# Forest Cover Type Prediction

February 23, 2016

## Introduction

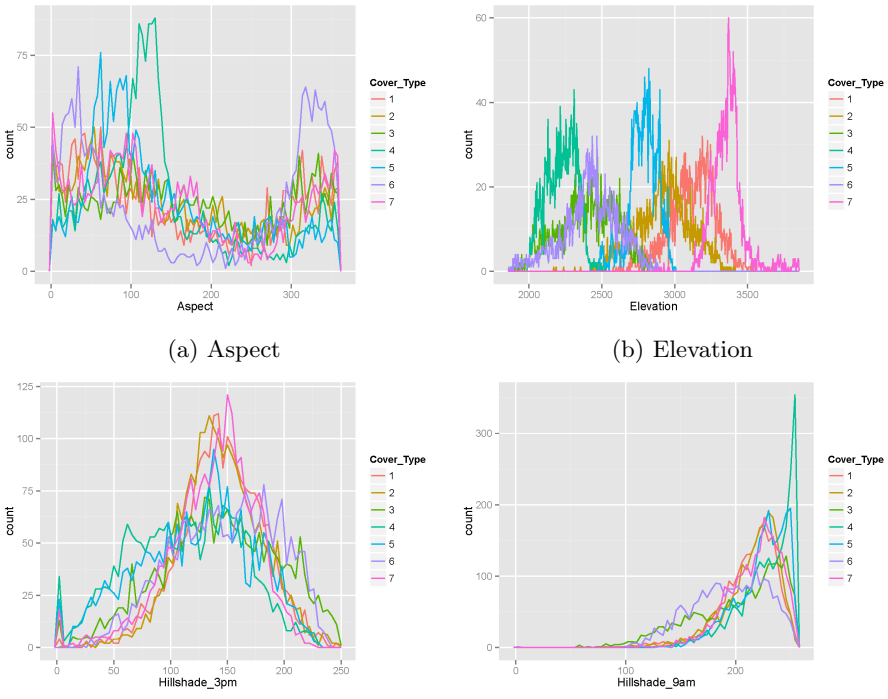
This is a Kaggle project.

## dataset

Check the dataset in this page.

## Exploratory analysis and feature selection

First I plot each feature's distribution of each category:



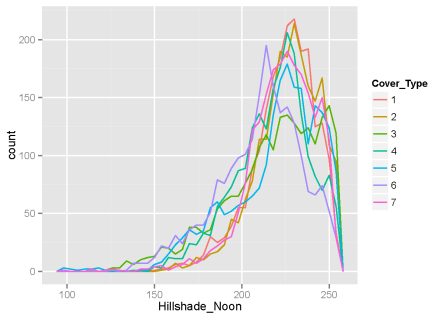

(a) Aspect

(b) Elevation

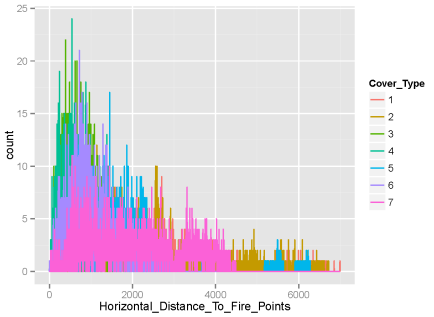
(c) Hillshade (3pm)

(d) Hillshade (9am)

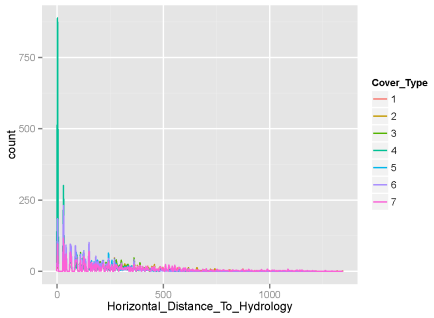
Figure 1: Distribution of features



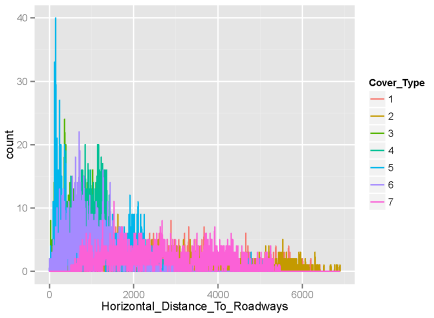

(e) Hillshade (Noon)



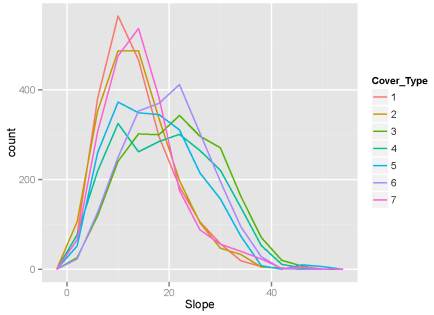

(f) Horizontal distance to fire points



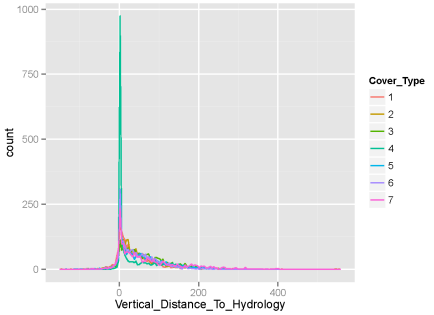

(g) Horizontal distance to hydrology



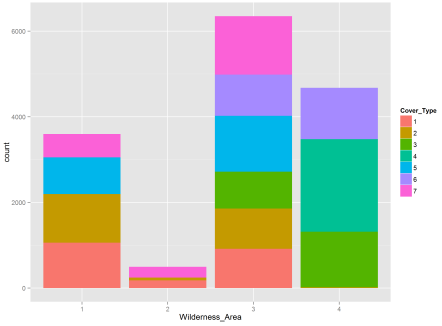

(h) Horizontal distance to roadways



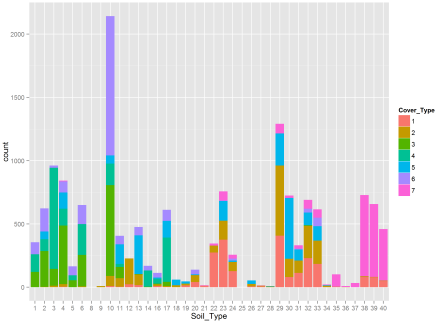

(i) Slope




(j) Vertical distance to hydrology




(k) Area




(l) Soil types

Figure 1: Distribution of features

Some insights or further guesses by reading the graphs here:

Guess-1 For continuous features, 'Aspect','Hillshade_9am','Hillshade_Noon','Hillshade_3pm' don't seem significantly different across the classes;

Guess-2 There seems a latent relationship between soils types and areas. A guess could be that 'Wilderness_Area1' contains 'Soil_Type20'-'Soil_Type34'; 'Wilderness_Area2' contains 'Soil_Type35'-'Soil_Type40', etc.

Insight-1 For discret features, 'Soil_Type7','Soil_Type8','Soil_Type9','Soil_Type15','Soil_Type21', 'Soil_Type27','Soil_Type28','Soil_Type34','Soil_Type36' have almost zero variance ($\leq 0.002$ after normalization). These features could be removed.

## Guess-1

To test Guess-1, I simply apply a t-test for these features. Here are the results:

We can see that even though in a few cases the difference is not significant, in most cases, they are significantly different. Those features can't be removed for this reason.

When mentioning Hillshade, however, it is acutally helpful to see how Hillshade is calculated:

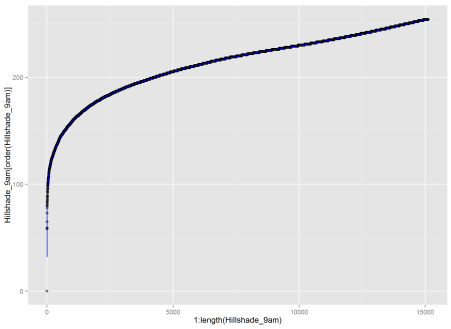
$$\begin{aligned} Hillshade = 255.0 * ((\cos(Zenith_{rad}) * \cos(Slope_{rad})) \\ + (\sin(Zenith_{rad}) * \sin(Slope_{rad}) * \cos(Azimuth_{rad} - Aspect_{rad}))) \end{aligned}$$

Where both Zenith and Azimuth only depend on time of a day! So, in theory, we can remove 'Hillshade_9am','Hillshade_Noon','Hillshade_3pm' because we already have 'Slope' and 'Aspect'.

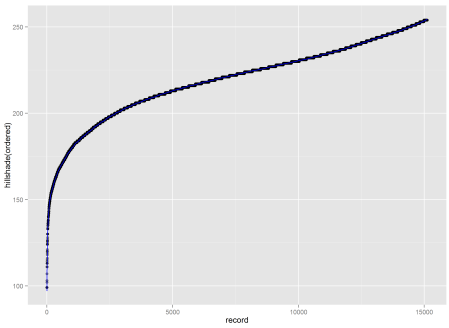
To validate that, I do a regression for Hillshade in three different time:

$$\begin{aligned} Hillshade_X/255 = \beta_{1X} \cos(Slope_{rad}) \\ + \beta_{2X} \sin(Slope_{rad}) * \cos(Azimuth_{rad} - Aspect_{rad}), X = 9am, Noon, 3pm \end{aligned}$$

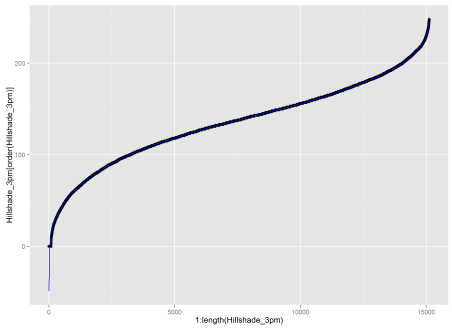
Azimuth can be calculated from here. Basically, it is about 90 at 9am, 40 at noon and 290 at 3pm (you can play with it to reach a higher R square because the actual value of Azimuth doesn't matter). For all of three regressions, I get a 1 R square ($\geq$0.999). The fitted value is plotted below:




(a) 9am




(b) noon




(c) 3pm

Figure 2: Hillshade and fitted value

So, when classifying, we can compare the difference between removing hillshade features and not.

## Guess-2

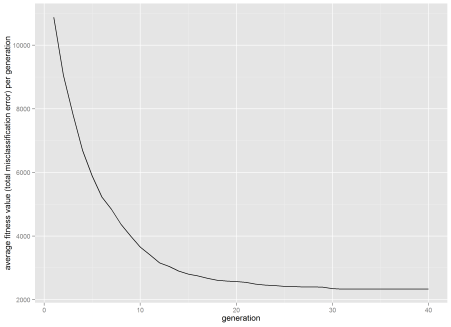
To test whether area features can be replaced by soil type features, it is acually to test **whether there exists a partition that divides** $U = 1, 2, \cdots, 40$ **into 4 sets** $U_i$**, so that we are satisified with the overall mis-classification error:**

$$agg(error(a_{ik}, \sum_{j \in U_i} s_{jk})), i = 1, 2, 3, 4; j \in U, k = 1, 2, \cdots 7$$

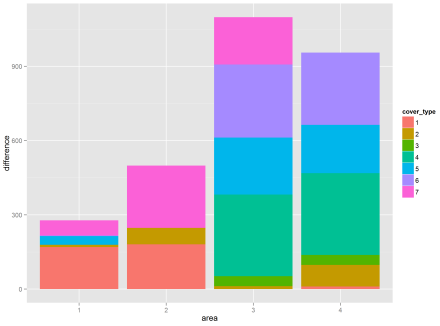
A simple version is to calculate the total mis-classification records, that is

$$\sum_i \sum_k \| a_{ik} - \sum_{j \in U_i} s_{jk} \|, i = 1, 2, 3, 4; j \in U, k = 1, 2, \cdots 7$$

This is a Combinatorial optimization problem. I get the suboptimal solution using genetic algorithm (population size = 400, generations = 40)



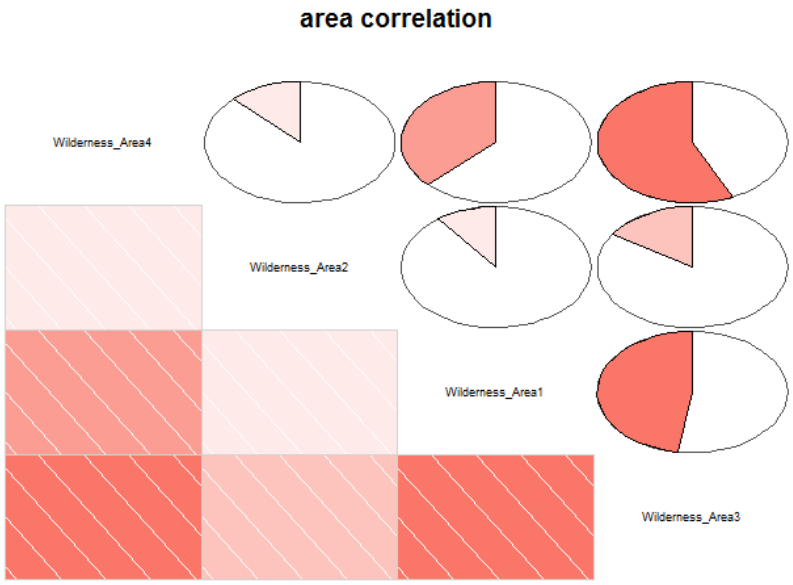

(a) Average total misclassification error for 'Area'-'Soil_Type' match against generation




(b) Difference for the suboptimal solution

One suboptimal solution can reach a total 2331 out of 15120 mis-classified. Considering it is a 7-class dataset, it is an acceptable level. So, when classifying, we can compare the difference between removing 'Wilderness_AreaX' features and not.

Finally, I need to remove one 'Wilderness_AreaX' feature and one 'Soil_TypeX' feature to prevent Multicollinearity. To do that, I plot corrleation graph of them:



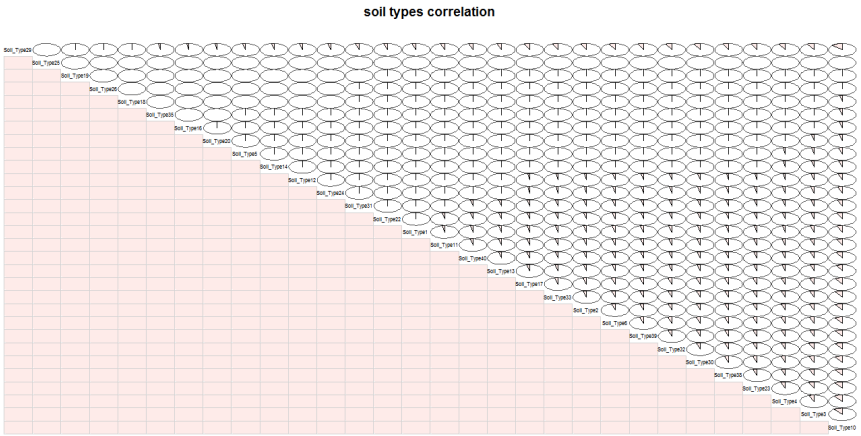



Figure 3: Project framework

I remove the features that have a high corrleation with other features, that is, 'Wilderness_Area3' and 'Soil_Type10'.

### summary

After the work in this section, I removed features 'Soil_Type7','Soil_Type8','Soil_Type9', 'Soil_Type15','Soil_Type21','Soil_Type27','Soil_Type28','Soil_Type34','Soil_Type36', 'Wilderness_Area3', 'Soil_Type10', and plan to build models comparing removing features Hillshade or Wilderness_Area versus not.

## Classification model

Here I tried SVM and Extra Trees Classifier (Random Forest is also tried but not so good as it in this project). First of all, for both models, removing hillshade features but reserving area features reach a better cross validation scores.

Then I tune the parameters by cross valiation and grid-serach. For SVM, the optimal paramters are $C = 2^{5.5}, \gamma = 2^{-18}$. For Extra Trees Classifier, they are $max\_features = 'log2', max\_depth = 40$. However, the cross validation score of SVM is much lower than that of Extra Trees. For the latter, I reach a **0.77293** score after uploading on Kaggle.

With the inspiration from this post, I notice that the distribution of 7 cover types in the training set is quite different from the original dataset. Plus, the predcited distribtuion in the test set is similar to that of the original dataset. So, I apply an iteration on fitting, predicting and sampling. Here is a general view:

**Algorithm 1** Sampling Iteration

**Require:** MINIMAL DATA SIZE, TRAINSET, TESTSET
  **while** length(TRAINSET) $\geq$ MINIMAL DATA SIZE **do**
    TRAIN_X, TRAIN_Y $\leftarrow$ 1 get_xy(TRAINSET)
    ClF $\leftarrow$ 1 init_classifier(params){//init classifier}
    fit(CLF,TRAIN_X, TRAIN_Y)
    PREDICT_Y $\leftarrow$ 1 predict(CLF, TESTSET){//get predicted values}
    CLASS_WEIGHTS $\leftarrow$ 1 get_weights(PREDICT_Y){//generate predcited distribution of classes in the test-set; normailize it into weights of classes}
    TRAINSET $\leftarrow$ 1 sample(TRAINSET, CLASS_WEIGHTS){//sample trainset based on class weights}
  **end while**
  **return** PREDICT_Y

Where CLASS_WEIGHTS is normalized such that the weight of the class with maximal records is 1. It is like an EM algorithm, penalting the misclassification based on the weights, and generating weights from the prediction results. It is also a trade-off between iteration times and remaining training set size. One more iteration may reach to the minimal closer (it does not necessarily coverage), while the training set may decrease a lot. After one iteration, I reach a test accuracy of **0.80570**.