**Student Number: 184516 & Kaggle Team Name: sf184516**

## 1. Introduction

In this report, I will be exploring a binary classification problem of memorable and non-memorable scenes in Brighton and London. I will be explaining the approach of final model, the methodology of how I got there and discuss and analyse the results.

## 2. Approach

In my final model, I used model selection to evaluate each classifier, the classifier I choose was the random forest classifier, which is a common classifier for image classification [8], and performed best after preprocessing etc. took place, *figure 3 and 4*. The hyperparameters of this classifier were tuned using 'RandomizedSearchCV' [1], which evaluates the classifier to find it's best parameters.

The additional data was incorporated with the training data, using an imputer [2] to fill in the missing values to generate a larger amount of data to train the classifier. The test proportion data was used to verify that the class distribution was unbalanced, this can influence the predicted labels leading to an inaccurate model, to balance this oversampling and under-sampling were used.

In feature selection, the high dimensionality of the training data was reduced by only using the CNN features due to their properties (section3) as well as implementing the filter chi squared. These selections took place to reduce overfitting, improve accuracy, reduce model complexity and reduce training time.

## 3. Methodology

I began by using the tagged training data to evaluate the classifiers: multilayer perceptron, logistic regressor, k nearest neighbours, random forest, supervised gradient decent and support vector [11], using a test-train split; training for model selection and testing for the final evaluation [3]. After using the test set to estimate the performance of the model, we can make use of all the labelled data and get the categorization accuracy from Kaggle using the test features, therefore, test train split is removed. In *figure1,* we can see that all models begin with no skill.

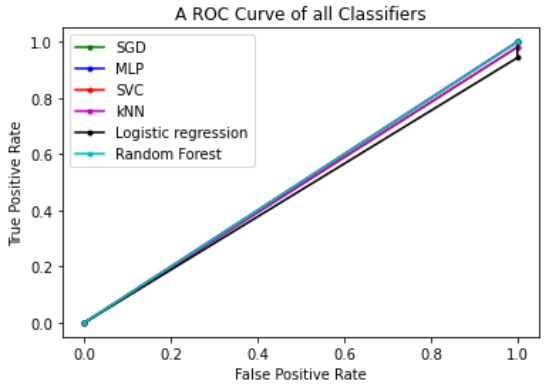


Figure 1: The ROC curve of each classifier using the labelled training data to train and test.

In the next step of the process, I decided to add more data to the training set by using both the 'training' and 'additional_training' files. 'additional_training' had features missing, to fill in these features I used a simple Imputer [2] that uses the mean of each column to fill in the missing values. Adding the additional data ensured a larger data set to train the model, producing the similar accuracies of the classifiers in *figure3 x-axis= 'original'*. The accuracies start very high, this is most likely due to overfitting of the data.

Comparing the additional information 'test_proportions' and the class distribution of the training data I found there was a massive imbalance. The testing and training class proportions can be seen in *figure2*. This bias in the training set can influence the labels that are given to the testing data; to tag more with the majority class (in this case 1) [4]. I decided to combine both over and under sampling techniques to balance the data, as over sampling may lead to overfitting the minority class and underfitting can result to losing too much vital information of the majority class. In *figure3 axis=* balanced, we can see that the model's accuracies have reduced, this is most likely due to the less information being passed into fitting the classifiers: the old class distribution (1: 2140 0:326) to the new balanced distribution of (1:535, 0:428). I submitted the best classifier (Random forest) into Kaggle and the category accuracy was 0.68935, compared to an accuracy of 0.38409 from the start.

| | Proportions | | |
|---|---|---|---|
| class | Test | Training | Balanced |
| 1 | 0.3848 | 0.8702 | 0.5556 |
| 0 | 0.6152 | 0.1298 | 0.4444 |

Figure 2: The class proportions for testing data ('Test') , original training and additional training data ('Training'), and the balanced training data ('Balanced'). All to 4sf.

Ideally, we want to select the best features from the data to build the model. After investigating the training data made up of CNNs and GIST features, I found that CNN features give far more detail and are a lot more important than GIST features. CNN has also been useful in other scene recognition applications, for example [5]. The CNN features range between 0 and 9, due to this I do not think the features need to be normalised, standardised, scaled or binarized before use as the range is so small. After performing this technique, most classifiers accuracies increased or stayed the same, seen in *figure3 x-axis=CNN,* except from the random forest classifier which reduced, this may be due to the over fitting of the data, as well as the reduction of information to train the classifiers.

Filter methods can be used to select features based on a calculated score. Chi-square achieves this by calculating the probability of correlation between the features to

choose the best ones. The scikit-learn library 'SelectKBest'[6] has been used to select these features. This method also reduces the high dimensionality of the dataset as it creates new combinations of attributes rather than removing them. Using this method increased more model's accuracies (*figure3 x-axis=Chi)* as well as increasing the categorization accuracy in Kaggle to 0.72237.

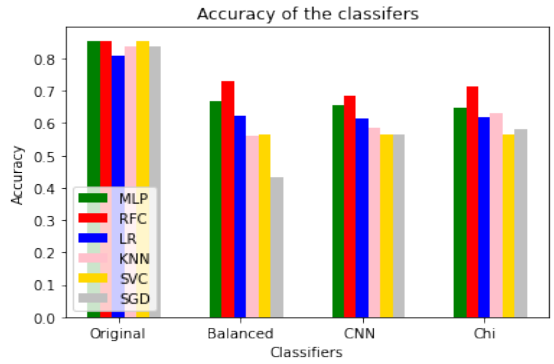


Figure3: A bar chart showing the accuracy of each of the models using different classifiers throughout steps of the report.

Another feature selection technique is wrapper methods, these are used in the training of the model, where the possible combination of features is evaluated, and then best result is chosen. The algorithm I chose to implement was recursive feature elimination, after each iteration this greedy algorithm keeps the best preforming features and removes the worst, the features are then ranked in order of their elimination [7]. Due to the characteristics of this model, it took a long time to train and it could not be completed on all classifiers (only random forest, logistic regression and supervised gradient decent) due to the classifier's characteristics. On the classifiers that it did execute on, the accuracies of the classifiers reduced, the Kaggle submission using a random forest classifier dropped to 0.62567, this could be due to the risk of overfitting, therefore I decided not to use a wrapper method.

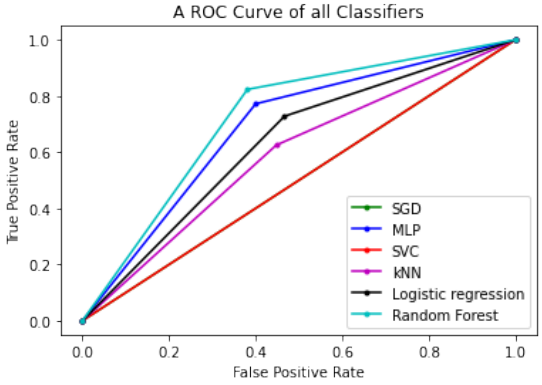


Figure4: The ROC curve of each classifier after feature pre-processing and selection has occurred: Using the additional data, balancing the classes, removing the GIST features and applying the Chi-squared filter method.

In *figure4*, the ROC curve shows all the classifiers, trained on training data and the additional data, where the GIST features have been removed, the class distribution has been balanced and the filter method 'chi' has taken place. The most skilful model produced is the random forest classifier. To complete model selection, I decided to fine tune this classifier's hyperparameters using 'RandomizedSearchCV'[1]. A parameter 'cv' is used to determine the number of k-folds which splits the training data to allow for validation tests of each combination. After using these validation tests to evaluate the performance of each models, 'best_params()' can be used to retrieve the best parameters of the final model. The categorization accuracy from Kaggle of the final model predicting the testing labels is 0.75269.

Due to the domain adaptation problem, where a model is trained on source data which is used on different but related target data, it may benefit the results to regularise the data. One way of doing this it to tune the parameters of Logistic regression (the 3nd best performing classifier, *figure4)* to have a penalty of L1. L1 looks at the simpler patterns of the data and discovers a common part. Using this method leads to sparse solutions which reduces the over fitting on the Brighton data to lead to more accurate results on the London data. Testing this on Kaggle using the test features, reduced our categorization accuracy of the model by 0.12(2dp) to 0.63039, therefore I decided to stay with the random forest regressor.

## 4. Analysis

Overall, I think my model is accurate, I have learnt to look carefully at the feature properties to understand what to execute on them to fit the model, resulting in a well thought out feature selection.

To get a better performance, a voting strategy could have been implemented by using multiple classifiers to balance out the individual weaknesses [12], for example random forest regression and logistic regression to add in the L1 penalisation. Solving the domain adaptation problem by implementing embedded methods such as Lasso, which is a combination of filter and wrapper methods, would have lowered the complexity of the model. I struggled to evaluate the classifiers due to this problem as the models would overfit to the source data resulting in a good accuracy, however when tested on the target data this was not the case.

I did not use the confidence levels of the training data as I was unsure how to incorporate this into the model, if I had more time, I would have done more research into this. There may also be an issue with adding the missing values to the additional data using the mean of the columns, due to the unbalanced class distribution; therefore, producing values similar to the majority class features.

In conclusion, my goals to create a model that classifies data in an accurate way has been achieved resulting in a score of 0.75269 on Kaggle.

## 5. References


[1] Scikit-learn.org. 2020. *Sklearn.Model_Selection.Randomizedsearchcv*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html> [Accessed 21 May 2020].

[2] Scikit-learn.org. n.d. *Sklearn.Impute.Simpleimputer — Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html> [Accessed 26 May 2020].

[3] Scikit-learn.org. n.d. *Sklearn.Model_Selection.Train_Test_Split — Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html> [Accessed 22 May 2020].

[4] Brownlee, J., 2020. *Random Oversampling And Undersampling For Imbalanced Classification*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/> [Accessed 22 May 2020].

[5] Zhou, B., 2015. *MIT Places Database For Scene Recognition*. [online] Places.csail.mit.edu. Available at: <http://places.csail.mit.edu/> [Accessed 22 May 2020].

[6] Scikit-learn.org. n.d. *Sklearn.Feature_Selection.Selectkbest — Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest> [Accessed 23 May 2020].

[7] Kaushik, S., 2016. *Feature Selection Methods With Example (Variable Selection Methods)*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/#:~:text=Filter%20methods%20use%20statistical%20methods,the%20best%20subset%20of%20features.> [Accessed 24 May 2020].

[8] Bosch, A., Zisserman, A. and Munoz, X., n.d. *Image Classification Using Random Forests And Ferns*. [online] Available at: <http://www.robots.ox.ac.uk/~vgg/publications/papers/bosch07a.pdf> [Accessed 17 May 2020].

[9] Manjula, K., Vijayarekha, K. and Vimaladevi, P., 2017. *Review On Classification Algorithms In Image Preprocessing*. [online] Available at: <http://file:///Users/stephiefoster/Downloads/REVIEW-ON-CLASSIFICATION-ALGORITHMS-IN-IMAGE-PROCESSING.pdf> [Accessed 18 May 2020].

[10] Raschka, S., 2018. *Model Evaluation, Model Selection And Algorithm Selection In Machine Learning*. [online] pp.10-33. Available at: <https://sebastianraschka.com/pdf/manuscripts/model-eval.pdf> [Accessed 29 May 2020].

[11] Rozran, J., 2017. *How To Build A Binary Classification Model | Jake Learns Data Science*. [online] Jake Learns Data Science. Available at: <https://www.jakelearnsdatascience.com/post/how-to-build-a-binary-classification-model/> [Accessed 23 May 2020].

[12] Scikit-learn.org. n.d. *1.11. Ensemble Methods — Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier> [Accessed 24 May 2020].