EE380L Project Report

Forest Cover Type Prediction

Yuchao Han, Abolfazl Hashemi, Xingyuan Wang, Xinyu Zhao, Zhiyuan Zou

Abstract

We develop an effective model to predict the tree cover type in Roosevelt National Forest of Northern Colorado. We extract new features from the original features that better describe the nature of factors influencing tree growth. Deep learning method such as linearly combining features of the same scale gives a better performance than former analysis. We find from the prediction result on test set that Cover_Type 1 & 2 consist of the majority of the data whereas in training set all 7 types are evenly distributed. So we use resampling method to change priors in the training set so that the model we get emphasizes more on Cover_Type 1 & 2. As a result of this we get the accuracy as high as 84.472%, which ranks 10th out of 902 teams on Kaggle.

Chapter 1 Introduction

1.1 Problem Description

Forest Cover Type Prediction is a challenge from Kaggle website. We are asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type.

1.2 Description of Data

1.2.1 Data Fields

Elevation - Elevation in meters

Aspect - Aspect in degrees azimuth

Slope - Slope in degrees

Horizontal_Distance_To_Hydrology - Horz Dist to nearest surface water features

Vertical_Distance_To_Hydrology - Vert Dist to nearest surface water features

Horizontal_Distance_To_Roadways - Horz Dist to nearest roadway

Hillshade_9am (0 to 255 index) - Hillshade index at 9am, summer solstice

Hillshade_Noon (0 to 255 index) - Hillshade index at noon, summer solstice

Hillshade_3pm (0 to 255 index) - Hillshade index at 3pm, summer solstice

Horizontal_Distance_To_Fire_Points - Horz Dist to nearest wildfire ignition points

Wilderness_Area (4 binary columns, 0 = absence or 1 = presence) - Wilderness area designation

Soil_Type (40 binary columns, 0 = absence or 1 = presence) - Soil Type designation

Cover_Type (7 types, integers 1 to 7) - Forest Cover Type designation

1.2.2 Data Size

The training data has more than 15,000 rows and test data has more than 565,000 observations.

Chapter 2 Data pre-processing

2.1 Quick Visualization

In order to get a general idea of variable type, mean, range, outliers, etc., we first do a quick visualization of the dataset. Figure 1 is the wilderness area distribution of different cover types. Figure 2 is the soil type distribution of different cover types.

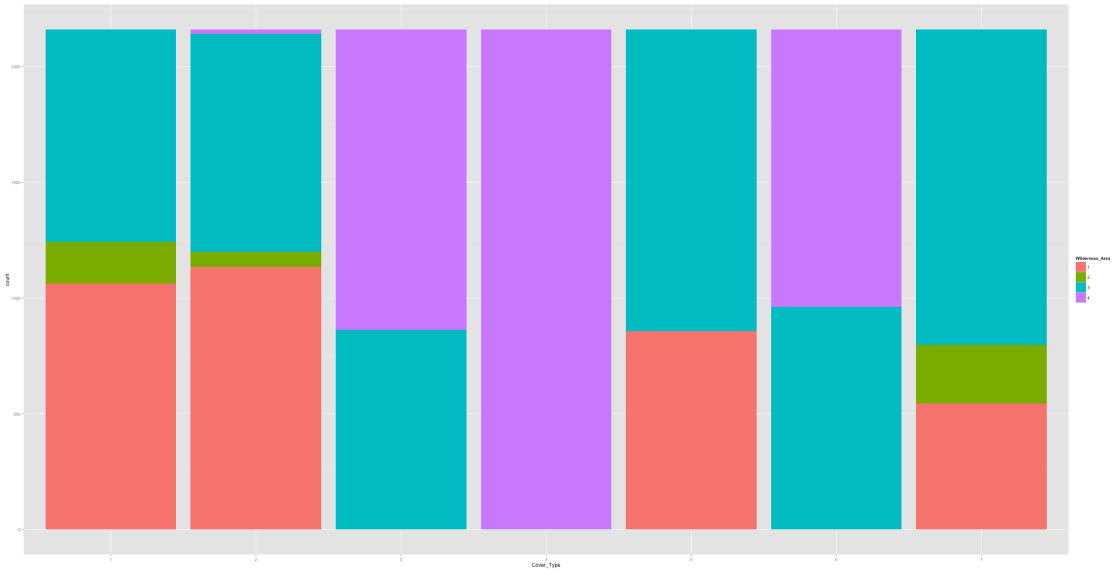

Figure 1 Wilderness Area Distributions of 7 Cover Tyoes

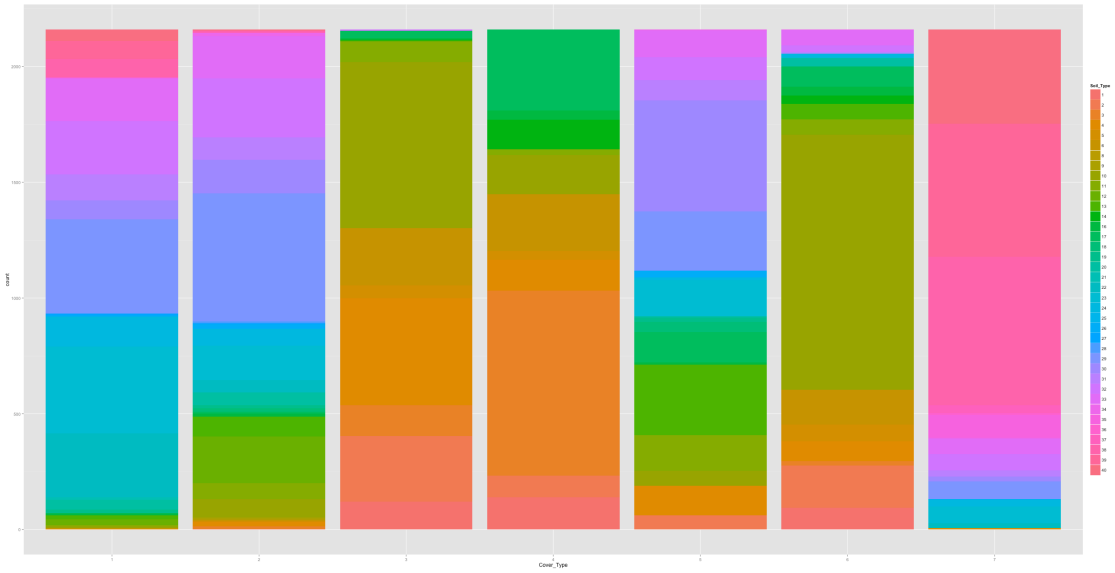

Figure 2 Soil Type Distributions of 7 Cover Types

From Figure 1 we can see that cover type 1,2,7 are similar and cover type 3,6 are similar in terms of wilderness area. Figure 2 shows that type 1,2 are similar in terms of soil type. And another obvious point we can find is that the number of different cover types are the same in training set. We can find in later analysis that it poses a big challenge to us when we use our models to predict test set.

To graphically glance at variables' location and spread, we use boxplot to illustrate this. From all 10 boxplots, we choose 5 representative features: Elevation, Aspect, Slope, Horizontal_Distance_to_Hydrology, Hillshade_Noon. These features are highly related to the

environmental factors that affect plant growth such as temperature, illumination and water. The 5 boxplots are as follows:

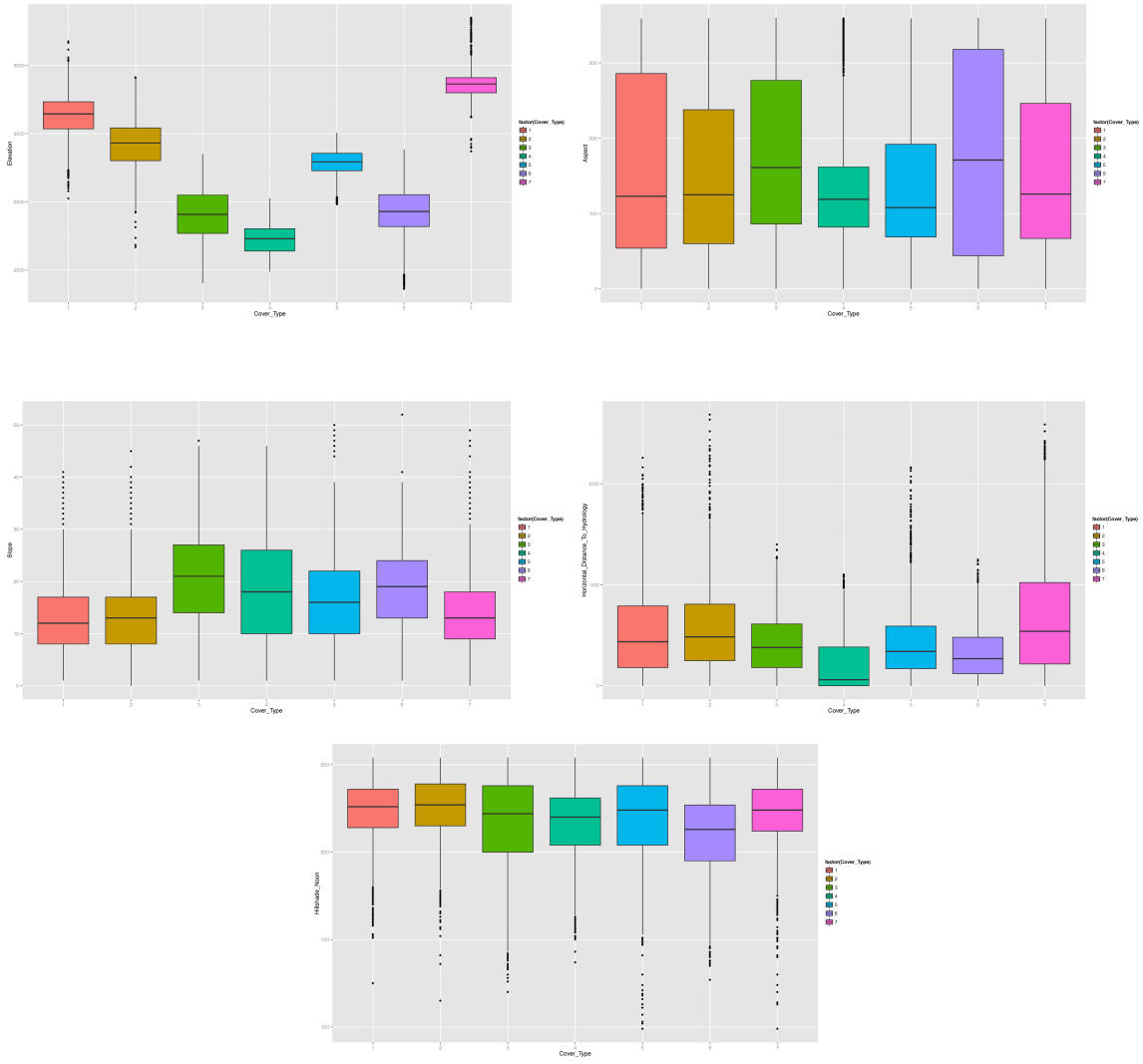

Figure 3 Boxplots of Typical Features

From Figure 3 Boxplots of Typical Features above we can't see very big difference between cover type 1 & 2.

2.2 Feature Extraction

The original features in the dataset are cartographic variables (as opposed to remotely sensed data). We need to extract features that explicitly relate to factors that affect tree growth, namely illumination, temperature, water, etc. Based on geologic knowledge we find on the Internet, we extract the following features:

2.2.1 Temperature

Temperature is affects the productivity and growth of trees (Temperature affects the rate of photosynthesis and respiration.). It is governed by many factors, including incoming solar radiation, humidity and altitude. For simplicity, the relationship between temperature and elevation can be roughly described by Equation (1),

$$T = T_0 - 6.5 \cdot H \quad (1)$$

where T_0 is the temperature at sea level (℃), H is the height above sea level (kilometers) and 6.5 is the lapse rate(℃/km).

2.2.2 Altitude

Altitude of the sun is the most important factor to determine the quantity of solar heat energy on the earth surface. The hillshade function obtains the hypothetical illumination of a surface by determining illumination values for each cell in a raster. The hillshade algorithm is described by Equation (2),

$$Hillshade = 255 * (cos\varphi cos\theta + sin\varphi sin\theta cos(\psi - \omega)) \quad (2)$$

where φ is zenith(rad), θ is slope(rad), ψ is azimuth(rad), ω is aspect(rad). Altitude α (rad) is the complementary angle of zenith, i.e.

$$\alpha = 90^{\circ} - \varphi \quad (3)$$

The relationship among zenith, altitude and azimuth are showed in Figure 4.

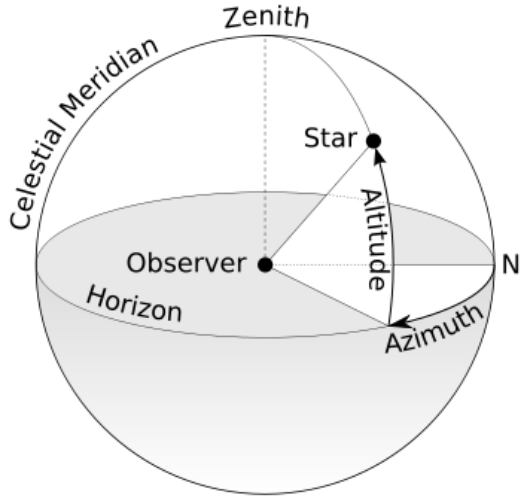

Figure 4 Zenith, Azimuth, Altitude Relationships

2.2.3 Climatic Zone and Geologic Zone

Each soil type corresponds to a unique 4-digit USFS ELU (Ecological Landtype Units) code. The first digit indicates the climatic zone and the second digit indicate the geologic zone. The

third and fourth ELU digits are unique to the mapping unit and have no special meaning to the climatic or geologic zones. We use the ELU code of these 40 soil types to extract two new features of climatic zone and geologic zone. This processing method has two advantages. The first is that it can reduce the 40 soil types to 16 zones (8 climatic zones and 8 geologic zones). The second is that it also resolves the problem that some soil types are missing in the training set.

2.2.4 Hillshade variation

The original feature includes 3 features related with hillshade: Hillshade_9am, Hillshade_Noon, Hillshade_3pm. Considering that the fluctuation of illumination can affect the growth of tree, We compute the variation of hillshade in a day as a new feature.

2.3 Validation Subset

Since number of observations in test dataset is huge, it is not practical to use test dataset to test our models every time. It is time-consuming to upload the test result to Kaggle to see our scores. So it is necessary to find a tiny subset to conveniently validate our models. We randomly sample 1/3 observations of the training set as our tiny test set, so that we can use it to get a rough picture of our models' performance. If the performance is relatively good, we then use the real test set to see our prediction accuracy on Kaggle.

Chapter 3 Single Classifiers

After pre-processing steps, we use different classifiers to tackle this problem, such as adaboost, logistic regression, extra tree and MLP. They are now explained in detail in following sections.

3.1 Adaboost

3.1.1 Brief Introduction

Adaboost algorithm first takes the input of size m, $(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)$, and does the following:

(1) Initialize $D_1(i) = \frac{1}{m}$.

(2) For $t = 1, 2, \cdots, T$ (different iteration)

I. Train a weak learner using distribution D_t with accuracy higher than 50%.

II. Get weak hypothesis h_t with error $\varepsilon_t = Pr_{x\sim D_t}[h_t(x) \neq y]$.

III. Choose $\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$.

IV. Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, where Z_t is the normalization factor.

(3) Finally, the output is $h(x) = sign(\sum_{t=1}^{T}\alpha_t h_t(x))$.

Adaboost essentially trains a lot of trees, each focus more on the samples that are mis-classified by previous trees. Generally, adaboost tends to yield good result because it reduces both variance and bias. In R, adaboost method is included in the gbm package.

3.1.2 Using binary classifier for multi-class classification

Adaboost is implemented in the gbm package in R. However, the gbm version of adaboost is only for two-class classification. In our project, the goal is to do a seven-class classification, which means in order to use the adaboost method we need to find a way to do multi-class classification using binary classifier.

3.1.2.1 One-Against-All Scheme

One of the obvious solutions is that we fit 7 binary classifiers. In each classifier, the data are group into two classes, namely class x (the target class) and class non-x, and returns the probability of being class x. After fitting 7 binary classifiers, the return result would be 7 probabilities, and we decide which class a sample should be according to the highest probability.

3.1.2.2 Round Robbin Scheme

A more complicated way is the so-called round robbin scheme, which fits pair wise classifiers. For this 7-class problem, 21 classifiers are fitted. To determine the final result, a majority vote scheme can be used. That is, when a new sample is to be predicted, each of the 21 classifiers determines which one of the two classes involved should the new sample be, and that class gets a point. The final winner will be the class that gets the most points. If there is a tie,

choose the class with higher frequency in the training set (or just flip a coin).

3.1.2.3 Error Correction Code Scheme

Another way to do multi-class classification using binary classifier is the Error Correction Code Scheme. For an m-class problem, an m*n matrix called the codebook is generated (usually randomly). Each column defines a binary classification problem and thus ends up with n binary classifier. To predict a result, combine the result of the n binary classifier into an n-byte binary code. Based on the Hamming distance between the resulting binary code and the binary codes of the m classes, the final result is chosen for the nearest class.

3.1.3 Test Results

We are trying to run an Adaboost using gbm. But because each run requires a very long time, it would be extremely difficult to do the Round Robin Scheme. In addition, we expect the classifier to return the probability of each class, which could be useful in hierarchical classification. Therefore, we use the One-Against-All Scheme. The result is showed in Table 1.

Table 1 Test Results of One-Against-All Scheme

n.trees	shrinkage	Overall accuracy	Cover_Type1 Accuracy	Cover_Type2 Accuracy
5000	0.01	79.8%	66%	59.6%
5000	0.02	82.1%	69%	61%

Overall accuracy means the 7-class accuracy on the validation set. Cover_Type1 Accuracy is the accuracy to predict Cover_Type1. Cover_Type2 is the accuracy to predict Cover_Type2. It's obvious that the accuracy for Cover_Type 1&2 is very low. Therefore we give up using Adaboost for 7-class classification.

3.2 Gradient boosted Logistic Regression

3.2.1 Introduction to Friedman's gradient boosting machine

Friedman (2001) and the companion paper Friedman (2002) extended the work of Friedman, Hastie, and Tibshirani (2000) and laid the groundwork for a new generation of boosting algorithms. In any function estimation problem the goal is find a regression function $\hat{f}(x)$ that minimizes the expectation of some loss function.

$$\hat{f}(x) = \underset{f(x)}{\arg\min}\, E_{y,x}\psi(y, f(x))$$

Friedman's gradient boost algorithm does the following:

(1) Initialize $\hat{f}(x)$ to be a constant,

$$\widehat{f}(x) = \arg\min_{\rho} \sum_{i=1}^{N} \psi(y_i, \rho)$$

(2) For $t = 1, 2, \cdots, T$ (different iteration),

I. Compute the negative gradient as the working response.

$$Z_i = \frac{\partial}{\partial f(x_i)} \psi\big(y_i, \mathrm{f}(x_i)\big)|_{f(x_i)=\hat{f}(x_i)}$$

II. Fit a regression model $g(x)$, predicting z_i from the covariates x_i.

III. Choose a gradient descent step size as

$$\rho = arg\min_{\rho} \sum_{i=1}^{N} \psi(y_i, \hat{f}(x_i) + \rho g(x_i))$$

IV. Update the estimate of $f(x)$ as

$$\hat{f}(x) = \hat{f}(x) + \rho g(x)$$

Basically, the gradient boost algorithm is doing gradient descent in the function space to minimize the loss function.

3.2.2 GBM package in R

The gbm package in R actually uses regression tree model to do gradient search in the function space. One tree is one iteration. It has the following parameter to choose:

- A loss function(distribution)
- The number of iterations, T(n.trees)
- The depth of each tree, K(interaction.depth)
- The shrinkage (or learning rate) parameter, λ
- The subsampling rate, p(bag.fraction)

To be more specific, the gbm package does the following:

(1) Initialize $\hat{f}(x)$ to be a constant,

$$\widehat{f}(x) = \arg\min_{\rho} \sum_{i=1}^{N} \psi(y_i, \rho)$$

(2) For $t = 1, 2, \cdots, T$ (different iteration),

I. Compute the negative gradient as the working response.

$$Z_i = \frac{\partial}{\partial f(x_i)} \psi\big(y_i, \mathrm{f}(x_i)\big)|_{f(x_i)=\hat{f}(x_i)}$$

II. Randomly select p*N cases from the dataset.

III. Fit a regression tree with K terminal nodes, $g(x) = E(z|x)$. This tree is fitted using only those randomly selected observations.

IV. Compute the optimal terminal node predictions, $p_1, \cdots, p_k$, as

$$p_k = arg \min_{p} \sum_{x_i \in S_k} \psi(y_i, \hat{f}(x_i) + \rho)$$

V. Update the estimate of $f(x)$ as

$$\hat{f}(x) = \hat{f}(x) + \lambda p_k(x)$$

3.2.3 Gradient boost logistic regression

In the gbm package, when selecting "bernoulli" as the distribution, the function to be estimated is $f(x) = log \frac{p}{1-p}$, which is actually doing logistic regression. Similar to Adaboost, the logistic regression in gbm package is also for binary classification. Therefore, to perform 7-class classification we use the one-against-all scheme. The experiment result is showed in Table 2.

Table 2 Test Result of Gradient Boost Logistic Regression

n.trees	Shrinkage	Overall accuracy	Cover_Type1 Accuracy	Cover_Type2 Accuracy
5000	0.01	80.1%	70.3%	58.2%
5000	0.02	82.3%	71.8%	59.4%

The result shows that using logistic regression is slightly better than Adaboost, but the accuracy for Cover_Type1 and Cover_Type2 is very low.

3.3 Multilayer Perceptron (MLP)

An MLP is a network of simple neurons called perceptron. Rosenblatt introduced the basic concept of a single perceptron in 1958. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then possibly putting the output through some nonlinear activation function.

A single perceptron is not very useful because of its limited mapping ability. No matter what

activation function is used, the perceptron is only able to represent an oriented ridge-like function. The perceptron can, however, be used as building blocks of a larger, much more practical structure. A typical multilayer perceptron (MLP) network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer. The signal-flow of such a network with one hidden layer is shown in Figure 5.

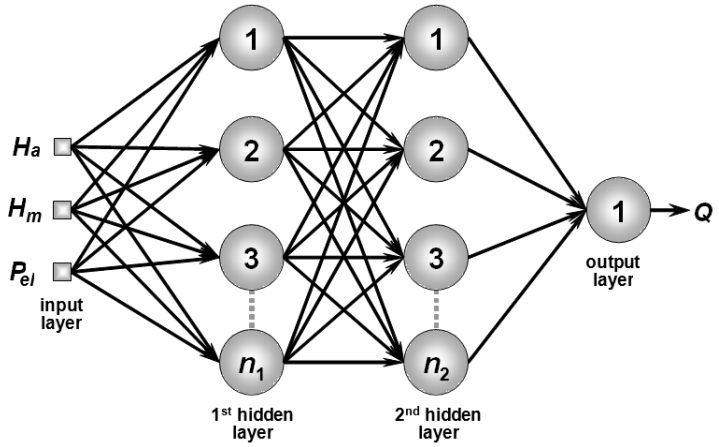

Figure 5 Multilayer Perceptron Network

In our project, we use the package "monmlp" provided by R community. The monmlp package implements the monotone multi-layer perceptron neural network (MON-MLP) model following Zhang and Zhang (1999). The main feature is the monotone constraint, which guarantees monotonically increasing behavior of model outputs with respect to specified covariates.

The key factor to MLP layer is to select suitable hidden nodes, hidden layer transfer function and output transfer function. Monmlp package mainly supports 3 types of transfer function: linear / linear.prime, logistic / logistic.prime, transig / transig.prime. With different parameter combination, using MLP as the single classifier, the performance is compared in Table 3.

Table 3 MLP Performance

	Th= Logistic	Th= Linear	Th= Logistic
hidden nodes	To= Linear	To= Logistic	Th= Logistic
10	0.45	0.49	0.49
50	0.71	0.69	0.65
100	0.76	0.72	0.7

(Note: (1) All models runs for 1000 iterations in maximum. (2) All models use only 1 hidden layer.)

From the results above, we can see that one simple MLP fails to give a satisfying accuracy,

compared to other tree methods and SVM. We tried to apply some general techniques to improve the model. The performance is presented in Table 4.

Ensemble is first introduced to MLP. It has been observed that besides performance difference, MLP with different transfer functions display inductive bias towards different prediction types. This serves as a good property for ensembles method. We used 3 MLP running 3 sets of transfer function, with same hidden nodes and layer and use average as the Meta Learner.

Bagging technique is then applied to increase the average ambiguity among separate single MLPs and hence reduce the variance of ensembles MLP.

We also tried MON-MLP (Monotone MLP). It is introduced by Zhang and Zhang (1999). The most important feature is to continuous increasing behavior of model outputs. Unfortunately it yields a very poor accuracy. We believe over-fitting is the possible reason.

Table 4 Performance of Improved MLP

Techniques	Simple MLP	Ensembles	Bagging	Monotone
Accuracy	0.76	0.764	0.781	0.715
Parameters	50 hidden nodes	Ensembles = 3	--	--

3.4 Random Forests

Bagging trees (or any high variance, low bias technique) improves predictive performance over a single tree by reducing variance of the prediction. Generating bootstrap samples introduces a random component into the tree building process, which induces a distribution of trees, and therefore also a distribution of predicted values for each sample. The trees in bagging, however, are not completely independent of each other since all of the original predictors are considered at every split of every tree. One can imagine that if we start with a sufficiently large number of original samples and a relationship between predictors and response that can be adequately modeled by a tree, and then trees from different bootstrap samples may have similar structures to each other (especially at the top of the trees) due to the underlying relationship. This characteristic is known as tree correlation and prevents bagging from optimally reducing variance of the predicted values. Reducing correlation among predictors can be done by adding randomness to the tree construction process. Each model in the ensemble is then used to generate a prediction for a new sample and these *m* predictions are averaged to give the forest's

prediction. Since the algorithm randomly selects predictors at each split, tree correlation will necessarily be lessened. Random forests' tuning parameter is the number of randomly selected predictors, k, to choose from at each split, and is commonly referred to as m_{try}. A general random forests algorithm for a tree-based model can be implemented as shown in Figure 6.

```
1  Select the number of models to build, m
2  for i = 1 to m do
3      Generate a bootstrap sample of the original data
4      Train a tree model on this sample
5      for each split do
6          Randomly select k (< P) of the original predictors
7          Select the best predictor among the k predictors and
           partition the data
8      end
9      Use typical tree model stopping criteria to determine when a
       tree is complete (but do not prune)
10 end
```

Figure 6 Random Forest Algorithm

3.5 Extremely randomized trees (Extra-Trees)

In this section, we are going to introduce a new tree-based ensemble method for supervised classification and regression problems. It essentially consists of randomizing strongly both attribute and cut-point choice while splitting a tree node. In the extreme case, it builds totally randomized trees whose structures are independent of the output values of the learning sample The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.

The Extra-Trees splitting procedure for numerical attributes is given in Figure 7. It has two parameters: K, the number of attributes randomly selected at each node and n_{min}, the minimum sample size for splitting a node. It is used several times with the (full) original learning sample to generate an ensemble model (we denote by M the number of trees of this ensemble). The

predictions of the trees are aggregated to yield the final prediction, by majority vote in classification problems and arithmetic average in regression problems. From the bias-variance point of view, the rationale behind the Extra-Trees method is that the explicit randomization of the cut-point and attribute combined with ensemble averaging should be able to reduce variance more strongly than the weaker randomization schemes used by other methods. The usage of the full original learning sample rather than bootstrap replicas is motivated in order to minimize bias.

From the computational point of view, the complexity of the tree growing procedure is, assuming balanced trees, on the order of $N \log N$ with respect to learning sample size, like most other tree growing procedures. However, given the simplicity of the node splitting procedure we expect the constant factor to be much smaller than in other ensemble based methods which locally optimize cut-points.

The parameters K,n_{min} and M have different effects: K determines the strength of the attribute selection process,n_{min}, the strength of averaging output noise, and M the strength of the variance reduction of the ensemble model aggregation. These parameters could be adapted to the problem specifics in a manual or an automatic way (e.g. by cross-validation).

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

Figure 7 Extra Tree Splitting Procedure

Chapter 4 Hierarchical Classification

4.1 What is Hierarchical Classification?

Hierarchical classification sequentially fits models with many independent classifiers so that it can take advantage of all good properties collected from simple classifiers.

It first maps input data into predefined categories, or Mega Types as referred in this report. Based on these predefinition, the classification occurs first on the low-level and highly specific pieces of input data. Each individual piece of data is then combined systematically and classified on a higher level iteratively until one output is produced. This final output is the overall classification based on original requirements.

Figure 8 gives an application of hierarchical classification in phytology. Linnaeus started a hierarchical system with large groups broken into smaller and smaller groups. Large groups are subdivided into smaller groups, based on key features and smaller groups are divided into still smaller groups based on other key features. The largest groups would be the domains, which are subdivided into

kingdoms→phyla→classes→orders→families→genera→species.

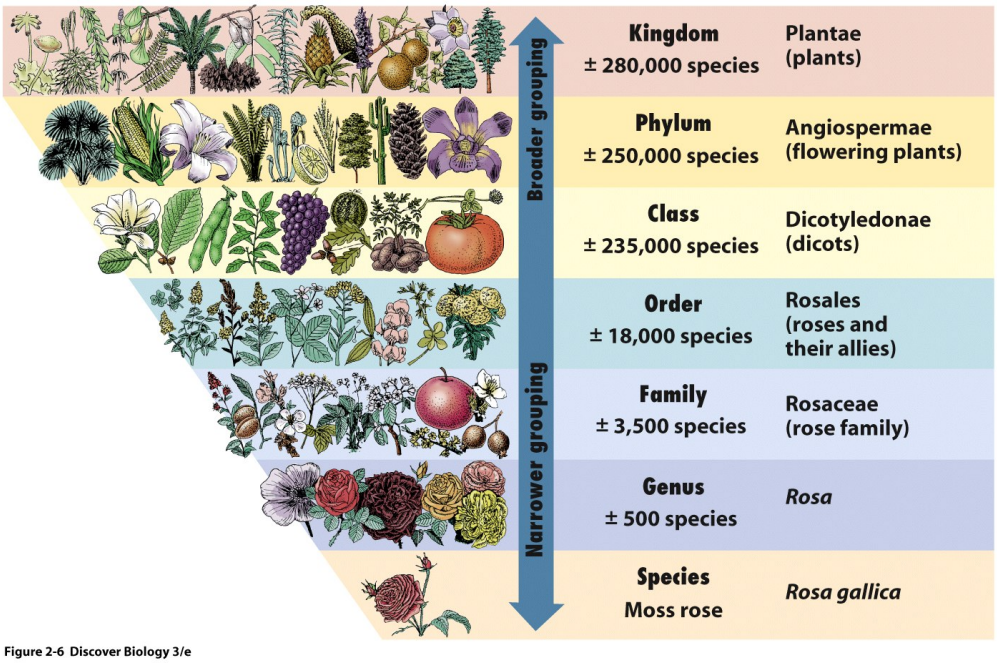

Figure 8 Modern Plant Classifications

4.2 How to define mega type?

In our project, we employ the two-level hierarchy, mega-type and binary classifiers. With the single classifier, one typical confusion matrix coming out of our classifiers is shown in Table 5. It is generated by a single MLP classifier with 500 hidden nodes and 1 hidden layer, running on a validation dataset of 5000 samples.

Table 5 Typical Confusion Matrix

		Reference						
		1	**2**	**3**	**4**	**5**	**6**	**7**
Prediction	**1**	473	165	0	0	10	0	49
	2	158	408	14	0	59	9	10
	3	1	13	532	29	16	120	0
	4	0	0	36	639	0	29	0
	5	22	100	17	0	650	9	1
	6	4	19	117	13	10	543	0
	7	74	11	0	0	0	0	680

Another criterion is the imbalanced distribution of prediction class in test dataset. In Figure 9, the left pie chart is class distribution of training data, while the right one is class distribution of test data. We can observe a uniform distribution in training set, where all types are balanced. In contrast, class 1 and class 2 predominates over all other types in test dataset, which means that means that almost predictions are class 1 and class 2, given a 80% accuracy on test dataset.

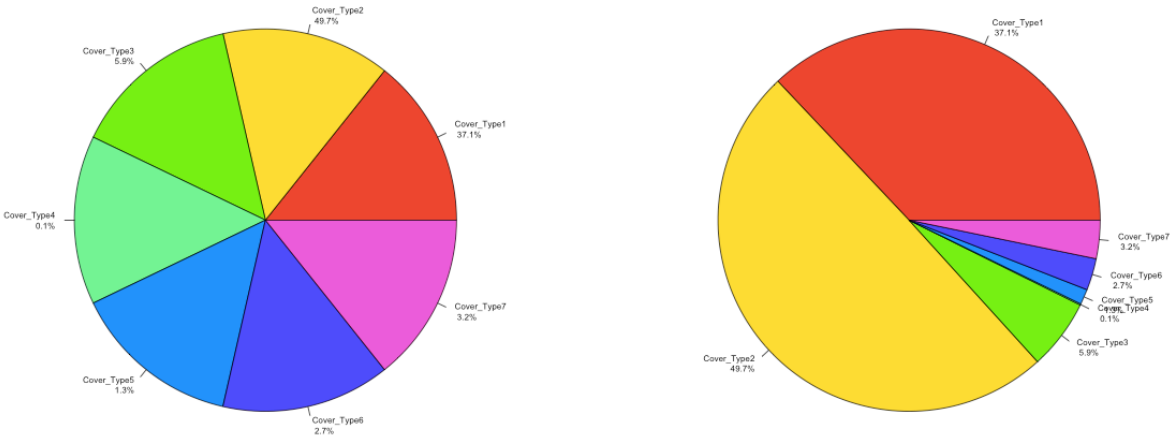

Figure 9 Distribution of Class in Train and Test Set

Our strategy is to combine Cover_Type 1 and Cover_Type 2 into one mega type. Figure 10 shows the mega type data structure in our project. In the first level, we use a 6-class classifier on

the whole training data and obtain 6 mega types. In the second level, we move on to the binary classifier on mega type 1 to finally separate Cover_Type 1 and 2.

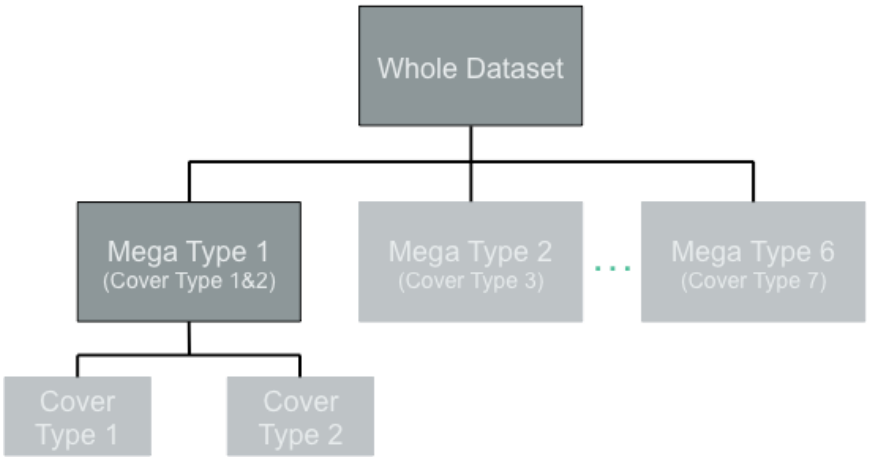

Figure 10 Mega Type Structure

On first level, we have tried four classifiers, inherited from single classifiers in previous sections. They are Random Forest, Extra Tree, Logistic Regression and Adaboost. On second level, we choose the binary classifier to be Extra Tree, Logistic Regression, Adaboost, MLP and SVM. The result is presented in Figure 11. Random Forest has the highest accuracy in 6-class classification. While in binary classification, extra tree has the highest accuracy.

One thing to note is that we add some new features in the second level. Back to single classifier, almost all models can output prediction possibilities of each type, including type 1 and type 2. The possibility might deliver useful information to binary classifier. So we feed possibilities of type 1 and type 2 to binary classifier as new features. Another way to look at this is that we actually take advantage of more than models in the second level.

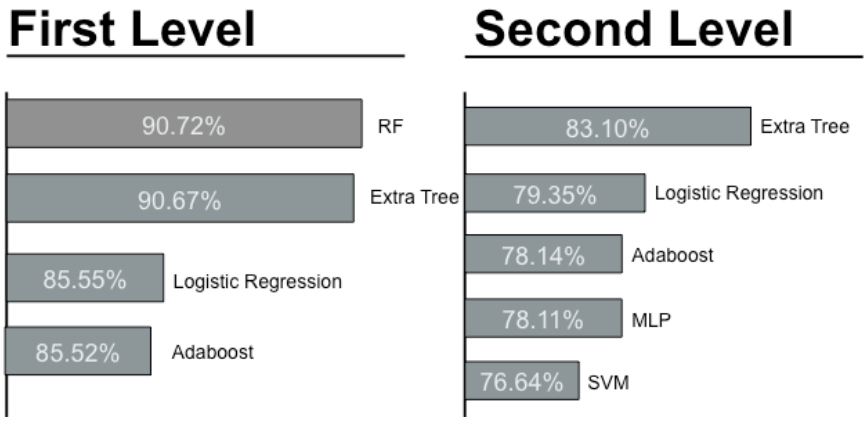

Figure 11 Result of Two-Level Classification

Details of classifiers we use in hierarchy are shown in Table 6 and

Table 7. For the sake of comparison, we set the tree number to be 500 and depth to be 5, for all tree method. As argued in last section, excessive number and depth will cause serious overfitting and decreases the accuracy.

Table 6 Parameters of Mega Type Classifier (6-Class)

Classifier	Adaboost	RandomForest	ExtraTrees	DecisionTrees
Tree Number	500 trees	500 trees	500 trees	--
Depth	5	5	5	--
Accuracy	85.5184	90.7200	90.6694	85.5469

Note: All training data and test data has not been preprocessed (feature engineering)

Table 7 Binary Classifier (on Cover_Type 1 & 2)

Classifier	Accuracy	Parameter
Adaboost	0.78~0.82	n.trees=5000, depth=6, shrinkage=0.01
Gradient Boost	0.79~0.82	n.trees=5000, depth=6, shrinkage=0.01
Extra Tree	0.83	Ntree=600, Mtry=10, Numrandomcuts=4
MLP	0.781	Nodes=50, layer=1, iter_max=500
SVM	0.76	Kernel: RBF, gamma 0.02, C 2.4

Based on the observations above, we can draw the conclusion that Binary Classifier (on Cover_Type 1 & 2). The best performance is around 79%, lower than 80%.

The performance is better than single classifiers, but not improved too much. The bottleneck lies in the binary classifiers. Class 1 and class 2 are highly mixed up so that our model cannot work well on even two classes.

Chapter 5 Feature Engineering

5.1 Introduction

With the extensive amount of free tools and libraries available for data analysis, everybody has the possibility of trying out advanced statistical models in a competition. As a consequence of this, what gives you most "bang for the buck" is rarely the statistical method you apply, but rather the features you apply it to. For most Kaggle competitions the most important part is feature engineering. Feature engineering means using domain specific knowledge or automatic methods for generating, extracting, removing or altering features in the data set. The features you use influence more than everything else the result. There are often some hidden features in the data which can improve the performance by a lot.

5.2 Sub-Problems of Feature Engineering

5.2.1 Feature Importance

You can objectively estimate the usefulness of features. This can be helpful as a pre-cursor to selecting features. Features are allocated scores and can then be ranked by their scores. Those features with the highest scores can be selected for inclusion in the training dataset, whereas those remaining can be ignored. Feature importance scores can also provide you with information that you can use to extract or construct new features, similar but different to those that have been estimated to be useful. A feature may be important if it is highly correlated with the dependent variable (the thing being predicted). Correlation coefficients and other univariate (each attribute is considered independently) methods are common methods. More complex predictive modeling algorithms perform feature importance and selection internally while constructing their model. Some examples include MARS, Random Forest and Gradient Boosted Machines. These models can also report on the variable importance determined during the model preparation process.

5.2.2 Feature Extraction

Some observations are far too voluminous in their raw state to be modeled by predictive modeling algorithms directly. Common examples include image, audio, and textual data, but could just as easily include tabular data with millions of attributes. Feature extraction is a process of automatically reducing the dimensionality of these types of observations into a much smaller set that can be modeled. For tabular data, this might include projection methods like Principal Component Analysis and unsupervised clustering methods. For image data, this might include line or edge detection. Depending on the domain, image, video and audio observations lend themselves too many of the same types of DSP methods. Key to feature extraction is that the methods are automatic (although may need to be designed and constructed from simpler methods) and solve the problem of unmanageably high dimensional data, most typically used for analog observations stored in digital formats.

5.2.3 Feature Selection

Those attributes that are irrelevant to the problem need to be removed. There will be some features that will be more important than others to the model accuracy. There will also be features that will be redundant in the context of other features. Feature selection addresses these problems by automatically selecting a subset that is most useful to the problem. Feature selection algorithms may use a scoring method to rank and choose features, such as correlation or other feature importance methods. More advanced methods may search subsets of features by trial and error, creating and evaluating models automatically in pursuit of the objectively most predictive sub-group of features. There are also methods that bake in feature selection or get it as a side effect of the model. Stepwise regression is an example of an algorithm that automatically performs feature selection as part of the model construction process. Regularization methods like LASSO and ridge regression may also be considered algorithms with feature selection baked in, as they actively seek to remove or discount the contribution of features as part of the model building process.

5.2.4 Feature Construction

Feature importance and selection can inform you about the objective utility of features, but those features have to come from somewhere. You need to manually create them. This requires spending a lot of time with actual sample data (not aggregates) and thinking about the underlying

form of the problem, structures in the data and how best to expose them to predictive modeling algorithms. With tabular data, it often means a mixture of aggregating or combining features to create new features, and decomposing or splitting features to create new features. With textual data, it often means devising document or context specific indicators relevant to the problem. With image data, it can often mean enormous amounts of time prescribing automatic filters to pick out relevant structures.

5.3 Feature Engineering in this project

We spent most of our efforts in feature engineering. We were also very careful to discard features likely to expose us to the risk of over-fitting our model. We spent a tremendous time visualizing the data, understanding which data could bring value, and elaborating a strategy to convert this insight in usable features before producing a first model.

Hence, we tried to define new features that may have geographical meaning and remove features that are not highly correlated with cover types (not important features).

There are several features that describe distances. We tried to define new features based on them. Thus we defined 10 new features by them.

By using the relationship between Aspect and Slope, and Also the way that Aspect has been measured, we were able to define 3new features.

It turned out the idea of adding new and relevant an important features, is actually very similar to building a Deep Artificial Neural Network, where the hidden nodes are actually are new features.

We kept continuing this method and we constructed and created 16 new features (Where can be found in the codes in more details.).

Chapter 6 Resampling Method

After utilizing the new features the prediction accuracy of the extra tree method has been increased to higher than 80%. When we looked at the prediction result on the test set we found that about 90% in the test set are Cover_Type 1 and 2. On the contrary, the number of each Cover_Type in the training set is pretty even. In another word, the prior in the training set and the

test set are different. The prediction result in the training set in based on the fact that the prior of each class is the same, while in the test set the prior of class 1 and 2 are extremely high. The prediction result is showed in Figure 12. The percentage of each class is shown in Table 8.

Table 8 Predicted Percentage in Test Set

Cover_Type	Predicted Percentage in Test set
1	37%
2	42.6%
3	6.4%
4	0.3%
5	4.6%
6	4.1%
7	4.8%

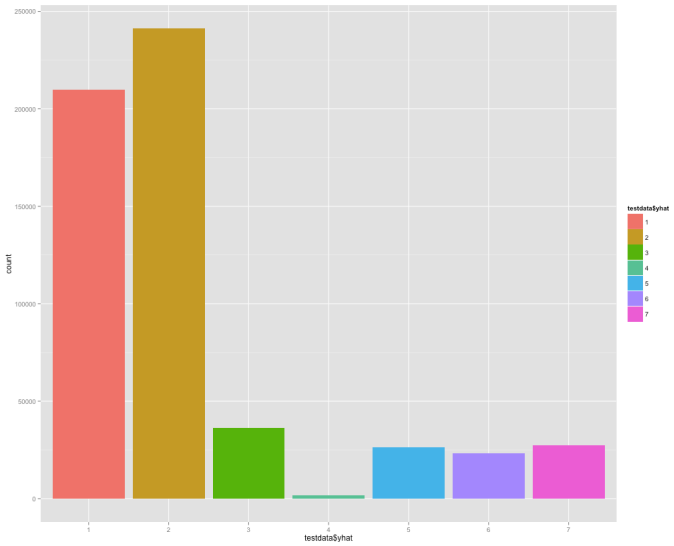

Figure 12 Bar Plot of Prediction Result on Test Set

And 7 classes in training set is evenly distributed mentioned above. To address this problem, one way is to calculate the probability from the extra tree model and then multiply it with the prior of the test set based on the result of our first prediction. However, because of the extremely large size of the test set, the extra tree package in R will encounter some problem and fail to return the probability of each class. Therefore, we use another way to address this problem, which is to change the prior of the training set to make it similar to that of the test set. That is, to reduce the number of class 3-7. Because in the training set each class has roughly 2000 samples, reducing class 3-7 according to the test set will make the number of class 3-7 less than 100 per class. In this way, the accuracy of predicting class 3-7 will be hampered. We have roughly 50 features so each class having 500 samples should be reasonable. Therefore, we resample 2500 samples from class 3-7 so that each class has roughly 500 samples. After adjusting the prior of the training set, the prediction accuracy on the test set is increased.

Another way to look at this method is that we are operating on the ROC curve. The sensitivity can be described using Equation (4) :

$$\text{Sensitivity} = \text{number of predicted class } 1,2 / \text{ number of real class } 1,2 \qquad (4)$$

Because of the large number of class 1,2 in the test set, low sensitivity will lead to a large number of mis-prediction. 1% of sensitivity drop will lead to hundreds of, even thousands of

mis-prediction. Therefore, we really need the sensitivity to be high. Adjusting the prior of training set essentially make the model to predict more class 1, 2 and thus increases the sensitivity as well as accuracy.

Chapter 7 Conclusion and Rethinking

After trying different classifiers and data-preprocessing method, we finally get the prediction accuracy to 84.472%, which ranks 10^{th} among all 902 teams up to Dec. 8, 2014. There are some possible ways for further improvement. The first is iteratively using test data for co-training. When using adaboost and logistic regression for binary classification, even though the percentage of truly predicted type 1,2/real type 1,2 is not high, if the confidence (or probability returned) is higher than 95%, the accuracy will be very high. Therefore, we can apply the classifier on the test set and pick out those results whose confidence is higher than 95%, and use them to co-train the models and effectively enlarge the training set. This method could possibly make use of the large test set and alleviate the problem that the training size is not large enough. The second is to try other randomized methods. We analyze the result of different methods and find that the randomized methods do better than other methods in this problem. A possible reason is because Cover_Type 1 and Cover_Type 2 are too similar. When training models using non-randomized methods the model can easily be trapped in the local optimum, while randomized methods can escape such pitfalls. Therefore, another way for possible improvement is to try out other randomized methods. Since the codebook for Error Correction Code is generated randomly, it may worth a try.

References

[1] Extremely randomized trees, P Geurts, D Ernst, L Wehenkel - Machine learning, 2006 – Springer.
[2] Applied Predictive Modeling, Max Kuhn, Kjell Johnson, Springer.
[3] Ridgeway, Greg. "Generalized Boosted Models: A guide to the gbm package." Update 1.1 (2007).
[4] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of Statistics (2001): 1189-1232.
[5] Friedman, Jerome H. "Stochastic gradient boosting." Computational Statistics & Data Analysis 38.4 (2002): 367-378.
[6] [ONLINE] Available at: https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.info. [Accessed 09 December 2014].
[7] AZ Master Gardener Manual: Environmental Factors. 2014. AZ Master Gardener Manual: Environmental Factors. [ONLINE] Available at: http://ag.arizona.edu/pubs/garden/mg/botany/environmental.html. [Accessed 09 December 2014].
[8] How Hillshade works. 2014. How Hillshade works. [ONLINE] Available at: http://edndoc.esri.com/arcobjects/9.2/net/shared/geoprocessing/spatial_analyst_tools/how_hillshade_works.htm. [Accessed 09 December 2014].
[9] Azimuth - Wikipedia, the free encyclopedia. 2014. Azimuth - Wikipedia, the free encyclopedia. [ONLINE] Available at: http://en.wikipedia.org/wiki/Azimuth. [Accessed 09 December 2014].
[10] Lapse rate - Wikipedia, the free encyclopedia. 2014. Lapse rate - Wikipedia, the free encyclopedia. [ONLINE] Available at: http://en.wikipedia.org/wiki/Lapse_rate. [Accessed 09 December 2014].
[11] Liaw A, Wiener M. Classification and Regression by randomForest[J]. R news, 2002, 2(3): 18-22.
[12] Svetnik V, Liaw A, Tong C, et al. Random forest: a classification and regression tool for compound classification and QSAR modeling[J]. Journal of chemical information and computer sciences, 2003, 43(6): 1947-1958.
[13] Zhang H, Zhang Z. Feedforward networks with monotone constraints[C]//Neural Networks, 1999. IJCNN'99. International Joint Conference on. IEEE, 1999, 3: 1820-1823.
[14] Gordon A D. A review of hierarchical classification[J]. Journal of the Royal Statistical Society. Series A (General), 1987: 119-137.
[15] Silla Jr C N, Freitas A A. A survey of hierarchical classification across different application domains[J]. Data Mining and Knowledge Discovery, 2011, 22(1-2): 31-72.

Appendix

Here is the Code Link.