Student Number: 213120 & Kaggle Team Name: Dylan Ruggier

1. Approach

The model I propose for the classification task at hand (to determine whether an image can be deemed memorable or not) makes use of a Multi Layer Perceptron (MLP) feedforward artificial neural network. All available training data was used, where missing values were imputed using the mean value from k-nearest neighbors.

A key obstacle in this was the domain adaptation problem – specifically, the significant class imbalance that exists between those class proportions found within the training data set, and those found within the testing data set. In order to bridge this gap, I factored in the annotation confidences; swapping entries within the training data set with low confidences, such that the relative proportions would more closely mirror those within the dataset the model would be tested on.

Since there didn't exist a sufficient amount of entries with low enough annotation confidence to warrant flipping their expected outputs, I further rectified the class imbalance by employing the use of a random under sampler. This allow me to match the training and testing class proportions with minimal loss of training data. Finally, all data is standardized before being fed in to the MLP.

2. Methodology

2.1. Data Formatting

I began by importing all data that was made available to me from their respective csv files, of which there were 5; training.csv, additional_training.csv, testing.csv, test_proportions.csv, and annotation_confidence.csv. In terms of the formatting that was required, I separated the CNN and the GIST features from all sets of training/testing data, before concatenating the lists resulting from both training data sets in accordance with their feature type. I now had all available CNN and GIST features stored in their respective variables in the correct format for pre-processing, and ultimately to be used as parameters for our classifier.

2.2. Imputation of Missing Values

If you are to inspect the contents of additional_training.csv, you will notice that a significant proportion of the values are missing - this will obviously pose a problem should we attempt to use it to train a classifier. Under normal circumstances it would be tempting to simply omit this data, however when we consider that it accounts for 89.98% of all training data we have, this would likely come at huge detriment to the performance of the classifier.

Although I will go in to more detail on this in section 3, I found best success with a KNNImputer, which imputes values based on the means of corresponding values of the data entry's k-nearest neighbours. I saw my best success with this methodology, combined with a low *k* value, as I hypothesize this to maintain more variation in the data where appropriate, allowing the class members to be differentiated from one another more clearly.

2.3. Model Selection

Before implementing any further pre-processing techniques, I used K-fold cross validation to assess the 'out of the box' performance of a few different classifiers; namely a Support Vector Machine (SVM), Multi Layer Perceptron (MLP), and Logistic Regressor – all from pythons Scikit-Learn library. Please refer to section 3 for the results, however I found best success with the MLP approach, and so this became my model of choice moving forwards.

Another reason for evaluating performance at this early stage is to serve as a control – I now had a metric to allow me to gauge the benefit, or even detriment, brought by any additional pre-processing methods implemented.

2.4. Domain Adaptation Problem

Since all training data were collected in Brighton, and test data in London, it was clear that we were going to face a domain adaptation problem. This became particularly apparent when models that were scoring in excess of 80% locally, using k-fold cross validation, saw significant drops in performance to approx. 50% when submitted to Kaggle. Because of this, k-fold cross validation became a relatively obsolete method by which to evaluate the performance of a classifier, due to its inability to factor in the challenges posed by the domain adaptation problem.

Upon investigating the nature of this issue, I noticed the extreme class imbalance between the training and testing data sets. Whilst 86.78% of training data entries belonged to the positive class, this dropped to only 38.48% in the testing data set. As was concluded in Japkowicz's publication [1], the sensitivity of an MLP to the class imbalance problem increases with the complexity of the domain. Since we are working with data of a very high dimensionality, our results would seem to be corroborative of this.

In terms of bridging this imbalance, although Japkowicz expressed the conjecture that both over-sampling the minority class, as well as down-sizing the majority class are effective methods[1], Zhao et al [2] and Marqués et al [3] culminate that oversampling methods perform better than any undersampling approaches in general. This, combined with the fact that if we undersampled the +ve class to the point where training and testing class proportions were equal, we would result in the loss of 78.5% of our training data, we will do whatever oversampling we can.

This is where I decided to make use of the data from annotation_confidence.csv. I flipped the expected output of every positive entry (that is one with an expected output of 1) within the training data that had an associated annotation confidence of less than 1. I was justified in doing this as a method of oversampling as these annotation confidences are based on the opinions of only 3 people, on what could be argued to be a fairly subjective classification task.

In order for the vast train/test class imblanace to be bridged, 1191 positive instances would have to have been swapped in the mannor described above. However an issue that came to light was the fact that there only existed 998 positive entries with an annotation confidence less than 1. Since flipping additional entries to a class that is almost certainly incorrect would likely skew the logic of the classifier, I made the decision to further bridge the imbalance throgh random undersampling. The class ratio's between the training and testing sets were now perfectly aligned, having only lost 15.6% of our training data through undersampling, as opposed to the 78.5% it would have been without the prior annotation confidence-based oversampling.

2.5. Additional Pre-Processing

The changes described in section 2.4 saw significant improvement in the classifier, as it was now able to surpass what could be achieved by simply returning 0 in all instances, which would achieve a 61.52% accuracy (as this is the relevant class proportion) without exhibiting any form of the complex decision-making logic expected from artificial neural networks. It was only surpassing this boundary, by little more than 5% however, so I continued experimentation with various pre-processing techniques to see if this could be further improved upon. I found best success with Standardization; this saw a significant jump in accuracy to 76%, a score I was unable to beat, and hence it becoming my model of choice.

3. Results & Discussion

3.1. Alternative approaches

In addition to standardization, I attempted to implement binarization as an additional pre-processing technique to the model I have described – using a threshold of 0.293 standard deviations above the mean, which equates to a division in line with the given testing class proportions. This saw a drop in performance to 66.3%, I hypothesize this could be because the conversion of data from a continuous to discrete format is lossy by nature. I have had success with this methodology in other applications in the past, however they have always been of a very objective nature. Whether something is memorable or not by comparison is more subjective, and therefore I would anticipate is best expressed continuously.

When I have implemented binarization into machine learning models in the past, I have used the threshold as a means of accounting for the testing class proportions. In this model however, I had already accounted for them through the over/under sampling described in section 2.4. Has employing both methodologies over-corrected the class imbalance to a detriment? This remains a point of further investigation I hope to pursue, but one I have left for now due to the strict limitations imposed by the permitted Kaggle submissions, as well as my suspicion that continuous data would be more suitable to training the network due to the increased precision, and the nature of the task which may rely on (combinations of) features having partial values as opposed to binarized ones.

Principle Component Analysis was another approach I attempted. Due to the very high dimensionality of our data, I anticipated PCA to be a good option for managing this. Upon implementation however, my model's accuracy suffered, dropping to 58.22%. I was suspicious that this was because PCA is linear, and the output we seek may lie in non-linear dependencies. I attempted to address this through the use of KernalPCA, and a polynomial kernel. The polynomial kernel considers vector similarities across dimensions, as opposed to exclusively within the same dimension, which in effect accounts for feature interaction which I suspected could be very well suited to the task at hand. Unfortunately I was not able to bring this approach to fruition, as I was only able to achieve an accuracy of 52.78%.

After conducting more research into the various available kernels, I decided to attempt one other, very different, approach to ensure I hadn't become blinded by tunnel-vision as a result of some of my earlier successes. The Chi-squared kernel is a prevalent option in computer vision applications of non-linear source vector machines, and so I trained an SVC on normalized GIST featured to make use of a chi-squared kernel. This approach scored an accuracy of 62.8%, which may seem respectable, however it is only marginally above the 61.52% threshold that can be achieved purely by outputting 0's - it has a very high rate of false negatives, and in turn suffers in terms of true positives. This approach remains one that I wish to revisit in the future, as I do still believe there are improvements that can be made here, and I anticipate that it may result in a model that is more robust than the one I have proposed, however it is possible that much more would need to be done before this is the case.

Another approach I attempted that is worthy of note, was to incorporate multiple classifiers. One method of doing so was for the output of each entry to be the result of a majority vote between 3 classifiers – much the same way the training data is originally hand labelled. In practice however, its success lay somewhere in between that of the 3 classifiers in use, meaning that in effect, the strongest classifier amongst them was being restricted by the other two. In all cases where the strongest classifier would be the only one get an instance correct, it would be outweighed by the other 2. I found similar results when attempting a weighted 2 classifier approach, weighting each proportionally to their relative tested accuracies, but this suffered from the same issue, causing me to abandon the multi-classifier approach.

3.2. Imputation of Missing Values

18% of all training data are missing values, this gives us an indication of the gravitas associated with the correct handling of them.

Figure 1 – Graph to show accuracies with various imputation methodologies

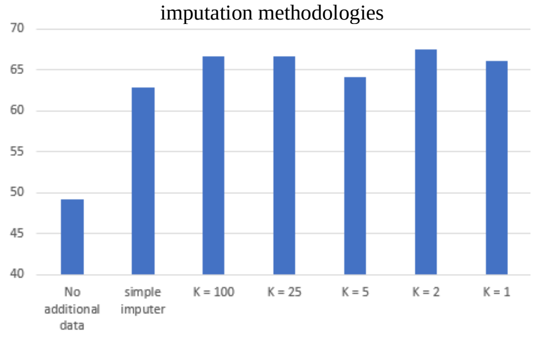


Figure 1 demonstrates not only the importance of imputing the missing values, allowing for the full training data set to be used, but also the importance of the method used to do so. As soon as the extra training data was incorporated with a simple imputer, the accuracy of the classifier on the testing data set jumped by 13.78%. This simple imputer works by calculating an average of all those values within the same column that are present within the entire training data set. Since these training images are likely rather varied, with different features present to different degrees, and these differences are what we rely on in order to correctly categorize each one – although it allowed for the data to now be incorporated, there seemed to be room for improvement.

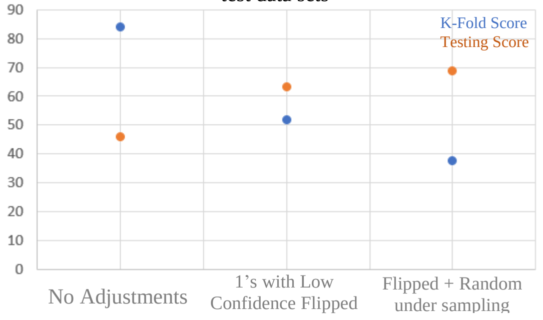
A KNN-imputer works by instead replacing a value with the corresponding average from the entry's k-nearest neighbours, rather than the entire dataset. This is advantageous as it is very likely that the nearest neighbours to each data entry also belong to the same class, meaning that the calculated average will be more accurate and class-specific, therefore maintaining more variation between the classes.

I hypothesized that a low k-value would yield the best results, as the calculated averages would be more specific to each entry. This certainly seemed to be the case to a degree, and I found my best success with a k value of 2. I suspect the sharp drop seen in figure 1 when k = 5 to be the result of poor chance, since the public leaderboards are only based on a random 25% of the total training data set. Under normal circumstances I would have averaged these results, however this was not an option when factoring in the Kaggle submission limitations, and the ineffectiveness we've seen of other local methods such as k-fold cross validation.

3.3 Domain Adaptation Problem

As outlined in section 2.4, perhaps the greatest obstacle we must overcome is the domain adaptation problem that exists here. Figure 2 illustrates the true extent of this domain gap, and the reason why I took the decision fairly early on to abandon K-fold cross validation on the entire training data set as a metric by which to evaluate the success of a classifier. One may argue that there is a strong inverse correlation between the scores achieved on each domain, and although this is true for the 3 classifiers depicted in Figure 2, this wouldn't be sufficiently reliable metric to base my efforts on, as there do exist classifiers that don't exhibit this relationship - for example in the case of a classifier that performs poorly across both testing domains.

Figure 2 – Discrepancy in scores achieved on different test data sets



It's important to note that there are more components to the domain adaptation problem than exclusively the train/test class imbalance. For example, the feature ranges are likely to differ between the two data sets, due to the photos being inherently different in accordance with the scenery each was captured in. I focused primarily on the class imbalance aspect as this appeared to carry the heavier detriment, whereas simple pre processing techniques such as standardization help in correcting differed feature ranges.

4. References

[1] N. Japkowicz, "The Class Imbalance Problem: Significance and Strategies," in *Proc. of the Int'l Conf. on Artificial Intelligence*, Halifax, 2000.

[2] S. X. B. H. K. Zongyuan Zhao, "Investigation of Multilayer Perceptron and Class," *International Journal of Computer and Information Technology ,* vol. 3, no. 4, pp. 805-812, 2014.

[3] V. G. J. S. S. Ana Isabel Marqués, "On the suitability of resampling techniques for the class imbalance problem in credit scoring," *Journal of the Operational Research Society*, vol. 64, no. 7, pp. 1060-1070, 2013.

5. Appendix

Submission name	**Notes**	**Score**
Attempt_1.csv	SVM model using training proportion weights	0.43564
Attempt_2.csv	Failed scaled + Binarized CNN, w/ weights + confidences, all 1's	0.38207
Attempt_2.csv	Failed, baseline acc of all 0's	0.61792
Attempt_3.csv	MLP, original pre processing methods, not overly hopeful	0.55121
Attempt_4.csv	Same MLP, adjusted binarization threshold	0.55121
Attempt_5.csv	MLP with impute and standardise	0.55121
Attempt_6.csv	MLP, KNNimputer, standardise	0.55121
Attempt_9.csv	K-Fold MLP, scores 80+ in testing there but looks wrong, mostly 1's	0.58827
Attempt_14.csv	Flipped all (991) training 1's with <1 confidence to 0, to help combat train vs test class imbalance. MLP using K-fold on Imputed +standardised data	0.68901
Attempt_15.csv	Flipped all (991) training 1's with <1 confidence to 0, to help combat train vs test class imbalance. MLP using K-fold on Imputed +standardised data	0.72641
Attempt_16.csv	Same as prev, but reduced KNN from 5 to 2 - will it improve due to more variation in the data?	0.76549
Attempt_17.csv	Same as prev, model trained on entire available set, rather than 4-fold of cross validation sets	0.76549
Attempt_18.csv	Same as prev, model trained on entire available set, rather than 4-fold of cross validation sets	0.64555
Attempt_20.csv	Attempt to replicate best score, where did it all go wrong	0.72035
Attempt_21.csv	PCA applied, scaling only applied to training data, not testing	Error
Attempt_22.csv	PCA applied, scaling only applied to training data, not testing	0.53975
Attempt_23.csv	PCA individually fit to data sets, components set to 100, max iterations doubled to 400	0.58221
Attempt_24.csv	PCA individually fit to data sets, components set to 100, max iterations doubled to 400	0.66442
Attempt_25.csv	Same as previous, with the addition of Standard Scaler, individually fit to train + test datasets.	0.67890
Attempt_26.csv	Same as previous, but Standard scaler fit once to training data, then applied to both sets (training + testing).	0.67048
Attempt_27.csv	Train/test class ratio's corrected, now with PCA (100 components) instead of standard scaler. (Max iter's increased to 500 due to convergence warning)	0.54278
Attempt_28.csv	Binarize test! Impute (k=2), standard scaler, then binarize with threshold 0.293 as that equates to 61.52% = class 0 ratio	0.66307
Attempt_29.csv	Train/Test class imbalance resolved. No other pre processing - control	0.66004
Attempt_31.csv	Standardisation + KernalPCA with polynomial kernel. Non-linear so maybe will do better than normal PCA. components = 2440. max iters = 1000	0.52796
Attempt_32.csv	Standardisation + KernalPCA with polynomial kernel. Non-linear so maybe will do better than normal PCA. components = 2440. max iters = 1000	0.62803
Attempt_33.csv	KNN imputer investigation, k = 1	0.66037
Attempt_34.csv	KNN imputer investigation, k = 5	0.62904