Capstone Project Report - Forest Cover Type Prediction

Mohan Prasath Chinnasamy

January 7, 2017

1 Definition

1.1 Project Overview

Our Earth is an unique planet in the solar system holding the key to sustain Life. Even now many parts of the oceans, and forests remain unexplored. Several species are discovered periodically in remote forests across the globe. Under further genetic studies it is revealed that we humans do posses some characteristics of the DNA of the mammals!. Exploration of such remote regions can provide answers for many unanswered questions regarding origin of life, evolution, and life in unpolluted environments[1].

However such explorations are costly and even dangerous for humans. Aerial surveillance have improved much during these last decades. These technological improvements can assist in human explorations by capturing images of unexplored regions reducing cost and human effort. But most forests are dense with vegetation and such aerial surveillance can provide only the top view. We can then use the surveillance data(tabular data) to predict and restrict the search space for human explorations. From these explorations an extrapolation[3] can be done about missing species in similar areas, thus reducing the search space further. This project uses the Covertype dataset obtained from UCI Machine learning Repository[2]. A list of all relevant publications made based upon the Covertype dataset can be found here.

1.2 Problem statement

This project aims to use surveillance data with cartographic information to train models that can predict and identify the forest cover types. The project also aims to create a classification model with more than 90% accuracy which can classify the forest cover type. The tasks can be fairly summarized and listed below,

- Download and analyze the Covertype dataset from UCI Machine Learning Repository[2]

- Identify the correlated variables in the dataset.
- Select and train a list of classifiers to identify those correlated variables.
- Evaluate those classifiers using metrics
- Select specific classifiers for further fine tuning using the metrics
- Evaluate the selected classifiers again for measuring their performance improvement or consistency.

From the above steps a single classifier can be chosen finally to solve any future Covertype dataset problem.

1.3 Metrics

Performance of various classifiers can be ranked and finally we can chose highest ranked classifier using all of the following metrics,

- Precision
- Recall
- Accuracy
- F1-score

Precision and Recall are actually used to measure the relevance of information and their measurement. In order to understand, the above listed metrics, we first need to understand the four measurement rates: True Positive(TP), True Negative(TN), False Positive(FP), False Negative(FN). Lets consider the well known story of The Boy Who Cried Wolf as an example here.

- If the boy has cried wolf, but on examination there was no wolf to be seen, then this is a False Positive(FP).
- If the boy didn't cry wolf, but on no examination there was a wolf, then this is a False Negative(FN).
- If the boy has cried wolf, and on examination there was a wolf, then this is a True Positive(TP).
- If the boy didn't cry wolf, and on examination there was no wolf, then this is True Negative(TN).

Here "The boy crying wolf" can be considered as a prediction, while the villagers examining the area for wolf can be considered a condition. Further discussion about this topic can be found here. From this explanation, all the listed metrics can be mathematically derived as follow,

- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $F1 - score = 2 * \frac{Precision*Recall}{Precision+Recall}$

All above scores can be calculated by measuring the occurrence of true positive, true negative, false positive and false negatives. Also the true positive rate(Sensitivity) and true negative rates(Specifivity) are used in the above calculations. The performance of the model can be evaluated by calculating the prediction accuracy. For testing, 20% of the total data set will be used which is not used to train the model. All classification models will be trained with remaining 80% of the data. This huge amount of data used to train the models can increase the prediction accuracy over the unseen test data. For example, a k-Nearest Neighbor model that can classify the features into k target variable can be applicable here. Here the k-NN can classify a data point into any one of the class of the target variable, since we already have the actual target variable provided with the data, we can easily measure the predicted value against the actual value. So the above metrics all can easily be calculated.

2 Analysis

2.1 Data Exploration

The cover type data set was acquired from UCI Machine Learning Repository[2]. The data was collected in four major areas of wilderness in Roosevelt National Forest of northern Colorado. This region is affected minimally by human intervention. So this data also holds information about ecological changes occurring over time.

The data set has 12 major features with one multivariate target variable with information about forest cover type. Two features Ẃilderness_Area; and Śoil_Typeáre further divided into 4 and forty binary valued sub features respectively. Thus adding up all features to a count of 54. There are 581012 instances of the data with no missing values.

The features and the target variable of the data set are listed below,

- Elevation-quantitative data measured in meters
- Aspect-quantitative data measured in degrees
- Slope-quantitative data measured in degrees
- Horizontal_Distance_To_Hydrology-quantitative data measured in meters
- Vertical_Distance_To_Hydrology-quantitative data measured in meters
- Horizontal_Distance_To_Roadways-quantitative data measured in meters
- Hillshade_9am - quantitative data from 0 to 255 index

- Hillshade_Noon - quantitative data from 0 to 255 index
- Hillshade_3pm - quantitative data from 0 to 255 index
- Horizontal_Distance_To_Fire_Points - quantitative data measured in meters
- Wilderness_Area (4 binary columns) - binary data with 0 (absence) or 1 (presence)
- Soil_Type (40 binary columns) - binary data with 0 (absence) or 1 (presence)
- Cover_Type (7 types) - integer data from 1 to 7

Above features are measurements of a forest and the landmass around the forests. This information is provided to us as a tabular numerical data. All the above data contains their respective forest type(our target variable) preclassified.

2.2 Exploratory Visualization

In this section, the data is studied visually in-order to understand the hidden relationships of various attributes. However, the visual explorations can only be done effectively on non-binary data. So we avoid columns related to wilderness_area and soil_type. The following picture1 is a scatter plot of all the variables against all attributes themselves. From 1 we can observer various relationship among the data attributes. Some of them are listed below,

- (aspect, hillshade_3pm) and (aspect, hillshade_9am) are two pairs of attributes exhibiting a sigmoid relationships2.
- (hillshade_9am, hillshade_3pm) and (hillshade_noon, hillshade_3pm) are two pairs of attributes exhibiting an ellipsoid relationships3.
- (vertical_distance_to_hydrology), (horizontal_distance_to_hydrology) exhibit a linear pattern relationship. 4

The above relationship have to be verified, so the correlation score among those attributes are calculated and higher correlation(absolute values above 0.5) are listed below:

- hillshade_9am and hillshade_3pm = -0.78
- aspect and hillshade_3pm = 0.65
- horizontal_distance_to_hydrology and vertical_distance_to_hydrology = 0.61
- hillshade_noon and hillshade_3pm = 0.59
- aspect and hillshade_9am = -0.58

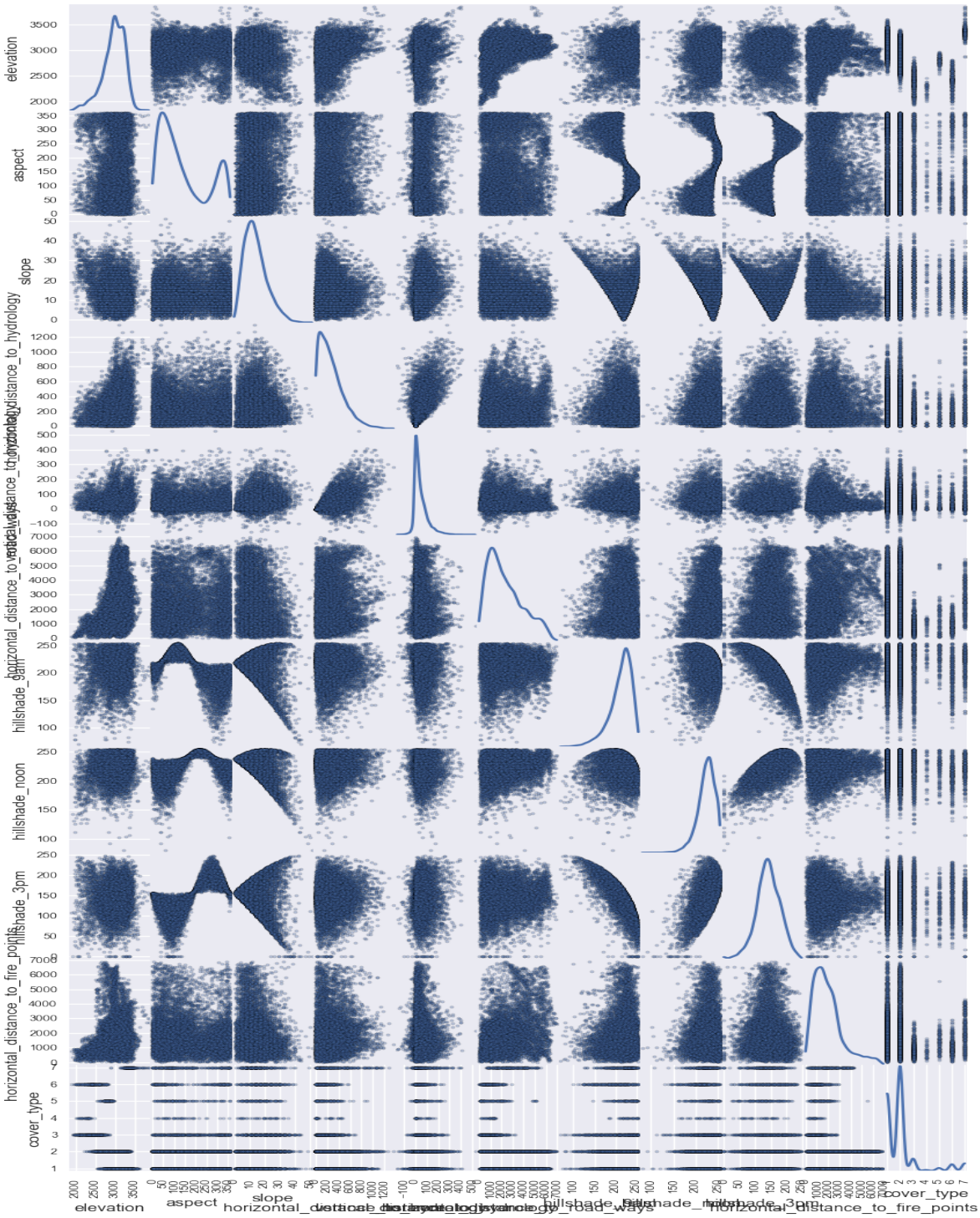
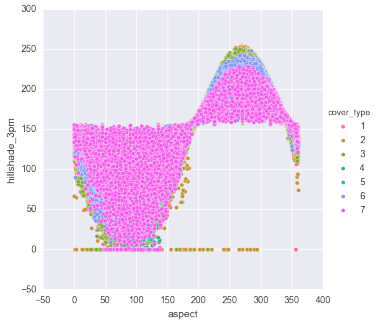
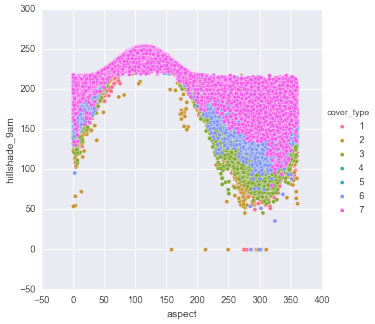


Figure 1: A scatter plot of all non binary attributes in the data

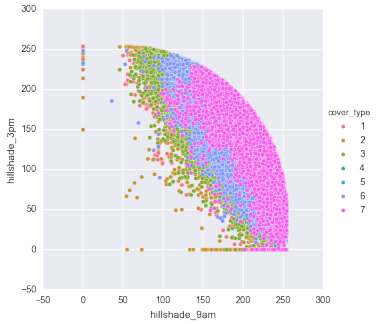


(a) Aspect vs Hillshade3PM

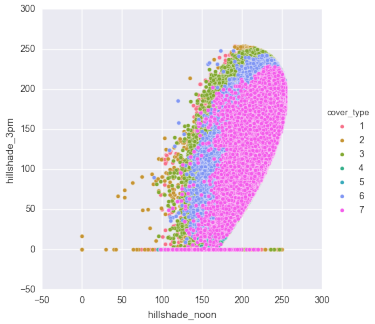


(b) Aspect vs Hillshade9AM

Figure 2: Aspect VS Hillshade 3PM and 9AM

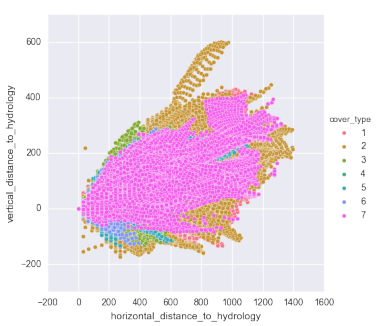


(a) Hillshade9AM vs Hillshade3PM

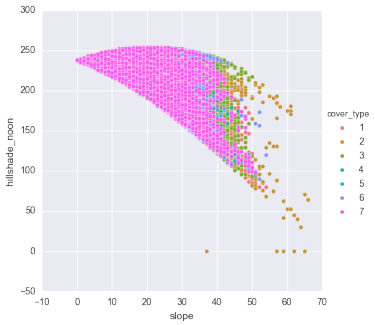


(b) HillshadeNoon vs Hillshade3PM

Figure 3: Hillshade VS Hillshade



(a) Horizontal vs Vertical distance to hydrology



(b) Slope vs HillshadeNoon

Figure 4: Distance to Hydrology and Slope VS Hillshade

- slope and hillshade_noon = -0.53

Further examinations can be done by viewing the attached images with this project report or by viewing the plotted results in the Ipython notebook named **Capstone Project Forest Cover Type Prediction.ipynb**

2.3 Algorithms and Techniques

2.3.1 Classification

Let's consider, we are given a data set with x, y pairs of data where value y is a function of x. The values of y are categorical and discrete and are called as classes. We have to identify groups of x data point that belongs to a particular class of y. We can train a model to do the classification process and later test the model for its accuracy of classification on an unseen data. This type of classification are called as Supervised learning. Also depending upon the number of classes in y, we can also classify this model as Binary classification(y has two classes) or Multi-class classification.

Forest cover type data set's target variable contains seven unique values. They are 1, 2, 3, 4, 5, 6, and 7. Also we are already given the target variable, so this project is a Supervised classification project. Since there are more than two classes in the target variable we consider this classification model as Multi-class classification. Some random algorithms are chosen to predict the target variable and from their basic predictions a mean accuracy score is calculated. Those score are listed below.

- LinearSVC
- SVC with sigmoid kernel
- SVC with rbf kernel
- SGDClassifier
- KNeighborsClassifier
- RandomForestClassifier
- XGBClassifier

From the above list of classifiers, we can notice that three classifiers, namely KNeighborsClassifier, RandomForestClassifier, and XGBClassifier shows highest classification accuracy(more than 0.7). We focus on these algorithms and fine tune their attributes to increase the accuracy further.

2.4 Benchmark

The forest cover type data set is split into test and train set at 80% and 20% respectively. This would create a training set with 464809 samples and test set with 116203 samples. We focus on training a classification model, with more than 90% accuracy because the implication of a wrong classification can be huge. For example, spending is huge in human forest explorations including medical, transport, insurance, machinery, navigation, etc., Explorations done based upon results of classifier is supposed to yield good results. Also we use the existing target variable for measuring the test set. This gives us scores for precision, recall, accuracy, and f1-score. We plan to select a best classifier that can score consecutively higher in all the scores and at low cost(running time).

3 Methodology

3.1 Data Pre-processing

Data pre-processing is done in a data to identify the presence of any missing values or Outliers in the data. These have to be removed before the algorithm starts classifying the data to avoid any skewed results. On examination of the data, no missing values and no Outliers are found in the data. This can be verified in the attached iPython notebook.

3.2 Implementation

In this section, all three classification algorithms chosen for fine tuning are discussed in detail along with their logic. Each of the following classifiers undergo three stages: training, testing and scoring respectively. In training phase, around random 80% of the data is used to fit the data. Later in the testing phase, the remaining 20% of the attributes data is used to predict the target variable(here it's the cover_type). Then the predicted target variable is compared against the actual target variable obtained with the raw data to calculate all the metrics listed in Section 1.3. Most of the algorithms were available for use from sklearn library and xgboost library. Major difficulties faced during implementation was during fine tuning of the attributes being passed to these classifiers. The relevant information about each of these attributes are discussed in following sections in detail.

3.2.1 KNeighborsClassifier

KNeighborsClassifier is developed based upon the concept of Nearest Neighbor Classification. Initially this classifier mode is trained with the training data. When unseen data is given as input to the model, it selects **k** nearest neighbors to a data point and identifies the common class for those n neighbors. The common class is then assigned as the appropriate class for a decision tree being investigated.

3.2.2 RandomForestClassifier

RandomForestClassifier is developed based upon the concept of random decision trees. Random Forest is an ensemble learning method which builds decision trees during training time and later during testing time, outputs the modes of the most occurring class for a data point. Both Random Forest classifiers and k-NN can be viewed as *weighted neighborhoods schemes*. The following paragraph(Quoted from scikitlearn webpage) best explains the Random Forest further.

"In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model."

3.2.3 XGBClassifier

XGBClassifier is based upon the famous machine learning concept called Gradient boosting. It was originally created on the idea of improving a weak learner into a better learner. Usually a weak learner is defined as the one whose performance is slightly better than random choice. Gradient boosting has three components: optimizing a loss function, a weak learner making predictions, and an additive model to add weak learners to minimize the loss function. Based upon these functions a booted classifiers is built and it's source code can be found in GitHub.

3.3 Refinement

Several refinements are done to the three classifiers used in the project. This refinements are specified in their Python class attributes during their initialization. All those required attributes are discussed in detail in the following subsections.

3.3.1 Attributes to fine tune for KNeighborsClassifier

The attributes for KNeighborsClassifier are listed below:

- n_neighbors was changed to 10, default was 5.
- weights was changed to distance, default was uniform. This gives more importance to closest neighbors rather than the count of most common class among the neighbors.

- algorithm remains unchanged, default is auto. This allows the model to pick the best approach to select nearest neighbors from the available options.
- leaf_size is set to 10000, default value was 30. This value is increased to a higher value, but the optimal value cannot be known.
- p uses the default value.
- metric uses the default 'minkowski' distance metric.
- metric_params was set to default value None.
- n_jobs was set to -1, default was 1. This makes the model to utilize all the cores in the hardware for calculations.

3.3.2 Attributes to fine tune for RandomForestClassifier

The attributes for RandomForestClassifier are listed below:

- n_estimators was set to 500, default was 10.
- criterion was set to default 'gini'.
- max_depth was set to default value None, this allows the tree to expand unbounded.
- min_samples_split was set to default value 2, at least two samples are required to split a node in the tree to two other sub nodes.
- min_samples_leaf was set to default value 1, at least one sample is required to form a leaf node.
- min_weight_fraction_leaf was set to default value 0.0
- max_features was set to default value 'auto', that is square root of number of features, $\tilde{7}.35$.
- max_leaf_nodes was set to default value None, allowing the tree to grow to the maximum size.
- min_impurity_split was set to default value 1e-07, thus stopping the tree from further growth.
- bootstrap was set to default value True.
- oob_score was set to default value False.
- n_jobs was set to -1, default was 1. This makes the model to utilize all the cores in the hardware for calculations.
- random_state was set to 42, default value was None.

- verbose was set to default value 0
- warm_start was set to default value False
- class_weight was set to default value None. This is set to 'None' because the target variable is not balanced. It has more data points for class 1 and 2. So building a tree for the target variable in balanced manner is not possible.

3.3.3 Attributes to fine tune for XGBClassifier

The attributes for XGBClassifier are listed below:

- max_depth was set to 50, default value as 3.
- n_estimators was set to 500, default value as 100.
- silent uses default value True.
- objective uses "multi softmax", default was "binary logistic". Since we use multiclass classification, binary classification wont work here. More related discussion.
- nthread was set to -1, default was 1. This makes the model to utilize all the cores in the hardware for calculations.
- gamma uses default value 0.
- min_child_weight uses default value 1.
- max_delta_step uses default value 0.
- subsample uses default value 1.
- colsample_bytree was changed to 0.3, from default value of 1.
- colsample_bylevel uses default value 1.
- reg_alpha uses default value 0.
- reg_lambda uses default value 1.
- scale_pos_weight uses default value 1.
- seed uses default value 0.
- base_score uses default value 0.5.
- missing uses default value None.

4 Results

4.1 Model Evaluation and Validation

Using only three classifiers reduced the need for fine tuning a large set of classifier attributes. However some attributes are similar in nature among these classifiers, for example, utilization of all cores in the computer. This attribute was easy to identify from the documentation of these classifiers. All the fine tuning were performed in a trail and error basis. A portion of the original raw data is sampled(n=100,000), then the sample data was split into training(80,000) and test(20,000) data respectively. Evaluating the small data was quick and provided valuable information on the attribute refinement process. This is how all the three classifiers was evaluated and validated.

4.2 Justification

Justification of the three classifiers can be done scientifically based upon the metrics: precision, recall, accuracy and F1-score. However another metric needs to be considered here. It's the running time of these classifiers. Since all algorithms were executed upon a single computer with multiple cores while no other additional processes running. Running time gets higher precedence. We are working on relatively huge set of data points, roughly more than five million. Even several algorithms were eliminated in the earlier phases of this project, because they required more running time. All these metrics are discussed in the following subsections.

4.2.1 Results of KNeighborsClassifier

Before fine tuning the algorithm generated a mean accuracy score of $\tilde{\mathbf{0}}$**.96870**. After fine tuning the mean accuracy score remained unchanged with a value of $\tilde{\mathbf{0}}$**.961154**. Using the scaled data, the algorithm had a mean accuracy score of $\tilde{\mathbf{0}}$**.916362**. This is a reduced score. However this algorithm has achieved our initial goal to create a classifier model with more than 90% accuracy. Running time of KNeighborsClassifier is $\tilde{2}3$ to $\tilde{3}0$ minutes. Finally, it seems that the algorithm cannot be fine tuned anymore. So we will look into the next algorithm.

4.2.2 Results of RandomForestClassifier

Before fine tuning the algorithm generated a mean accuracy score of $\tilde{\mathbf{0}}$**.943143**. After fine tuning the mean accuracy score was slightly increased with a value of $\tilde{\mathbf{0}}$**.955982**. Using the scaled data, the algorithm had an increase in mean accuracy score with a value of $\tilde{\mathbf{0}}$**.956042**. The algorithm responds to fine tuning and attribute scaling. Running time of RandomForestClassifier is $\tilde{3}$ minutes.

This is the fastest algorithm for classifying the forest cover type dataset. So RandomForestClassifier can be recommended to be used on similar data sets in the future for accurate predictions. This algorithm has achieved our initial goal to create a classifier model with more than 90% accuracy.

4.2.3 Results of XGBClassifier

Before fine tuning the algorithm generated a mean accuracy score of $\tilde{\mathbf{0}}$**.74403**. After fine tuning the mean accuracy score was increased highly with a value of $\tilde{\mathbf{0}}$**.95894**. Using the scaled data, the algorithm had a increase in mean accuracy score with a value of $\tilde{\mathbf{0}}$**.95895**. The algorithm responds to fine tuning only. Running time of XGBClassifier is $\tilde{4}$1 minutes. **This is the slowest algorithm for the forest cover type data set.** This algorithm has also achieved our initial goal to create a classifier model with more than 90% accuracy.

5 Conclusion

All classifiers were successfully fine tuned to achieve an accuracy score above 90%. From the three classifiers, we can notice that after fine tuning, XGBClassifier performance is increased by a huge margin. Also an accuracy increase from $\tilde{0}$.74 to $\tilde{0}$.95 is observed. Also our initial plan to achieve an accuracy of above 90% is also achieved here. However, **RandomForestClassifier** performs consistently better when considered to all other classifiers. It produces an accuracy score of $\tilde{0}$.96 which is the highest observed in this project. Also it is the fastest classifier in this project. **So from the list of classifiers used in this project I would recommend RandomForestClassifier to be used for further analysis in similar type of forest cover identification. Since it has consistent highest accuracy score.**The following subsection summarizes all the steps taken in this project.

5.1 Free Form Visualization

Most of the results were based upon the metrics and also ROC curve plot was not feasible for multi-class classification. So more importance were given to the metrics and this can be studied in detailed in the attached iPython notebook. The time consuming tasks are performed by the classifiers. After scaling the data, it takes more amount of time to train the classifiers. Since a new metric was introduced later in the project: running time, it would be useful to discuss them using bar charts. The following image5 shows the running time of all three classifiers with respect to their training and testing time. Here the data used by the classifiers is of two type: un-scaled(represented as u_classifier) and scaled(represented as s_classifier) in the X-axis of the image. The Y-axis represents the total running time in seconds.

From the figure5, it can be observed, that for both scaled and un-scaled versions of the data, RandomForestClassifier scores very minimum running time. This is a great indicator of the efficiency of the RandomForestClassifier over all the remaining classifiers. This confirms that RandomForestClassifier is the best of all classifiers considered for further fine tuning.

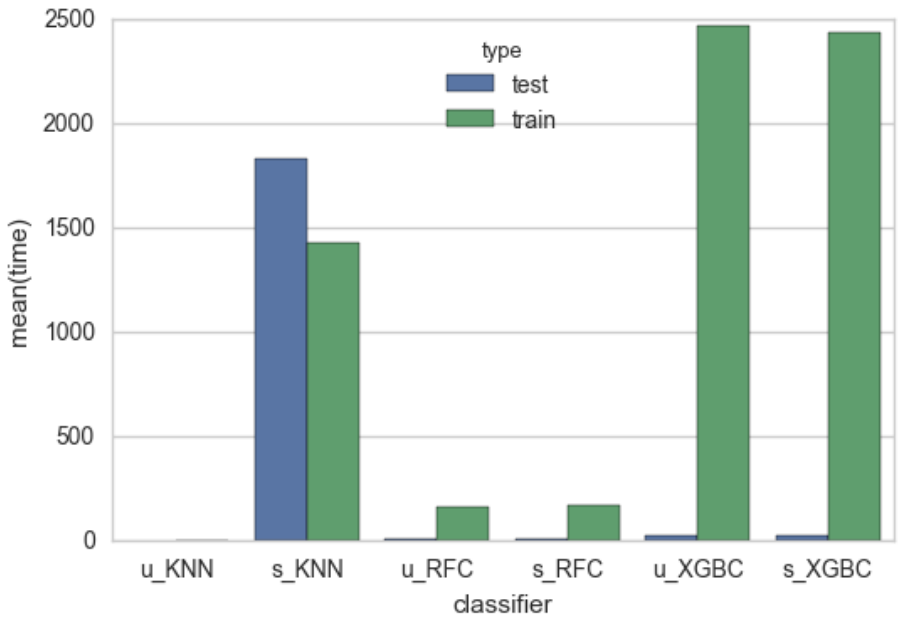


Figure 5: A bar plot of training and testing time of un-scaled(u) and scaled(s) data by the classifiers

5.2 Reflection

All the processes involved in completing this project is listed below,

1. Forest Covertype dataset was found in UCI Machine Learning Repository. [2]
2. The dataset was downloaded and preprocessed.
3. All the metrics required for the project analysis were defined.
4. A benchmark model based upon the metrics was created.
5. A initial set of classifiers were chosen to be evaluated against the preprocessed data.
6. Three classifiers with the highest mean accuracy score were chosen for fine tuning.
7. All three classifiers were fine tuned and fitted with the training data.
8. The test data is the used by the classifiers to predict the target variable.
9. The predicted target variable is then evaluated against the actual target variable.

10. New metrics were calculated for all the above three classifiers.

There were some difficulties during the process of fine tuning. The step 7, required more analysis time in terms of sampling the data and recording the changes when various attributes were fine tuned. Here the documentation of these classifiers were highly influential in my decisions. I was very surprised by the final mean accuracy score of the classifiers.

5.3 Improvement

The RandomForestClassifier still has an error rate of 4%. As mentioned earlier in the Project Overview, the results of this project can be used to guide a human team of forest explorers. In such case, a 96% accuracy is appreciated, but we still have to acknowledge an error rate of 4% is high. This can still be improved by training the model with more data points. Our Earth has a huge amount of forest land mass that still remain unexplored. Collecting data from those regions and training the model with more data can improve the accuracy of the RandomForestClassifier.

References

[1] Christopher W. Dick and W. John Kress. Dissecting tropical plant diversity with forest plots and a molecular toolkit. *BioScience*, 59(9):745–755, 2009.

[2] M. Lichman. UCI machine learning repository, 2013.

[3] Rosanne Tackaberry, Nicholas Brokaw, Martin Kellman, and Elizabeth Mallory. Estimating species richness in tropical forest: the missing species extrapolation technique. *Journal of Tropical Ecology*, 13(3):449–458, 005 1997.