# Data Mining and Machine Learning Assignment

## Forest Cover Type Prediction

## 2020

Rivnyák Tímea

CPX2X1

# 1 Exploratory Data Analysis and Data Pre-processing

## 1.1 Information

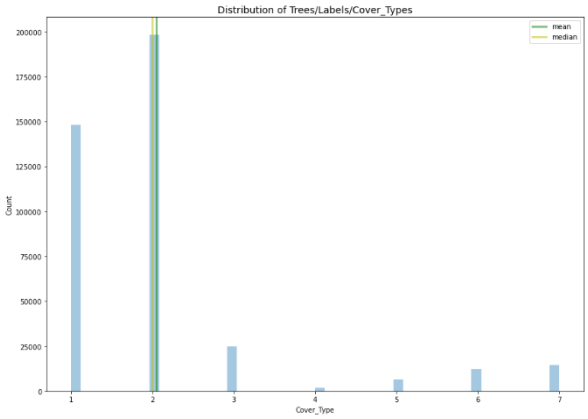
The dataset includes 7 forest cover types located in the four wilderness areas of the Roosevelt National Forest of northern Colorado. The dataset consists of cartographic independent variables from twelve measures with the determined actual forest cover types as dependent variables. The database was introduced by Blackard et al. and two techniques, namely a neural network model and a linear discriminant analysis, were used to predict the cover types.

## 1.2 Description of the data

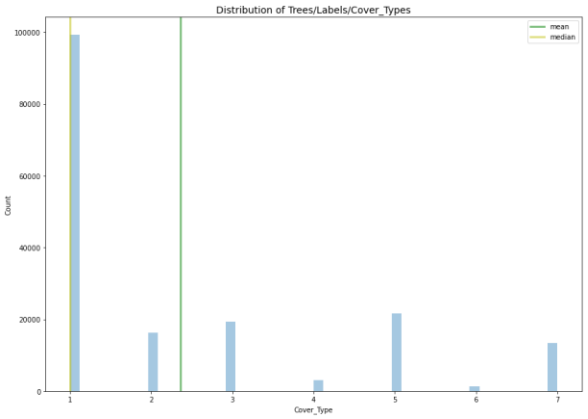
The number of observations is 581.012 from which the training data contains 406.708 instances while the test dataset contains the rest, 174.304 instances. The number of attributes in the set of features is 55 with the *"id"* column out of 12 measures. There are 10 quantitative variables and from two qualitative variables 4 binary wilderness areas and 40 binary soil type variables were created. In the set of labels, there are 2 attributes also with the *"id"* column. From the summary of the dataframe (*dataframe.info()*) it can be seen that there are no null values and every column has a numeric data type (int64), which means that the data is well-formatted and clean, there are no missing values. The *"id"* columns were removed because they are not relevant in the prediction. The descriptive statistics of the dataset (*dataframe.describe()*) has shown that there are some negative values in the *"Vertical_Distance_To_Hydrology"* column, there are no constant columns, the scales are not the same for all attributes and the *"Wilderness_area_"* and *"Soil_type_"* columns mostly contains zeros. Checking the sum of the soil type and wilderness area columns, which are binary columns with 0 and 1 values, both of them gave back the number of instances, hence they are one-hot-encoded. This means that a tree can only belong to one soil type and one wilderness area.

## 1.3 Feature selection

Further examining of the *"Vertical_Distance_To_Hydrology"* column, it measures the vertical distance to nearest surface water features, thus negative values indicate that the nearest surface water is below the examined data point, so it is meaningful. Moreover, for determining the outliers I used the extreme outliers method. The extreme outliers are any data values which lie more than 3 times the interquartile range below the first quartile or above the third quartile. The detected outliers in the one-hot-encoded columns does not count because their values are always 0 or 1, the *"Hillshade_"* columns also have a fixed range from 0 to 255. The highest number of outliers identified in the *"Vertical_Distance_To_Hydrology"* column, not just in the training data but in the test data too, therefore I will not eliminate this column. The distribution of the *"Cover_Type"* values shows an imbalanced distribution of labels in both the training and the test data, but every forest cover type has over a thousand representatives.

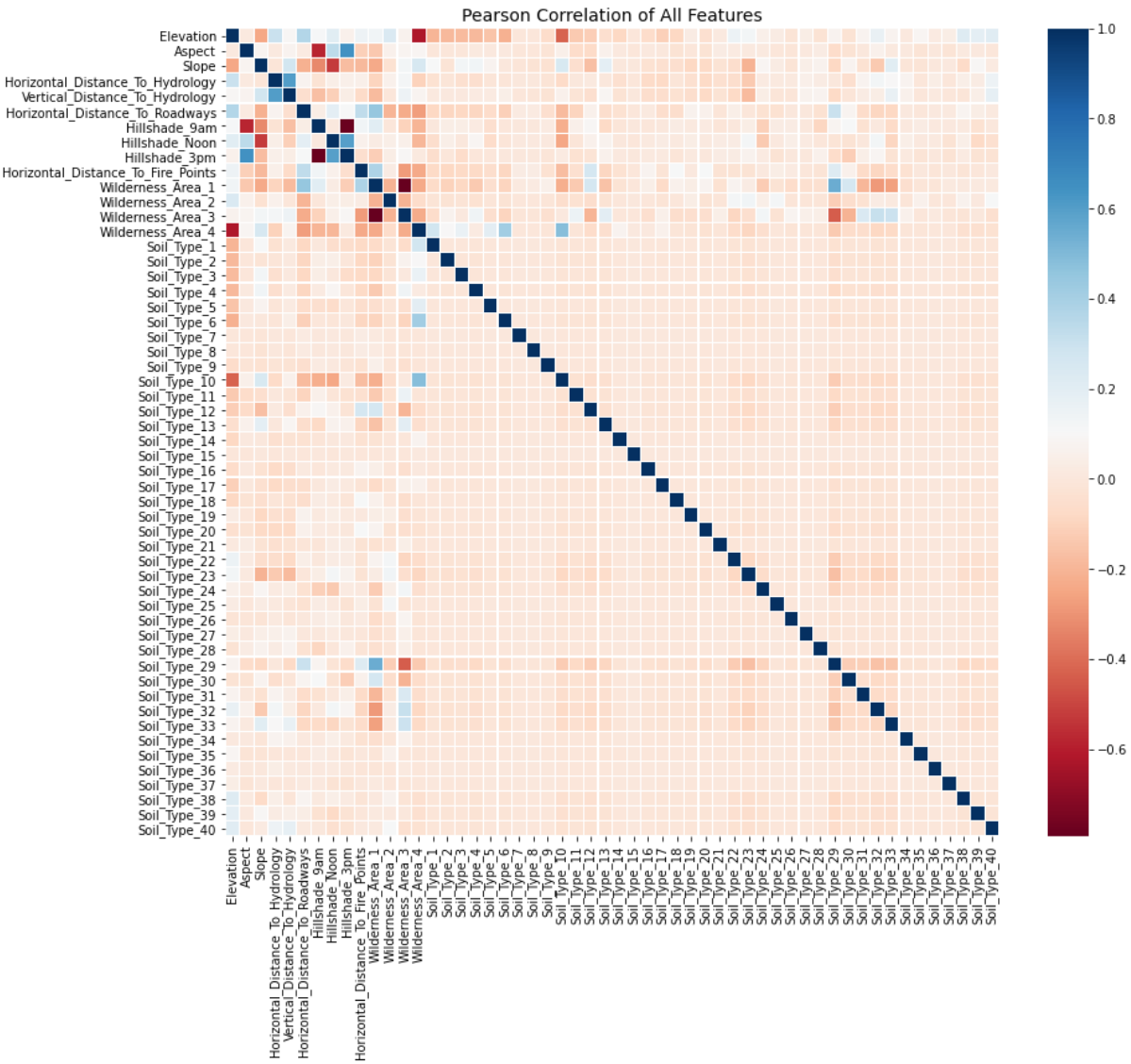



*1. Distribution of „Cover_type" values in the training set*




*2. Distribution of „Cover_type" values in the test set*

Computing the correlation with Pearson method, the resulting matrix shows strong negative correlation between the *"Hillshade_9am"* and the *"Hillshade_3pm"* features, which is logical, since the morning shade and afternoon shade are correlated, thus I dropped the *"Hillshade_9am"* column.




*3. Pearson correlation of all features in the training set*

# 2 Algorithms

Since features and target are available in the datasets and there are seven unrelated categories, the seven forest cover types, therefore it is a multi-class classification supervised machine learning problem. First, I tried the models on the training set splitted this way:

```
x_train, x_valid, y_train, y_valid = train_test_split(X, y, test_size = 0.30, random_state = 10, stratify=y['Cover_Type'])
```

I chose stratified sampling because some cover types have too few representatives.

## 2.1 Tree-based ensemble algorithms

I have chosen them because they are non-parametric so the data do not have to follow a particular distribution, therefore I will not transform the data to be normally distributed, since I have an imbalanced dataset. It is also robust against overfitting and scaling the data is not necessary. Moreover, they are computationally inexpensive, so they are faster and outperforms their weak learners.

### 2.1.1 AdaBoost Classifier

I have considered using AdaBoost, but probably it was very sensitive to the noise and the outliers, because its classification report got this result, also two class was never predicted:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.44 | 0.54 | 0.48 | 44487 |
| 2 | 0.76 | 0.29 | 0.42 | 59493 |
| 3 | 0.38 | 0.23 | 0.29 | 7508 |
| 4 | 0.00 | 0.00 | 0.00 | 577 |
| 5 | 0.00 | 0.00 | 0.00 | 1994 |
| 6 | 0.20 | 0.76 | 0.31 | 3647 |
| 7 | 0.12 | 0.74 | 0.21 | 4307 |
| accuracy | | | 0.40 | 122013 |
| macro avg | 0.27 | 0.37 | 0.25 | 122013 |
| weighted avg | 0.56 | 0.40 | 0.42 | 122013 |

*4. Classification report of AdaBoostClassifier*

### 2.1.2 XGBoost Classifier

XGBoost is faster and usually outperforms other algorithms, however extreme values lead to underfitting the model. Using the default parameters, the classification report looked like this:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.73 | 0.73 | 0.73 | 44487 |
| 2 | 0.76 | 0.82 | 0.79 | 59493 |
| 3 | 0.68 | 0.85 | 0.75 | 7508 |
| 4 | 0.78 | 0.55 | 0.65 | 577 |
| 5 | 0.76 | 0.10 | 0.18 | 1994 |
| 6 | 0.54 | 0.11 | 0.18 | 3647 |
| 7 | 0.84 | 0.54 | 0.66 | 4307 |
| accuracy | | | 0.74 | 122013 |
| macro avg | 0.73 | 0.53 | 0.56 | 122013 |
| weighted avg | 0.74 | 0.74 | 0.73 | 122013 |

*5. Classification report of XGBoostClassifier*

### 2.1.3 Random Forest Classifier

Since boosting algorithms was not performed as I envisioned, I tried a method, which is using bagging. The random forest algorithm is relatively robust to noise and outliers, and since I left the outliers in the dataset, I expected better performance, which I got eventually.

0.9474892019702819

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.96 | 0.93 | 0.95 | 44487 |
| 2 | 0.94 | 0.97 | 0.96 | 59493 |
| 3 | 0.94 | 0.95 | 0.94 | 7508 |
| 4 | 0.92 | 0.84 | 0.88 | 577 |
| 5 | 0.93 | 0.74 | 0.83 | 1994 |
| 6 | 0.92 | 0.89 | 0.90 | 3647 |
| 7 | 0.97 | 0.94 | 0.95 | 4307 |
| accuracy | | | 0.95 | 122013 |
| macro avg | 0.94 | 0.90 | 0.92 | 122013 |
| weighted avg | 0.95 | 0.95 | 0.95 | 122013 |

*6. Classification report of RandomForestClassifier*

## 2.2 K-nearest neighbours algorithm

This algorithm assumes that similar data points are near to each other and this is also a non-parametric method. It is a simple algorithm and can be used for classification. It is also a little bit slow as there are a lot of instances and features, however its prediction performance is the best up to this point.

0.9556276790178095

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.96 | 0.95 | 0.96 | 44487 |
| 2 | 0.96 | 0.97 | 0.96 | 59493 |
| 3 | 0.94 | 0.96 | 0.95 | 7508 |
| 4 | 0.89 | 0.73 | 0.80 | 577 |
| 5 | 0.89 | 0.85 | 0.87 | 1994 |
| 6 | 0.92 | 0.90 | 0.91 | 3647 |
| 7 | 0.97 | 0.95 | 0.96 | 4307 |
| accuracy | | | 0.96 | 122013 |
| macro avg | 0.93 | 0.90 | 0.92 | 122013 |
| weighted avg | 0.96 | 0.96 | 0.96 | 122013 |

*7. Classification report of KNeighborsClassifier*

# 3 Hyperparameter Tuning

I decided to tune the parameters for the two best performed algorithms.

## 3.1 Random Forest Classifier

The n_estimators parameter is the number of trees in the forest, for which the more the merrier is true, yet with increased number of trees the computation will take longer. Trying the parameters for the max_features, min_samples_split and min_samples_leaf I found that the default values are the best. With tuning the RandomForestClassifier it turned out that the best hyperparameters are these:

```
Best Hyper Parameters:
 {'criterion': 'entropy', 'n_estimators': 500, 'n_jobs': -1, 'random_state': 42}
0.9510789833870161
              precision    recall  f1-score   support

           1       0.96      0.94      0.95     44487
           2       0.95      0.97      0.96     59493
           3       0.94      0.96      0.95      7508
           4       0.91      0.85      0.88       577
           5       0.93      0.76      0.84      1994
           6       0.93      0.89      0.91      3647
           7       0.97      0.95      0.96      4307

    accuracy                           0.95    122013
   macro avg       0.94      0.90      0.92    122013
weighted avg       0.95      0.95      0.95    122013
```

*8. Best hyperparameters and classification report of RandomForestClassifier*

It can be also seen that with parameter tuning the performance of the classifier increased, however the k-nearest neighbour algorithm still outperforms it.

## 3.2 K-nearest neighbours algorithm

For this algorithm, the n_neighbors parameter says that the k closest data will be searched for which the optimal is 3 for this dataset. I found that the weight function ‘distance’ is better than the default, in which case, closer neighbours of a query point will have a greater influence than neighbours which are further away. Using Manhattan distance performed better than Euclidean and the leaf_size was set to 7, so within 7 leaf nodes it will switch to brute force. The others stayed at default parameters. With the chosen hyperparameters the method outperformed all of the other models.

```
Best Hyper Parameters:
 {'algorithm': 'auto', 'leaf_size': 7, 'n_jobs': -1, 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Accuracy: 0.9621433781646218
              precision    recall  f1-score   support

           1       0.96      0.96      0.96     44487
           2       0.97      0.97      0.97     59493
           3       0.95      0.96      0.96      7508
           4       0.90      0.83      0.86       577
           5       0.89      0.87      0.88      1994
           6       0.93      0.92      0.93      3647
           7       0.96      0.96      0.96      4307

    accuracy                           0.96    122013
   macro avg       0.94      0.92      0.93    122013
weighted avg       0.96      0.96      0.96    122013
```

*8. Best hyperparameters and classification report of KneighborstClassifier*

### 3.3 Voting Classifier

I also tried voting classifier with the best models, however it did not perform better than the best model.

## 4 Conclusion

Since on the splitted training set the k-nearest neighbour algorithm with the hyperparameter tuning outperformed every other model, I chose to fit the whole training dataset on it and make a prediction with the test dataset. This model performed with the score of 0.96826 on the Kaggle competition.

## 5 References


https://scikit-learn.org/stable/index.html

https://xgboost.readthedocs.io/en/latest/

https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f

https://towardsdatascience.com/ensemble-learning-using-scikit-learn-85c4531ff86a

https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761