

APPLICATION OF MACHINE LEARNING ALGORITHMS ON THE FOREST COVER DATASET

Kunal Chandiramani (16BCE1396), Dhruv Garg (16BCE1190), Nikhilesh Prabhakar (16BCE1158), Divyansh Choudary (16BCE1152), Pruthviraj Patil (16BCE1096), Aditya Chitlangia (16BCE1143).

ABSTRACT

Deforestation is an issue being faced for a long time now, and there is no stopping it. Majority of things now-a-days are the products of wood. Paper is being used in abundance and there is no stopping it, no one has yet come up with an alternate which is as good as paper. Most of the furniture and many more products are made up of wood. Classifying the forest covers into various types will help the authorities in maintaining the forests and protecting them by controlled deforestation and regrowing. This will help in improving the quality of forest covers.

INTRODUCTION

A large number of animal species are residents of the forest which are under constant threat of deforestation, even after the strict laws to execute deforestation in controlled manner the authorities are not able to manage the task efficiently. Data collected by US Forest Services is processed and used for classifying the different forests. Our main goal is to classify the forests into the 7 predefined classes using the data. Many different machine learning models were compared in the dataset for attaining the best accuracy. The data was taken from the uci database.

METHODOLOGY

In this paper we are going to implement multiple machine learning model's on Forest Cover Type dataset. To start with first we are going to do data pre-processing so that we can fill out missing values and also remove noise from our dataset. So, after the preprocessing step we are going to do feature engineering to create more features, for better accuracy. After which we will apply regularization so that all the attributes are in the same range. At last we are going to apply multiple machine learning algorithms and check which performs the best among them. Also we are going to use GridSearchCV for hyper parameter optimization, to select the best parameters for better accuracy.

DATASET

The train dataset chosen has 15K records with 55 attributes. The attributes include:

- Elevation
- Aspect
- Slope
- Horizontal distance to hydrology
- Vertical distance to hydrology
- Horizontal distance to roadways
- Hillshade 9am, noon, and 3pm
- Horizontal distance to firepoint
- Wilderness level
- Soil Types

The wilderness_Areas include 4 binary columns, soil_type contains 40 binary columns, and the cover types include 7 classes. The 7 forest cover type classes are spruce/Fir(1), Lodgepole Pine(2), Ponderosa Pine(3), Cottonwood/Willow(4), Aspen(5), Douglas-fir(6), Krummholz(7).

Data preprocessing and Feature Engineering:

The most important step before applying any machine learning algorithm is doing data pre-processing. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. Some of the steps taken on forest cover type dataset are as follows:

1. Looking for missing values:

- The first step was looking for missing values. Since, there were no missing values and data was already clean no steps were taken.

2. Looking into description of the data:

-when we looked at the description of the data, we found that the SOIL_TYPE 7 and SOIL-TYPE 14 had 0 variance i.e all the values of columns were same and 0. So we removed both the columns from the dataset.

3. Normalizing the data:

-We used sklearn standardscaler to standardize the data. Normalizing is very important step to get good results.

After the preprocessing step the main challenge faced by us was that, the 7 forest cover types were not separable and were overlapping. In order to solve this problem we did feature engineering, to create more features from the given data.

To start with feature engineering, we initially found the feature importance using random forest algorithm, the bar chart for the top important features is shown in Figure.01. In addition to that we also found the correlation matrix to look for the features which are highly correlated as shown in Figure.02.

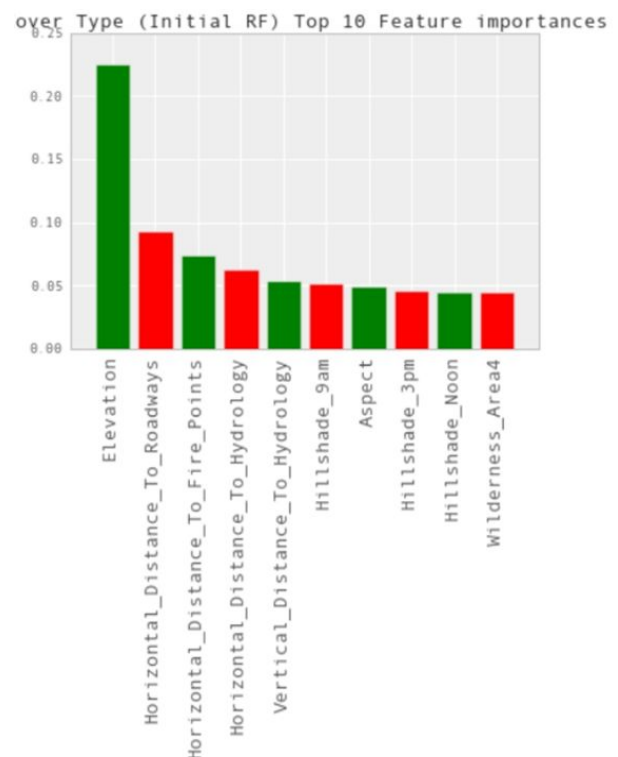


Figure.01

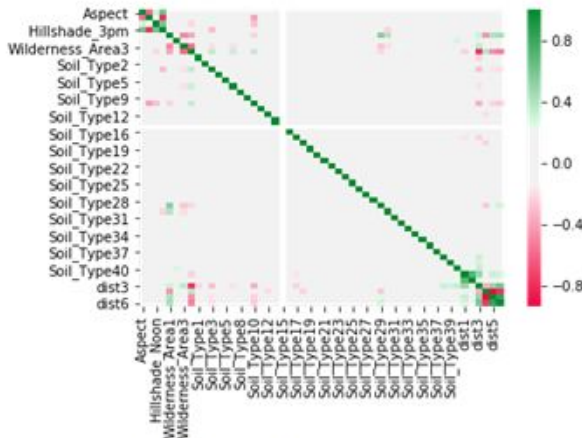


Fig. 2 Correlation between attributes

Figure.02

For creating more features, we took the most important features which were elevation, horizontal_distance_to_roadways, aspect, hillshade_at_3pm, etc and added/subtracted the two which were highly correlated. For example, horizontal_distance_to_roadways and horizontal_distance_to_firepoint had high correlation and were important so we added and subtracted the two to create two new features and so on. Some of the features that we created are shown in Figure.03 and Figure.04.

```
df['HF1'] = df['Horizontal_Distance_To_Hydrology'] + df['Horizontal_Distance_To_Fire_Points']
df['HF2'] = (df['Horizontal_Distance_To_Hydrology'] - df['Horizontal_Distance_To_Fire_Points'])

df['HR1'] = (df['Horizontal_Distance_To_Hydrology'] + df['Horizontal_Distance_To_Roadways'])
df['HR2'] = (df['Horizontal_Distance_To_Hydrology'] - df['Horizontal_Distance_To_Roadways'])
df['FR1'] = (df['Horizontal_Distance_To_Fire_Points'] + df['Horizontal_Distance_To_Roadways'])
df['FR2'] = (df['Horizontal_Distance_To_Fire_Points'] - df['Horizontal_Distance_To_Roadways'])

df['EV1'] = df.Elevation + df.Vertical_Distance_To_Hydrology
df['EV2'] = df.Elevation - df.Vertical_Distance_To_Hydrology
df['Mean_HF1'] = df.HF1/2
df['Mean_HF2'] = df.HF2/2
df['Mean_HR1'] = df.HR1/2
df['Mean_HR2'] = df.HR2/2
df['Mean_FR1'] = df.FR1/2
df['Mean_FR2'] = df.FR2/2
df['Mean_EV1'] = df.EV1/2
df['Mean_EV2'] = df.EV2/2
df['Elevation_Vertical'] = df['Elevation'] + df['Vertical_Distance_To_Hydrology']
df['Neg_Elevation_Vertical'] = df['Elevation'] - df['Vertical_Distance_To_Hydrology']
```

Figure.03

```
#Shadiness
df['Shadiness_morn_noon'] = df.Hillshade_9am/(df.Hillshade_Noon+1)
df['Shadiness_noon_3pm'] = df.Hillshade_Noon/(df.Hillshade_3pm+1)
df['Shadiness_morn_3'] = df.Hillshade_9am/(df.Hillshade_3pm+1)
df['Shadiness_morn_avg'] = (df.Hillshade_9am+df.Hillshade_Noon)/2
df['Shadiness_afternoon'] = (df.Hillshade_Noon+df.Hillshade_3pm)/2
df['Shadiness_mean_hillshade'] = (df['Hillshade_9am'] + df['Hillshade_Noon
ade_3pm']) / 3

# Shade Difference
df['Hillshade-9-Noon_diff'] = df["Hillshade_9am"] - df["Hillshade_Noon"]
df['Hillshade-noon_3pm_diff'] = df["Hillshade_Noon"] - df["Hillshade_3pm"]
df["Hillshade-9am_3pm_diff"] = df["Hillshade_9am"] - df["Hillshade_3pm"]

# Mountain Trees
df["Slope*Elevation"] = df["Slope"] * df["Elevation"]
```

Figure.04

ALGORITHMS IMPLEMENTED:

- **Random Forests**

Random Forests were used for its bagging approach, it improves the result when compared to a single decision tree which is the base model used for bagging in random forest. The dataset is first bootstrapped in this approach on which different decision trees are then applied to classify the forests on those attributes. Random forests was chosen because the cases were highly overlapped and an exhaustive approach to classify them was needed hence Random forests was the first thought that came to be applied on the data that is being preprocessed.

Implementation: After the Random forests were applied for the data without the given feature engineering, we used Grid Search CV to approximate the best number of estimators. We found out that 900 trees works the best.

- **Naive bayes**

Naive Bayes is among the simplest probabilistic classifiers. It often performs surprisingly well in many real world applications, despite the strong assumption that all features are conditionally independent given the class. In the learning process of this classifier with the known structure, class probabilities and conditional probabilities are calculated using training data, and then values of these probabilities are used to classify new observations.

Naive Bayes performs well when we have multiple classes.

NB is used because its simple and if conditional independence of data assumption actually holds, a Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data. And even if the NB assumption doesn't hold. It requires less model training time.

Implementation: As the dataset was not normalized, scaling was done.

Then feature engineering was done along with addition of new features.

Also there were many independent features so NB optimisation was applied with hyper-parameters like `n_folds=7`, `learning rate=0.02` and `n_estimators=10000`.

After that Light GBM was used for boosting. Light GBM is a gradient boosting framework that uses tree based learning algorithm. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max

delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

- **SVM**

Algorithm: Support Vector Machine (SVM) is a supervised learning algorithm. In SVM we plot all the data points in the n-dimensional space, and then perform classification by finding a hyperplane which will separate the classes well. Optimal plane is the one which separates the classes well and has the largest possible margin on both sides with respect to the support vectors. Also, SVM is robust to outliers. For classifying data that is not linearly separable, the SVM has a technique called the kernel trick, which converts a non-separable problem to a separable problem. Important parameters in SVM are (i) Kernel: kernel options include linear, rbf and poly, (ii) Gamma: Higher value of gamma will try to exact fit the training data which could lead to overfitting and cause generalization error and (iii) C: This is the penalty parameter of the error term. It controls the trade-off between smooth decision boundary and classifying the training points correctly.

Implementation: As the dataset was not normalized, scaling was done. First SVM implementation on the dataset involved tuning of hyper parameters without feature engineering. Thus, to get a decent training accuracy, hyper parameters had stringent limits. The penalty parameter (C) was set to 4500, and tolerance was 0.0001. Kernel used was the radial basis function. In the second

implementation of SVM, feature engineering was done, and better accuracy was obtained with less stringent hyper parameters for the same kernel type. The penalty parameter was 4000 and tolerance was 10 times higher, at 0.001.

- **MLP**

Algorithm: Multi-layer perceptron is a part of “feedforward” neural networks. It consists of At Least 3 layers of neurons – Input layer (no. of neurons = input features + 1 for bias), hidden layers (can be one or multiple) and output layer (no. of output neurons is equal to the number of output classes). The MLP uses the backpropagation technique for training. Usually, even a single hidden layer is sufficient for majority of the problems. In this case the number of neurons in the hidden layer should be equal to the mean of the number of neurons in the input and output layer. If multiple hidden layers are used, the total number of neurons in the hidden layer should be lesser than the total number of (input + output) layer neurons. Important parameters in MLP are (i) hidden layer sizes: represents the number of neurons in the hidden layers, (ii) activation function options include identity, logistic, tanh and relu, (iii) solver for weight optimization- SGD, lbfgs and adam, and (iv) alpha as the L2 regularization parameter.

Implementation: The multi-layer perceptron classifier was trained after feature engineering on the dataset. Since the data is not linearly separable, we used the thumb rules along with hit and trial to determine the number of hidden layers, and neurons in

each layer. We implemented a neural network with 4 hidden layers having 300, 150, 75 and 40 neurons respectively. For the other hyper-parameters like alpha, solver, learning rate and tolerance, GridSearchCV was implemented to get the best possible parameters. Although GridSearchCV was time consuming and computationally intensive, it gave us the best tuning of parameters. Thus, activation function used was tanh, alpha was set to 3, solver as lbfgs, learning rate was adaptive and tolerance was 0.00001.

- **Extra trees**

The main objective of Extra Trees model is further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree. With respect to random forests, the method drops the idea of using bootstrap copies of the learning sample, and instead of trying to find an optimal cut-point for each one of the K randomly chosen features at each node, it selects a cut-point at random. From a statistical point of view, dropping the bootstrapping idea leads to an advantage in terms of bias, whereas the cut-point randomization has often an excellent variance reduction effect. This method often leads to increased accuracy because to its smoothing and at the same time significantly reduces computational burdens linked to the determination of optimal cut-points in standard trees and in random forests.

Implementation: After the preprocessing step and featuring engineering step, the next step is applying the model by choosing the best hyper parameters. To select the best parameters we used GridSearchCV, after applying GridSearchCV the features which gave us best parameters are criterion = 'gini', n_estimators = 1000, max_features = 10, min_samples_split = 2, warm_start = False.

- **Decision trees with Adaboost**

Algorithm: AdaBoost, short for “Adaptive Boosting”, is the first practical boosting algorithm proposed by Freund and Schapire in 1996. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one. The final equation for classification can be represented as

$$F(x) = \text{sign}\left(\sum_{m=1}^M \theta_m f_m(x)\right),$$

where f_m stands for the m -th weak classifier and θ_m is the corresponding weight. It is exactly the weighted combination of M weak classifiers.

Adaboost retrains the algorithm iteratively by choosing the training set based on accuracy of previous training and the weightage of each trained classifier at any iteration depends on the accuracy achieved.

Implementation: After the preprocessing step and featuring engineering step, we choose Decision trees as a base model for the Adaboost Classifier. We scaled the data using Standard Scaler. Instead of using decision stumps, i.e. decision trees of unit height, we grew a tree

with 25 features and used entropy as a criteria for splitting. We passed this model to the Adaboost classifier with a 0.1 learning rate and trained the model for 1000 iterations.

- **Voting**

Voting is an ensemble approach where we train the model using diverse algorithms and then ensemble them to predict the final output. We can use previously trained base classifiers like, a Random Forest Classifier, SVM Classifier, Linear Regression etc. Later models are pitted against each other and selected upon best performance by voting using the Voting Classifier. This finds out the consensus among a group of base classifiers as to what is the label for a given observation. Voting does not work well when the base classifiers have bad training accuracy. There are two types of voting methods. Hard voting is where a model is selected from an ensemble to make the final prediction by a simple majority vote for accuracy. Soft Voting can only be done when all your classifiers can calculate probabilities for the outcomes. Soft voting arrives at the best result by averaging out the probabilities calculated by individual algorithms.

Implementation: On the dataset that we used, we combined 3 base classifiers which were having good accuracies. We performed hard voting on those 3 classifiers. The classifiers were Random Forest, ExtraTrees with Adaboosting and Extreme Gradient Boost.

- **Stacking**

Stacking is concerned with combining multiple classifiers generated by using different learning algorithms L_1, \dots, L_N on a single dataset S , which consists of examples $s_i = (x_i, y_i)$, i.e., pairs of feature vectors (x_i) and their classifications (y_i). In the first phase, a set of base-level classifiers C_1, C_2, \dots, C_N is generated, where $C_i = L_i(S)$. In the second phase, a meta-level classifier is learned that combines the outputs of the base-level classifiers to generate a training set for learning the meta-level classifier, a leave-one-out or a cross validation procedure is applied to the classifiers.

Implementation: On the dataset that we used, we stacked the classifiers that had previously given a high accuracy when they ran separately. The base-classifiers that were used were Extra Trees, Random Forest Classifier and Naive Bayes. The number of estimators in Extra Trees were 1000 and 800 for Random Forest which was optimized by using a Grid Search Function. The meta-classifier used was eXtreme Gradient Boosting which is widely used for ensemble classifications and is good at learning models.

EXPERIMENTS:

- **Naive bayes**

Feature engineering and feature addition increased the performance of the model and gave training accuracy of **90.3%** but Light GBM boosting was not a good choice because boosting algorithms reduce Bias and Variance while Naive Bayes doesn't

create variance as such. Finally giving a poorer performance than other models with a test accuracy of **70.3%**.

- **SVM**

Scaling was found to be very important for SVM. Without scaling, the model gave just 14% accuracy on training data, which increased to 73% after scaling. The SVM only weighs instances (support vectors), not features. Thus, normalizing the features properly is a big issue. Also, importance of feature engineering was clearly visible. With feature engineering, the model gave a training accuracy of **74.52%**, a 2.2% increase over the model without feature engineering and more stringent parameters.

- **MLP**

In artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly. Since the 7 classes were overlapping and not linearly separable, the hidden layers as mentioned above were used. A good neural network was thus formed which gave us **77.32%** accuracy. Since the MLP classifier did a better job than many other implementations, we can say with confidence that for overlapping classes/non-linearly separable classes, multi-layer perceptron must be implemented at least once.

- **Random forest**

After the Random forests were applied for the data without the given feature engineering, taking 500, 1000, 1500 as n times, we got around 76% of accuracy on

the test data. After the feature engineering, we got 79% of accuracy.

- **Extra trees**

Among all the tree based model's extra trees performed the best because of its extremely randomised approach to build trees which has an excellent variance reduction effect. The predictions were made on the given test data and was submitted to kaggle, which gave an accuracy score of **80.72%**.

- **Decision trees with Adaboost**

The predictions were made on the given test data and was submitted to kaggle, which gave an accuracy of **80.26%**.

- **Voting**

With Voting we got an accuracy 81.9%. We noticed that the accuracy of Extra Trees was not much different from that of the accuracy of Voting. By this we decided that Voting was not the most suitable model for predicting the classifier.

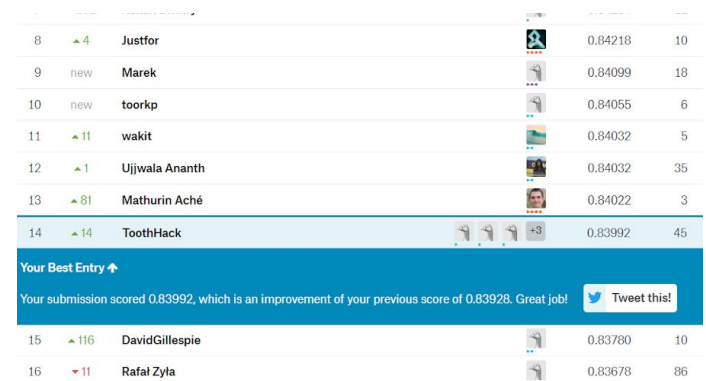
- **Stacking**

Through Stacking the classifiers using XGBoosting Classifier as the meta-classifier we had got a an accuracy score of 84% which was the highest among the list of classifiers used

RESULT:

Out of the **45 kernels** submitted **Stacking** proved to be the most effective of all models. Our final rank in the Kaggle competition was **14 out of 359 teams**.

| Model Used | Accuracy |
|----------------|---------------|
| Random Forests | 79% |
| Naive Bayes | 70.3% |
| SVM | 74.52% |
| MLP | 77.32% |
| Extra Trees | 80.72% |
| Decision Trees | 80.26% |
| Voting | 81.9% |
| Stacking | 83.99% |



The screenshot shows a Kaggle competition leaderboard. The top entries are:

| Rank | Change | Team | Score | Rank |
|------|--------|----------------|---------|------|
| 8 | ▲ 4 | Justfor | 0.84218 | 10 |
| 9 | new | Marek | 0.84099 | 18 |
| 10 | new | toorkp | 0.84055 | 6 |
| 11 | ▲ 11 | wakit | 0.84032 | 5 |
| 12 | ▲ 1 | Ujjwala Ananth | 0.84032 | 35 |
| 13 | ▲ 81 | Mathurin Aché | 0.84022 | 3 |
| 14 | ▲ 14 | ToothHack | 0.83992 | 45 |

Below the leaderboard, a blue banner reads: "Your Best Entry ↑ Your submission scored 0.83992, which is an improvement of your previous score of 0.83928. Great job! Tweet this!"

The bottom entries of the leaderboard are:

| | | | | |
|----|-------|----------------|---------|----|
| 15 | ▲ 116 | DavidGillespie | 0.83780 | 10 |
| 16 | ▼ 11 | Rafal Zyla | 0.83678 | 86 |

CONCLUSION:

The accuracy which we achieved through the different machine learning models were increased through the use of pre-processing steps like feature engineering. Using a random-forest feature importance algorithm we found out that many of the features that were created had a higher importance and thus contributed to a better prediction. We used a wide array of models from Naive Bayes which uses a probabilistic approach to

predicting the classifier to ensemble classifiers like Extra Trees and Random Forest which proved to have a much better accuracy when we combined it with a higher number of estimators. Extratrees worked better than random forest because of its extremely randomised approach to build trees which has an excellent variance reduction effect. After doing a bunch of base classifiers we moved on to Stacking the classifiers using a meta classifier to further improve our accuracy by a few points which used 5 classifiers among which it the labels were classified. The Stacking of classifiers

proved to be the most effective with the highest accuracy score which we received. Further improvements could have been done by selecting important features using Principal Component Analysis and Linear Discriminant Analysis, although on previous trials feature selections gave us lesser accurate results than when all the attributes were used for prediction.