Exploration and automated classification of forest cover from cartographic data using machine learning

Abel Kahsay Gebreslassie (Kaggle username: Abel Kahsay)

Introduction

Forest cover type dataset was originally collected by Remote Sensing and GIS Program at Colorado State University. Data collection was conducted in Roosevelt National forest of northern Colorado where twelve different measurements were taken along with the forest cover type. Ten of the measurements are quantitative while the remaining two are qualitative, Wilderness_Area (four different types) and Soil_Type (forty different types). However, the qualitative attributes are one hot encoded, as that is desirable for machine learning algorithms, resulting in a total of 54 columns ('attributes') for a every entry in the dataset.

The owners of the dataset have used machine learning algorithms for automated predictions of cover type from the remaining measurements and their best classification accuracy was 70.58% using a neural network. This document is a report on the exploration and use of different machine learning algorithms classification for a Kaggle competition organized by Pázmány Péter Catholic University in the year 2020 on a subset of the forest cover type dataset.

Exploratory Data Analysis

First things that need to be checked before exploring the data in-depth are sanity checks, that is making sure data types match the expected ones, there are no missing values and so on.

i. Attributes (columns) check: Printing the length and names of the columns gave the expected number and names.
ii. Data types and null values: Those were checked with the help of Pandas info function. For all the 406708 entries of the dataset none of the values in any of the columns are null. As for data types, all columns of type 'int64' which is the expected data type for quantitative(numeric) and Boolean represented with binary values. Further checking is required for the Boolean values to make sure they are composed of only the values zero and one. Checking the columns thar are consist of only zero or one yields the expected 44 Boolean (qualitative) columns. Hence, no preprocessing needs to be done regarding data types and null values.

The next step was checking if there is class imbalance between the cover types in the dataset. Actual class counts, percentages and count plot for visualization were utilized for this purpose. About of 85% of the forests are of cover type 'Lower montane dry' or 'lower montane' and this indicates a huge class imbalance as the remaining 5 cover types make up only 15% of the dataset. As a result, it is necessary to consider the class imbalance when training the machine learning algorithms.

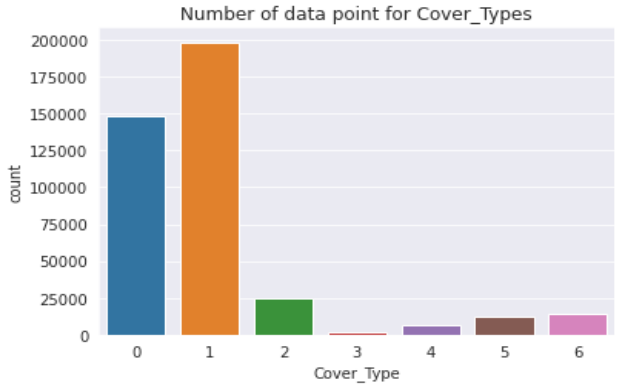

Figure 1Number of data points for Cover_Types

In the dataset we have both quantitative (numeric) and qualitative (categorical, Boolean in this case) attributes that need to be explored separately for analysis purposes.

A. Quantitative(categorical) attributes

Counts of instance having each attribute, count plot and count plots with corresponding classes were used to gain insight about the attributes. Most of the forests in the dataset have Rawah(45%) or Comanche(43.5%) wilderness area and the remaining, about 11.5%, have Neota or Cache wilderness area.

The most common soil type, relatively speaking, are Soil types 29, 23, 32 and 33 which are found in 46% of the forests. Some of the soil types such as soil type 7, 36, 8 and 37 can be rarely be found. Soil type 15 is the rarest type and is only present in two of the forests.

Note:

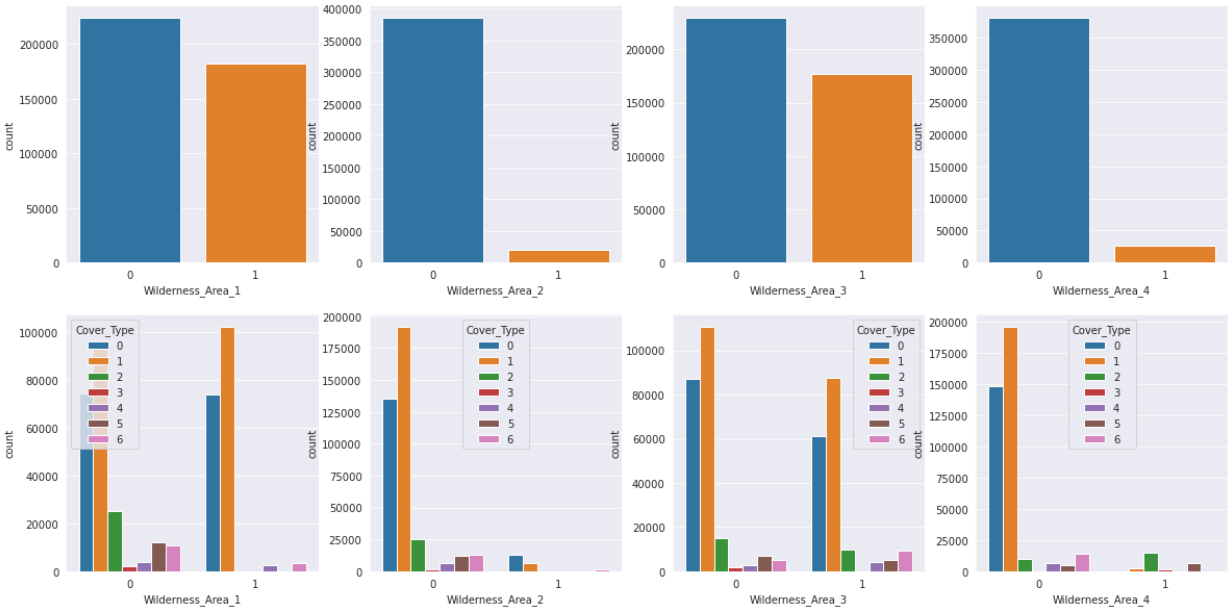

Figure 2count plot for wilderness type

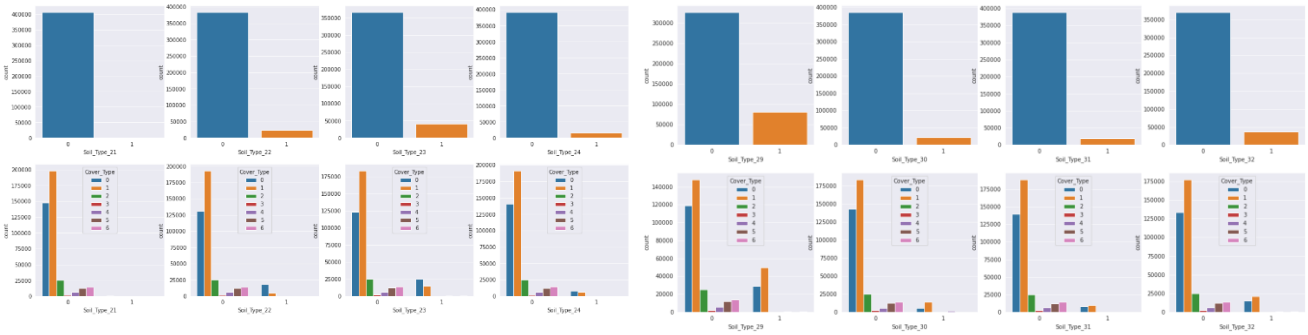

Figure 3 some of the count plot for soil types

Note: When there are many attributes the complete plots can be found in the notebook and only sample plots of interest are shown in this document.

B. Qualitative (numeric) attributes

Numeric attributes can be characterized by their statistical parameters and visualization with box and distribution plots.

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways
count	406708.000000	406708.000000	406708.000000	406708.000000	406708.000000	406708.000000
mean	2959.329730	155.763309	14.103413	269.485181	46.467994	2349.097488
std	280.028028	111.925462	7.490411	212.462577	58.327213	1559.559608
min	1859.000000	0.000000	0.000000	0.000000	-173.000000	0.000000
25%	2809.000000	58.000000	9.000000	108.000000	7.000000	1103.000000
50%	2995.000000	127.000000	13.000000	218.000000	30.000000	1994.000000
75%	3163.000000	261.000000	18.000000	390.000000	69.000000	3328.000000
max	3858.000000	360.000000	66.000000	1397.000000	599.000000	7117.000000

Figure 4Description of some numerical attributes

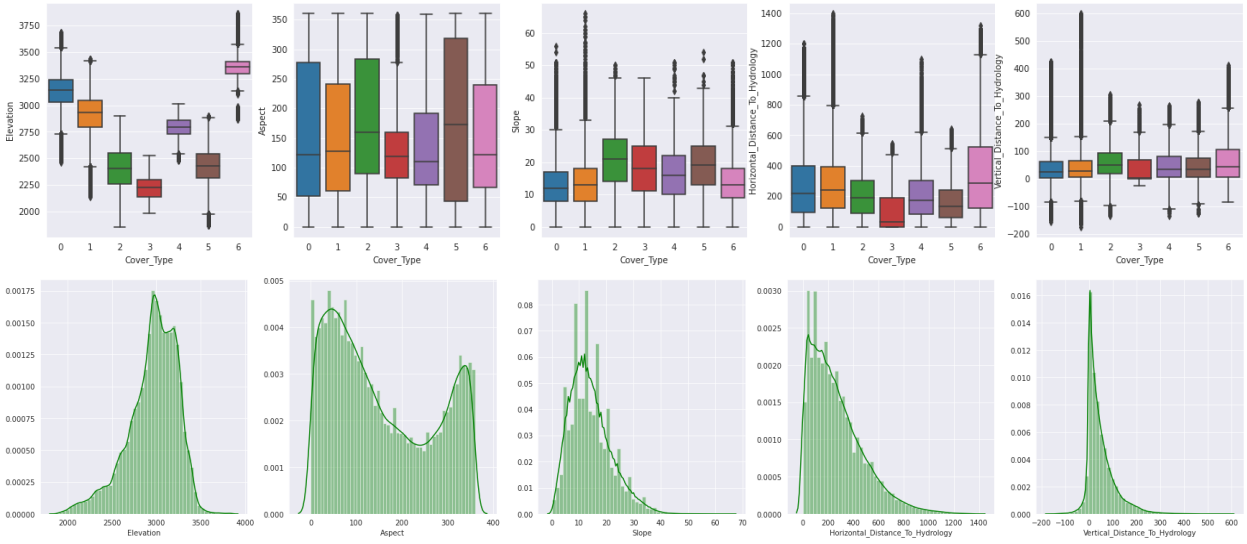

Figure 5some box and distribution plots

Most of the attributes are skewed to one side while the aspect attribute resembles the combination of two gaussian distributions. Outliers can be observed both in the boxplots and distribution plots and relatively speaking the slope attribute has a noisier distribution. As can be seen in the box plot range distribution for the numeric attributes varies. Furthermore, some have smaller variance, and some have larger variance with outliers. This needs to be addressed before feeding data to the machine learning algorithms, which will be discussed later.

In addition, exploring correlation between numeric attributes will gives more insight on how the attributes are relate to each other. After computing correlation values and identifying attributes that have stronger correlation pair plots were drawn to extract more intuitive patterns of correlation.

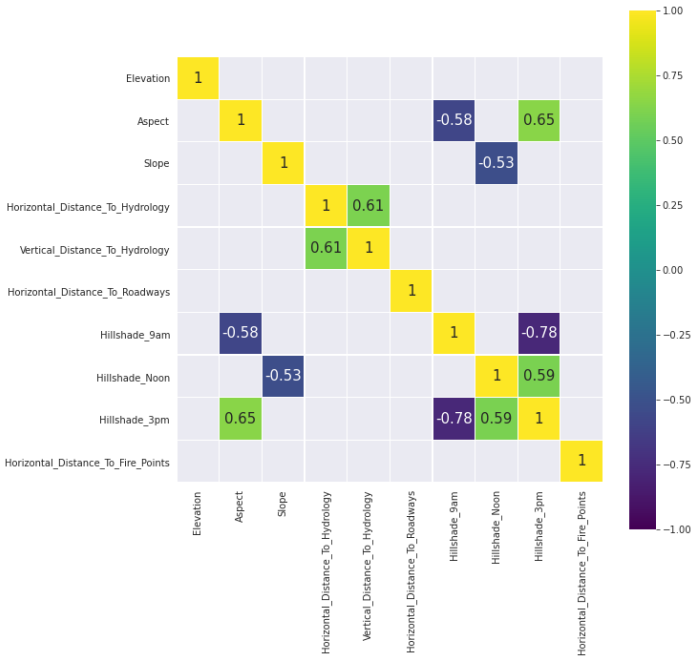

Figure 6Heatmap for attributes with correlation larger than 0.5

Aspect has a stronger positive correlation with hill-shade at 3 pm and a negative correlation with hill-shade at 9 am. That is, a larger aspect indicates a larger hill-shade at 3 pm and smaller hill-shade at 9 am.

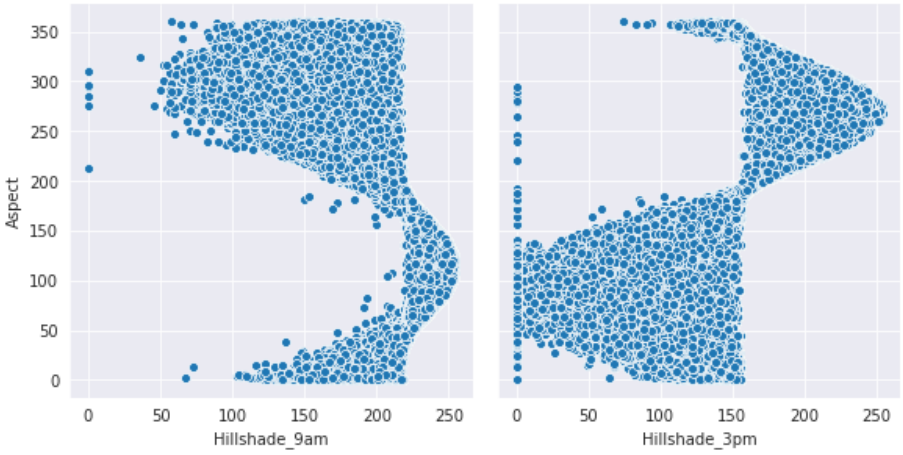

Figure 7Aspect pair plots to visualize correlation

Similarly, slope and hill-shade at noon as well as horizontal and vertical distance to hydrology have stronger negative correlations.

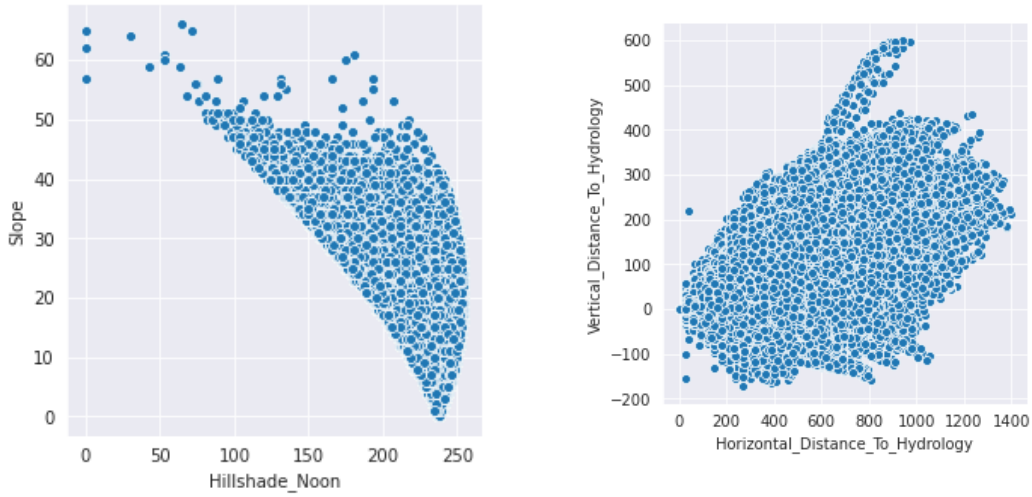

Figure 8Pair plots for slope, hillshade at noon and distances to hydrology

Hill shade at noon and 3 pm have a positive correlation and they both have a negative correlation with hill shade at 9 am. This is correlation is expected as hill shades take the sun's position into account and the sun is at opposite directions at 9 am and 3 pm.

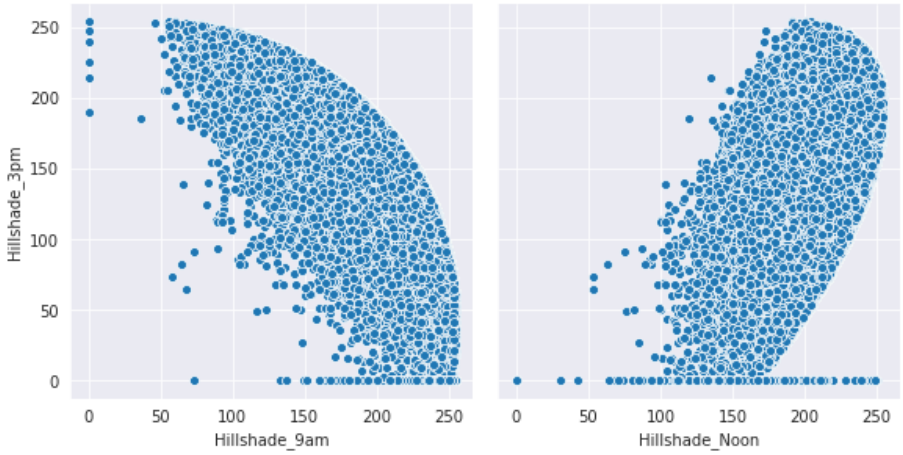

Figure 9hill shade correlations

Feature Engineering

Feature engineering is regarded as the art of identifying features that are important for better performance and introducing new ones that are derivatives or manipulated versions of the existing ones. Introducing new features that are combinations of given ones is more likely to give better performance. In case they end up being redundant most algorithms are able to discriminate and ignore those in the training process. In terms of computational cost those are a few extra columns and won't incur a significant computational cost. Hence we're more likely to gain by adding those new features. Idea for new features: Introduce sum and difference of distances that are in the same direction, i.e. horizontal ones with horizontal ones and vertical ones with vertical ones. In case of distance to hydrology use Pythagorean theorem to calculate as both the vertical and horizontal distance are available.

i.e $D_{hydrology} = \sqrt{VD^2_{hydrology} + HD^2_{hydrology}}$

Data splitting and feature scaling

To train and evaluate different machine learning algorithms the data was split with 60%-30%-10% train-test-validation split. From exploratory data analysis it can be observed the numerical attributes have different ranges, mean and variance. This is almost always the case and larger weight will be given to the features that are at a larger scale and have larger variance. Hence, scaling features (often to zero mean and variance of one) is necessary for meaningful and better performance.

For scaling features robust scaler from scikit learn was used which is robust to outliers, hence the name robust scaler. Scikit learn has a standard scaler that returns features with zero mean and unit variance but isn't robust to outliers as outlies can significantly affect the mean and variance. In the case of robust scaler, it uses the median and inqaurtile range to produce features that are in the range between the first and third quartiles of the the given feature. This insures robustness of the scaling to outliers. [1]

The numeric feature have a more standardized statistical parameters after scaling.

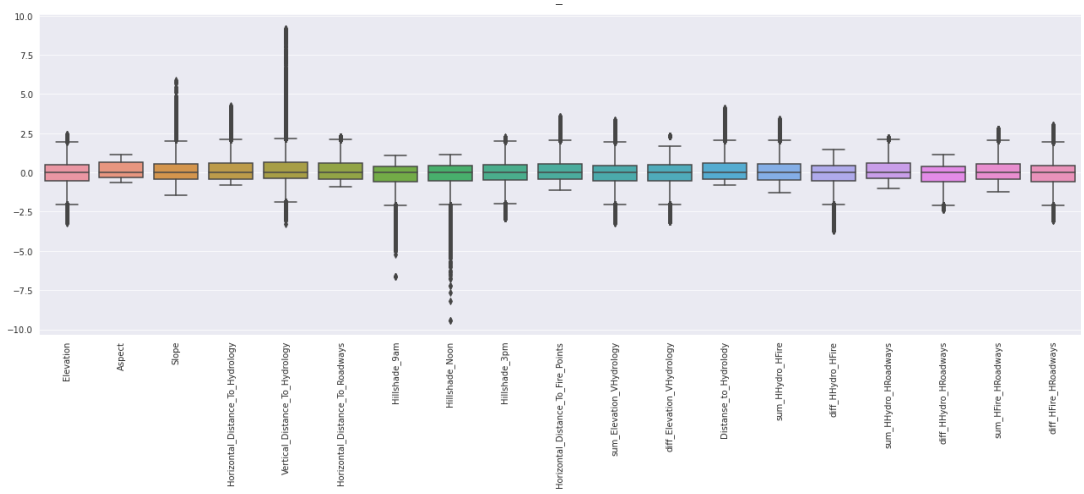

Figure 10Scaled numeric features

Training, Parameter tuning and evaluation

A. Machine learning algorithms utilized

Algorithms that were utilized for the classification task are logistic regression, decision tree, random forests, extra trees, xgboost, boosting and bagging of logistic regression, stacking and deep neural networks (DNN). Most of the algorithms used are variants of decision tree with randomized bagging and boosting. The main reason for choosing those is the fact that they have better performance in practice and tend to have less overfitting issues compared to state of art algorithms such as DNNs. Training a support vector machine classifier took longer than expected and was interrupted after training for more than half an hour.

i. Logistic regression classifier is a simple classifier that makes use of cross entropy loss (for multiple classes categorical cross entropy) and sigmoid or softmax(which is an extension of sigmoid to multiple classes) activation to get probabilities for every class. It results in a smaller

sized and computationally less expensive model. Better performance can be achieved by boosting or bugging multiple logistic regression classifiers.

ii. Random forest classifier is a randomized bagging model that uses CART as its base algorithm. It trains multiple small CART trees, with no pruning, on a randomly selected set of features on a random sample drawn from the dataset (sampling with replacement) and those small trees are bagged together to give a better performance. It evaluates the feature that gives the most discriminative results when making a split at every level. [2]
Random forests, and randomized bagging trees in general, have less overfitting issues in practice and that's the main reason for evaluating them on forest cover type dataset.

iii. Extra trees classifier is also a randomized bagging model similar to random forests. However, instead of evaluating all features for maximum discriminative results when splitting it randomizes the feature selection for splitting. This is why the original title of the paper is extremely randomized trees, both features for a tree and splitting feature selection are randomized. [3]

iv. Xgboost is a boosted ensemble of decision trees that is faster to train and optimized to take advantage of parallel computing capability, can be used with GPUs as well. It popular for it's superior performance in competitions especially on structured datasets.

First decision tree, random forest, extra trees and logistic regression classifier were trained on original features only and all features, including derived features. The models then were evaluated in the validation set and execution time (training + prediction) was recorded.

Model	Original features only		Original + engineered(derived) features	
	Val acc	Time (sec)	Val acc	Time (sec)
Decision tree	92.32%	6.6	93.90%	13
Extra trees	94.20%	77.7	96.97%	75.2
Random forest	94.59%	85.5	96.90%	108.4
Logistic regression	72.13%	17.0	72.19%	17.9

Table 1 Accuracy of different algorithms in validation set and time taken for training and prediction

Derived features resulted in better performance, up to 2.77% accuracy improvement in extra trees, and a minimum time overhead, about 23 seconds in case of random forests. Therefore, introducing new features results in better performance with minimal overhead if features are chosen wisely.

B. Fine tuning

Fine tuning a single parameter can easily be done with a loop and a list of possible values for that parameter. But when trying to fine tune multiple parameters at the same time scikit-learn's grid search comes handy instead of using multiple nested for loops. Grid search with cross validation was carried out for decision tree and xgboost. For xgboost classifier conducting a grid search for parameters is better than training it with the default ones or random tweaking as xgboost has many parameters that can be tuned. The best parameters were then used to train the models and as the base model for bagged and boosted decision tree. In addition, the best parameters for decision tree were used for random forest and extra trees classifiers.

From the best grid search parameters, we can see that training with equal weight for all classes results in better performance than weighing for class imbalance. Grid search for decision takes about 7 minutes for decision and 5 hours 30 minutes for xgboost.

Parameters with best performance for decision tree are the default ones with the exception of criterion, which has a default value of gini but entropy results in better accuracy.

Disclaimer: Original parameter tuning for xgboost took 5 hours and 30 minutes but as kaggle clears cell outputs on restart and I had less time to re-run the cell to have a demo output so I had to reduced the parameters and took about 15 minutes. However, when training the actual model I used the best parameters from the first tuning, the one that took more than 5 hours. The parameter dictionary was

```
parameters = {
        "eta"    : [0.10, 0.20, 0.3, 0.5] ,
        "max_depth"        : [10, 20],
        "gamma"          : [0.0, 0.1 , 0.4],
        "subsample"        : [0.5, 1]
```

} and the best parameters were eta=0.2, max_depth=20, gamma=0.1, subsample=1 (default value).

Xgboost classifier was trained on training + validation data and had an accuracy of 97.41% on test data. It took 5006 seconds (83 minutes) to train.

C. Ensembles

Bagged and boosted decision tree and logistic regression, a stacking of extra trees classifier and bagged decision trees with logistic regression at level-1 of the stacking were trained and evaluated on the dataset.

Model	Base learner	Val acc	Time (sec)
Boosting	Decision tree	93.93%	13.3
	Logistic regression	49.56%	333.4
Bagging	Decision tree	96.78%	259.9
	Logistic regression	72.15%	518.0
Stacking	Level-0 [Bagged decision tree + Extra trees classifier] Level-1 [Logistic regression]	97.15%	1459.5

Boosting improves performance by very small margin for decision tree and degrades performance significantly when used with logistic regression. Bagging on the other hand improves the performance of decision trees, about 3% accuracy increase. In terms of time, boosting training and validation requires a little more time compared to training the base decision tree model while bagging results in a significant time overhead. This is due to the use of sampling with replacement in bagging. Both bagging and boosting take a long time to train with no performance gain, performance loss in the case of boosting, for logistic regression.

The stacking classifier improves accuracy by about 0.2% and takes 26 minutes to train and predict.

D. Deep Neural Network (DNN)

The main challenge with training DNNs is having too many parameters to tune, network structure and learning rate being the most important ones, and overfitting. Increasing or dropping the number 'slowly' is results in better performance while dropping number of neurons significantly from a layer to the next one degrades performance. It is common practice to use powers of two for number of neurons in a layer. A

DNN with 4 hidden layers that consist of 64, 128, 64 and 32 neurons and 54 input and 7 output neurons was trained on the forest cover type dataset. To reduce overfitting 10% of the training data was used for validation while training the neural network. Number of epochs was originally set to 300 but training was interrupted at 99th epoch because change in training loss was getting smaller and smaller. Training loss is optimistic and prone to overfitting but when the rate of change stays small for longer then further improvement is low. Validation accuracy of the DNN was 93.76% and took 982.5 seconds to train and predict.

E. Performance

Confusion matrix and accuracy reports for validation set were generated with scikit-learn for the trained models. Most classifiers, with the exception of logistic regression, have high precision and recall that is comparable to their accuracy. Finally, test set (30% of the original data, not the provided test set) performance was evaluated and predictions were generated for the actual test set and uploaded.

Model	Test accuracy	Actual prediction accuracy	Test + prediction time (sec)
Decision tree	93.78%	93.97%	1.63
Extra trees	97.02%	97.04%	7.4
Random forest	96.92%	96.92%	5.17
Logistic regression	72.61%	72.39%	0.65
Boosted decision tree	93.8%	93.90%	0.75
Boosted logistic regression	49.25%	49.10%	11.72
Bagging decision tree	96.73%	96.73%	3.42
Bagging logistic regression	72.65%	72.40%	3.31
Stacking(bagging decision tree + extra trees)	97.13%	97.17%	7.0
Xgboost	97.38%	97.446%	77.6
Multi layer perceptron(DNN)	93.55%	93.55%	3.77

Conclusion

Generally, from the experiments and predictions carried out we can say knowledge was extracted from the dataset with exploratory data analysis, adding derived features, if done carefully, results in better performance and fine tuning was carried out to search for best parameters.

Finally, from my exploration the best algorithm for the predictions task for forest cover type was Xgboost, proving itself once more. 😊

References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html, last accessed 06/12/2020.

[2] L. Breiman, "Random Forest," University of California, Berkeley, January 2001.

[3] P. Geurts, D. Ernst and L. Wehenkel, Extremely randomized trees, Springer Science + Business Media, 2006.