FINAL PROJECT: Forest Cover Type Prediction

Table of Contents

Introduction 2
Data selection 3
K-Nearest Neighbor (KNN) 5
Support Vector Machine (SVM) 6
Random Forest 7
Conclusion 8
Using Random Forest model on test dataset 9

Introduction

The main goal of this project is to use and compare alternative techniques for predicting forest cover types (the predominant kind of tree cover) from strictly cartographic variables.

I used a train and test datasets in CSV format. Train dataset has 15120 observations with 55 features (including Id) and target. Test dataset has total 565892 observations with 55 features (including Id). There are total 7 forest cover types. So this is a multiclass classification problem as I need to classify the instances into those 7 classes.

Following is the list of methods that I used to find out the best working method for us:

1. K-Nearest Neighbor
2. Support Vector Machine
3. Random Forest

I have also performed fine tuning of parameters for all of these methods.

To accomplish this task, I have used Jupyter Notebook with Python 3. Following libraries are used to perform analysis and get desired outputs.

1. Pandas – useful to import and clean the data up. Pandas dataframes are easier to use than raw numpy arrays since they give us column names and many methods to manipulate data.
2. Sklearn – contains modules for dividing the data into parts, using different models and calculating accuracies.
3. Seaborn – useful for plotting graphs. It is a wrapper around matplotlib providing nicer graphs.

Data selection

Before running any machine learning algorithm, the most important task is to have good dataset. There are chances that some values are missing or invalid in the dataset. Sometimes I need to introduce new features as well to make it easier to process the dataset or to apply specific algorithm. The activities included in data selection are: visualize the data, analyze it and clean it up. It also includes understanding the features, normalizing the data, dimensional reduction.

First the data is read from “train.csv” file and a **pandas** dataframe is created. If I look at the data, I find that there is a feature: “Id”. It is just for uniquely identifying each observation. This feature does not play any important role in training or testing any algorithm. So I start by removing this feature.

There are 10 quantitative features that are not scaled and 44 categorical features with values 0 or 1. I found no missing data in any of the columns. I scaled quantitative data using **StandardScaler** (removing the mean and scaling to unit variance) found in **preprocessing** module of **sklearn** library.

Let’s look at the distributions of the quantitative data to get a feeling for it.

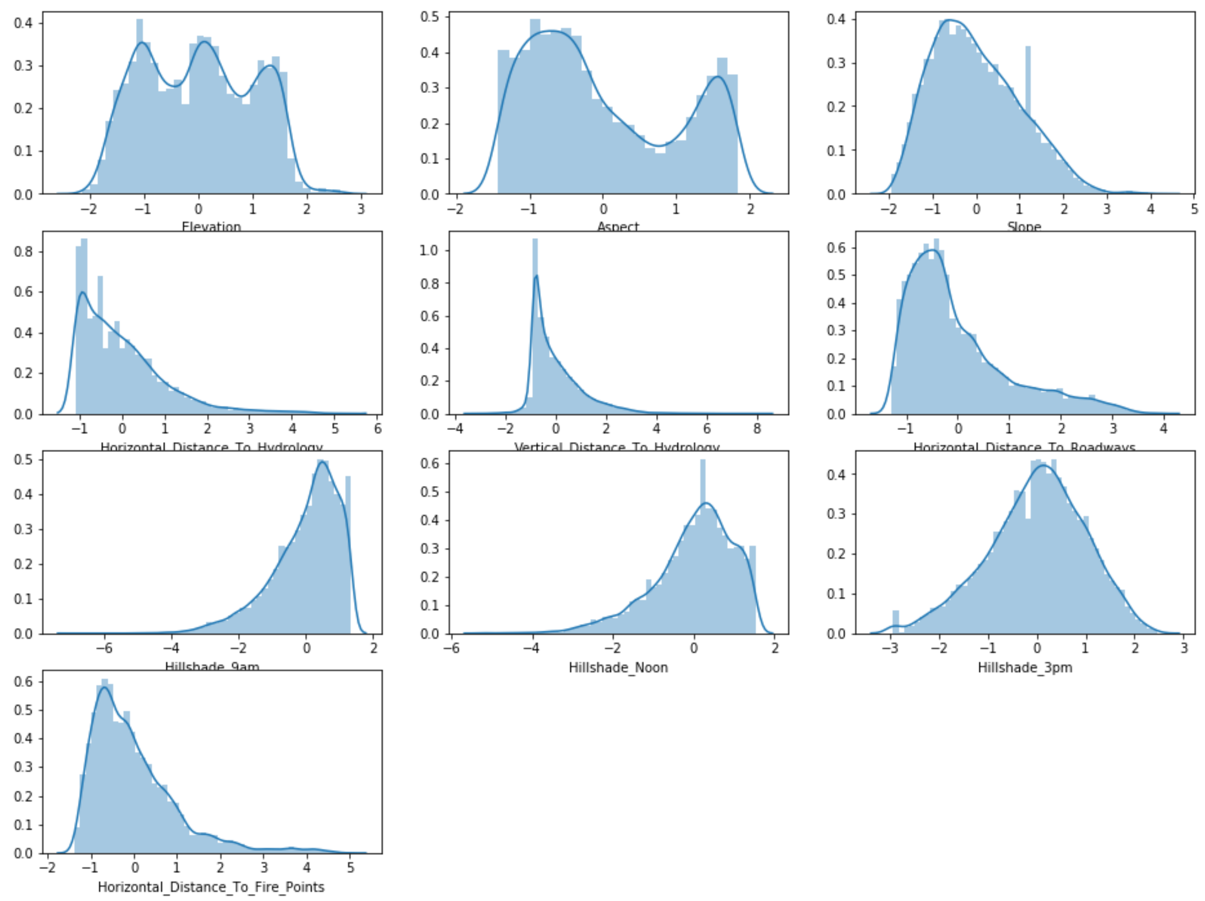


Figure 1 Histogram of quantitative data

Most of the data seem to be a skewed normal distribution with the exception of **Aspect** which seems to be bimodal. **Elevation** also seems like multi-modal distribution.

Let's plot the distribution of the quantitative variables according to their class.

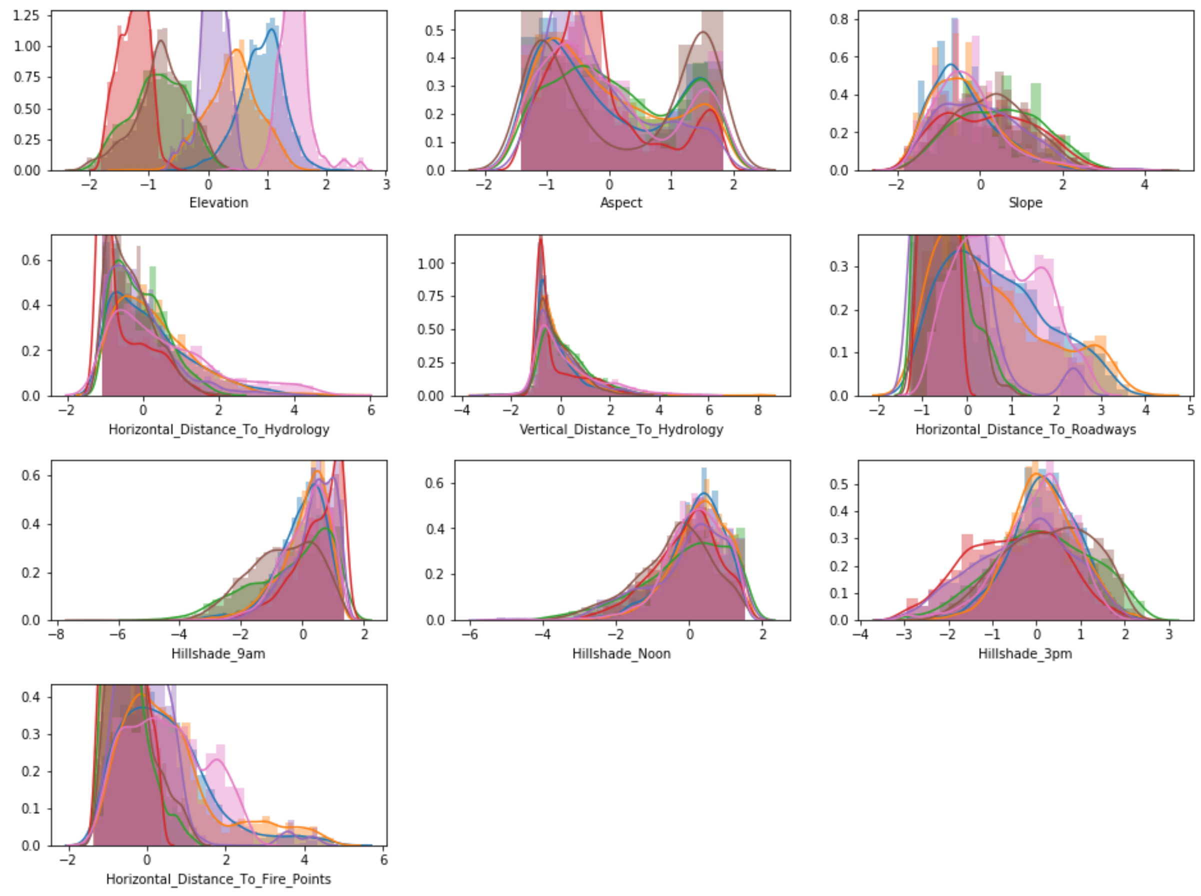


Figure 2 Quantitative data class-wise distribution

As we can see, **Elevation** is the most potent indicator of the class.

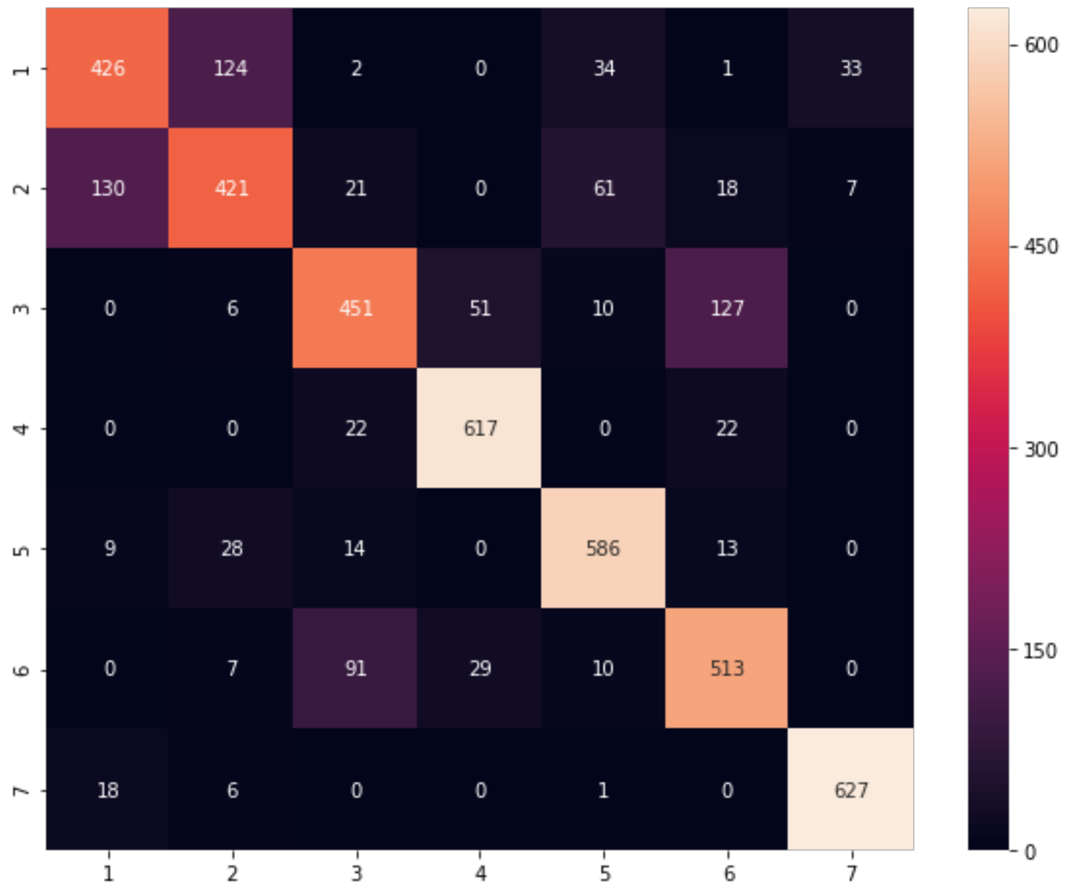
I store the normalized data into **cleaned_train.csv** file. Finally, I split the normalized data into training and testing sets using **train_test_split** method found in **model_selection** module of **sklearn** library.

K-Nearest Neighbor (KNN)

This algorithm is one of the simplest supervised machine learning algorithms. K-Nearest Neighbour (KNN) algorithm is used for regression as well as classification problems. It classifies the data point based on how their neighbours are classified. **K** in this algorithm is a parameter that specifies the number of nearest neighbours. This algorithm looks promising for the kind of dataset I have.

I used standard **sklearn** method to train our **KNN** classifier model. To fine tune parameters, I set the value of **k** from *2 to 10*. I used the **weights** parameter set to *distance*. With weights='distance', closer neighbours of a data point will have a greater influence than neighbours which are further away. I also set the number of jobs to -1, so that it will use all the available cores and complete the operation faster.

The best result I achieved using KNN is with **k=2**. It scored **80.27%** accuracy. Following plots the confusion matrix I get for the KNN model with fine-tuned parameters.



	1	2	3	4	5	6	7
1	426	124	2	0	34	1	33
2	130	421	21	0	61	18	7
3	0	6	451	51	10	127	0
4	0	0	22	617	0	22	0
5	9	28	14	0	586	13	0
6	0	7	91	29	10	513	0
7	18	6	0	0	1	0	627

Figure 3 KNN - Confusion matrix

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning model in machine learning that analyse data used for classification and regression. Normally SVM is used for binary classification. But there is an extension to this SVM which allows us to use it even for multi-class classifications.

In **sklearn** library, I get support for **SVC** (Support Vector Classification) model. It implements "one-against-one" approach for multi-class classification. I used **RBF** kernel as it suits the best for our dataset and classification problem. To fine tune its other parameters, I tried with different values of **C** and **gamma**. Here, **C** is a regularization parameter that controls the influence of each individual support vector and **gamma** is kernel coefficient.

To train our model, I used all the combinations of C and gamma where C is in [1, 10, 100, 1000] and gamma is in [1e^{-3}, 1e^{-4}]. After trying all these with RBF kernel, I get the best accuracy of **75.88%**. This accuracy is achieved with **C=1000** and **gamma=1e^{-3}**. Following plots the confusion matrix I get for the SVC model with fine-tuned parameters.

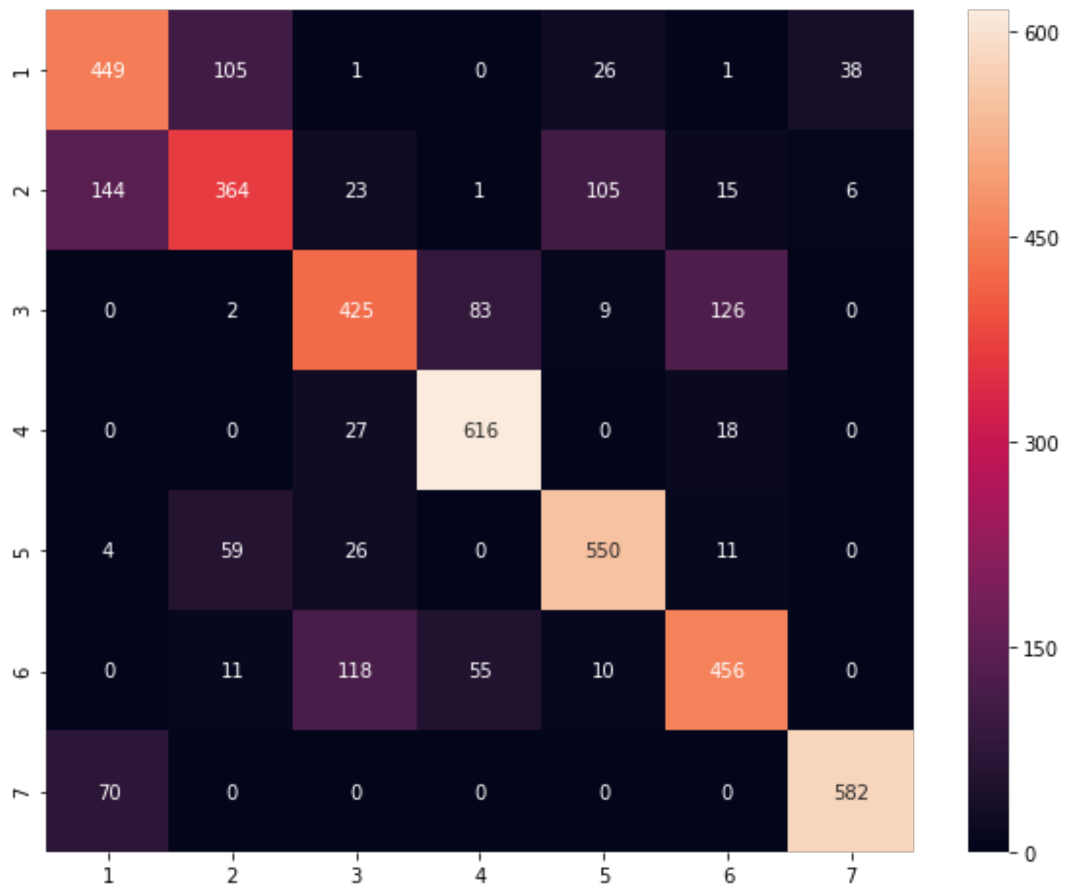


Figure 4 SVC - Confusion matrix

Random Forest

Decision trees are used for both classification and regression. The model behaves with "if this then that" conditions ultimately yielding a specific result. These trees send us down a specific route to find out an answer by using sequential questions. This model can get very good accuracy but decision trees are prone to overfitting, especially when a tree is particularly deep.

To solve this error in decision tree model, **Random Forest** model comes into the scene. A random forest is simply a collection of decision trees. The results of these decision trees are aggregated into a single final result. Random forest is one of the best models in machine learning to solve classification problems. Though, this model can be very slow for large amount of data.

I have used **RandomForestClassifier** model provided in **ensemble** module in **sklearn** library. I concentrated on fine tuning the most important two parameters: **n_estimators** and **max_depth**. The first one, n_estimator, specifies the number of trees in the forest. The second one, max_depth, specifies the maximum depth of the tree that the model should process to findout the result.

To train our model, I used all the combinations of n_estimators and max_depth where n_estimators is in [100, 200, 500] and max_depth is in [10, 25, 50, 60]. After trying all these, I get the best accuracy of **86.51%**. This accuracy is achieved with **n_estimators=200** and **max_depth=50**. Following plots the confusion matrix I get for the Random Forest model with fine-tuned parameters.

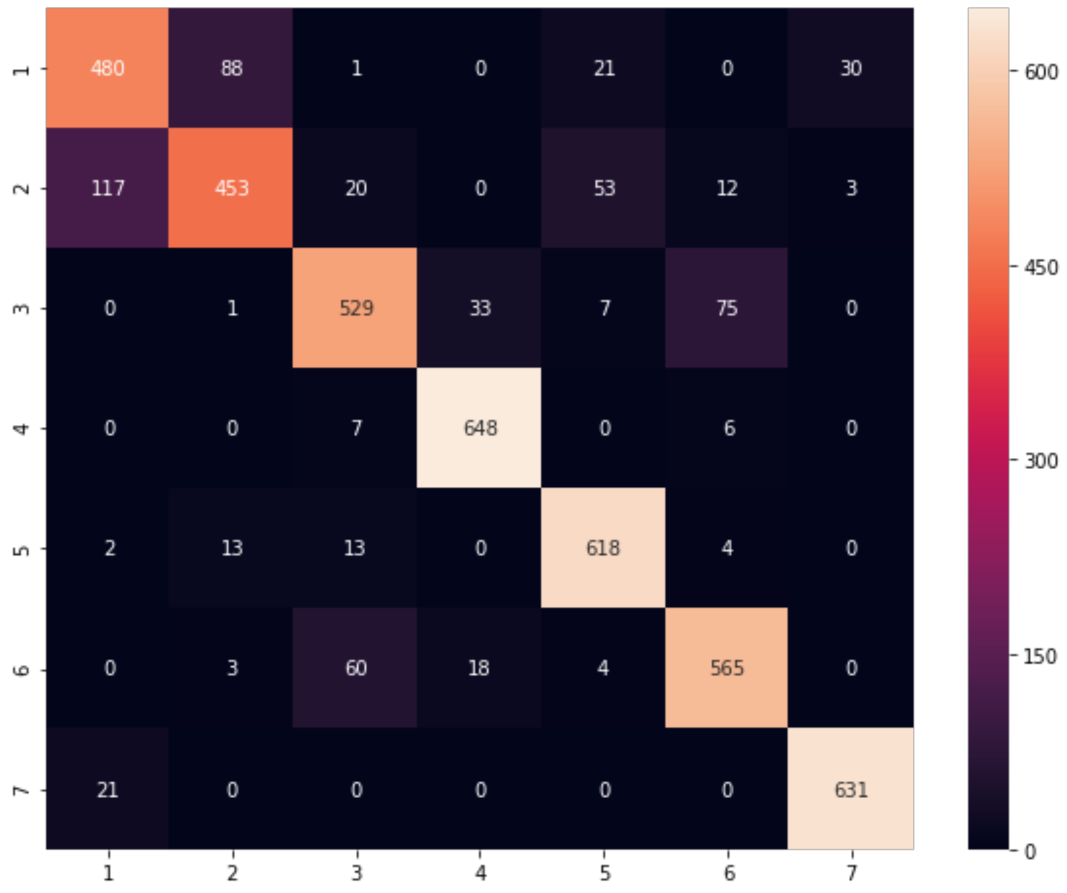


Figure 5 Random Forest - Confusion matrix

Conclusion

I performed data reading, visualization, data cleansing & normalization. I split the data into train and test sets. Then I prepared some models and fine-tuned best parameters for them by training them and predicting labels. Following is the list of all the models I tried:

1. K-Nearest Neighbor (KNN)
2. Support Vector Machine (SVM)
3. Random Forest

After applying the best parameters to the models, I found following accuracy:

Model	Best Parameters	Accuracy
K-Nearest Neighbor (KNN)	k=2,	80.27%
Support Vector Machine (SVM)	kernel=rbf, C=1000, gamma=1e^{-3}	75.88%
Random Forest	n_estomators=200, max_depth=50	86.51%

As far as algorithms go, for data with this much training features and samples, **Random Forest** classifier seems to do the best. K-Nearest Neighbour (KNN) also performed good. Though, random forest is very slow for large dataset, I think **Random Forest** is a clear winner for this project.

Using Random Forest model on test dataset

At last, I read observations from **test.csv** file and prepared a pandas data frame in Jupyter Notebook. There are total 565892 observations. I faced **Memory Error** while processing all the observations at once. To solve this issue, we divided all the observations into slots with 100000 observations in each slot. We then performed normalization on each slot data. I already have the Random Forest model trained with fine-tuned parameters on train dataset. So I used the same trained model on each slot to predict the labels. The labels are predicted and are stored in separate CSV file for each slot. These CSV files include only two columns: **Id** and **Cover_Type**.