Student Number: 195105 & Kaggle Team Name: Your Favourite Seminar Teacher

1. Introduction

This project was centered around a binary classification problem. The task was to train a classifier that given an image could predict if the image was 'sunny' or 'not sunny'. The training data images provided were taken by the beach whereas the testing data images were taken in urban areas. This added an interesting dimension to the problem as the source and target domain differed slightly. The training data was made up of 259 labeled data instances with 4608 features. The testing data was made up of 2818 unlabeled data instances of the same dimensionality. Additional data that was available for this task included additional labelled training data with some missing values, annotation confidence for all labeled data and the proportions of 0's and 1's in the training data labels.

2. Approach

Two slightly different approaches were used in tackling this problem. Both approaches involved the use of decision tree-based ensemble methods.

The first approach used was that of a Random Forest classifier. To understand a random forest one much first define a decision tree. An example decision tree can be seen in Figure 1. Decision trees suffer heavily in regard to overfitting to their training data, especially as they become more complex. Random forests look to resolve this issue by constructing multiple decision trees, utilizing the fact that using a group of classifiers can help to alleviate the bias present in each classifier as a standalone [1]. Random Forests introduce randomness in a model by splitting each iteration on a subset of features instead of the most important features in the whole feature set. This randomness works to deal with some of the issues decision trees have with overfitting. Classification in a Random Forest comes from the summation of the probabilities at each of the leaf nodes in the tree. By averaging the probabilities in this manner, a Random Forest classifier often achieves good results across a number of classification and regression problems.

The second approach used was that of a Gradient Boosted Ensemble. Much like a Random Forest this is made up of decision trees. However, instead of just splitting on a random subset, a Gradient Boosted model adjusts the weights assigned to the observations so that the features that are more difficult to classify carry more weight. In splitting at each iteration in this way by calculating the classification error from the new weighted model, an additional tree can be added, and this process starts again. This is often repeated until the calculated error stops improving or a finite limit is set.

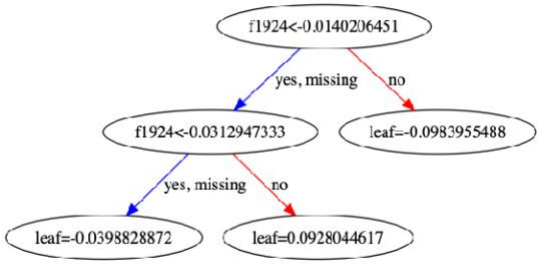

Figure 1 - Example Decision Tree

3. Methodology

3.1 Pre-Processing

As decision tree ensembles deal with high-dimensional data well the dimensionality of the data was not reduced using techniques like Feature Extraction or Principle Component Analysis (PCA). Moreover, both random and gradient boosted forests incorporate feature importance into their calculations, so the most important features are used to split the trees. Figure 2 demonstrates an example graphing of feature importance from a Random Forest ensemble. As a result, little value would have come from performing the aforementioned pre-processing techniques.

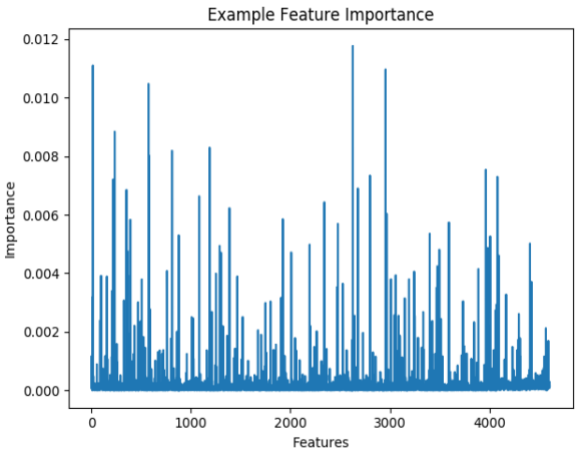

Figure 2 - Example Feature-Importance Plot from a Random Forest

Standardization was performed to scale the variance (leave standard deviation of 1) and remove the mean for

each feature with the motivation of making the imputation of missing values easier downstream. This standardization was performed using the StandardScaler from the Python library sklearn.preprocessing [2].

3.2 Dealing with NaN values

Although the additional training data set is labelled, it also contains a number of missing values (NaN values) across a number of the features. To make use of this much larger data set these NaN values were replaced. The data set was split into data labeled '0' and data labeled '1' so that the aggregated values used to replace the NaN values are better suited to the class label. The motivation being that better suited values with will provide a more valuable imputation. The values are replaced using the mean for each feature, utilizing the fillna method of NumPy class from the Python library SciPy [3]. Once imputed, the data set was rebuilt.

3.3 Dealing with the Class Imbalance

There is a distinct class imbalance in both the training and the test sets of data. The majority class in the testing data is '1'. Three methods were attempted to resolve this issue: leave the data oversampled, under-sample for balance, under-sample to make '0' the majority class. Figure 3 demonstrates the findings.

Class Weight	'1's	Balance	'0's
Accuracy ± SD (%)	93.59 ± 1.32	92.66 ± 1.59	91.79 ± 1.73

Figure 3 - Classify Accuracy of Different Class Balanced in Training Data

The K-Fold Cross Validation [4] performed suggests that leaving the class imbalanced with '1' as the majority class was the most effective as it achieved the highest accuracy on the validation set (50:50) taken from the training data. However, using the test data proportions provided, the test data set is imbalanced in the opposite direction containing 67.67% '0' as the majority class with the rest '1'. With this in mind, the testing data was imbalanced in favor of the class '0' to better reflect the reality of the test data. This imbalance was created by removing all data instances with a label of '1' and an annotation confidence of '0.66'.

3.4 Annotation Confidence

The confidence of the annotation given to each vector labelled as either '1' or '0.66'. This information was used for the sample_weights parameter for the RandomForestClassifier's fit() method from the Python library sklearn.ensemble [5]. This weighting will get the trees to sample those labeled with confidence '1' more often and this should improve the classifiers accuracy.

3.5 Test Proportions

The XGBClassifier class from the Python library XGBoost [6] has a hyper-parameter scale_pos_weight. This controls the balance of positive and negative weights. The values '0.67' and '0.32' were used for this to see which performed best. Given the imbalance in the test data, '0.67' brought better results and was used in the final implementation.

3.6 Domain Adaption Problem

The slight difference in domain between the training and test data sets (seaside images vs urban images) poses an interesting problem. Transfer Component Analysis [7] provides a means by which a shared feature space can be established between a labeled source domain and unlabeled target domain that maintains the most value across the two. The transfer_component_analysis() method of the TransferComponentClassifier class from the Python library libtlda.tca [8] was used to achieve this. This provided both a list of transfer components and the source and target domain kernel distances. As demonstrated in Figure 4, there are two features that show a considerable variance in distance when compared to the rest of the data set. These two components where isolated as features 2299 and 2432. With the intuition that these features may hold more value in the mapping across domains the XGBClassifier was leveraged to keep these features from interacting during training in an attempt to maintain their independent value. The 'interaction_constraints' hyper-parameter allowed this to be achieved. With these constraints in place, the gradient boosted classifier model did marginally improve in performance against the test set.

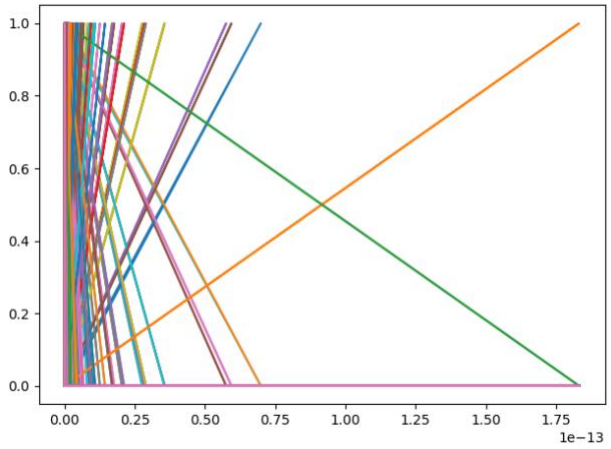

Figure 4 - Source and Target Domain Kernel Distance

3.7 Optimization of Hyper Parameters

Both the random forest and gradient boosted classifiers have a variety of hyper-parameters that can be set to constrain the number of trees, tree depth, number of estimators etc…

The hyper-parameters of the RandomForestClassifier [9] were optimized using a randomized grid search [10]. A randomized grid search was selected because of the wide range of values that a number of the hyper-parameters (such as max_depth, n_estimators and max_features) is very large. A grid search was not used due to the computational expense [11] of iteracting over all of these values. A randomized approach provides similar results with much less computational expense and was therefore used to find the best parameters for the random forest ensemble model. It was evaluated using precision. The returns parameters did require some slight manual alteration through experimentation to achieve a model that performed well on the test set and this is largely down to overfitting that will be discussed later. This was achieved using the RandomizedSearchCV class from the Python library sklearn.model_selection [12]

The hyper-parameters of the XGBClassifier [6] were optimized using a grid search. This was selected as the range of useful values for the hyper-parameters was considerably less dense. Instead a more directed search was used, performing 864 fits before deciding on a best parameter set.

3.8 Threshold Optimization

Both of the models that were used allow for a probability instead of a label to be returned when predicting. This introduced another parameter, the prediction threshold, i.e. what probability will act as the boundary between negative or positive label. The default is set to '0.5', so threshold optimizations were carried out using values around this as demonstrated in Figure 4.

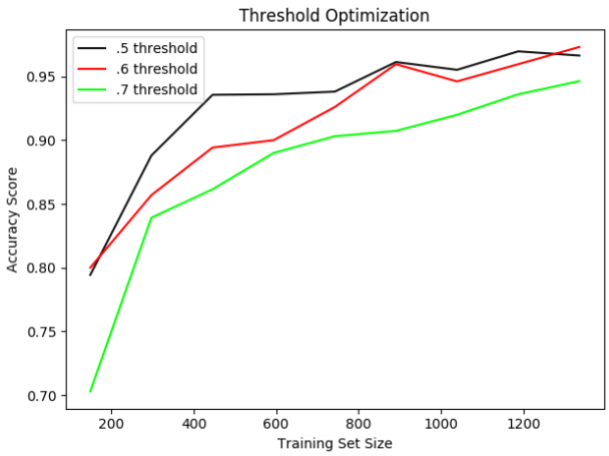

Figure 5 - Prediction Threshold Learning Curves

The optimizations highlighted a threshold of '0.6' to provide the best accuracy, with the trend for precision being exactly the same. The threshold was placed on the '1' label and not the '0' label as '1' is the minority class in the testing set. It is therefore more problematic to misclassify a '0' as a '1' than the alternative. This threshold compensates against them by requiring a higher confidence in terms of prediction for a '1' label to be produced.

4. Results and Discussion

4.1 Random Forest Ensemble Model

The highest performing model on the public facing test data was the Random Forest Ensemble with an accuracy of 72.301%. The model demonstrated similar success on any split of training/validation on the training set producing an accuracy of 85.10% as demonstrated in Figure 6. This is a somewhat promising score as it suggests that the model has not overfitted to the training data too much which would manifest itself in very high accuracy scores. The learning curve of the Random Forest Classifier shown in Figure 7 shows a steady increase in learning as more training data is given to the model. Its relative stability would suggest that the model could benefit from more training data without too much consequence in classifying both in the source and target domain.

Classifier	RF	XGB	XGB + RB
Accuracy (%)	85.10	92.02	90.48

Figure 6 - Accuracy of 3 Final Models

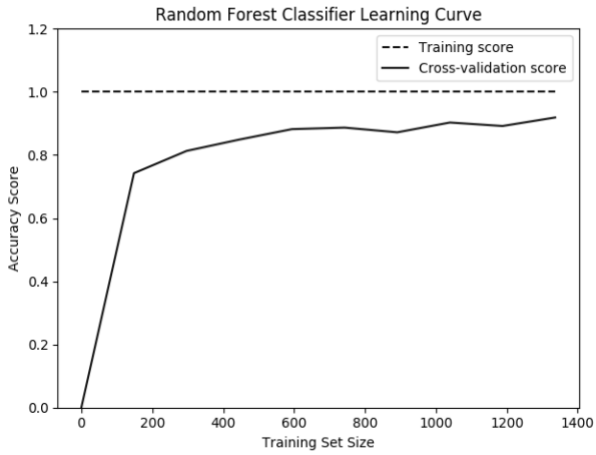

Figure 7 - Random Forest Model Learning Curve

4.2 Gradient Boosted Forest Model

The Gradient Boosted Forest Model achieved reasonable accuracy on the public facing test data achieving 68.181%. The Gradient Boosted Classifier model was selected by plotting the cross-validation score of classifiers with the 'best' parameters according to the grid search with

changing learning rate. As can be seen in Figure 8, the learning rate can have a reasonable impact on the fluctuation in the accuracy score of a classifier. With fluctuations at either end of the range, the classifier with the learning rate of '1.0' was selected as it was both the median value and close to the median accuracy score.

Looking at Figure 6 however, the model does appear to be overfitting (given the high accuracy of 92.02% on the validation split), and this would explain the diminished performance on the public facing data. Even with the encoding of some useful features across the source and target domain this model still did not perform as expected. A more extensive parameter tuning exercise with a greater range for each parameter is likely the solution to getting more value out of this model.

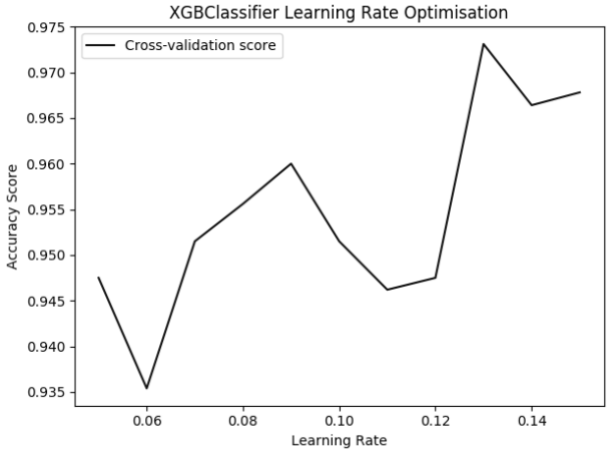

Figure 8 - XGBClassifier Model Selection Using Learning Rate

4.3 Mean of Random Forest and Boosted Gradient Model

Despite its suspected overfitting, the Boosted Gradient model does hold some value, especially in the potential mapping from source to target domain. As a result, a third model was devised using the well-fitted Random Forest to dampen some of the Gradient Boosted models overfitting by taking the mean prediction for each data point and applying a threshold of '0.6' for minority class classification. Variable success was had against the public testing data with this model ranging from 71.02% to 69.6%. The model that achieved the highest score did not perform as strongly in cross-validation, so the model chosen to classify the whole data set had an accuracy of 70.45%.

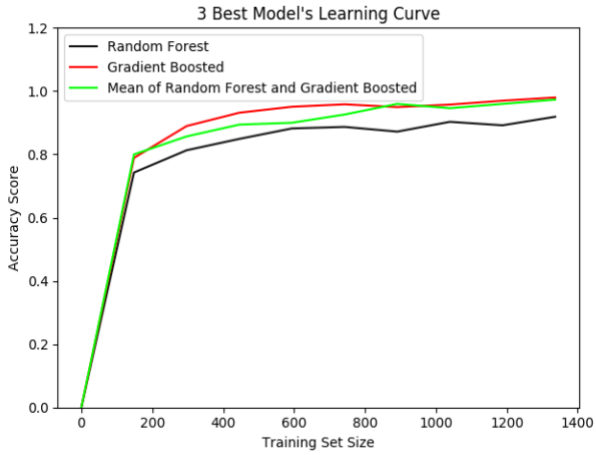

Figure 9 - Learning Curve of 3 Final Models

In looking at Figure 9 one can see the model fluctuate between the learning curves of the other two models and therefore could show promise. To improve this model further one could imagine creating a wrapper for both of the models and performing an extensive hyper-parameter tuning grid/random search that optimized the hyper-parameters of each against each other alongside the shared threshold, producing a single optimized model.

Moreover, the data returned from the Transfer Component Analysis is used quite sparingly. Further analysis into this, using it to motivate a more drastic mapping across the domains may result in a model that performs slightly worse on the source domain but reasonably better on the target. This could be considered an acceptable trade-off.

4.4 Final Criticisms

In performing standardization at the beginning of the pre-processing pipeline a lot of useful data that may have in utilized in the imputation stage was lost. In further iterations of this project it would be recommended to instead normalize the data before replacing the NaN values with either the mean or through the use of some manner of logistic regression. The latter would be interesting, providing additionally estimated information may provide a better ground by which a feature space that accurately maps both the source and target domain could be established.

Moreover, given that some overfitting has occurred in the model's future iterations may look to prune the trees in the forests in an attempt to get the best yet smallest model. Perhaps tradeoffs could be made regarding some cross-validation accuracy at the benefit of less chances of overfitting.

References

[1] T. K. Ho, "Random Decision Forests," *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995.

[2] scikit-learn, "sklearn.preprocessing.StandardScaler," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed 14 05 19].

[3] SciPy, "NumPy v1.16 Manual," [Online]. Available: https://docs.scipy.org/doc/numpy/user/quickstart.html. [Accessed 14 05 19].

[4] scikit-learn, "sklearn.model_selection.KFold," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html. [Accessed 14 05 19].

[5] scikit-learn, "Ensemble Methods," [Online]. Available: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble. [Accessed 14 05 19].

[6] XGBoost, "XGBoost Documentation," [Online]. Available: https://xgboost.readthedocs.io/en/latest/index.html. [Accessed 14 05 19].

[7] I. W. T. J. T. K. Q. Y. S J Pan, "Domain Adaptation via Transfer Component Analysis," 2009.

[8] W. M. Kouw, "Transfer Component Classifier," [Online]. Available: https://libtlda.readthedocs.io/en/latest/classifiers/tca.html.

[9] scikit-learn, "sklearn.ensemble.RandomForestClassifier," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed 14 05 19].

[10] Y. B. James Bergstra, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281-305, 2012.

[11] H. L. R. P. A. Jasper Snoek, "Practical Bayesian Optimization of Machine Learning Algorithms".

[12] scikit-learn, "3. Model Selection and Evaluation," [Online]. Available: https://scikit-learn.org/stable/model_selection.html. [Accessed 14 05 19].