Mars 2022

Machine Learning

Forest cover type prediction

MASSENET Victor , MELLERIO Antoine

Context **1**

Data exploration **1**

Numerical variables 1

Categorical variables 2

Target 3

Model **3**

Preprocessing 3

Feature engineering 4

Model optimization and selection 4

Conclusion **7**

Annexes **8**

Code 8

I. Context

Having an idea of the diversity of the territories and of its composition is important in order to preserve it effectively. The objective of this project is to develop a model capable of predicting the forest cover type of 30 meters by 30 meters cells of land.

To achieve this, 13 independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data, such as the aspect, the elevation, the distance to Hydrology, the slope or the soil type. Moreover, the US Forest Service (USFS) Region 2 Resource Information System (RIS) provides us with data concerning areas which forest type we know. This data will be used to train the machine learning model we aim to develop.

The forest cover type classification is composed of 7 classes: Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir and Krummholz.

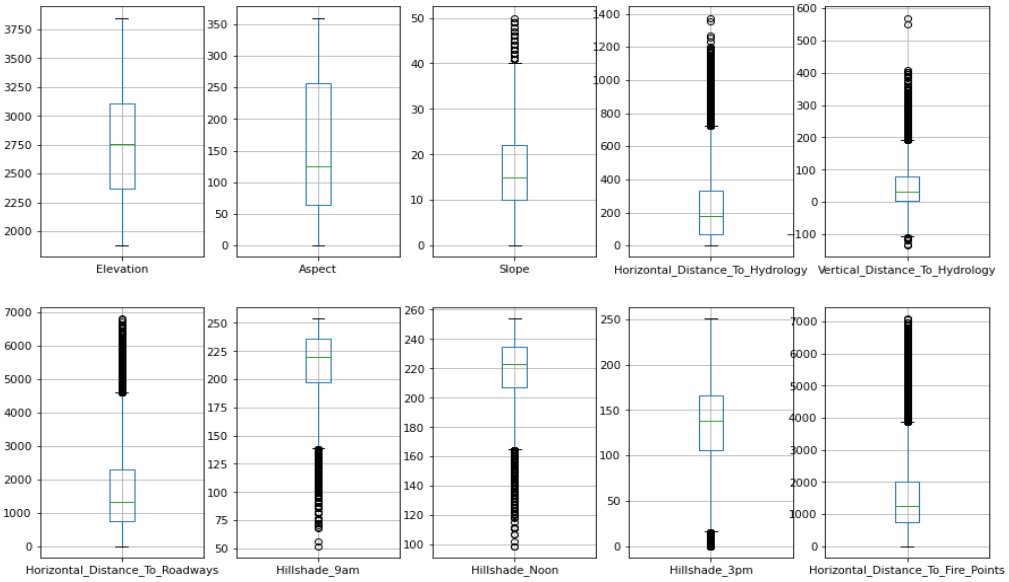
This project is monitored through a Kaggle competition. We are provided with a training set, a test set and a sample submission with the correct format. Our model will be scored based on the accuracy of our prediction over the test set.

II. Data exploration

Ten of the twelve variables available are numerical : Elevation, Aspect, Slope, horizontal and vertical distance to Hydrology, horizontal distance to roadways, hillshade at 9am, at noon and at 3pm, and horizontal distance to fire points. The two others are categorical: Wilderness_Area and Soil_type.

A. Numerical variables

Let's have a look at the distribution of the numerical variable values.



These boxplots allow us to visualize the repartitions of the values and to spot the variables that may have some outliers to be removed in the future. The concerned variables are the slope, the distances to hydrology, the hillshades and the horizontal distance to fire points.

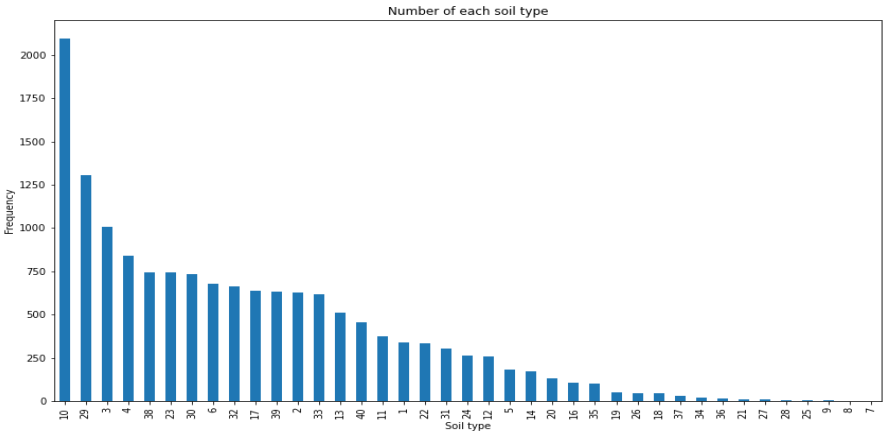
The other important point is the independence of the variables. It is thus relevant to apply a pairplot to the dataset in order to spot potential correlations.

Even though some unreachable areas are identifiable in the plots, there is nothing for us to question the independence of the numerical variables.

B. Categorical variables

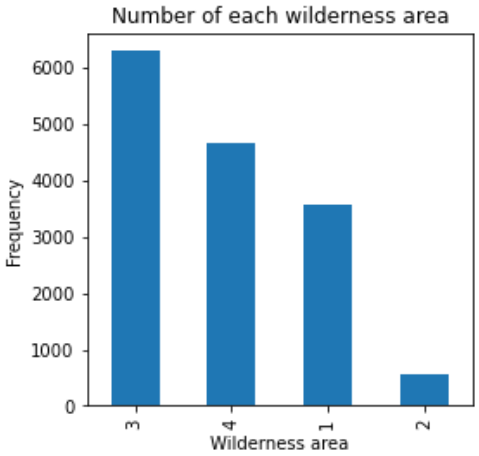
These are already one-hot-encoded with respectively 4 and 40 binary columns. Let's have a look at their repartitions.

Soil type :



Some soil types are excessively under-represented relatively to others. For instance, the soil type n°7 (ELU 3501, Gothic) is represented only once, whereas the soil type n°10 (ELU 4703, Bullwark - Catamount - Rock outcrop complex, rubbly) is represented 2096 times.

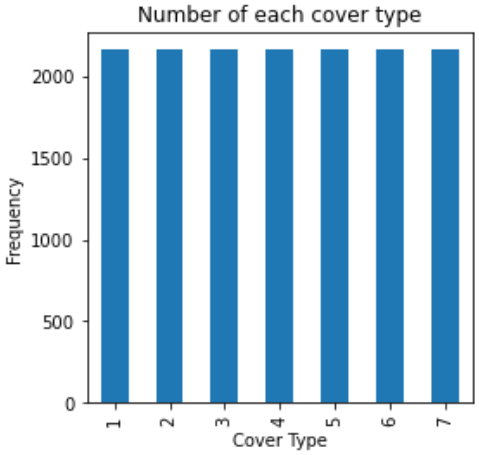
Wilderness area :



The available data is also unbalanced concerning the wilderness area : the type 2 (Neota) is represented 569 times whereas the type 3 (Comanche Peak) is represented 6302 times.

C. Target

As explained in the context, the objective is to predict the forest cover type of multiple land cells. 7 types are identified by integers. Let's have a look at the cover type repartition of the available data.



The target is perfectly balanced with each cover type represented 2160 times.

III. Model

A. Preprocessing

For the preprocessing phase, we first check if there is any null or NA value. The training and the test data set was already clean out of those null values. We separate the forest cover type, our target value, with the rest of the features.

We evaluate the number of outliers for each feature. We build a function that first detects the third quartile and the interquartile range for a given column of a data frame. Then, it calculates upper and lower limits to determine outliers conservatively. We run this function and get rid of outliers. But as it didn't improve our accuracy we did not include in the final code. Below, you can see the code of this detecting outlier method.

```
def outlier_function(df, col_name):
    first_quartile = np.percentile(np.array(df[col_name].tolist()), 25)
    third_quartile = np.percentile(np.array(df[col_name].tolist()), 75)
    IQR = third_quartile - first_quartile

    upper_limit = third_quartile+(3*IQR)
    lower_limit = first_quartile-(3*IQR)
    outlier_count = 0

    for value in df[col_name].tolist():
        if (value < lower_limit) | (value > upper_limit):
            outlier_count +=1
    return lower_limit, upper_limit, outlier_count
```

Then, we run a Principal Component Analysis on the train and test data to evaluate the important axis of the whole data. We add these explanatory variables in the whole data set (train + test). Using these merged dataset, we are now going to do some feature engineering.

B. Feature engineering

Extracting new features based on the existing ones and eliminating features with some methods and algorithms are called feature engineering. Basically, we will do four main feature engineering tasks.

First, we are given Horizontal and Vertical distance to the nearest surface water features. Therefore, we compute the euclidean distance to the water surface.

Second, we are given hill shade for three different hours in the day, we decide to add the mean and the standard deviation of the distribution of these shades during the day.

Also, adding linear combinations of the numeric features is a common practice in feature engineering. Therefore, we decide to combine in different ways the three features describing the horizontal distance to three different places (fireplace, road and water) .

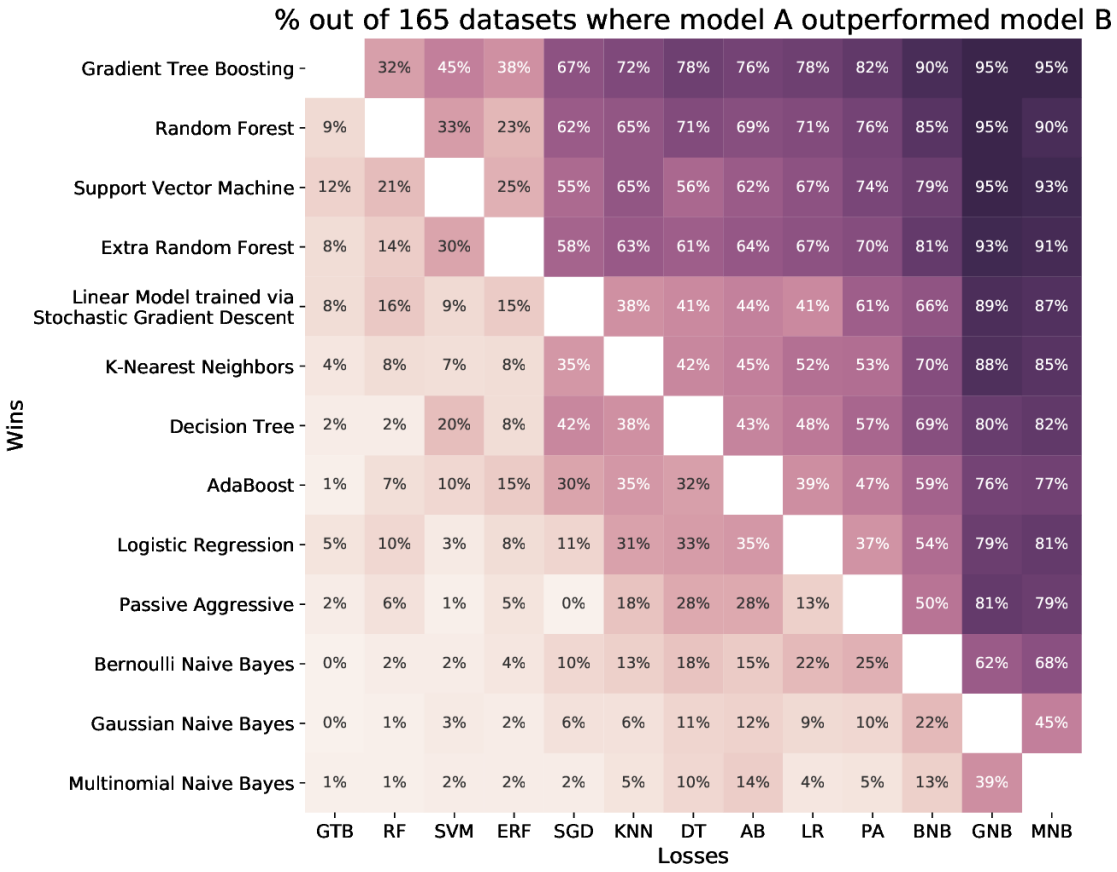
The last feature engineering technique is the computation of a linear combination regarding the height of the tree patch. We add and subtract the distance from the water to the elevation of the patch.

C. Model optimization and selection

In order to evaluate the performance of our model, we split the given train dataset into train and test set. And the given test dataset will be treated as a validation set for the Kaggle competition.

A study[1] shows the performance of classification models across 165 publicly available datasets. You can see the results of this study below.

As you can see, the tree based models are very efficient algorithms for classification, and are probably as well for our dataset. We first train the K-nearest neighbor (using euclidean distance to cluster labels) and the SVM model. This gives us a pretty low accuracy (<75%) for the validation set on Kaggle. This is why we decide to use only tree based models.



We try on four models:

1. Random Forest Classifier from the scikit-learn library
2. Light Gradient Boosting Machine (LightGBM) Classifier from the light gbm library
3. Extra Gradient Boosting (XGBoost) Classifier from the xgboost library
4. Extra Trees (Random Forests) Classifier from the scikit-learn library

For each of these models, we optimize the hyper-parameter n_estimator by using gridSearch. we end up with the values stated in the table page 9.

1

1. Olson RS, Cava W, Mustahsan Z, Varik A, Moore JH. Data-driven advice for applying machine learning to bioinformatics problems. Pac Symp Biocomput. 2018;23:192-203. PMID: 29218881; PMCID: PMC5890912.

Moreover, you can see in our code that we use a class weight array. This distribution of the class in the test set. We saw before in this report that the classes are perfectly distributed in our training set. We assume that this balancedness will not stand in the testing data. Therefore, from our research and using different model predictions we computed, we came up with an expectation of the distribution of the classes in the testing data.

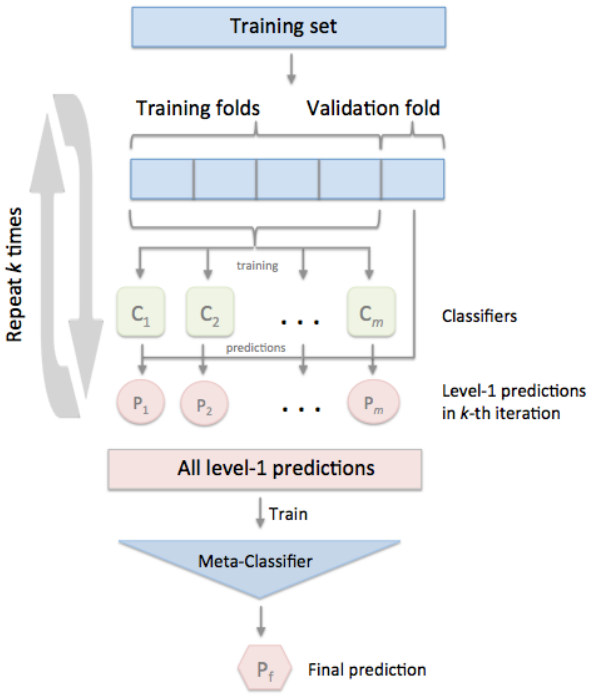
Here are the results obtained, using stratified folds for the cross validation in order to evaluate all the models that we ran.

Model	Random Forest	LightGBM	XGBoost	Extra trees
n_estimators	250	600	50	250
Accuracy	0,76	0,79	0,79	0,80

We obtain a good accuracy (around 80%) for each one of the models. Since several models perform well, we decide to stack them together, using a cross validation to avoid overfitting.

Stacking is a method of ensembling classification model. It consists in processing one after the other two layers of estimators. The first layer consists of all the basic models used to predict the output of the test dataset. The second layer consists of a MetaClassifier. They take the predictions of all the underlying models as input and generate new predictions. A classic stacking classifier is available in the sklearn library.

However, this StackingCVClassifier – available in the mlxtend (machine learning extensions) library – allows us to avoid overfitting by using cross validation, as the picture below describes.



We choose the three models having had the best accuracies previously to stack: ExtraTrees and LGBM as the first-layer models, and XGBoost as the Meta-classifier.

This stacking model got us to reach an **accuracy of 0.84**, the best so far.

IV. Conclusion

In this report, we provided highlights of the end-to-end machine learning workflow application to a supervised multi-class classification problem, predicting forest cover. We started with understanding and visualizing the data with the EDA and formed insights about the cover type dataset. With the outputs of the EDA, we performed feature engineering where we transformed, combined, added and removed features.

Extra trees classifier, LightGBM and XGBoost models matched well to this classification problem with the accuracy metric. We hence decided to stack them together. We managed to reach the first place of the leaderboard (based on 30% of the test data) with an accuracy of 0.86, by tuning some hyperparameters : having the sample_weight insight and optimizing the n_estimators parameter.

Some improvement leads exist, by for instance being more exhaustive on the hyper-parameter tuning, using GridSearch or RandomSearch on more hyper-parameters. We could as well use auto-ML algorithms to find more complex model structures. Finally, extending the dataset by searching for more information on the land cells is always a good way to improve our predictions.

V. Annexes

A. Code

Importing the needed libraries

In [1]:
```
import numpy as np
import pandas as pd
pd.set_option("display.max_columns", None)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import xgboost as xgb
import lightgbm as lgb
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.model_selection import cross_val_score, cross_val_predict, cross_validate, G
from sklearn.decomposition import PCA
from mlxtend.classifier import StackingCVClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, make_scorer
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Defining a weighted score

We create a weighted scorer that takes into account the class distribution obtained by the leaderboard. The classes more represented will have a bigger impact on the scorer, and hence the optimization will be more efficient on accuracy.

In [2]:
```
# obtain test set class distribution through probing predicitons
class_weight = {1: 0.370530,
                2: 0.496810,
                3: 0.059365,
                4: 0.001037,
                5: 0.012958,
                6: 0.026873,
                7: 0.032427}
```

In [3]:
```
def balanced_accuracy_score(y_true, y_pred):
    return accuracy_score(y_true, y_pred, sample_weight=[class_weight[label] for label in
balanced_accuracy_scorer = make_scorer(balanced_accuracy_score, greater_is_better=True)
my_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

Loading the data

In [4]:
```
X_train = pd.read_csv("train.csv")
X_test = pd.read_csv("test-full.csv")
```

In [5]:
```
X_train.drop("Id", axis=1, inplace=True)
test_ID = X_test["Id"]
X_test.drop("Id", axis=1, inplace=True)
```

In [6]:
```
y_train = np.array(X_train['Cover_Type'])
X_train.drop('Cover_Type', axis=1, inplace=True)
```

In [7]:
```
#Check that the data is well one-hot encoded on wilderness area and
```

```
assert np.all(X_train.loc[:, "Wilderness_Area1": "Wilderness_Area4"].sum(axis=1) == 1)
assert np.all(X_train.loc[:, "Soil_Type1": "Soil_Type40"].sum(axis=1) == 1)
```

In [8]:
```
X_train.head()
```

Out[8]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_D
0	2881	130	22	210	54	
1	3005	351	14	242	-16	
2	3226	63	14	618	2	
3	3298	317	8	661	60	
4	3080	35	6	175	26	

Dimension reduction : PCA

In [9]:
```
num_train = X_train.shape[0]
all_data = pd.concat([X_train, X_test])
```

In [10]:
```
pca = PCA(n_components=0.95).fit(all_data)
pca_trans = pca.transform(all_data)
pca_trans.shape
```

Out[10]:
```
(596132, 2)
```

In [11]:
```
for i in range(pca_trans.shape[1]):
    all_data["pca" + str(i)] = pca_trans[:, i]
```

Feature Engineering

Computing the distance to hydrology

In [12]:
```
all_data["Distance_to_Hydrology"] = np.sqrt(np.square(all_data["Vertical_Distance_To_Hydr
                                    np.square(all_data["Horizontal_Distance_To_Hydrology"
```

Mean and standard deviation of hillshade

In [13]:
```
hillshade_cols = ["Hillshade_9am", "Hillshade_Noon", "Hillshade_3pm"]
all_data["Hillshade_mean"] = all_data[hillshade_cols].mean(axis=1)
all_data["Hillshade_std"] = all_data[hillshade_cols].std(axis=1)
```

Combination of horizontal distances

In [14]:
```
cols = ["Horizontal_Distance_To_Hydrology",  "Horizontal_Distance_To_Roadways", "Horizonta
names = ["H", "R", "F"]
for i in range(len(cols)):
    for j in range(i + 1, len(cols)):
        all_data["Horizontal_Distance_combination_" + names[i] + names[j] + "_1"] = all_da
        all_data["Horizontal_Distance_combination_" + names[i] + names[j] + "_2"] = (all_d
        all_data["Horizontal_Distance_combination_" + names[i] + names[j] + "_3"] = all_da
```

```
        all_data["Horizontal_Distance_combination_" + names[i] + names[j] + "_4"] = np.abs
all_data["Horizontal_Distance_mean"] = all_data[cols].mean(axis=1)
```

Elevation of hydrology

In [15]:
```
all_data["Elevation_Hydrology_1"] = all_data["Elevation"] + all_data["Vertical_Distance_To
all_data["Elevation_Hydrology_2"] = all_data["Elevation"] - all_data["Vertical_Distance_To
```

In [16]:
```
all_data.head()
```

Out[16]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_D
0	2881	130	22	210	54	
1	3005	351	14	242	-16	
2	3226	63	14	618	2	
3	3298	317	8	661	60	
4	3080	35	6	175	26	

Comparing models

Defining training and test set

In [17]:
```
X_train = all_data[:num_train]
X_test = all_data[num_train:]
```

RandomForestClassifier

In [18]:
```
clf_rfc = RandomForestClassifier(n_estimators=250, random_state=0, n_jobs=-1)
scores_rfc = cross_validate(clf_rfc, X_train, y_train, cv=my_cv,
                            fit_params={"sample_weight":[class_weight[label] for label in y_t
                            scoring=balanced_accuracy_scorer, return_train_score=True)
print(np.mean(scores_rfc["train_score"]), np.std(scores_rfc["train_score"]))
print(np.mean(scores_rfc["test_score"]), np.std(scores_rfc["test_score"]))
```

```
1.0 0.0
0.7591144495370371 0.0049230332238218485
```

XGBoost

In [19]:
```
clf_xgb = xgb.XGBClassifier(n_estimators=50, random_state=0, n_jobs=-1)
scores_xgb = cross_validate(clf_xgb, X_train, y_train, cv=my_cv,
                            fit_params={"sample_weight":[class_weight[label] for label in y_t
                            scoring=balanced_accuracy_scorer, return_train_score=True)
print(np.mean(scores_xgb["train_score"]), np.std(scores_xgb["train_score"]))
print(np.mean(scores_xgb["test_score"]), np.std(scores_xgb["test_score"]))
```

```
[11:29:30] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:29:37] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
```

```
old behavior.
[11:29:44] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:29:51] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:29:58] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
0.9402220582175925 0.0018215630145176362
0.7875722592592593 0.005860095892003223
```

ExtraTreesClassifier

In [20]:

```
clf_etc = ExtraTreesClassifier(n_estimators=250, random_state=0, n_jobs=-1)
scores_etc = cross_validate(clf_etc, X_train, y_train, cv=my_cv,
                            fit_params={"sample_weight":[class_weight[label] for label in y_t
                            scoring=balanced_accuracy_scorer, return_train_score=True)
print(np.mean(scores_etc["train_score"]), np.std(scores_etc["train_score"]))
print(np.mean(scores_etc["test_score"]), np.std(scores_etc["test_score"]))
```

```
1.0 0.0
0.8018592287037037 0.009538528269943518
```

LGBMClassifier

In [21]:

```
clf_lgbmc = lgb.LGBMClassifier(n_estimators=600, random_state=0, n_jobs=-1)
scores_lgbmc = cross_validate(clf_lgbmc, X_train, y_train, cv=my_cv,
                            fit_params={"sample_weight":[class_weight[label] for label in y_t
                            scoring=balanced_accuracy_scorer, return_train_score=True)
print(np.mean(scores_lgbmc["train_score"]), np.std(scores_lgbmc["train_score"]))
print(np.mean(scores_lgbmc["test_score"]), np.std(scores_lgbmc["test_score"]))
```

```
1.0 0.0
0.7908917750000002 0.009332265061177264
```

StackingCVClassifier

In [22]:

```
clf1 = ExtraTreesClassifier(n_estimators=250, random_state=0, n_jobs=-1)
clf2 = lgb.LGBMClassifier(n_estimators=600, random_state=0, n_jobs=-1)
clf = StackingCVClassifier(classifiers=[clf1, clf2],
                           meta_classifier=xgb.XGBClassifier(
                               n_estimators=50, random_state=0, n_jobs=-1),
                           cv=my_cv, random_state=0, use_probas=True, use_features_in_sec
scores_stack = cross_validate(clf, X_train, y_train, cv=my_cv,
                              fit_params={"sample_weight": [
                                  class_weight[label] for label in y_train]},
                              scoring=balanced_accuracy_scorer, return_train_score=True)
print(np.mean(scores_stack["train_score"]),
      np.std(scores_stack["train_score"]))
print(np.mean(scores_stack["test_score"]), np.std(scores_stack["test_score"]))
```

```
[11:32:15] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:33:34] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
```

```
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:35:08] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:36:38] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[11:37:40] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
0.9999409990740741 0.00011428933801491637
0.8424386476851853 0.00849603508790092
```

Predicting

In [23]:

```
clf.fit(X_train, y_train, sample_weight=[class_weight[label] for label in y_train])
pred = clf.predict(X_test)
```

```
[11:39:11] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

Saving results

In [24]:

```
submission = pd.DataFrame({'Id':test_ID, 'Cover_Type':pred}, columns=['Id', 'Cover_Type'])
submission.to_csv("output.csv", index=False)
```