Forest Cover Type Prediction

Kaggle Competition

Joe Sieber

Table of Contents

Table of Contents 1

Abstract 2

Definition 3

- Introduction 3
- Problem Statement 3
- Metrics 3

Analysis 5

- Data Exploration 5
- Data Visualization 8
- Algorithms and Techniques 10
- Benchmark 11

Methodology 12

- Preprocessing 12
- Implementation 12
- Refinement 13

Results 18

- Model Evaluation and Validation 18
- Justification 18

Conclusion 19

- Free-Form Visualization 19
- Reflection 20
- Improvement 20

References 21

Abstract

Kaggle has a competition based on data provided by Colorado State University to classify the type of tree cover that are growing on a plot of land. There are 7 types of tree cover classifications. Kaggle provides a test and training set. The training set contains 15120 examples and the test set contains 565892 examples both with 54 features. The training set was further split into a cross validation set and a training set.

The goal of this project was to create a classification algorithm that does better predicting on the test set then 71.1%. This benchmark comes from the original study of this data by Colorado State University. Kaggle and this paper will use multi-class classification accuracy as the metric to determine how well any algorithm generated performs.

Many different machine learning algorithms were tried with varying success. The Scikit library for Python was used for these algorithms. There was no preprocessing of the data. The best performing methods were gradient boosting, random forest, and extra random forest models. Each of these models was optimized on the cross validation training and test set. After tuning the models, their predictions were each fed into a voting systems which found the most frequently occurring predicted tree cover type and made that the final output classification. The final voting model was trained on all of the training data and then predicted on the test set. This prediction was submitted to Kaggle for evaluation and the multi-class classification accuracy for 76.5%. This significantly outperformed the benchmark.

Definition

Introduction

There are many national and state parks in Colorado. These parks contain some types of forest cover that can be narrowed down to five different groups. The parks can be very remote and hard to navigate the entire park. With such a large area to cover, it would be desireable to be able to know what types of forest cover are a specific locations without having to send people to inspect. This would be very expensive and time consuming. Knowing what type of forest cover is at a given location helps the United States Forest Service (USFS) allocate resources and time to areas that might need more assistance. This also helps the USFS determine if manmade features affect the outcome of what type of forest cover is present for a given location.

The data for this problem was put together by the Colorado State University in order to facilitate answering these questions. This problem was also put forth to the Kaggle community in order to help refine the answer and make better predictions.

Problem Statement

If the forest cover type for all of Colorado was known, it would be easier to allocate resources and time to areas in need. However, classifying the forest cover type for all of the land area in Colorado is a daunting task. The goal of this project is to predict what the tree cover type is for a plot of land based some general features of that area. A machine learning classification algorithm will be used to solve this problem. The final solution to this problem will be an optimized and tuned algorithm that will classify the plot of land, based on its features, into 7 different tree cover types. This algorithm will be used to classify the test set to generate a submission file to be evaluated on Kaggle.

Solving this problem with machine learning, although not perfect but reasonably close, would allow a small subset of the land to be sampled in order to create a training set. If successful, this would be a tiny fraction of money need compared to do this manually.

Metrics

In order to determine how successful the machine learning algorithm is the dataset will be split up into three groups. The data received from kaggle was already split into a training set and test set. The training set will be further split into a cross validation training set and a cross validation test set. After the model is optimized and tuned on the two cross validation sets using the same multi-class classification accuracy, the model will be trained on the original full training

set from Kaggle. The overall performance of the algorithm will be determined on the test set using multi-class classification accuracy. This metric was used by the Kaggle competition. The reason Kaggle used this metric was because it is a straightforward and easy to interpret metric allows for comparison between algorithms. The accuracy is calculated by summing up all of the correct answers that the algorithm classified and dividing by the total number of examples. As the accuracy approaches 100%, the algorithm is getting more and more classifications correct.

The single accuracy score does not show how well each type of tree cover is classified. This metric will just give the average of all of the individual accuracies of each tree cover type. For example, if there are three types of tree cover with equal number of examples and the algorithm is able to classify two of those types with 100% accuracy and the other type of tree cover with 0% accuracy the multi-class classification accuracy would be 67%. This metric is appropriate for this type of problem because this is a classification problem and the goal is to try to classify the most number of plots of land to the best of our ability.

Analysis

Data Exploration

The data provided for this analysis has a training and test set already split. Table 1 shows some basic statistics of those data sets.

Number of Training Examples	15120
Number of Test Problems	565892
Number of Features	54
Number of Classifications	7

Table 1 - Basic Statistics

The 54 features are shown in Table 2. The Soil Type is broken up into a 40 different binary features. They are represented as with a 0 or 1 to show if that type of soil exists at that location. The Wilderness Area is also similar. It is broken up into 4 different binary features.

Feature Name	Feature Description
Elevation	Elevation in meters
Aspect	Aspect in degrees azimuth
Slope	Slope in degrees
Horizontal Distance To Hydrology	Horizontal distance to nearest water feature
Vertical Distance To Hydrology	Vertical distance to nearest water feature
Horizontal Distance to Roadways	Horizontal distance to nearest roadway
Hillshade 9 am	Hillshade index at 9 am on summer solstice
Hillshade Noon	Hillshade index at noon on summer solstice
Hillshade 3 pm	Hillshade index at 3 pm on summer solstice
Horizontal Distance to Fire Point	Horizontal distance to nearest ignition point
Wilderness Area	Wilderness area designation (1-4)
Soil Type	Soil type designation (1-40)

Table 2 - Data Set Features

The data does not have any missing information and each example is complete. The 7 cover types represent various types of trees. Table 3 shows those tree types and what the cover type number is. For the training set there are an equal number of examples of each cover type.

Cover Type Number	Type of Tree
1	Spruce / Fir
2	Lodgepole Pine
3	Ponderosa Pine
4	Cottonwood / Willow
5	Aspen
6	Douglas - fir
7	Krummholz

Table 3 - Cover Types

A few statistics about some of the features are listed in Table 4.

	Min	Average	Max
Slope	0	17	52
Aspect	0	157	360
Elevation	1863	2749	3849
Horizontal Distance To Hydrology	0	227	1343
Vertical Distance To Hydrology	-146	51	554
Hillshade Noon	99	219	254

Table 4 - Basic Statistics of Select Features

Data Visualization

The entire training data set was looked at. There were several features that seemed to show interesting findings. However, some of the features seemed to have no influence on the cover type. For example, Aspect seemed to be equally distributed amongst all of the cover types. This is show in Figure 1.

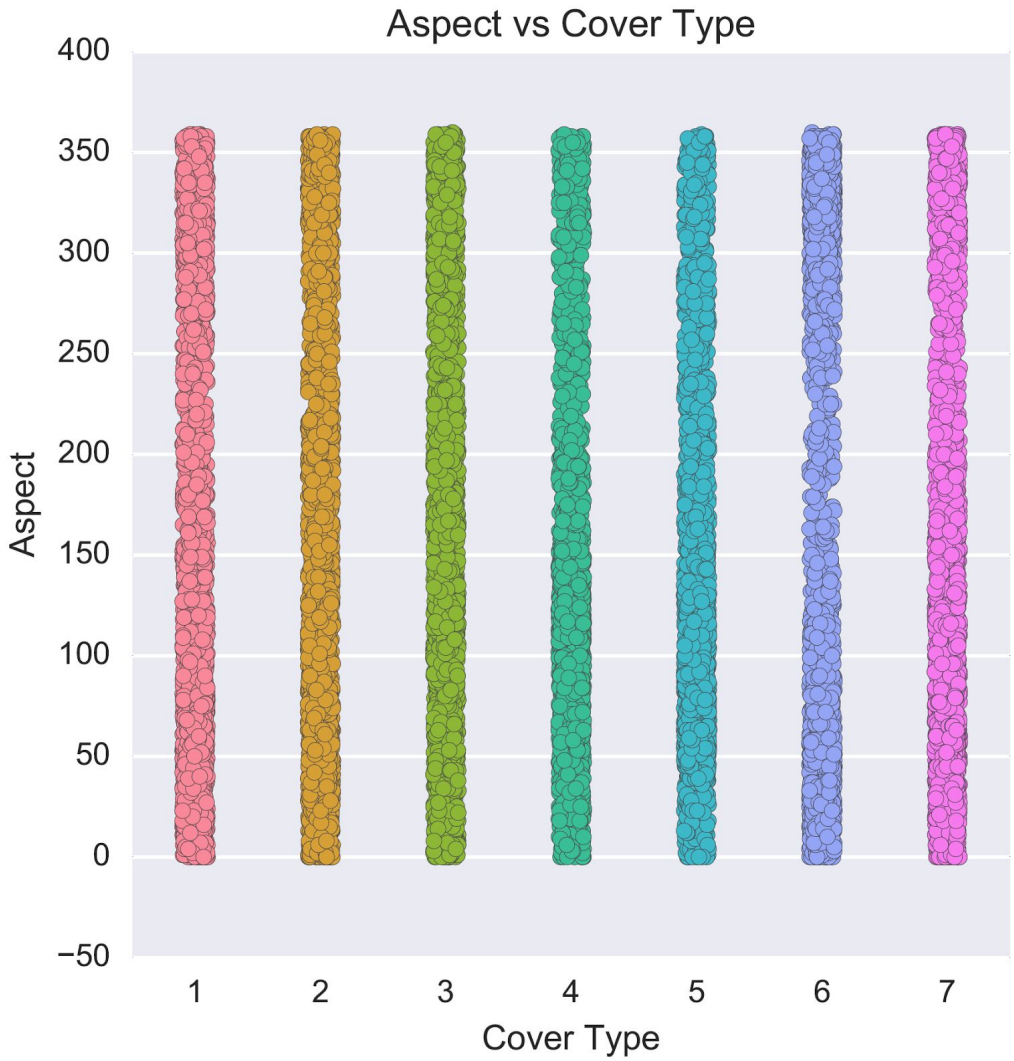

Figure 1 - Aspect vs Cover Type

Elevation seems to be an important feature. Figure 2 shows the elevation compared to cover type. This would make sense since different trees grow at different elevation.

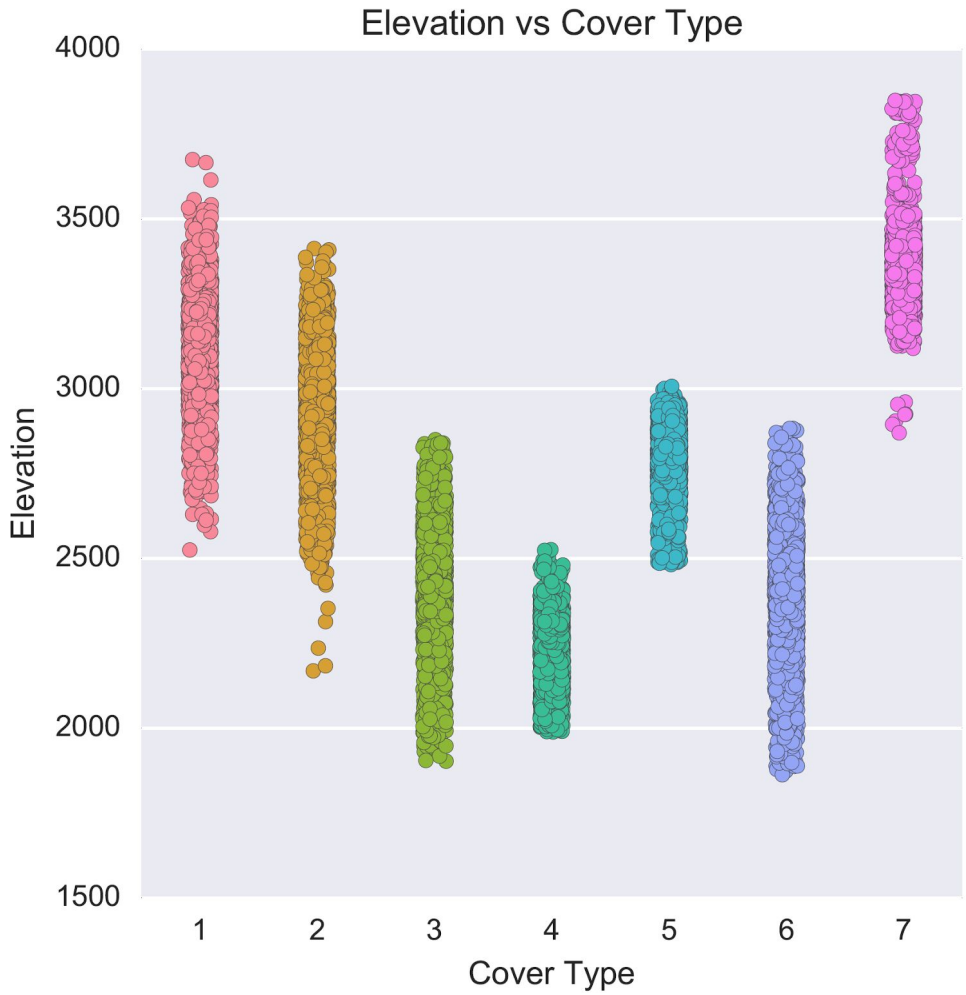

Figure 2 - Elevation vs Cover Type

The location of the sampled plot of land also seems to also determine what type of cover is at that location. Figure 3 shows what type of cover is a located at the 4 different wilderness areas. Each wilderness area, one through four, represent different parks. These are Rawah, Neota, Comanche Peak, and Cache la Poudre Wilderness Areas.

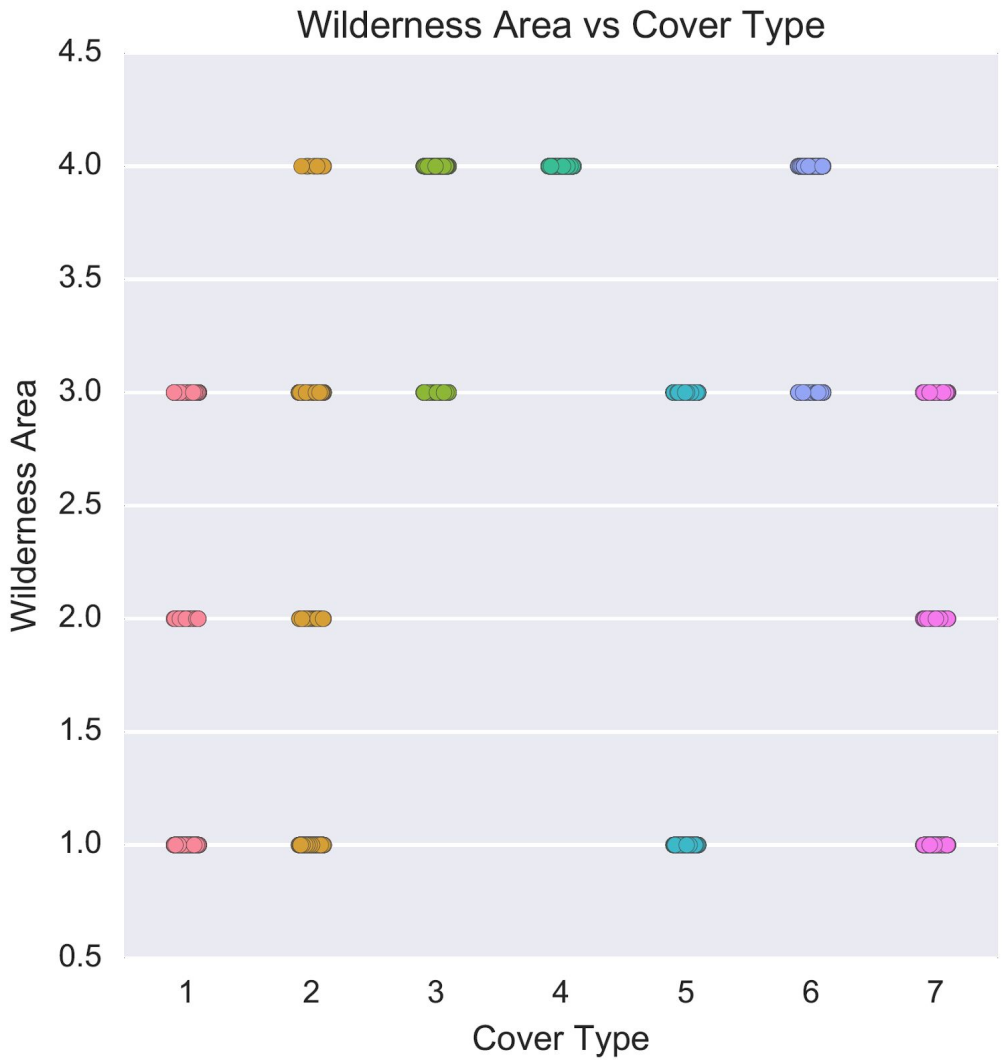

Figure 3 - Wilderness Area vs Cover Type

Algorithms and Techniques

Many different types of classification algorithms will be used to predict the cover type. The main classifications algorithms that will be tried will be Gradient Boosting, Random Forest, Extra Random Forest, and a voting method based on the previous three. After the Gradient Boosting, Random Forest, and Extra Random Forest models are tuned, two different models of each one will be trained with different random seeds to start them. This will result in 6 total trained models (2 of each). These 6 models will then make predictions on the class of each example in the test set and these 6 classifications will be looked at by the voting model to determine the final classification. The voting model will take the most commonly predicted classification out of the 6 models and make that it's classification prediction. The voting method was chosen because individual algorithms might occasionally get single examples wrong but on average they do well. The voting method will try to remove the occasional anomalies. The Python SciKit library will be used to provide these algorithms.

With each of these many different number of estimators will be tried. All of the features will be used with no preprocessing. Each of these algorithms were selected due to their great ability to classify. They are quick to train and predict. Also, these models normally do not overfit the data. Overfitting is a concern with this dataset due to the wide overlapping of possible tree cover types that could exist in similar conditions.

Benchmark

This dataset has been studied before and papers have been published on how classification models. In the 1999 paper by Jock Blackard and Denis Dean on comparative accuracies of artificial neural networks, they achieved an accuracy of 71.1%. [2] This will be the benchmark.

Methodology

Preprocessing

None of the previously mentioned classification algorithms require any preprocessing. However, if some preprocessing could be done then this could speed up the algorithms during training and during classification. There are 54 features and using PCA to reduce the number of critical features did not help. Using the gradient boosted model as the benchmark algorithm, PCA was used to reduce the number of features down to 5 from the 54. This kept 99.92% of the variance of the original dataset. With gradient boosting on this new dataset the accuracy was at 80.6%. Without PCA this same algorithm had an accuracy of 84.8%. PCA was tried with a varying amount of features to reduce to. With PCA reducing the number of features to 20, the gradient boosted model had a 83.6% performance. If the dataset was larger the time saving might make up for the slightly reduced accuracy. Since the goal of this is to create the absolute best performing algorithm, PCA was not used.

Implementation

This code was implemented using Anaconda, a version of Python, and several libraries such as Scikit, seaborn, numpy, and pandas. At a high level, there are a few steps for this code. This is shown below in, figure 4, the Main routine. Functions were created to create an easy to understand and highlevel Main program.

After downloading the data from Kaggle, the first step was to read in the data. Kaggle had already split the data into a training and a test set. The next step was to take the training set and split it again into a training set and a cross validation test set. Basic statistics were then shown for the original training and test sets. The original data was then plotted in various ways to try to find important features. These plots were save to a local hard drive for later analysis.

Then an iterative approach was used to find the best model from the newly created training and cross validation test sets. Different models were tried from the Scikit library with varying degrees of success. The top three best performing were a gradient boosted, random forest, and a extra random forest algorithm. The three methods were then completed twice with different random seeds set for each model. This resulted in 6 models and a voting method was used to find the most popular classification. This voting method resulted in the best performing algorithm. After finding the optimum model, that algorithm was retrained on the entire original training set. This newest model was then used to classify the test set. Finally, this was saved to a local hard drive in order to submit the final results to Kaggle.

```
def Main():

    #Get Data
    X_train, X_test, y_train, test_ids = Load_Data()

    #Split Data into CV sets
    X_cvtrain, y_cvtrain, X_cvtest, y_cvtest = Create_CVset(X_train, y_train)

    #Basic Statistics of Data
    Explore_Data(X_train, y_train, X_test)

    #View Data
    Plot_Data(X_train,y_train)

    #Find Best model with CV sets
    fit_predict_model(X_cvtrain,y_cvtrain,X_cvtest,y_cvtest, "gbm")
    fit_predict_model(X_cvtrain,y_cvtrain,X_cvtest,y_cvtest, "rf")
    fit_predict_model(X_cvtrain,y_cvtrain,X_cvtest,y_cvtest, "xrf")
    fit_predict_model(X_cvtrain,y_cvtrain,X_cvtest,y_cvtest, "vote")

    #Train best model and get predictions on training set
    pred = fit_predict_model(X_train,y_train,X_test,None, "vote")

    #Send it! - Create Submission File
    Send_it(test_ids, pred)
```

Figure 4 - Main

Refinement

For each of these models the main parameter used to tune the model was the number of estimators. The number of estimators was iterated through in order find the optimum value. The score vs the number of estimators was plotted and with this the optimum number of estimators was selected. Figure 5, 6, and 7 show the plots for the three models. For this section, each model was trained on the cross validation training set and tested against the cross validation test set.

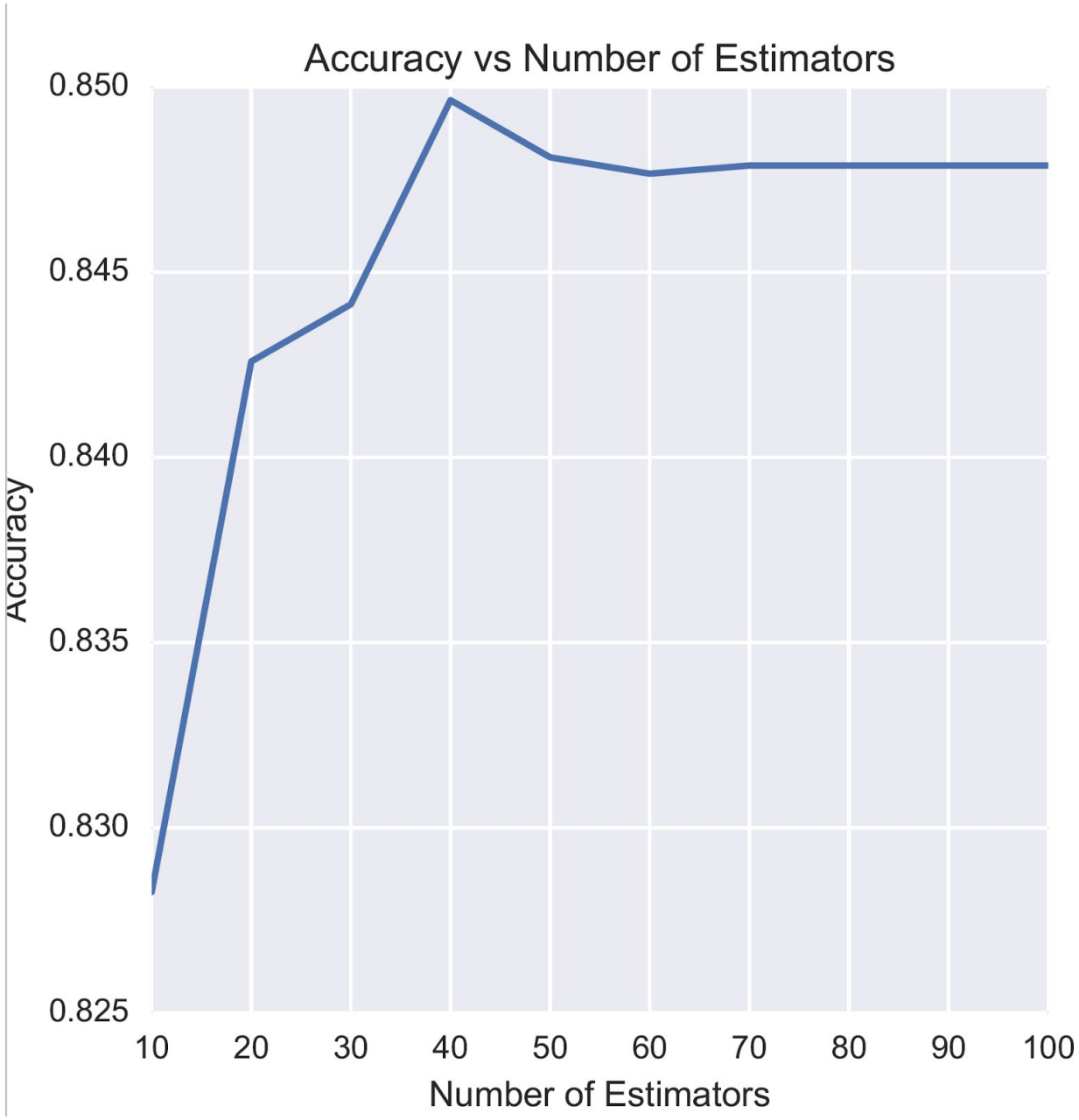

Figure 5 - Accuracy vs Number of Estimators for GBM

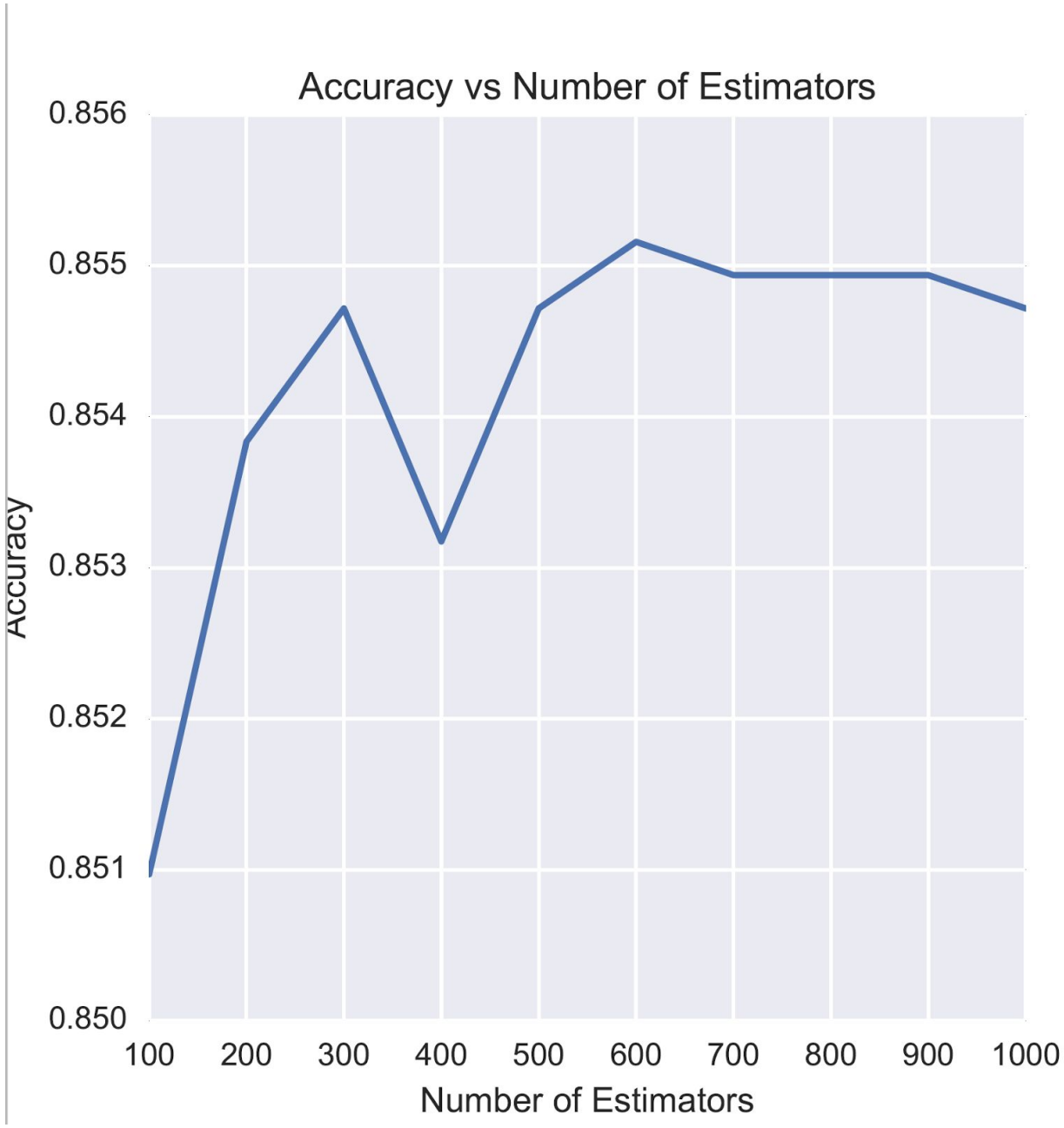

Figure 6 - Accuracy vs Number of Estimators for RF

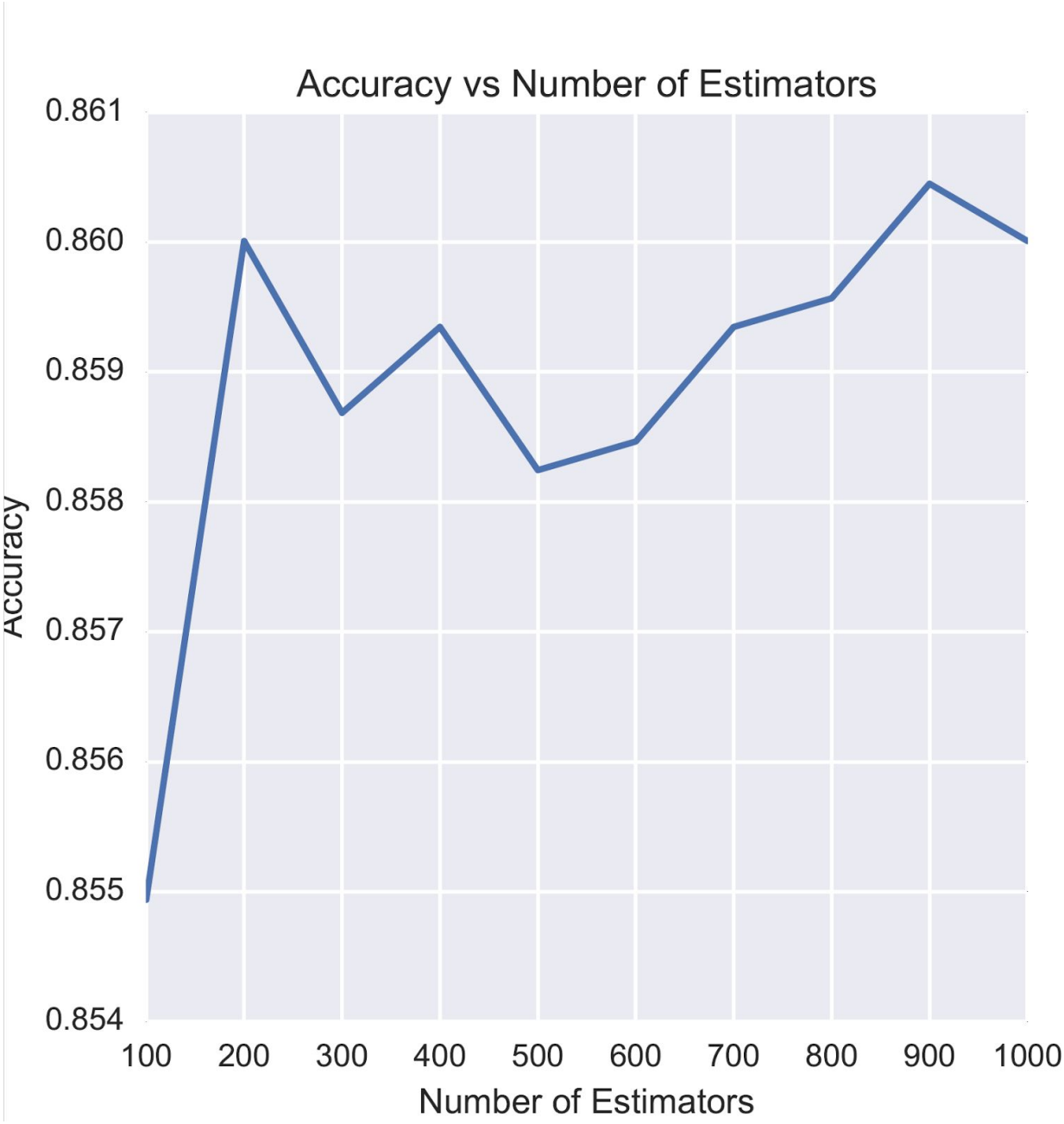

Figure 7 - Accuracy vs Number of Estimators for XRF

For each model, the optimal number of estimators is shown in table 5. The multi-class classification accuracy is also shown based on that number of estimators. The voting method, described in the implementation section above, took the optimal estimators from each model to generate the 6 final models that were feed into the voting method. The accuracy of this final model on cross validation test set is also listed in table 5.

Model	Estimators	Accuracy
Gradient Boosting	40	84.96%
Random Forest	600	85.52%
Extra Random Forest	900	86.04%
Voting	Same as above for each model	86.29%

Table 5 - Model Refinement Results

Results

Model Evaluation and Validation

After optimizing the individual models a voting method was used on the top three models. The top three models were run twice each with different random seeds set. This voting model exceeds the benchmark of 71.1%. The multi-class classification accuracy for this model is 86.3%. This is a significant improvement on the benchmark. The 86.3% was the result of optimizing models on the cross validation data. The model generalizes well and is robust. Since the training set is has a smaller number of examples compared to the test set, there might be some things that the model did not learn well enough. The training set has 15120 examples and the test set has 565892. It would be more ideal to split the data into 60% to 75% for training and test on the remainder.

Justification

The final model was trained on all of the training data set. This was then used to classify the test set provided by Kaggle. When uploaded to Kaggle, this resulted in a score of 76.5%, shown in figure 8.

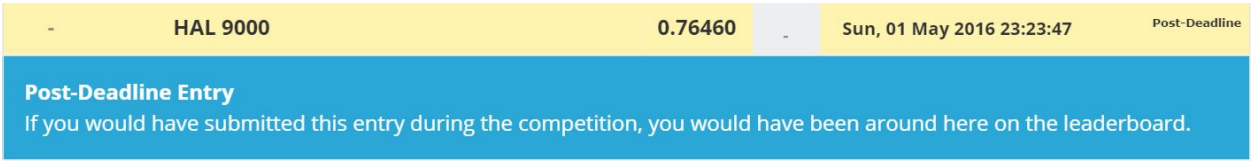

Figure 8 - Kaggle Score

This is a significant improvement on the benchmark of 71.1%. The difference between the 76.5% received on the final test set and the 86.3% on the cross validation test set is most likely due to some overfitting. This could be resolved by giving the algorithm more data to train on.

Conclusion

Free-Form Visualization

One problem with Kaggle style competitions is that after the competition is done the test set is never revealed. This makes it difficult to truly understand where the model was struggling to find the right classifications. As mentioned in the metrics section, one downside of the multi-class classification accuracy is that it does not show how the model is doing on each tree cover type. The model might be doing well predicting some classes but struggling on others.

One way to view the accuracy in more detail is with a confusion matrix. A confusion matrix compares what is predicted by the model and what the actual classification was. With this matrix more detail is shown on how the model is actually performing. The confusion matrix for the cross validation test set is shown in table 6. Interesting and useful information can also be inferred from the confusion matrix. For example, with this model it appears that it is struggling with differentiating between the type 1 and 2 classifications. It predicts type 1 when it is actually type 2 quite often. The same is true for type 2, it predicts type 1 many times. One way to improve the model would be to research and focus on trying to find a feature or create a new feature that will help differentiate these two classifications.

	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Pred 6	Pred 7
Actual 1	509	95	1	0	7	1	33
Actual 2	113	451	21	0	43	10	6
Actual 3	0	1	557	32	9	60	0
Actual 4	0	0	13	621	0	8	0
Actual 5	1	20	11	0	584	6	0
Actual 6	2	5	76	18	9	561	0
Actual 7	18	3	0	0	0	0	631

Table 6 - Confusion Matrix for Cross Validation Test Set

Reflection

This problem was to try to take general features of a plot of land a make predictions about the type of tree cover that was on that land. This project's dataset was provide by Colorado State University through a Kaggle competition. The dataset included a training set with 15120 examples and test set with 565892 examples. The goal was to create a model that would classify each example of the test set properly in order to achieve the highest multi-class classification accuracy possible.

Several machine learning algorithms were tried. Gradient Boosting, Random Forest, Extra Random Forest, and a voting method based on the previous three models were used. The final model and best scoring was the voting method. This method took the predictions of the first three models and found the most common classification and used that as its output. Each model, except the voting method, was optimized using the number of estimators. After optimizing, the voting method model used those optimized models. In order to optimize the models, the initial training set was split into a cross validation training and test set.

After optimization, the final voting method model was trained using the entire training set. The final model exceeds the expectations set by the benchmark. The benchmark was 71.1% and this model was able to score 76.5% It can be used in a general setting to solve this problem and could be implemented into an app that could help managers of these wildlife areas make critical decisions.

Improvement

The biggest improvement that could be made is to redistribute the data sets. As mentioned earlier the training set should be between 60% to 75% of the total data. Currently the training set is 2% of the total data. More data will generally result in better models. Even a subpar algorithm that had 75% of the data to train on might outperform an ideal algorithm that only trained on 2% of the data.

If the results published in this paper are the new benchmark, there could be a better score with more feature engineering. For this project there was no feature engineering. One example of feature engineering that could be looked into would be to create a feature based on level of human interaction. It could be some combination all the features that involve human activity, such as distance to roadway and distance to firepoint. Another example could be to look at the wildlife area and determine if that area's climate is more arid or temperate and combine that with how close the plot of land is to a water source.

References

Kaggle Competition:
https://www.kaggle.com/c/forest-cover-type-prediction

Original Research Paper:
http://www.fs.fed.us/rm/ogden/research/publications/downloads/journals/1999_compag_blackard.pdf

All code for this project can be found at:
https://github.com/J-Sieber/Forest-Cover-Type

Forest Cover App:
https://sieber.shinyapps.io/CoverType/