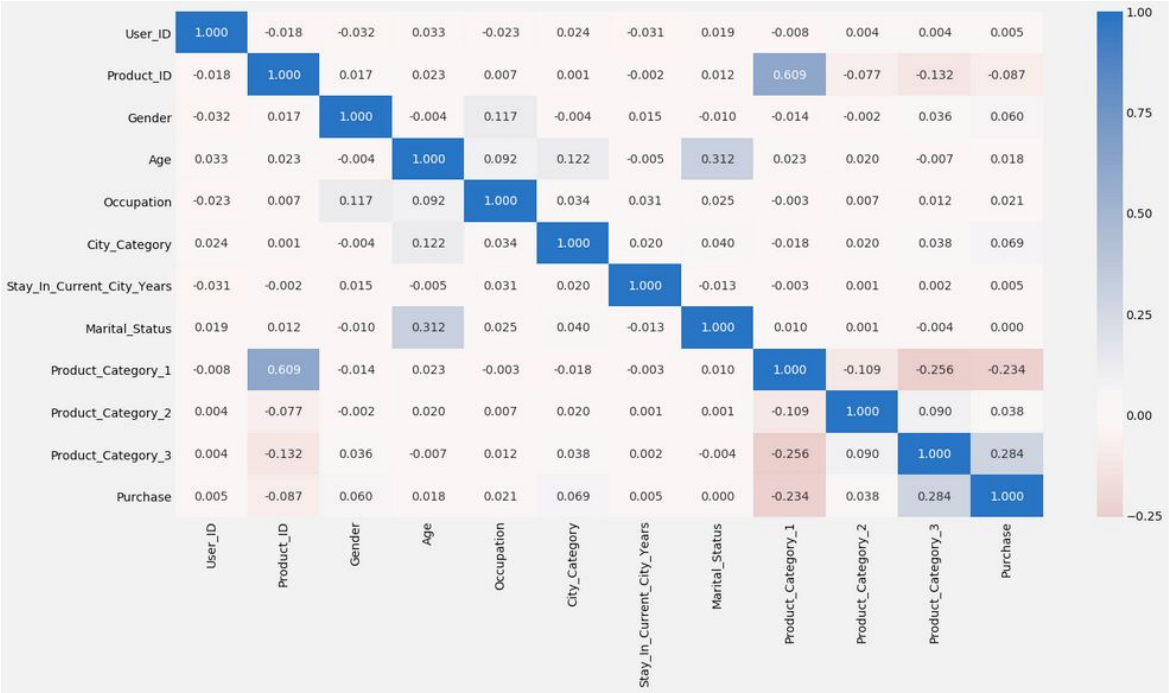
Decision Tree:

Most of our features were converted using one-hot encoding on the columns. The most important features for our configuration of the decision tree included the `Age`, `Occupation`, and `Purchase`columns. After grouping the data frame by product_id, we used a combination of taking the first row of the group and other aggregation methods. In our opinion, taking a step back from only looking at accuracy score and different adjustment methods was what brought us to optimizing the decision tree classifier. We thought that purchase would inherently more useful for a product category, so processed that and used it as a feature. It was interesting to see that this column boosted the accuracy score from ~25% to ~75% on the validation set.

#add your purchase column method or process and who it boost performance - cool got it

Random Forest:

As above, we wanted to see the correlation between all the columns, so we used LabelEncoder's fit_transform for preprocessing string data. Then we used correlation matrix as below.



For 'Product_Category_1' column, Product_ID, Product_Category_2, Product_Category_3 and Purchase column looks somewhat correlated. We ran a random forest method first and see the

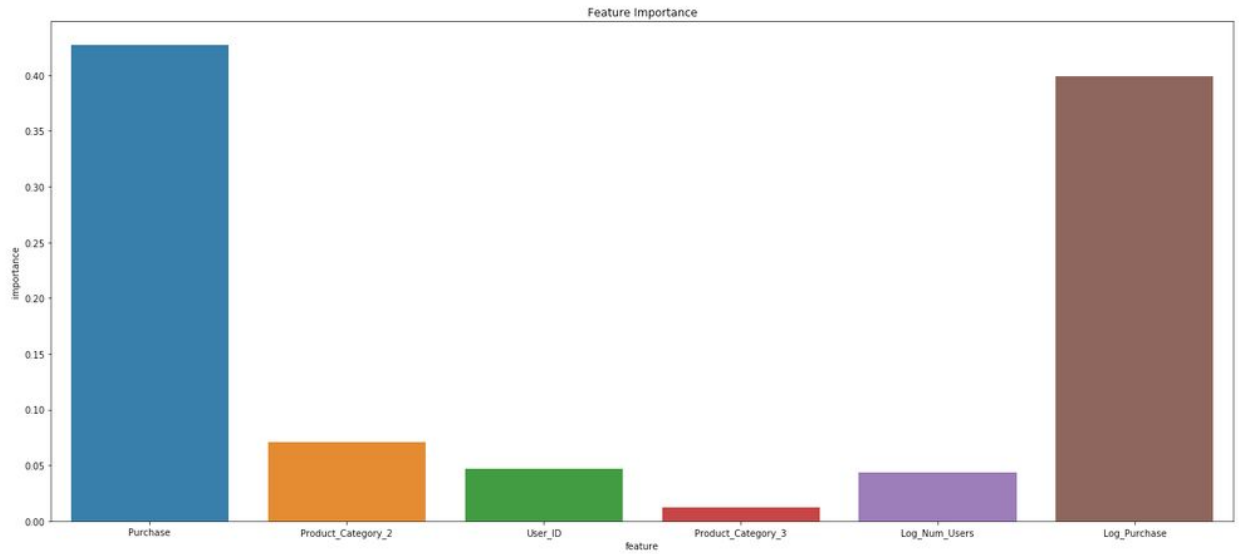
feature importance to follow up with this correlation matrix. While doing this step, we grouped by Product_ID column with Product_Category_2, Product_Category_3, User_ID count plus log value of user count and log number of Purchase column. Personally, we thought this step is the most important thing in this assignment. For this method, we played with many options and ended up choosing the maximum value from purchase. This eventually made a lot of improvements than other approaches. And for Product Categories, we picked the most frequent value. We picked the number of users as well. We did not use any other features but these. In addition to these ways, we added log converted columns with user_count and purchase.

```
target = (df.groupby(by='Product_ID', as_index=False)
          .agg({'Purchase': lambda x: x.max(),
                'Product_Category_1': lambda x: x.value_counts().idxmax(),
                'Product_Category_2': lambda x: x.median(),
                'User_ID': pd.Series.nunique,
#                 'Gender': lambda x: x.value_counts().idxmax(),
#                 'Age': lambda x: x.value_counts().idxmax()
#                 'City_Category': lambda x: x.value_counts().idxmax(),
#                 'Occupation': lambda x: x.value_counts().idxmax(),
                'Product_Category_3': lambda x: x.value_counts().idxmax()
                }))

target['Log_Num_Users'], target['Log_Purchase'] = np.log(
    target.User_ID), np.log(target.Purchase)
```

Ended up getting almost the same result from correlation matrix. It was a good follow up.

```
Feature ranking:
              feature  importance
0            Purchase    0.426791
5        Log_Purchase    0.399147
1  Product_Category_2    0.071066
2             User_ID    0.046680
4       Log_Num_Users    0.043640
3  Product_Category_3    0.012676
```



Models / Techniques

Which models did you use? What was your rationale for using them? Were there any specific models that you found to be better suited for this problem (either theoretically or experimentally)?

Support Vector Machine Model:
Since the we had a complex problem (trying to predict product category) based on features like Age and Gender we had to make the assumption that this problem was not going to be able to be modeled linearly. But what if it is? This is why one of the models selected for the problem was a support vector machine. If the problem was able to be linearly model the support vector machine would capture this. This was not the case. The SVM was tuned using GridSearchCV going through linear, poly and rbf kernels. The penalty factor ranged from 0 to 1000. For the rbf and poly kernel, the kernel coefficient ranged from 0.01 to 0.5 and for the poly the degree was searched from 3 to 8, the gridsearch was performed using three cross validations. If the linear model were to be the best one it would be apparent in the best parameters function but it was not. Instead the problem was indeed complex and had to be solved using an rbf kernel.

DT:

Our decision tree actually had a training accuracy score of 100%. By adding certain key features which were really important for the training set itself, the model became overfitted. However, it still faired well for our validation split, which brought about a score of 75%. This is both surprising and expected, as decision trees are known have better results with practical features that make sense at the data science problem level. When looking at the main issue of predicting Black Friday products, certain columns are more meaningful and bring better results. Decision trees did not really prove to be highly successful in the ensemble methods we tried at the start. For example, using an average between the SVM and decision tree, and then rounding did not do much to improve our accuracy on Kaggle. In terms of configuration, we started by testing different max_depths for the classifier, but decided to simply let the decision tree to go to pure leafs for the maximum accuracy. Since the set was extremely performance heavy, doing so will enable the highest accuracy in unseen data.

Basically, it's a classification problem. We tried four different approaches. First, we tried some methods we learned from class, such as Decision Tree, Random Forest. We were also curious with Decision Tree with bagging and Random Forest, so we tried it. Lastly, we used voting classifier with above three methods.

Theoretically, we thought Random Forest will have a better accuracy than Decision Tree.

Results

How did you evaluate your model? What metrics did you use? What were your results? If you tried out different models (as is expected), feel free to include all of your results.

Support Vector Machine Model:
As mentioned previously the model performed poorly when one hot encoding the categorical attributes and not normalizing the remaining numerical attributes. The accuracy (which was the metric used to evaluate the model since it was the metric to be used in kaggle) was in the order of 30%. After normalizing the data to a range of [0, 1] using a Min Max scaler the model performance boosted with an improvement of 40% in accuracy having ***a final accuracy of ~70%***. The final model after hyperparameter tuning and grid searching using cross validation (cv=4) was the following:

clf_svm = SVC(C = 100, kernel='rbf', gamma=0.4)

Where the C represents the error penalty, kernel the model used in hyperspace and gamma means the kernel coefficient range.

Decision Tree:

As we discussed before, our decision tree reacted best to using features that really mean something in the real world. For instance, the `Purchase` column in general would be a really good indicator of a product. Thus, using this column really improved the splits in the dataset. Using the accuracy score was ideal as this was used in the actual competition, as well as a very common way to judge a model's effectiveness. For the training set, as we mentioned before, we had an accuracy of 100% which is not surprising as we did not set a limit on the depth of the tree. Overfitting still didn't decrease performance in the validation set that we had an accuracy score of ~75%, which coupled with other models would do a great deal of differentiation.

These are the results from each different classifiers below. We compared classifiers' predictions with y_test which is a gold label for given X_test.

- Random Forest
    - Accuracy Score of Random Forests on train set 100.0 %
    - Accuracy Score of Random Forests on test set 94.11027568922306 %
- Decision Tree
    - Accuracy Score of Decision Tree on train set 100.0 %
    - Accuracy Score of Decision Tree on test set 95.6140350877193 %

- Bagging with Decision Tree
    - Accuracy Score of Bagging on train set 98.70210135970335 %
    - Accuracy Score of Bagging on test set 94.9874686716792 %
- Voting Classifier with three approaches
    - Accuracy Score of Voting Classifier on train set 100.0 %
    - Accuracy Score of Voting Classifier on test set 95.6140350877193 %

In the meantime, we wanted to know time efficiencies with these approaches and this is the follow up.

```
CPU times: user 5.1 s, sys: 27.3 ms, total: 5.13 s
Wall time: 5.18 s
Random Forest:  Score= 0.20589
CPU times: user 192 ms, sys: 1.95 ms, total: 194 ms
Wall time: 198 ms
Decision Tree:  Score= 0.21630
CPU times: user 4.92 s, sys: 19.2 ms, total: 4.94 s
Wall time: 4.98 s
Bagging with Decision Tree:  Score= 0.19815
CPU times: user 10.2 s, sys: 41.7 ms, total: 10.3 s
Wall time: 10.4 s
Voting Classifier:  Score= 0.19203
```

We used RMSLE (Root Mean Squared Logarithmic Error) with testing. In terms of evaluating scores, lesser the better. Guess RMSE can be a good alternative.
Obviously Decision Tree was the fastest and Voting classifier was the most time consuming method. This was expected because voting classifier performs all the other approaches under the hood. Bagging with Decision Tree took less time than Random Forest.

Conclusion

Summarize in short your experiments with the data and the results you obtained.

At first, we tested using just the SVM and decision tree models and using a simple combiner. We averaged the result of the two models and rounded the values to receive our final prediction. This did a decent job with a score of ~70% accuracy in the test set. But, we knew that we could do better with another model and also tweaking our ensemble method.

After training the models independently and tuning the hyper parameters using the logic mentioned in the Models/Techniques and Results section, we decided to use the ensemble method of majority vote. The combined modeled using the SVM, decision tree, and random forest gave an accuracy of 0.95580 when using majority vote. This was the same accuracy

obtained by performing the voting classifier using a decision tree with bagging and random forest.

In conclusion, there are a variety of ways to do ensemble and still obtain similar results. In this case we trained different models using Support Vector Machines, Decision Trees and Random Forests, Decision Trees with bagging and Random Forest, etc. At the end we decided to go for the latter which had the same performance without having to tune the SVM and DT by themselves.