EE669 Project

Forest Cover Type Prediction Classification Problem

8th December , 2015

Irene Mary Davis
USC ID : 9239327800
irenedav@usc.edu
Individual Project

Project Homepage :

I created my Project Repository in Github .
Link : https://github.com/Irenedvs/EE660_Project.git

Abstract:

Decision making process is facilitated by ecosystem management strategies. Information regarding forest land is very useful in developing these strategies. But sometimes it becomes difficult for the concerned authorities to collect forest land related data, specially when the land is outside the immediate jurisdiction. Machine learning techniques come in rescue in such situations. One of the primal approaches used for obtaining this information is through the use of predictive models like decision trees and neural networks. The goal of this project is to predict the forest cover type from strictly cartographic variables (as opposed to remotely sensed data). This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.
In order to be able to accurately predict the forest cover type I performed certain feature engineering techniques to improve the classification rate. With this the rate of classification by the extremely randomized trees classifier improved to around 81% .

Problem Statement and Goals:

The goal of the project is to determine the classifier model which best predicts the type of tree found on a 30X30 meter squares of the Roosevelt National Forest in northern Colorado from strictly cartographic variables.
This is a classical supervised-learning multi-class classification problem. This project aims to conduct a comparative study of the performance of various classifiers on the forest cover dataset.
There are a couple of reasons why this dataset is very interesting to work on.
(i) Large number of samples
(ii) unequal class distribution for the response
(iii) relatively large number of attributes and classes for the response
As the class distribution is skewed we would require to perform processing on the data to obtain better results . Also taking this into consideration the accuracy measure score on the test data is evaluated as a multi-class classification
In my project I have implemented feature engineering techniques , sub sampling and cross-validation to deal with the issues arising due to the use of this dataset.

Literature Review:

This problem has been visited a number of times by individuals . Most of the publications citing this data is from 2000-2001 . The first is the work by Jock A Blackard and Denis J Dean [1] in comparing the performance of neural networks and Discriminant Analysis on the forest cover datatype. The accuracy of classification for neural networks was found to be 70% . In the case of LDA the accuracy was reported as 68.6%[2]. In this project I aim to get a better classification error than these reported methods.

Prior and Related Work :

No prior or related work

Project Formulation and Setup:

In this project I have assumed the modulate be a non-linear thresholding function applied to a linear variable model . The relation between the output class label and the input feature vector for a classifier can represented as :

$$y=f(w_0x_0 + w_1x_1 + w_2x_2+ \ldots + w_dx_d)$$

where
d - is the number of features/attributes
the function f(.) is defined as

$$f(z)=\begin{cases}1 & if\ z\geq 0\\ -1 & if\ z<0\end{cases}$$

The aim of this project is to predict accurate output labels ‘y’ for each test sample. The only unknown in our equation is the weight vector ‘w’ = $[w_0 , w_1 , w_2 , \ldots , w_d]^T$. So we first need tp obtain the weight vector from our training samples . This defines our model of our classifier . We will then examine the performance of the resultant mapping between feature space and output labels characterized by the vector ‘w’ on our testing data set which comprises sample images that were not used in the training stage. Then, we will use the obtained vector ‘w’ to classify the samples in the testing data set.
I have implemented variations of the decision tree classifiers and the Support vector machine for the classification purpose . In the following sections I describe in details the model of each

classifier and the parameters of each . These parameters need to be tuned to get the best performance results.

Decision Tree

Decision trees are are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

The mathematical formulation of a decision tree is as follows :

Given training vectors xi and label vector y , a decision tree recursively partitions the space such that the samples with the same labels are grouped together.

Let the data at node m be represented by Q. For each candidate split theta = (j, t_m) consisting of a feature j and threshold t_m, partition the data into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ subsets

$$Q_{left}(\theta) = \quad (x, y) \mid \quad x_j <= t_m$$
$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta)$$

The impurity at m is computed using an impurity function H(), the choice of which depends on the task being solved .

$$G(Q,\theta)=\frac{n_{left}}{Nm}H(Q_{left}(\theta))+\frac{n_{right}}{N_m}H(Q_{right}(\theta))$$

We select the parameters that minimize this impurity function.

$$\theta* = \operatorname{argmin}_{\theta} G(Q, \theta)$$

One of the parameters we can tune for the decision tree is the maximum depth . In this particular implementation I set it to none , the nodes are expanded until all leaves are pure.

Random Forest

Random forest is an ensemble of decision trees. It contains multitude of decision trees where each tree is trained on different subset of training data with different random subset of features. The final prediction is done by voting of predictions across the forest. The main parameters that affect the performance of the random tree forest classifier are :

Ne - number of estimators

max_depth - maximum depth of the tree

min_samples_split - The minimum number of samples required to split an internal node.

The main advantage of a random forest classifier is that it does not over fit . In implementation I observed that as we increase the number of estimators the classification accuracy increased and saturated at about 1500 trees . When I decreased the maximum depth , the classifier accuracy reduced considerably. The best parameter values were found to be Ne=1500 ,

max_depth = None (then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples) , min_samples_split = 1;

Adaptive Boosting

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights w_1, w_2, ..., w_N to each of the training samples. Initially, those weights are all set to w_i = 1/N, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the re-weighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence .
We first train a decision tree for the best parameters and then apply AdaBoost to the decision tree model . The main parameter to tune is the Ne - the number of estimators .
In the case of this dataset the Ne=300 gave the best results.

Support Vector Machine

The next model I applied was the support vector machine.SVMs are recommended as reliable classification tools, and before reinvention of Neural Networks(CNNs) was state of the art classification techniques.
SVM is more advanced model with simple idea: large margin classification.Our assumption is there exists a hyperplane , $\psi(x) = w^T x + b$ that separates classes:

$$\begin{cases} \psi(x^{(i)}) > 0 & if\ y^{(i)}=1 \\ \psi(x^{(i)}) < 0 & if\ y^{(i)}=-1 \end{cases}$$

And we want this margins ρ(x(i)) between point and hyperplane be as much as possible :

$$\varrho(x^{(i)}) = \frac{y^{(i)}\psi(x^{(i)})}{\|w\|}$$

However, since perfect separation is not always possible, we introduce slack variable:

$$\xi_i = |y^{(i)} - \psi(x^{(i)})|$$

This is a measure how far we are beyond hyperplane, and we want this overall score be as low as possible. Our optimization objective J is:

$$J = c \sum_{i=1}^{N} \xi_i + \|w\|$$

Running this SVM with linear kernel to train on the training set, unfortunately, failed. Then I normalized the features using min-max approach and used the rbf kernel to get the best performance . The parameters we can tune for the SVM are mainly the kernel , C - slack parameter . In this project I evaluated the SVM performance for the linear and rbf kernel . I used cross validation to obtain the best value for the slack parameter . I found the best results for C=15000 and kernel='RBF'.

Extremely Randomized Trees

The latest model for this data set was Extremely randomized trees. The main difference of Extra-trees is that it splits nodes by choosing cut-points fully at random and it uses whole learning sample(instead of using bootstrap) to grow the trees.[3] There are three main parameters that affect performance of Extra trees:

K - number of random splits screened at each node

Nmin - number of minimal splits required for splitting a node

Ne - number of estimators(trees) in model.

Generally smaller the K , the stronger randomization of trees.Similarly, larger values of Nmin lead to smaller trees, higher bias and smaller variance. And finally, larger the number of estimators Ne is the better overall accuracy would be. For this dataset parameters that gave the best results were following: K = 1, nmin = 3, Ne = 500

Methodology:

In this section I intend to present the methodology involved in selecting a model , training it , cross validation and evaluating its performance on test data. In this project we are following a linear classification framework to obtain the results . The model of a linear classifier is defined as :

$$\hat{f}(\mathbf{x}) = \text{sign}\{\mathbf{w}^T\mathbf{x} + w_0\}$$

The detailed methodology followed is presented below

Dataset :

The dataset used for this project has 581012 instances , 15120 training data samples and the rest are provided as test data samples. Each sample is represented in the feature space as 54 attributes vector, in which 44 features are binary valued and the rest are continuously valued. The completes description of the dataset and attributes have been done in the implementation section of this report.

Since this dataset was part of the Kaggle competition the test set does have the true labels . Hence to evaluate the performance of our classifier , I initially divided my training dataset of 15120 samples into a training set and test set . While doing so I have maintained that each class is uniformly represented. This separation was done only for the needs of the project in order to get an estimate of the out of sample error.

Pre-Processing :

The data samples were first factorized . The soil-type attribute has 40 binary columns . These were factorized into a single column with the feature value ranging as an integer between 1-40. The same was done to the wilderness_area feature. Next I performed normalization on all the feature values . Then I analyzed the importance of the relative features and performed some feature engineering techniques to create new features of increased importance and filled in the missing value for the Hillshade_3pm feature .The details of the pre-processing step is entailed in the implementation section of this report.

Hypothesis Set:

As said we implement the project as a linear classification problem defined as :

$$\hat{f}(\mathbf{x}) = \text{sign}\{\mathbf{w}^T\mathbf{x} + w_0\}$$

The weight vector ‘w’ characterizes the mapping between feature space and output labels and are to be determined in the classification problem.Using this model, our hypothesis set H will have the following form:

$$\mathcal{H} = \{\ h_{\mathbf{w},w_0}(\mathbf{x}) = \hat{f}(\mathbf{x}) \mid \mathbf{w} \in \mathbb{R}^D\ ,\ w_0 \in \mathbb{R}\ \}$$

We use different classifiers to obtain the weight vectors for different cases.

Learning Procedure:

Initially the training set is divided disjoint and cross-validation sets as illustrated in figure below

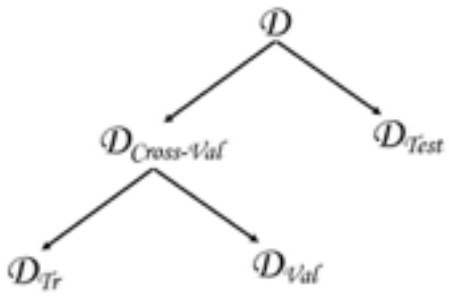


Figure 1: Dividing data set into training, validation and testing sets

The learning procedure that I used comprises of one stage of cross-validation. The cross validation algorithm details are presented below . The cross validation was done in order to get the best parameter values for the different classifier parameters.

Cross-Validation Procedure

For i = 1:K

$D_{Cross-Val,i} = \mathcal{D}_{Tr,i} \cup \mathcal{D}_{Val,i}$

For j = 1:M

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} f_{\text{obj}}(\mathbf{w}, \lambda_j)$$

$$E_{\text{Val}}^{i,j} = \frac{1}{N_{Val}} \sum_{n=1}^{N_{Val}} (\hat{f}(\mathbf{x}_n^{\mathcal{D}_{Val,i}}) - y_n^{Val,i})^2$$

END

END

As outlined above in each outer iteration (over i) of the learning algorithm we randomly divide the cross-validation data set into disjoint training and validation sets. Then in the inner loop (over j) we train the model with one of the values of the parameter say slack parameter C in the case of SVM . Afterwards in the validation stage of the inner loop, we compute the classification error over the validation set using the trained model. As can be see, we have used the residual sum of squares definition for the classification error. It can be seen

that after each outer iteration (over i) a vector of computed errors corresponding to each of the C's s is resulted and these vectors are concatenated in the form of a matrix whose rows contain the error vectors of the inner cross-validation loop . We now need to average over the rows of the resultant matrix of validation errors (Ein) to obtain the following error vector:

$$\bar{\mathbf{e}}_{\text{Val}} = \frac{1}{K}\sum_{l=1}^{K} E_{\text{Val}}^{l,1:M} = [e_{\text{Val}}^{1}\, e_{\text{Val}}^{2} \cdots e_{\text{Val}}^{M}]$$

After completing the cross-validation procedure, we pick the regularization factor (λ) corresponding to the minimum validation error in the averaged error vector (ēVal) as the optimal regularization parameter (Copt).
Then we use this optimal value of C to train the model over the entire training dataset.
Once we have done the training we compute several measures to evaluate our classifier performance.

(i) In - Sample Error

The in-sample error is calculated on the training dataset and is given by

$$E_{\text{in}} \approx E_{\text{Training}} = \frac{1}{N_{\text{Tr}}}\sum_{n=1}^{N_{\text{Tr}}}(\hat{f}(\mathbf{x}_n^{\mathcal{D}_{\text{Tr}}}) - y_n^{\mathcal{D}_{\text{Tr}}})^2$$

(ii) F-Score

In statistical analysis of binary classification, F-score is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results, and r is the number of correct positive results divided by the number of positive results that should have been returned. The F - score can be interpreted as a weighted average of the precision and recall, where an F-score reaches its best value at 1 and worst at 0.
In the case of a skewed multi-class classification problem the F-score is considered as a good measure to predict the models accuracy. In our case the F-score is calculated on the training data . F-score is defined as

$$F\text{-}score = 2.\frac{precision.recall}{precision+recall}$$

(iii) Out-of Sample Error

The out-of-sample error is calculated on the test data . Initially I had divided the 15120 training samples into training and test data . The out of sample error is calculated on this test data .After learning the model using the procedure that was described, we need to examine the performance of the trained model over the testing data set that was not learned by the algorithm in cross-validation stage, and obtain approximation of the out-of-sample error which can be a

good indication of the reliability and generalizability of the trained model. In this project the out-of-sample error is formulated as below :

$$E_{\text{out}} \approx E_{\text{Test}} = \frac{1}{N_{\text{Test}}} \sum_{n=1}^{N_{\text{Test}}} (\hat{f}(\mathbf{x}_n^{\mathcal{D}_{\text{Test}}}) - y_n^{\mathcal{D}_{\text{Test}}})^2$$

(iv) Multi-Class classification accuracy

Since this is a Kaggle competition the evaluation of the dataset done by Kaggle is done by evaluating the multi-class accuracy . This is measure is calculated on the test dataset (565892 samples) . In the case of this particular problem where we have skewed class distribution the multi-class accuracy is score is a good measure of the accuracy of the classifier performance . It compares the correct classifications per the total samples in each class and then performs an weighted average of those values to give us the complete correct classification value.

Implementation :

This section contains the implementation details of the project including the feature engineering techniques used to extract features , training , testing and model selection. The entire code was implemented in Python using the Sci-kit Learn package to implement the various classifiers . Matplotlib and python packages were used to plot the various results and other plots required for feature analysis . Pandas was used to load the dataset.

The entire implementation can be said to be mainly comprised of the following steps :

1. Data Preprocessing : In this step, I analyzed and visualized the data to get intrinsic understanding of the dataset. For this, various data cleansing techniques and feature engineering was applied on the dataset. Detailed explanation of these methods is further in this report.
2. Modeling : After processing the raw data, labeled observations were divided into training, test and validation set. Then, various machine learning techniques were applied on the training set to build a suitable model for the data.I used the validation set for fine tuning and parameter setting.
3. Evaluation : After building a model, the forest cover type was predicted for the test data. We submitted the prediction on Kaggle and reported the accuracy of our model.

Feature Space

This is a multivariate dataset consisting of an output label which is indicative of the type of tree that is present in the area of consideration.

The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and

USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.
The entire dataset consists of 581012 instances which are divided into 15120 training set and 565892 comprises the test dataset. Each feature vector can be represented using the 54 attributes which are categorical and integer valued . We have continuous values and binary valued features.
The description of all the features is as given below.

Feature Name	Data Type	Measurement	Description
Elevation	quantitative	metres	Elevation in meters
Aspect	quantitative	azimuth	Aspect in degrees azimuth
Slope	quantitative	degrees	Slope in degrees
Horizontal_Distance_to_Hydrology	quantitative	metres	Horz Dist to nearest surface water features
Vertical_Distance_to_Hydrology	quantitative	metres	Vert Dist to nearest surface water features
Horizontal_Distance_to_Roadways	quantitative	metres	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_noon	quantitative	0 to 255 index	Hillshade index at noon, summer soltice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_Distance_to_Fire_Points	quantitative	metres	Horz Dist to nearest wildfire ignition points
Wilderness_Area(4 binary columns)	qualitative	0 (absence) or 1 (presence)	Wilderness area designation
Soil_type(40 binary columns)	qualitative	0 (absence) or 1 (presence)	Soil Type designation
Cover_Type(7 types)	integer	1 to 7	Forest Cover Type designation

The output label is an integer ranging between 1-7 inclusive representing the type of tree

Cover_Type (integers 1 to 7)
1 - Spruce/Fir
2 - Lodgepole Pine
3 - Ponderosa Pine
4 - Cottonwood/Willow
5 - Aspen
6 - Douglas-fir
7 - Krummholz

Kaggle provides a labeled training data set of 15,120 observations, and an unlabeled test data set of 565,892 observations to be used for the submission. These 15,120 observations are divided into a training and test data set and the training samples are used to train the classifier and the performance of the trained classifier is tested on the test set.

Pre-processing and Feature Extraction

The Pre-processing was mainly done in 3 steps : factoring , normalization and feature engineering . The detailed implementation of each step is presented below.

Factoring

The dataset contains 4 binary columns for wilderness area and 40 binary columns for the soil type. These columns for wilderness area and soil type are disjoint or mutually exclusive among themselves, i.e., for example, only one of 40 columns of soil type will be present and rest will be absent. Factoring was done on the dataset to combine these 4 wilderness columns into one variable and 40 soil variables into another single variable.

The new features wilderness_area and soil_type is indicative of which binary component is active . Hence they are integer valued between (1-4) and (1-40) respectively. This reduces the dimension of the feature space from 524 to 12 which reduces the computational effort.But, this dataset does not give good performance for most of the classification algorithms. Thus we need to further process the data.

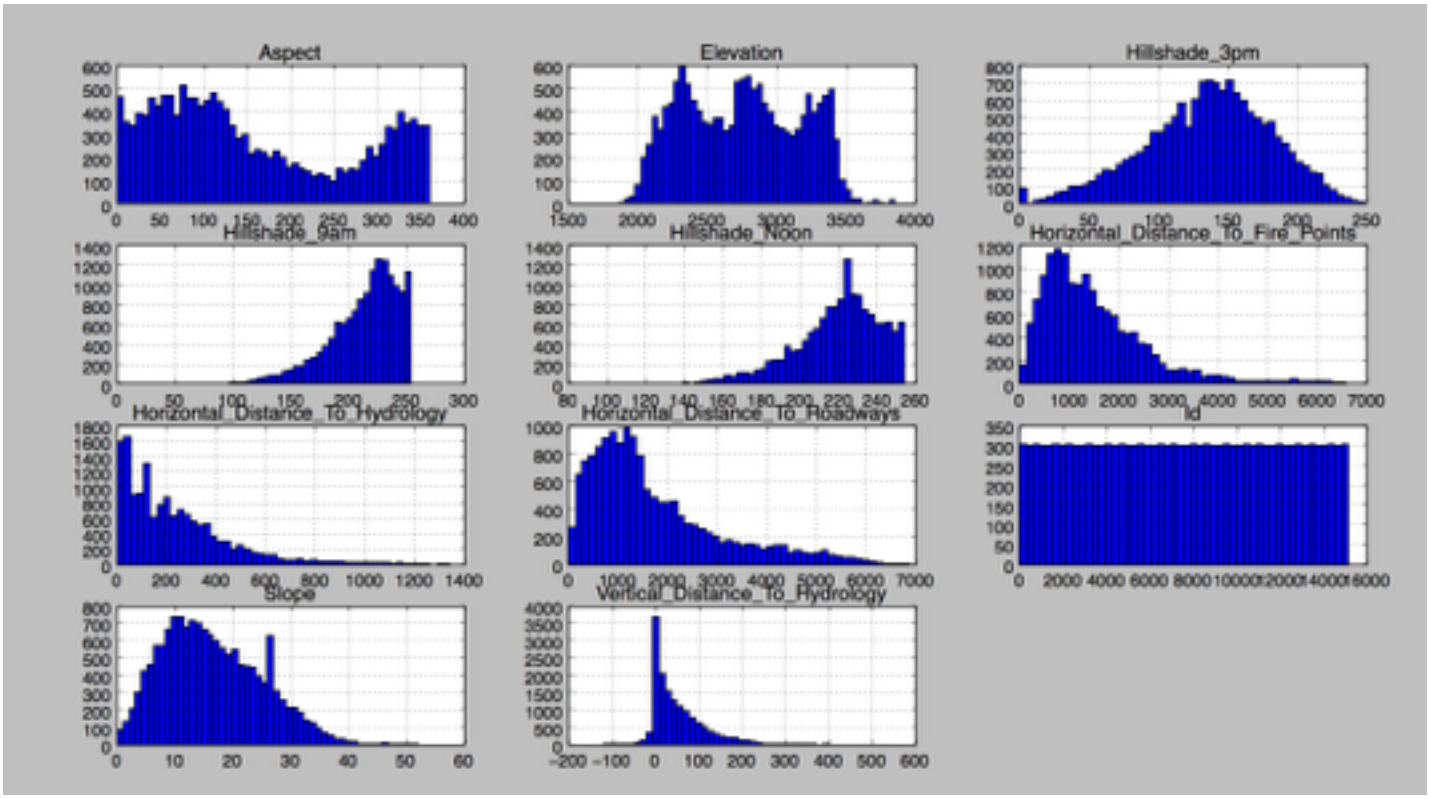
Normalization

The data was normalized and de-correlated in this step. Feature scaling was performed to standardize the range of features of the input data so that each independent feature contributes proportionally to the final classification. For this, data was first transformed to have zero mean and unit variance.

Feature Engineering

Even with the above pre-processing steps the results of classification seemed to be bad mainly due to the fact that the training samples are very less compared to the test data . Also the

skewed nature of the presence of classes needs to be handled for obtaining good classification result.
First I started by analyzing the distribution of each feature in the training dataset . Th plot of the distributions is as shown below :



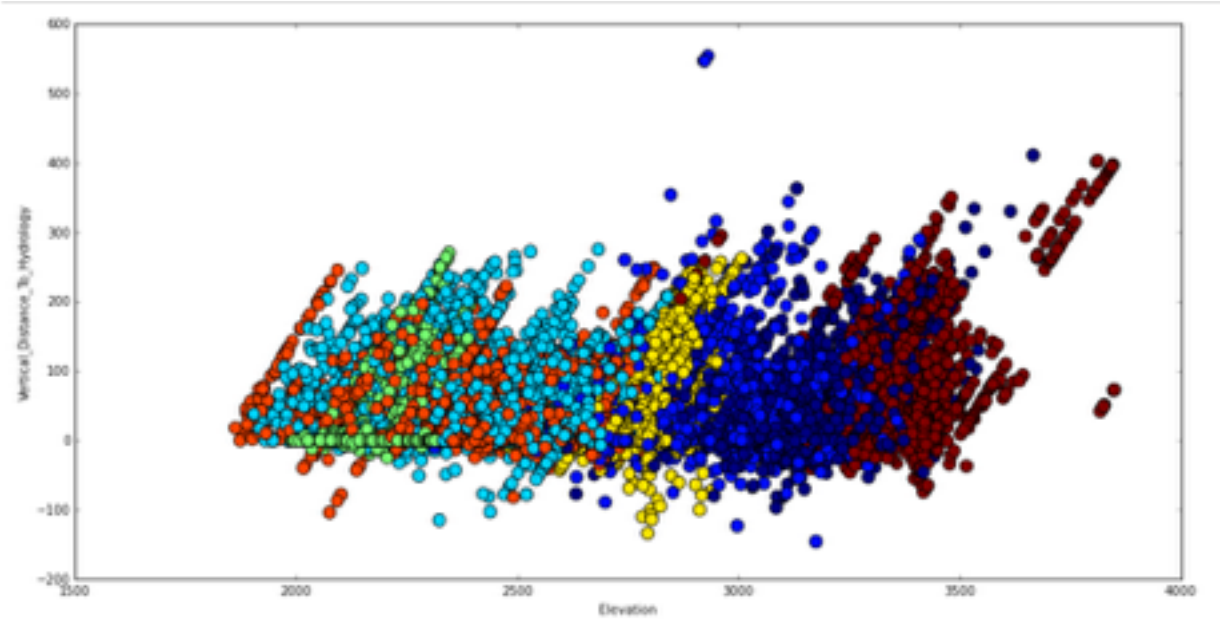
We can observe that the Hillshade_3pm feature has some missing features which are represented by the index value ‘0’ . Now we need to deal with these missing values for this I replace the zero values with the median values.
Now we know that feature ‘aspect’ is in degrees azimuth , so we can try shifting it by 180 . Now we try to determine the most important features for classification using the pd.DataFrame() . The top 5 important features are given as below :

Feature Name	Importance
Elevation	0.22461
Horizontal_Distance_To_Roadways	0.092400
Horizontal_Distance_To_Fire_Points	0.073394
Horizontal_Distance_To_Hydrology	0.062296
Vertical_Distance_To_Hydrology	0.054044

We can see that elevation is one the major features that contributes to classification .It is 2.5 times more important than the second feature in the Top 10. If we can manufacture more

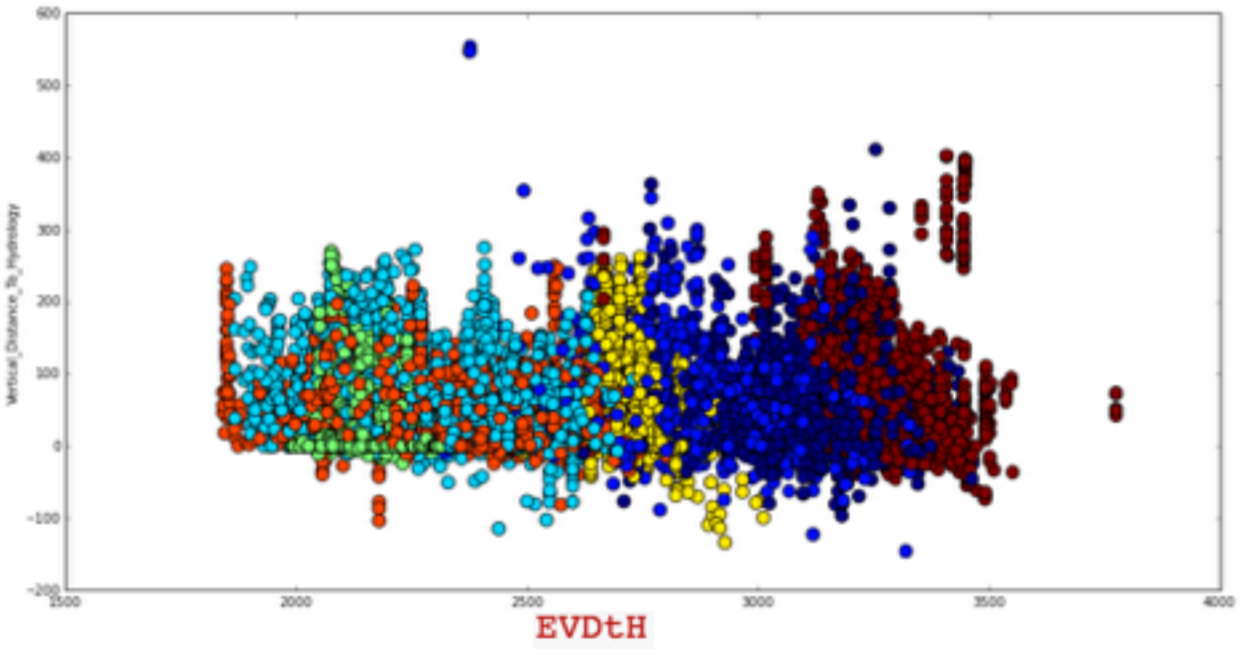
features with Elevation , we should get better classification results. I then analyzed the variation of features with elevation.

The plot of the relation of elevation with the Vertical_Distance_to_Hydrology is shown below :



From this we can conclude that if we created a new feature that reduced the elevation in relation to the Vertical_Distance_to_Hydrology we probably could get more similar labels to align with each other.

I created a new feature 'EVDtH' as the difference of elevation and Vertical_Distance_To_Hydrology . The relation of this new feature to Vertical_Distance_To_Hydrology is as shown below :



Similarly I created new features for the Horizontal_Distance_To_Hydrology and other features . This considerably improves the classifier performance which increased my Leader Board rank from 800 to 126 .

Training Process

Initially we split the training dataset into training and test set . The training set is then further divided into training and validation sets . The validation set size is set to be 0.2*training dataset size . We train the classifier using the training data samples and evaluate the performance for different classifiers on the validation set. We then choose the optimal parameters as the ones which gives us minimum in-sample error.

Overfitting is generally not an issue because the classifiers implemented like the random forest classifier make sure that we do not have

Testing, Validation and Model Selection

The test dataset contained huge set of data compared to the training dataset and since it was a Kaggle competition dataset the labels of the test data were not available . Hence the validation and the estimate of the out-of sample error was calculated on the dataset separated from the training set initially before we decided on the Hypothesis set . This set of samples were not used for pre-processing estimation or training so as to give us a good estimate of the out-of sample error.

The best set of parameters were chosen by cross validation . The model was chosen as the one that generated the lowest in-sample error.

Once the model was trained using our training data we predicted the labels of the test data using the model and wrote the output into a .csv file . I then submitted this file on to Kaggle to obtain an evaluation of classifier . This was given in the terms of a multi-class classification accuracy score. The definition of the accuracy was provided in the previous section. The best model was chosen as the one which gave the best multi-class classification accuracy score.

Final Results:

Initially I implemented the random forest classifier as this one of the simplest and robust classifier . I first trained the model with the default parameters available and then determined the in-sample error and f-score on the training dataset and obtained the results as below :

default Number of trees = 10

Precision	Recall	F-score	Support
0.80	0.79	0.79	412

Classification Accuracy on Training data is 0.8393

This was the error on the training data . I submitted the results on Kaggle and got test accuracy of 63% .

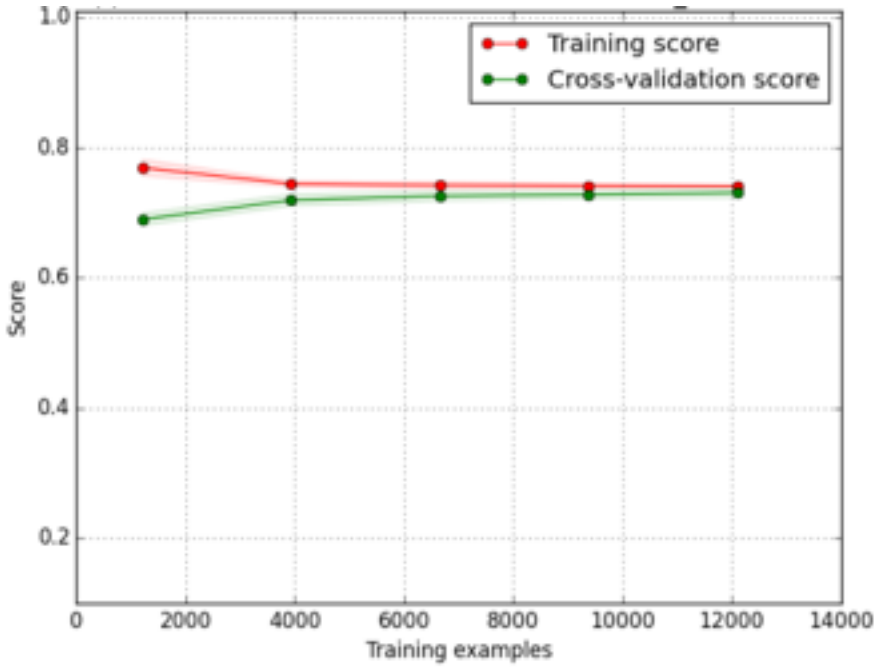
Next I implemented cross-validation to estimate the best number of trees parameter to obtain the classification result and obtained the following results on the training dataset.

Precision	Recall	F-score	Support
0.79	0.76	0.77	430
0.76	0.69	0.72	406
0.85	0.80	0.82	447
0.91	0.97	0.94	437
0.88	0.96	0.92	429
0.84	0.84	0.84	439
0.95	0.97	0.96	436

Classification Accuracy on Training data is 0.857474
Classification Accuracy on Test data is 0.705688
In-sample error : 0.1426
Out-of-sample error estimate : 0.29431

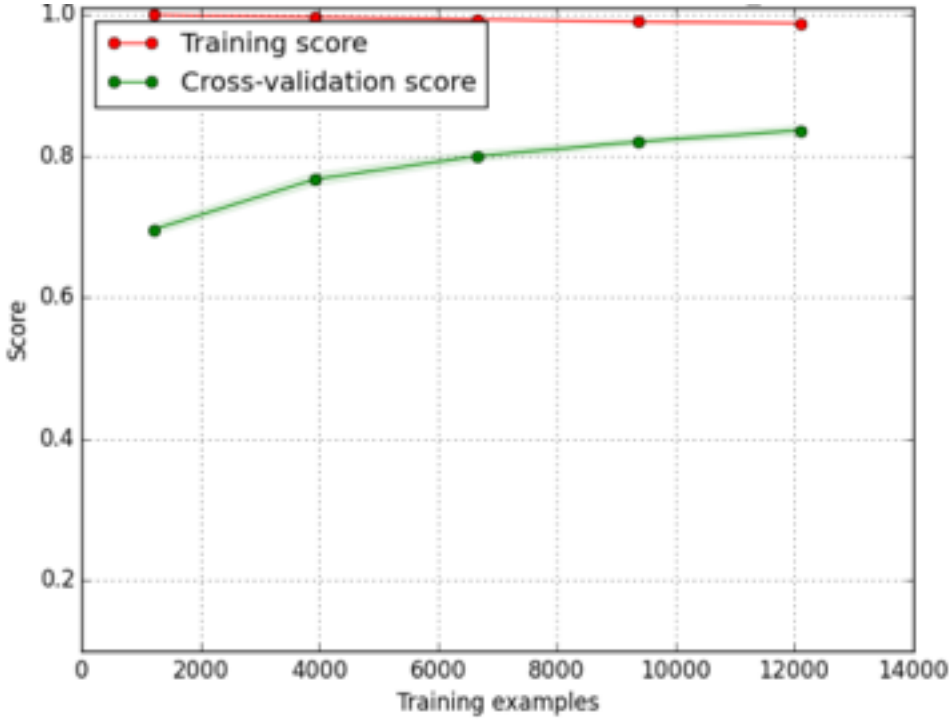
This test data was obtained by partitioning the training dataset initially . This provides an estimate of the out-of-sample error.
I submitted the results on Kaggle and got test accuracy of 75.194% with an optimal number of trees being 1500 and was 800 th on the Leader Board .
Next I performed the feature engineering to obtain better classification results for all the classifiers.
I implemented various variations of decision trees because the random forest classifier seemed to give good results . I also implemented the SVM classifier . Running the linear classifier gave quite bad results with classification rate of only 53% .To see if I can improve the results of the SVM I implemented the RBF kernel SVM which transforms the feature space into an infinite dimension space.There was an improvement in the multi-class classification rate on the test data compared to the linear SVM ,60.217% but was not as good as the tree-classifiers . This put me on a low rank on the Leader Board.
A plot representing the training and cross-validation parameters are presented below :

SVM C=1, Kernel = ‘Linear’



I performed cross validation to obtain the best parameters for the SVM and obtained the training and cross-validation score presented below as the best one :

SVM C=15000, Kernel = ‘RBF’



The random forest and other decision - tree variations seemed to perform better on the given dataset .
The classification accuracy results for the different type of classifiers is presented in the table below :

Classifier	Best Accuracy
Decision Tree	60.718%
Random Forest	75.194%
AdaBoost	60.663%
SVM	60.217%
Extra Trees	80.936%

These are the results after submitting the .csv file containing the predicted test data labels on Kaggle . The extra trees classifier has the best correct classification rate of 81% This put me on 126 th position on the Leader Board.
I also analyzed the important features that contribute to the classification
The table gives us the important features in the classification in decreasing order . This can be easily obtained by using the DataFrame function from pandas.

Feature Name	Importance
Elevation_Horizontal_Distance_to_Hydrology	0.0975
Elevation	0.0967
Elevation_Vertical_Distance_to_Hydrology	0.09256
Wilderness_Area	0.0462
Horizontal_Distance_To_Roadways	0.0313

We can see that the newly added features are also present as important features to classify the data. Hence the features we created gave us better classification results .

Summary and Conclusions :

In this project I implement a series of classifier models to accurately predict the labels of the test data. It was seen that the ensemble and randomized methods worked the best on the given data, Extra Trees Classifier has given the best accuracy of 81% on the feature engineered data. This gave a rank of 126 out of 1694 teams .
The SVM did not provide good results as expected this may be due to the fact that some minority classes are surrounded and overlapped by some majority classes with respect to

features. Ensembling based technique like random forest better prune such type of data as compared to SVM.

The main challenge in this work is to solve the discrepancy of the different in size of the training and test data (15K vs 565K) . This can be handled by practicing good feature engineering techniques.

Feature Engineering and pre-processing turned out to be helpful in most algorithms and the best accuracy was obtained on the feature engineered data. We can observe that the new features created are important to the classification and improve the classification accuracy on the test data considerably.

One would expect that the AdaBoost technique to produce better result than the random trees but it is not so. We have however achieved better accuracy than most of the work presented in literature.

One interesting future prospective work is to use semi-supervised learning to perform the classification as we have huge amount of unlabeled test data to work with. We can also use two-way classification approaches to distinguish between majority class groups with minority.

Reference:

[1]Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Computers and electronics in agriculture, 24(3):131–151, 1999.

[2] Joao Gama, Ricardo Rocha, Pedro Medas, and xxx cx. Accurate decision trees for mining high-speed data streams. In KDD, page 523, 2003.

[3] Pierre Geurts. Extremely randomized trees. In MACHINE LEARNING, page 2006, 2003.

[4] Paul Geladi and Bruce R Kowalski. Partial least-squares regression: a tutorial. Analytica chimica acta, 185:1–17, 1986.

[5] http://home.iitk.ac.in/~shefalig/g8p5.pdf

[6] http://blog.arttechresearch.com/tech/2014/05/28/forest-cover-type-prediction/

[7] https://shankarmsy.github.io/posts/forest-cover-types.html

[8] http://cseweb.ucsd.edu/~jmcauley/cse255/projects/Yerlan_Idelbayev.pdf

[9] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10]D.J. Newman A. Asuncion. UCI machine learning repository. https: //archive.ics.uci.edu/ml/datasets/Covertype,2007.

[11]Alexander Guschin. Feature engeenering benchmark. “http://nbviewer.ipython.org/github/aguschin/ kaggle/blob/master/forestCoverType_featuresEngineering.ipynb, 2014.