**COMP 472**
**ARTIFICIAL INTELLIGENCE**

# MINI-PROJECT 2 REPORT

**Submitted to**
Professor Leila Kossom

**Submitted by**
Ryan Nichols 29787739
Jiayin Liu 27532628

November 18, 2018

## Experimental Setup

The experiment is performed using the Python library Scikit-learn which contains many built-in ML algorithms, among which Decision Tree, Naive Bayes, and Random Forest have been used in the scope of this project.

### Decision Tree

The hyperparameters tuned in Decision Tree classifier are:

- **Criterion**: this parameters has the options of Entropy and Gini Impurity. The Entropy values in information gain, whereas Gini Impurity computes for the amount of impurity in a set, ie. the possibility of an item being classified in the wrong set [1]. Some studies however mentioned that, in practice, both criteria "[...]disagree only in 2% of all cases [4]."
- **Max_depth**: defines the maximum depth of the tree. If not defined, the tree will be expanded until all the leaves are pure.
- **Max_features**: defines the maximum number (or portion) of the features pool to consider for the best split
- **Min_samples_leaf**: minimum number of samples needed to define a leaf.
- **Random_states**: set to a fixed integer so that result would be the same for identical hyperparameters configuration

The last 4 parameters are defined to help preventing overfitting of the data model.

During the experiment, for both datasets, a set of combinations of the above hyperparameters have been tested to obtain the combination that yields the highest accuracy.

### Naive Bayes

There are many different classifiers available in the sklearn framework for Naive Bayes. The BernoullieNB model was chosen because it is optimized for data with binary features.This model has the parameters:

- **Alpha**: For smoothing
- **Binarize**: Option for mapping feature values to boolean if they meet a certain threshold
- **Fit_prior**: For non uniform priors if true will learn class probabilities
- **Class_prio**r: Priors are normally adjusted to fit the data but can be specified here[8]

From reviewing the available parameters above it is expected the alpha or smoothing parameter should be the most important and indeed this can be backed up by experimental data. The parameter **binarize** appears unimportant since our data we know is binary. Parameter **class_prior** if set is not adjusted according to the data so doing so wouldn't be taking advantage of those automatic adjustments which leaves **alpha** as well as **fit_prior**.

### Random Forest

The Random Forest algorithm is a combination of Decision Tree and bagging method, the algorithm creates and merges multiple decision trees into one final tree which will yield stable and accurate predictions. Similarly to Decision Tree, Random Forest applies the concept of splitting on the most important feature over groups of random features, then combine the groups together, forming a wider and shallower tree classifier model.

The reason on choosing the Random Forest is that this algorithm is easy to use, as its parameters and basic concept are very similar to that of Decision Tree, and it provides results with high accuracy even without much tuning.

Hyperparameters to analyze[6]:

- **N_estimators**: number of subtrees to build. More subtrees will generally result in higher stability in prediction.
- **Max_features**: number of features considered when computing for best split
- **Min_sample_leaf**: minimum number of leafs needed to split a node

## Results Analysis

**Accuracy of Untuned Models on Test Sets**

| | Decision Tree | Naive Bayes | Random Forest |
|---|---|---|---|
| Dataset 1 | 0.29571984435797666 | 0.5992217898832685 | 0.4163424124513619 |
| Dataset 2 | 0.7665 | 0.8005 | 0.838 |

**Accuracy of Best Tuned Models on Test Sets**

| | Decision Tree | Naive Bayes | Random Forest |
|---|---|---|---|
| Dataset 1 | 0.3345 | 0.610894941634241 | 0.5292 |
| Dataset 2 | 0.774 | 0.804 | 0.875 |

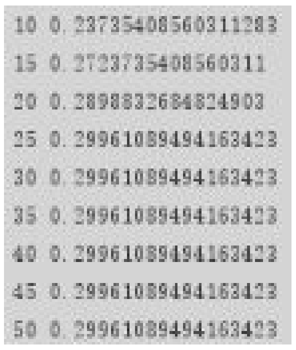
<u>Decision Tree</u>

Among the tested algorithms, Decision Tree outputs the lowest accuracy. Decision Tree tries to branch the features by determining on their purity or on information gain, and by doing so, the features could be divided into very specific and small groups, expanding the tree to deep levels, thus overfitting the training data.

By tuning the hyperparameter *Max_depth* while running Dataset 1 and 2, it has been observed that while all other parameters are the same, Dataset 1 needs more depth (20~25) than Dataset 2 (15~20) in order to yield a higher accuracy. Even after pruning, the classifier with too many features (Dataset 1) still overfits the training data, while the one with much less features (Dataset 2) exhibits less of the overfitting characteristic.

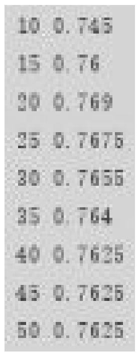
Results showing max_depth and accuracy:

Dataset 1



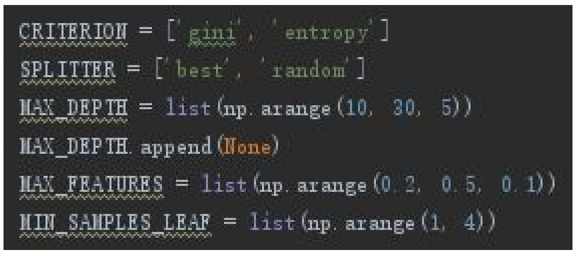
```
10 0.23735408560311283
15 0.2723735408560311
20 0.2898832684824903
25 0.29961089494163423
30 0.29961089494163423
35 0.29961089494163423
40 0.29961089494163423
45 0.29961089494163423
50 0.29961089494163423
```

Dataset 2



```
10 0.745
15 0.76
20 0.769
25 0.7675
30 0.7655
35 0.764
40 0.7625
45 0.7625
50 0.7625
```

The hyperparameters *Criterion, Splitter, Max_depth, Max_features,* and *Min_samples_leaf* have been evaluated on their contribution to the classifier model's accuracy. All tests have been conducted using classifier being configured with the hyperparameter under observation, while setting *random_state*=0 to ensure that the output stays constant throughout each run.



```
CRITERION = ['gini', 'entropy']
SPLITTER = ['best', 'random']
MAX_DEPTH = list(np.arange(10, 30, 5))
MAX_DEPTH.append(None)
MAX_FEATURES = list(np.arange(0.2, 0.5, 0.1))
MIN_SAMPLES_LEAF = list(np.arange(1, 4))
```

Tuning the above hyperparameters only increases the accuracy by a very negligible amount for both test sets 1 and 2.
For example, using different criterion:

| Dataset 1 | | Dataset 2 | |
|---|---|---|---|
| Gini Impurity | Entropy | Gini Impurity | Entropy |
| 0.2801 | 0.3035 | 0.7665 | 0.754 |

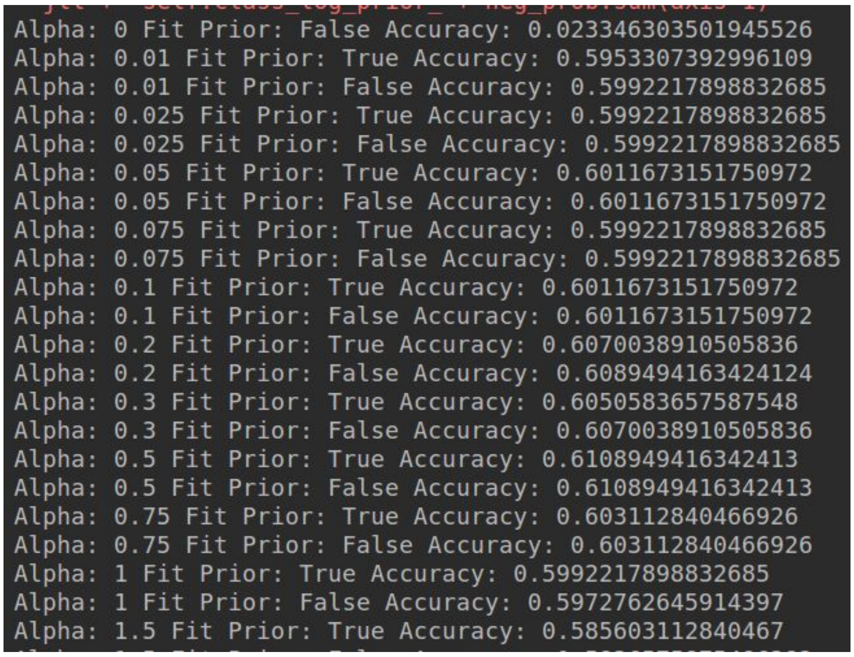
The difference between models using either criterion is very minimal, confirming the statement that these 2 criteria only have ~2% difference in practice.

The rest of the hyperparameters are used to prevent overfitting, after trying all the combinations of these parameters, the best accuracy of a tuned classifier improves by a small increase.

## Naive Bayes

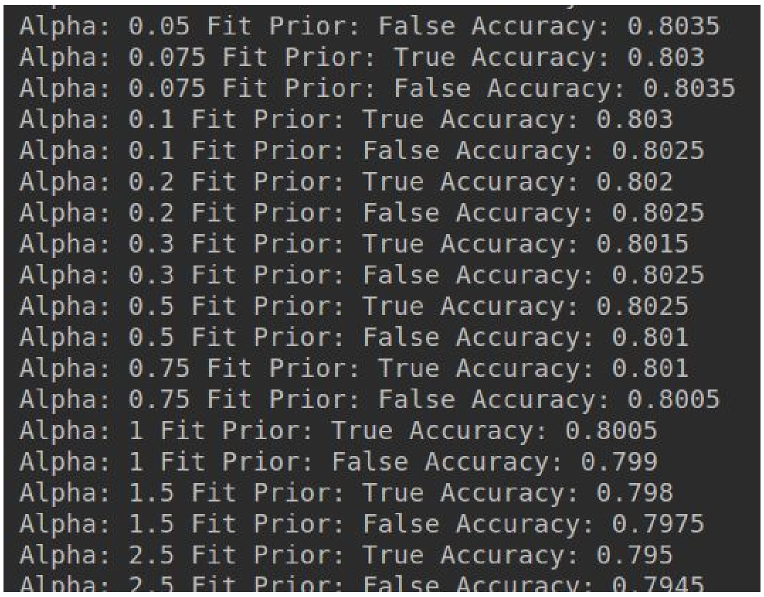
When trying different values for the parameters **alpha** and **fit_prior** the results below were found when the model was run on data set 1.

It is obvious the largest gain to be made is adding a smoothing factor when there was none previously. From there increasing the smoothing factor improves accuracy up to the point of diminishing returns at around a value of 0.5. This is likely due to the binary nature of the data and may not be a coincidence that the optimal value for smoothing is exactly half of the non-zero values.



```
Alpha: 0 Fit Prior: False Accuracy: 0.023346303501945526
Alpha: 0.01 Fit Prior: True Accuracy: 0.5953307392996109
Alpha: 0.01 Fit Prior: False Accuracy: 0.5992217898832685
Alpha: 0.025 Fit Prior: True Accuracy: 0.5992217898832685
Alpha: 0.025 Fit Prior: False Accuracy: 0.5992217898832685
Alpha: 0.05 Fit Prior: True Accuracy: 0.6011673151750972
Alpha: 0.05 Fit Prior: False Accuracy: 0.6011673151750972
Alpha: 0.075 Fit Prior: True Accuracy: 0.5992217898832685
Alpha: 0.075 Fit Prior: False Accuracy: 0.5992217898832685
Alpha: 0.1 Fit Prior: True Accuracy: 0.6011673151750972
Alpha: 0.1 Fit Prior: False Accuracy: 0.6011673151750972
Alpha: 0.2 Fit Prior: True Accuracy: 0.6070038910505836
Alpha: 0.2 Fit Prior: False Accuracy: 0.6089494163424124
Alpha: 0.3 Fit Prior: True Accuracy: 0.6050583657587548
Alpha: 0.3 Fit Prior: False Accuracy: 0.6070038910505836
Alpha: 0.5 Fit Prior: True Accuracy: 0.6108949416342413
Alpha: 0.5 Fit Prior: False Accuracy: 0.6108949416342413
Alpha: 0.75 Fit Prior: True Accuracy: 0.603112840466926
Alpha: 0.75 Fit Prior: False Accuracy: 0.603112840466926
Alpha: 1 Fit Prior: True Accuracy: 0.5992217898832685
Alpha: 1 Fit Prior: False Accuracy: 0.5972762645914397
Alpha: 1.5 Fit Prior: True Accuracy: 0.585603112840467
```

Fit prior appears to have less of an affect on the results.



```
Alpha: 0.05 Fit Prior: False Accuracy: 0.8035
Alpha: 0.075 Fit Prior: True Accuracy: 0.803
Alpha: 0.075 Fit Prior: False Accuracy: 0.8035
Alpha: 0.1 Fit Prior: True Accuracy: 0.803
Alpha: 0.1 Fit Prior: False Accuracy: 0.8025
Alpha: 0.2 Fit Prior: True Accuracy: 0.802
Alpha: 0.2 Fit Prior: False Accuracy: 0.8025
Alpha: 0.3 Fit Prior: True Accuracy: 0.8015
Alpha: 0.3 Fit Prior: False Accuracy: 0.8025
Alpha: 0.5 Fit Prior: True Accuracy: 0.8025
Alpha: 0.5 Fit Prior: False Accuracy: 0.801
Alpha: 0.75 Fit Prior: True Accuracy: 0.801
Alpha: 0.75 Fit Prior: False Accuracy: 0.8005
Alpha: 1 Fit Prior: True Accuracy: 0.8005
Alpha: 1 Fit Prior: False Accuracy: 0.799
Alpha: 1.5 Fit Prior: True Accuracy: 0.798
Alpha: 1.5 Fit Prior: False Accuracy: 0.7975
Alpha: 2.5 Fit Prior: True Accuracy: 0.795
Alpha: 2.5 Fit Prior: False Accuracy: 0.7945
```

We can see from the results to the left of the same model run on data set 2 that similar results were found regarding the best value for smoothing though the accuracy is clearly much better. This is likely due to the fact that data set 2 included more data points to train from. More data means more accurate models. That there were fewer possible labels to assign each data point may also play a role though the size of the data set should be considered more important.

A performance increase from parameter tuning on data set 1 yielded roughly a 2% gain in accuracy. For data set 2 the gain was minimal if not non-existent. This again is likely due to the size of the datasets in combination with the default settings for the model. It can be inferred that experiments on smaller data sets may benefit more from tuning that larger data sets. This is likely due to the extensive performance tuning that goes into coming up with the default values for the model parameters.

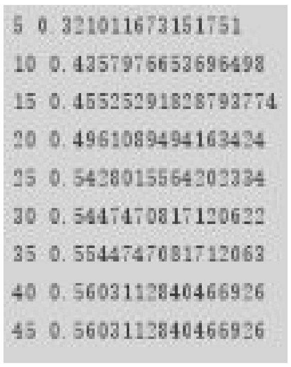
<u>Random Forest</u>

As predicted, the Random Forest classifier yields relatively high accuracy for both Datasets even before tuning. Since the Random Forest algorithm groups random number of features forming a smaller tree, and combining these small trees into a final tree that is shallower and less sensitive than a classic Decision Tree, the classifier model is therefore significantly less overfitted.

In the case of Random Forest, tuning the hyperparameters does improve the accuracy significantly. The most important parameters to tune are the number of estimators,

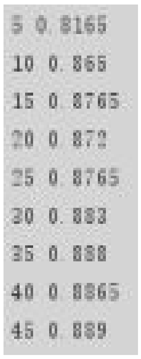
N_estimator:

Dataset 1



```
5 0.321011673151751
10 0.4357976653696498
15 0.4552529182879774
20 0.4961089494163424
25 0.5428015564202334
30 0.5447470817120622
35 0.5544747081712063
40 0.5603112840466926
45 0.5603112840466926
```

Dataset 2



```
5 0.8165
10 0.865
15 0.8765
20 0.872
25 0.8765
30 0.883
35 0.888
40 0.8865
45 0.889
```

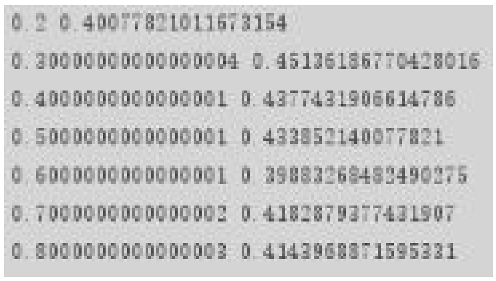
Left column: n_estimator value

Right column: prediction accuracy

The *n_estimator* parameter defines the number of subtrees to build. For training data with large feature size, a higher number of subtrees will yield a more stable classifier model, as the final tree is shallower and wider, it is not as sensitive to new data point as Decision Tree would be, and thus the model is less overfitted. This is proven by the improving accuracy of Dataset 1 with augmenting *n_estimator* value. In the case of Dataset 2, since the feature size is much smaller (10), the impact of increasing the number of subtrees is much less observed, as there are fewer ways to divide the features into subgroups.

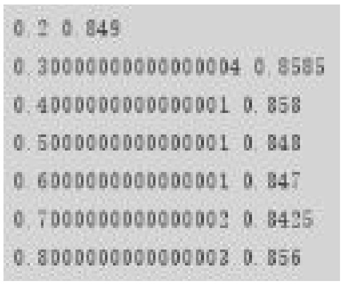
Max_features:
*Max_features* is the number of features looked at when looking for the best split, in the case of Random Forest, it defines the maximum number of features that can be tried in one subtree.

Dataset 1



```
0.2 0.40077821011673154
0.30000000000000004 0.45136186770428016
0.4000000000000001 0.4377431906614786
0.5000000000000001 0.433852140077821
0.6000000000000001 0.39883268482490275
0.7000000000000002 0.4182879377431907
0.8000000000000003 0.4143968871595331
```

Dataset 2



```
0.2 0.849
0.30000000000000004 0.8585
0.4000000000000001 0.858
0.5000000000000001 0.848
0.6000000000000001 0.847
0.7000000000000002 0.8425
0.8000000000000003 0.856
```

Left column: % of total feature as max_features
Right column: prediction accuracy

From the above results we see that there is no point at which we see a law for diminishing returns for the number of estimators. From our results one could conclude that increasing the number of estimators will further improve the accuracy, but also increases the run time of training models, so 1000 is a good number to set the limit at to balance model performance and program run time.[6]

The results above pertaining to max features indicate that examining roughly a third of the features will give the best results but again it’s apparent that tunning this parameter has a greater effect on the smaller data set. From best to worst regarding data set 1 is roughly 10% difference whereas for data set 2 the difference is much smaller.

## Conclusion

In conclusion, the Random Forest classifier outperforms Naive Bayes as well as Decision Trees. The later is not surprising since Random Forest as based on Decision Trees and should therefore provide an improvement else it has no reason to exist. The experimental results seen here indicate that all of our models more accurately predict the larger of the two data sets, confirming the importance of having more data when increasing the accuracy of a models prediction power. What was learned that’s less intuitive is that it’s the models trained on smaller data sets that stand to benefit the most from parameter tuning.

The future work to explore would be how to solve the problem of inaccurate models due to small data sets. Can we take the data points available and change them in some small way while leaving the labels to produce more data? Are all the features important in this data set or can we drop some of them by using thresholding?

## References

[1]"Decision tree learning", *En.wikipedia.org*, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity. [Accessed: 14- Nov- 2018].

[2]"What does "splitter" attribute in sklearn's DecisionTreeClassifier do?", *Stack Overflow*, 2018. [Online]. Available: https://stackoverflow.com/questions/46756606/what-does-splitter-attribute-in-sklearns-decisiontreeclassifier-do. [Accessed: 14- Nov- 2018].

[3]"sklearn.tree.DecisionTreeClassifier — scikit-learn 0.20.0 documentation", *Scikit-learn.org*, 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Accessed: 15- Nov- 2018].

[4]L. Raileanu and K. Stoffel, "Theoretical Comparison between the Gini Index and Information Gain Criteria", Annals of Mathematics and Artificial Intelligence, vol. 41, no. 1, p. 92, 2004.

[5]"sklearn.RandomForestClassifier — scikit-learn 0.20.0 documentation", *Scikit-learn.org*, 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: 15- Nov- 2018].

[6]"How to tune parameters in Random Forest, using Scikit Learn?", stackoverflow.com, 2018,[Online]. Available: https://stackoverflow.com/questions/36107820/how-to-tune-parameters-in-random-forest-using-scikit-learn
[Accessed: 12- Nov- 2018].

[7]N. Donges, "The Random Forest Algorithm", *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd. [Accessed: 15- Nov- 2018].

[8]"sklearn.Naive Bayes — scikit-learn 0.20.0 documentation", *Scikit-learn.org*, 2018. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html
[Accessed: 17- Nov- 2018].