Student Number: 213886 & Kaggle Team Name: Dan Watson

1. Approach

The general approach that I have taken is the use of a voting classifier, specifically the VotingClassifier function [1]. This is an ensemble technique included in the sklearn python package which essentially makes use of a combination of several unfitted estimators at once which "vote" on the outcome of the model. The estimators used in the voting classifier process include multiple classifiers from the linear model section of the package including LogisticRegression [2] and RidgeClassifier [3]. Ridge classification converts the class targets to real values and performs a regression which minimises a sum of squares term that is accompanied by a regularisation parameter attached to the weights. Hence it is a modification of the ordinary least squares procedure. This is represented in the equation:

$$\min_w \|Xy - w\|^2 + \alpha\|w\|^2$$

where α is the regularisation parameter and w represents the weights. Regularisation helps reduce overfitting by discouraging unnecessary complexity.

Logistic regression makes use of the logistic function which is a type of sigmoid curve and uses the cross-entropy loss function instead mean squared error which for multi-class problems the result is:

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

where M is the number of classes, y is the indicator function and p is the predicted probability that the observation o is of class c as seen in [18].

This ensemble technique tends to produce a respectable score on the public Kaggle leaderboard of around 0.78 and performs the best out of all of the possible methods that I have tried. The most influential reason for the use of this classifier is that it helps minimise certain biases that may be produced by individual classifiers towards a certain prediction as well as the fact that logistic regression and ridge classification seem to be the best performing estimators. Sklearn's StackingClassifier [4] and BaggingClassifier [5] estimators also worked well, but the voting classifier produced marginally higher scores and so this was preferred. I also attempted to incorporate the use of the XGBClassifier function in the xgb package [6] due to its reputation of producing excellent scores across a large amount of Kaggle competitions. This produced similar scores to the approach mentioned above but the overwhelming amount of available parameters made it a daunting task and so this was abandoned.

The reason for neglecting the use of classifiers such as support vector machines and the perceptron/MLP is that they did not seem to perform as well as the previous estimators. I believe that this is mostly attributed to the lack of a clear decision boundary that exist between the two classes. This can be seen later in Figure 1. Utilising random forests [7] and also did not seem to be as effective as the previous estimators which I believe is due to the class imbalance and domain adaptation problem, as well as a clear lack of binary "decisions" in the data due to the fact that they are extracted features.

2. Methodology

Before our classifier could be trained, some preprocessing had to be done. This was mainly to address the four additional characteristics that were provided with the competition dataset which were:

- Additional training data with missing features
- Test label proportions
- Training label confidence
- Domain adaptation problem

2.1. Additional training data with missing features

A simple logistic regression with the base data produced a poor score of around 0.36 - significantly worse than a random guess. Collating this with the additional training data provided while replacing the missing features with simply zero values produced a significantly better score, around 0.55-0.6. However, replacing these missing features with zero values is a rudimentary approach and so I first opted to instead replace them with the mean of its corresponding row, where each row represents an image. I believed this to be the most intuitive way of dealing with the missing values but this was changed so that the NaN value was replaced with the mean of the row in its respective group of features - a more intuitive approach. For example, NaN values that lie in the CNN features would be replaced by the mean of the CNN features of that row and the same for the GIST columns.

2.2. Training label confidence

The next characteristic that was addressed was the training label confidence. This data provides a human annotated score for the training data and hence is a significant indicator of the true label. However, it was unclear how to proceed with this additional data due to the fact that adding it to the training data would provide an extra feature when compared to the test data causing a difference in data dimensionality that would prevent a model from being able to be fit. To

get around this, I took of the mean of the true class labels and the confidence data, creating a new "combined" column which contained values of 0.33, 0.5, 0.83 and 1. This meant that our problem shifted from a binary classification to a multi-class classification - now with four classes instead of two.

2.3. Test label proportions

The test label proportions provided were a good indicator on the potential score of the model if it were to be uploaded to Kaggle and so I created a function named "proportions" that asseses the proportions of predictions that were generated by the model. If the proportions produced were similar to that of the test label ones given then usually good scores would be produced. However, just because the proportions are similar to the desired proportions, that does not mean that the predictions of the estimator are also correct. Despite this, it was still the best way of assessing the potential score of the model and outweighed the usefulness of scores produced by cross-validation.

2.4. Domain adaptation problem

This was perhaps the most difficult problem faced throughout the competition. The large difference in proportions of the class labels in the training data when compared to the test data made it difficult to account for this shift when producing a model. This was mainly tackled through the use of class weightings whereby more weighting was applied to the zero class label (not memorable predictions) which resulted in the model favouring these zero class labels during prediction therefore giving proportions closer to the real test proportions supplied. Without this weighting, the model would heavily favour class labels of ones (memorable images) due to the high proportions of one labels that exist in the training data. We could use the "balanced" parameter when specifying the class weights which provides weights to classes inversely proportional to their frequencies in the data and so is suitable for a class imbalance/domain adaptation problem. However, manually setting the class weights provides a better result overall and if we use weights that give perfect test proportions then the model will work well in the competition. The usual weights used were something around: $\{0 : 12, 1 : 1.5, 2 : 1, 3 : 2\}$ where 0 is a confidently not memorable image, 1 is not confidently not memorable, 2 is not confidently memorable image and 3 is a confidently memorable image. A small amount of weighting is applied to labels 1 and 2 whereby these are the non-confident predictions. This is to prevent the prediction of a label with a small amount of confidence that the prediction is correct.

I also tried using the SMOTE function from the imblearn python package [8] to tackle this issue by the use of resampling although specifying the class weights seemed to produce much better results more quickly. Due to the large difference in proportions of labels between the training and test set, this meant that decomposing the original training data into further training and test data using the train_test_split function in sklearn would not be as useful as it is in usual machine learning problems [9]. In other words, the distributions of this validation set and the original test data are not very similar and so the potential benefits of this approach are limited. We can go one step further and perform cross validation instead, whereby the model can train on multiple splits of itself and can allow for the fine-tuning of hyperparameters if desired. This led to the use of GridSearchCV [10], a very powerful tool that performs cross validation and searches for the parameters that maximise the type of score that the user is most concerned with (accuracy, f1, roc_auc etc). We will talk about this more in the results section.

2.5. Further preprocessing

Now that our problem had become a multi-class classification due to the meaning of the confidence and prediction columns, this provided a problem for our classifiers. The problem was that our y_train now contained continuous values which meant that fitting was not possible. To get around this, label encoding was used to transform the values of 0.33, 0.5, 0.83 and 1 to 0, 1, 2, 3 respectively [11]. This allowed fitting to occur. To reverse back to binary predictions which is required for uploads to Kaggle, inverse transforming is used on the predicted values which are then rounded using the numpy round function. Interestingly, this rounds values of 0.5 to 0, giving a bias towards the image being non-memorable. This is useful in the context of our problem as a shift in proportions is required anyway and made sure that images were only predicted to be memorable if there was strong evidence for this being the case. However, the use of label encoding produces an ordering of classifications which could be harming the overall predictive power of the model but another way to incorporate the confidence values was not obvious and so this method was kept. One-hot encoding would usually be used instead but this was not available due to the fact that it was the target labels being encoded and not features. In addition to the above, I also opted to use PCA as a dimensionality reduction technique [12]. Our training data has around 4600 features meaning that a model that is fit on this data will be very complex resulting in high bias and so we aim to use PCA to reduce the impact of possible overfitting. It also allows us to reduce our data to two components which means we can plot these components with a hue of the prediction value, as well as improving the run-time of the model. Hence a possible decision boundary can be found, with an example for the data provided being available in Figure 1.

It is standard practice to apply some form of standardisation, normalisation or scaling to the data before applying

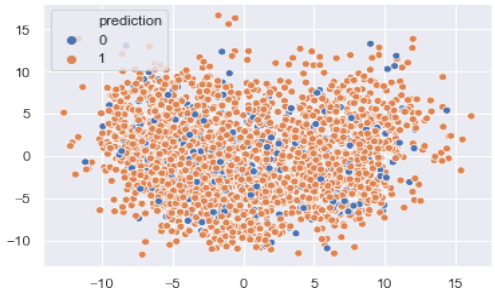

Figure 1. Plot of PCA components when reduced to two dimensions with a class label hue.

PCA. This is because when PCA is undertaken, we are interested in the number of components that maximise the retained variance. Originally, I used the StandardScaler function [13] which results in features being standard normally distributed with a mean of zero and variance one. This approach is suited well for sparse data which ours looks to be. Upon further inspection however, through the use of df.describe() it seems that the CNN features tend to have higher means and variance than the GIST features. This resulted in me opting the use robust scaling in the as it is more suited to data with outliers [14]. Due to the sparseness of the GIST features, I opted to train a model with just the CNN features which produced a much lower score than the current model, indicating that the GIST features were still important for prediction.

The next largest issue was determining the optimal number of components to specify when using PCA, which will be discussed further in the results section. To visually decide whether standard or robust scaling was a better fit, I produced plots of the cumulative explained variance ratio when using ten components.

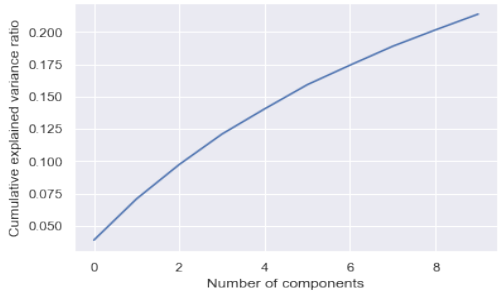

Figure 2. Plot of the cumulative explained variance ratio against the number of components for standard scaling.

In Figure 3, we can see that robust scaling results in PCA retaining almost all of the variance of the data whereas standard scaling only retains around 20% of the variance of the data as seen in Figure 2. Hence we would naturally prefer

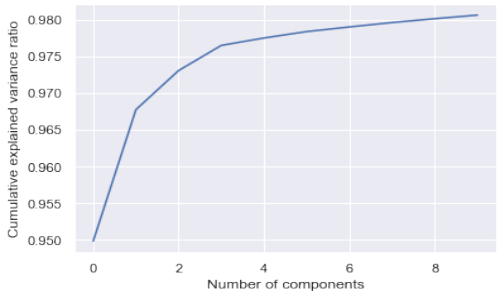

Figure 3. Plot of the cumulative explained variance ratio against the number of components for robust scaling.

robust scaling over standard scaling but it seems as though despite this fact, standard scaling still performs much better than that of robust scaling, achieving a much higher score on Kaggle. Other types of scaling such as min-max scaling [15] and max-abs scaling [16] also retain a large portion of the explained variance of the original data but do not achieve scores as high as those achievable through standard or robust scaling. Another technique experimented with was normalisation of the data in addition to scaling, as well as normalisation without scaling [17]. However, the scores produced by these models were sub-par and it seemed best to proceed with just standard scaling before applying PCA.

3. Results

3.1. Grid search cross validation hyper-parameters

In this section we present the results of the model selection. As mentioned before, grid search cross validation was used in order to try and find the hyper-parameters that maximise the performance of the model. We produce several figures that detail this, where the parameters of concern are the regularisation values of both the logistic regression and ridge classification, aswell as the different possible solvers available for both estimators. This was done by performing the grid search and then saving the cross validation results to a dataframe with which plots can be produced.

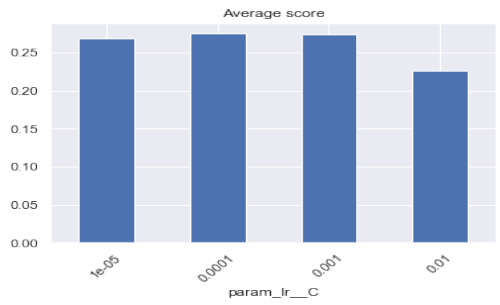

Figure 4. Plot of the average score achieved by different logistic regression regularisation parameters

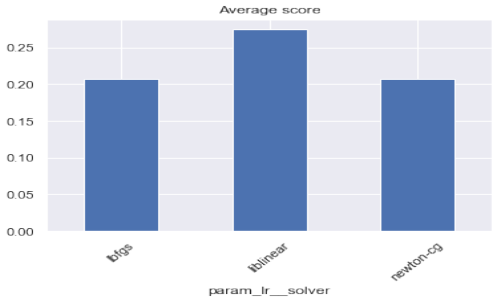

Figure 5. Plot of the average score achieved by different logistic regression solvers

It is clear to see from Figures 4 and 5 that the different hyper parameters do not make a very large difference on the score attained during cross validation with 10-folds. For the logistic regression, the liblinear solver has a small edge on the other parameters. In addition, it is clear that a C parameter of 0.01 performs worse than the others by a discernible amount. The alpha value and solver for ridge classification also causes almost no difference in cross validation scores and hence the plots were omitted. It should be noted that the C parameter is the inverse of regularisation strength whereas alpha is the regularisation strength, hence alpha = 1/(2C). This is why the regularisation parameters are very different across the two estimators.

3.2. Drawbacks of cross validation and grid search

It is important now to note the drawbacks of using this method. Due to the fact that the distribution of the original training data is very different to the distribution of the test set (domain adaptation problem), these plots are not indicative of the true effectiveness of different parameters when using our model. For example, regularisation seems to have little effect on the scores produced by cross validating the training data whereas this is not true for the final model. Through submission variations, different regularisation parameters do in-fact have a large difference on the final score produced on Kaggle, with stronger regularisation parameters being favoured. This is most likely due to the fact that the impact of overfitting on the training data is reduced which is ideal in this situation as producing similar results to the training data is not the aim in this competition. It is important to note that the model was specifically designed to actually underfit whereby it purposefully does not fit the training data very well and so generalises better. This is of course due to the shift in proportions of predictions that takes places - the domain adaptation problem.

During model selection, grid search chooses low levels of regularisation and so for the final model we specify the regularisations manually to increase the strength of regularisation. A preference of certain solvers, however, does not seem to cause a significant increase in score. This does not mean that cross validation is useless as it would be better than performing a simple train-test-split or none at all but it most certainly is not as effective as it usually is for machine learning problems whereby a domain adaptation problem is not apparent. As mentioned before, the best way to asses the predictive power of the model was to check the proportions of the predictions to see if it matched the proportions of the test set given with the original data, whereby we would prefer models that produced proportions of a 0.65/0.35 split for the labels 0 and 1 respectively. Of course this shift in proportions made the models predictive power on the data it trained on very poor, resulting in the low cross validation scores of around 0.25-0.4.

This was what led to the abandonment of grid search cross validation in favour of just normal cross validation of the relevant estimators. In this case, I used RidgeClassifierCV [19] and LogisticRegressionCV [20] with the usual voting classifier method as described in the beginning of the report in the approach section. This allowed for the use of leave-one-out cross validation whereby the number of folds is equal to the number of samples in the data. However, this did not work any better than the normal versions of the estimators and in additon to the fact that cross validation is not very useful for this particular problem, I opted to stick with the standard functions instead.

3.3. PCA

As mentioned earlier, another large problem in this competition was choosing the correct number of components to reduce the data to when using PCA. The explained variance and cross-validation scores tend to increase with the number of components which can be seen below in Figure 6. However, this does not seem to be valid for the true scores produced on Kaggle, with the number of components not making a significant difference in scores. Hence we take the number of components for the final model to be 10 in consistency with Occam's razor so as to limit the impact of unnessecary complexity of the model and to help limit the bias.

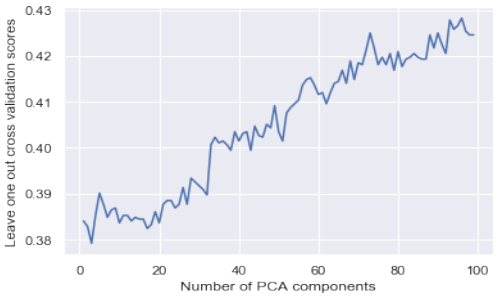

Figure 6. Plot of the average cross validation scores achieved with an increasing number of PCA components

3.4. References

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
6. https://xgboost.readthedocs.io/en/latest/python/python_api.html
7. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
8. https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html
9. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
10. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
11. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
13. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
14. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
15. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
16. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html
17. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
18. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#loss-cross-entropy
19. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifierCV.html
20. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html