Machine Learning Tree Based Classification Documentation

Predicting Forest Cover Type Using Machine Learning

MBD S2 - Machine Learning II

Prepared by Group F

Aditya Gavali Sunil
Alexandra Maria Mathay
Celia Assmuth Oreja
Eric von Stockar
Fernando Suárez Pinto
Mikael Dzhaneryan
Paul Ribes

TABLE OF CONTENTS

Introduction and Goal

The goal of this machine learning project is to build a model that is able to **predict the forest cover type**, which is the predominant type of tree cover, from cartographic variables.

There are seven different cover types, namely, 1 = Spruce/Fir, 2 = Lodgepole Pine, 3 = Ponderosa Pine, 4 = Cottonwood/Willow, 5 = Aspen, and 6 = Douglas-fir & 7 =Krummholz.

In this project we will go through the machine learning process of **understanding the data**, followed by **data cleaning and preparation**, **feature engineering**, **model selection**, **evaluation,** then **optimization**, to find the best model to predict the classes.

Understanding the Data

To understand the dataset we went through two main steps, **looking at the contents of the data**, and **doing an initial analysis** of the dataset.

Looking at the data

The dataset consists of 15,120 observations, with no null values, from the 4 wilderness areas in the Roosevelt National Forest in Northern Colorado, and 56 columns containing the following features:

- **Elevation** - Elevation in meters
- **Aspect** - Aspect in degrees azimuth
- **Slope** - Slope in degrees
- **Horizontal_Distance_To_Hydrology** - Horz Dist to nearest surface water features
- **Vertical_Distance_To_Hydrology** - Vert Dist to nearest surface water features
- **Horizontal_Distance_To_Roadways** - Horz Dist to nearest roadway
- **Hillshade_9am (0 to 255 index)** - Hillshade index at 9am, summer solstice
- **Hillshade_Noon (0 to 255 index**) - Hillshade index at noon, summer solstice
- **Hillshade_3pm (0 to 255 index)** - Hillshade index at 3pm, summer solstice
- **Horizontal_Distance_To_Fire_Points** - Horz Dist to nearest wildfire ignition points
- **Wilderness_Area (4 binary columns, 0 = absence or 1 = presence)** - Wilderness area designation
- **Soil_Type (40 binary columns, 0 = absence or 1 = presence)** - Soil Type designation
- **Cover_Type (7 types, integers 1 to 7)** - Forest Cover Type designation

```
In [3]: dataframe = pd.read_csv('train.csv')
        df = dataframe.copy()
        df
```

Out[3]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade
0	1	2596	51	3	258	0	510	
1	2	2590	56	2	212	-6	390	
2	3	2804	139	9	268	65	3180	
3	4	2785	155	18	242	118	3090	
4	5	2595	45	2	153	-1	391	
...	...	...	...	...	...	...	...	
15115	15116	2607	243	23	258	7	660	
15116	15117	2603	121	19	633	195	618	
15117	15118	2492	134	25	365	117	335	
15118	15119	2487	167	28	218	101	242	
15119	15120	2475	197	34	319	78	270	

15120 rows × 56 columns

Fig 1: Reading the CSV file

Initial Analysis

For the initial analysis, the team looked at **missing values**, the **distribution**, and **descriptive statistics**.

Regarding the distribution of the features and the target variable there are some interesting insights. Features are **mostly categorical** (all soil types and area wilderness types). From the 56 variables only 10 of them are continuous variables and the other 46 are categorical. There are some Soil types like 'Soil_Type7' and 'Soil_Type15', which only contain one type of value. This might be interesting to keep in mind for the data cleaning/feature engineering part as it might be a good idea to drop them if they do not add any explainability to the model and so reduce the noise of the dataset. The team also saw that **many of the features are skewed and not normally distributed**. Lastly, the target variable 'Cover Type' is evenly distributed, therefore we can say that we have a **balanced dataset**.

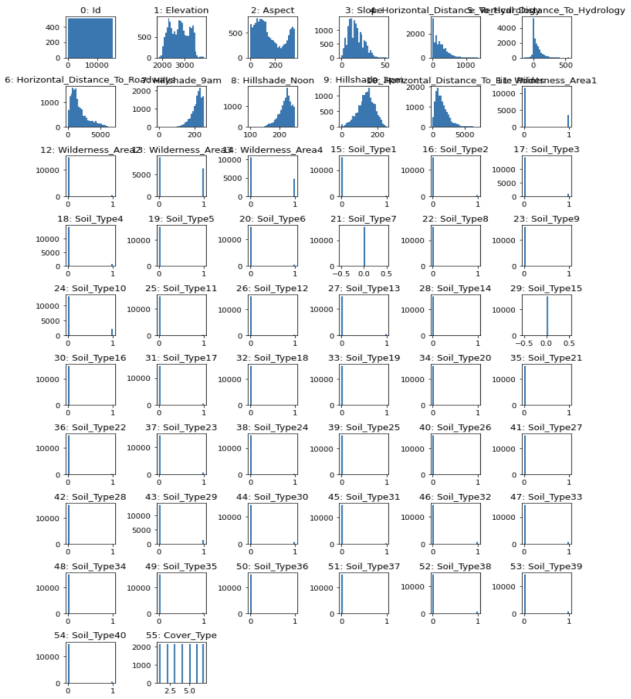

Figure 2: Variables distribution

The team also used Pandas Profiling to look more into the data. Some important things to note are the statistics of the data, which shows that there are **no missing values**, and **no duplicate rows.**

Dataset statistics

Number of variables	56
Number of observations	15120
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	6.5 MiB
Average record size in memory	448.0 B

Variable types

Numeric	12
Categorical	44

The team also observed that the features are mostly **highly correlated** with each other, for example, elevation and slope are negatively correlated with each other, and elevation and hillshade are positively correlated with each other. These correlations will be taken into consideration in the feature engineering part.

Data Cleaning and Preparation

Data cleaning and preparation was done in three steps, **splitting the data**, **creating a baseline model, analysing outliers**, and then **analysing the scale**.

Splitting the data

First step to data preparation and cleaning was to split the train dataset into train and test, with a train size of 80% and a test size of 20%. This is done to avoid data leakage when fitting and transforming the train and test data.

Creating a baseline model

The next step was to create a baseline model which was used to compare and decide between different transformations applied sequentially to the data in order to improve our classification score. After running the SVM, the RandomForest Classifier and the decision classifier the team decided to stick to the Decision Tree Classifier as it provided a simple model with fast training time and an initial **baseline model score of around 0.79**.

```
In [9]: from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import f1_score

        clf = DecisionTreeClassifier()
        clf.fit(X_train, y_train)
        prediction = clf.predict(X_test)

In [10]: print(f"Baseline model score:  {f1_score(y_test, prediction, average = 'macro')}")

         Baseline model score:  0.7950883528718004
```

Figure 3: Baseline model screenshot

Analysing outliers

After analysing the outliers, in order to determine if outliers should be taken into account, the team trained a baseline decision tree model and calculated the F1 Score and compared it with 4 different outlier detection algorithms to determine which one should be used (if any).

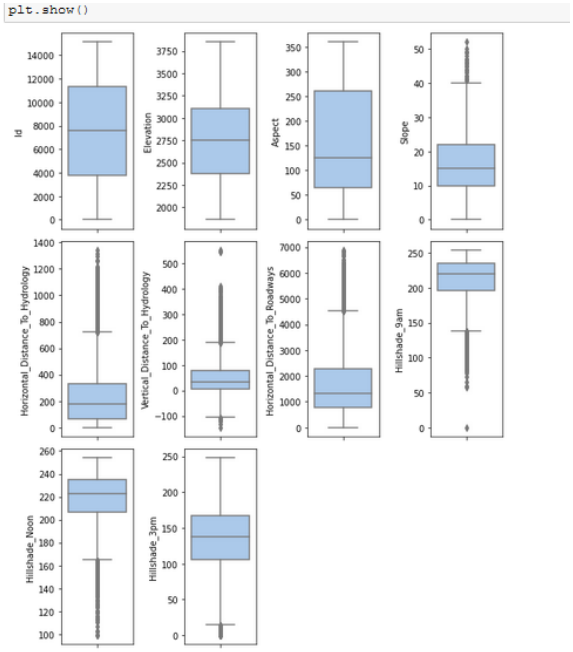

Figure 4: Box plot outlier detection numerical variables

As we can see from Figure 4 some of the numerical variables have outliers. For this reason, the team decided to run an outlier detection with the following algorithms (IsolationForest, EllipticEnvelope, LocalOutlierFactor, OneClassSVM) to see if the F1 score would improve by removing outliers.

Outlier Detection Model	F1 Score
IsolationForest	0.74
EllipticEnvelope	0.73
LocalOutlierFactor	0.74
OneClassSVM	0.73

Figure 5: Outlier Detection Model F1 Scores

As we can see from figure 5 all algorithms which remove the outliers give lower performance than our initial baseline model. So the team decided to keep the outliers within the dataset.

Analysing scale

Usually by using a decision tree classifier there is no need in scaling the data at this stage of the project.

While running the prediction after scaling the model indeed the team can affirm that scaling the variables does not affect the model as the model score after scaling is still similar to the baseline model score.

```
Baseline model score:  0.7859509363604039

After-Scaling model score:  0.7878280105581593
```

Figure 6: Baseline model score vs after scaling model score

Feature Engineering

Feature engineering consists of two parts, **feature generation,** and **feature selection**.

Feature Generation

For feature selection, the team generated as many variables as they could using the column transformer. The new variables were created through four main ways: **arithmetic**, **normalizer**, **polynomial features**, and **exponential features**.

First the team created difference arithmetic and statistic combinations, using addition, and subtraction. Then, using polynomial features, normalization, and exponentials, the team created multiple new features. This was done with the goal of reducing dimensionality using different feature selection methods, in order to find the best features.

```
In [17]: from itertools import combinations

X_train2['linear_distance_to_hydrology'] = (X_train2['Horizontal_Distance_To_Hydrology']**2+X_train2['Vertical_Distance
X_test2['linear_distance_to_hydrology'] = (X_test2['Horizontal_Distance_To_Hydrology']**2+X_test2['Vertical_Distance_To

for i in list(combinations(numerical, 2)):
    X_train2[i[0]+"_add_"+i[1]] = X_train2[i[0]]+X_train2[i[1]]
    X_train2[i[0]+"_mean_"+i[1]] = (X_train2[i[0]]+X_train2[i[1]])/2
    X_train2[i[0]+"_subs_"+i[1]] = X_train2[i[0]]-X_train2[i[1]]
    X_test2[i[0]+"_add_"+i[1]] = X_test2[i[0]]+X_test2[i[1]]
    X_test2[i[0]+"_mean_"+i[1]] = (X_test2[i[0]]+X_test2[i[1]])/2
    X_test2[i[0]+"_subs_"+i[1]] = X_test2[i[0]]-X_test2[i[1]]

for i in list(combinations(numerical, 3)):
    X_train2[i[0]+"_add_"+i[1]+i[2]] = X_train2[i[0]]+X_train2[i[1]]+X_train2[i[2]]
    X_train2[i[0]+"_mean_"+i[1]+i[2]] = (X_train2[i[0]]+X_train2[i[1]]+X_train2[i[2]])/3
    X_test2[i[0]+"_add_"+i[1]+i[2]] = X_test2[i[0]]+X_test2[i[1]]+X_test2[i[2]]
    X_test2[i[0]+"_mean_"+i[1]+i[2]] = (X_test2[i[0]]+X_test2[i[1]]+X_test2[i[2]])/3

for i in list(combinations(numerical, 4)):
    X_train2[i[0]+"_add_"+i[1]+i[2]+i[3]] = X_train2[i[0]]+X_train2[i[1]]+X_train2[i[2]]+X_train2[i[3]]
    X_test2[i[0]+"_add_"+i[1]+i[2]+i[3]] = X_test2[i[0]]+X_test2[i[1]]+X_test2[i[2]]+X_test2[i[3]]


generated = list(set(list(X_train2.columns)) - set(numerical) -set(categorical))
```

Figure 7: Creating new features using itertools library

With these transformed features the pipeline score increased to about 0.80.

```
In [41]: pipe = Pipeline([('transformations', transformations),
                          ('Classifier', DecisionTreeClassifier())])
         pipe.fit(X_train, y_train)
         pipe.score(X_test, y_test)

Out[41]: 0.8035714285714286
```

Figure 8: Score after transformations

Feature Selection

In this part the team wanted to reduce the number of input variables for the development of a predictive model in order to decrease the complexity of the model and to avoid overfitting. With the feature selection, the team also intended to reduce the number of input variables to both reduce the computational cost of modelling, reduce variance, and to improve the performance of the model we will be training later. We will use the following two feature selection methods to achieve that namely:

- Filtering method - SelectFpr
- Dimensionality Reduction method - PCA

First, **SelectFpr** was used to filter the total features by ranking each feature based on their p-values, then dropping the ones that are not significant or less than 0.05. Then, to create uncorrelated linear features that capture as much information as possible and to further reduce the complexity of the model, the team used **PCA**.

```
In [21]: print(f"Features after L1: {features_pipe.steps[1][1].n_features_in_}")
         print(f"Features after filtering: {len(features_pipe.steps[1][1].get_feature_names_out())}")
         print(f"Features after PCA: {len(features_pipe.steps[2][1].explained_variance_ratio_)}")

         Features after L1: 29920
         Features after filtering: 4794
         Features after PCA: 50
```

Figure 9: Feature selection, complexity reduction.

Model Selection

To find the best model, the team tested five different multiclassification algorithms and achieved the following scores:

Algorithm	Score
RandomForestClassifier	0.85
KNeighborsClassifier	0.80
ExtraTreesClassifier	0.87
AdaBoostClassifier	0.86
GaussianNB	0.51

Figure 10: Scores for each algorithm

After comparing the scores, the team decided to choose **ExtraTreesClassifier** as the best model, with a score of .87, in which the team will perform further hyperparameter tuning to.

Optimization

Finally, hyperparameter tuning was done to the model in order to find the best parameters, this was done through **random_search.best_params_**. After running this, the following parameter were chosen, that gave an F1 score of around **0.89.**

- 'n_jobs': -1
- 'n_estimators': 100
- 'min_samples_split': 2
- 'min_samples_leaf': 1
- 'max_samples': None
- 'max_features': 0.6
- 'max_depth': None
- 'bootstrap': False
- 'verbose': 5

```
clf = ExtraTreesClassifier(**random_search.best_params_)
start_time = time.time()

pipe2 = Pipeline([('Classifier', clf)])

pipe2.fit(X_train_features, y_train)
print(f"{clf}, {pipe2.score(X_test_features, y_test)}")
print("--- %s seconds ---" % (time.time() - start_time))
```

```
ExtraTreesClassifier(bootstrap=True, max_features=0.6, n_estimators=300,
                     n_jobs=-1), 0.892526455026455
--- 253.01914429664612 seconds ---
```

Figure 11: Score after optimization

Deployment

Finally, after finding the right model and identifying the best parameters, the team used this model to predict the given test set. The team used the whole dataset in order to train the final classifier, and removed the principal components transformation as this will allow them to provide the best possible model with as much data as possible, this will make it more likely for the model to generalise well.