CS109A Introduction to Data Science:

Homework 8 AC 209 : Trees and ensemble methods

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

In [1]:
```
# RUN THIS CELL FOR FORMAT
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-C
HTML(styles)
```

Out[1]:

In [2]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets
import sklearn
import pandas as pd
%matplotlib inline
```

Question 7

7.1 Describe the main differences between bagging and adaptative boosting.

7.2 Why do we use the word "gradient" in gradient boosting?

7.3 Describe three improvements of XGBoost over the conventional implementation of Boosted Trees.

Question 8

Here, we will compare some of the top ensemble methods for classification. We will look at AdaBoost, XGBoost, LGBM and CatBoost.

- To install XGBoost, run `pip3 install xgboost`.

- To install LGBM, run `pip install lightgbm`
- To install CatBoost, run `conda -c conda-forge install catboost` if using conda, or `pip install catboost` if not.

We will be using a different dataset than what we're used to, so as to test the capabilities of these advanced classifiers. We will be playing with the Forest Cover Type dataset, a classification dataset where observations from 30mx30m patches of forest are associated with the type of tree that grows there. We will be trying to predict the primary species of those patches based on 54 predictors, e.g. elevation, slope, distance to water, etc.

Here are the main predictors of the dataset:

- Elevation
- Aspect
- Slope
- Horizontal_Distance_To_Hydrology
- Vertical_Distance_To_Hydrology
- Hillshade_9am
- Hillshade_Noon
- Hillshade_3pm
- Horizontal_Distance_To_Fire_Points
- Wilderness_Area (one-hot encoded, 4 binary columns)
- Soil_Type (one-hot encoded, 40 binary columns)

Response: Cover_Type (7 types), integer, 1 to 7

For more details on the dataset, visit [http://archive.ics.uci.edu/ml/datasets/Covertype (http://archive.ics.uci.edu/ml/datasets/Covertype)](http://archive.ics.uci.edu/ml/datasets/Covertype)

8.1 Import the coverage type dataset from sklearn.datasets with `datasets.fetch_covtype` . Use return_X_y=True and split the data into train and test sets (30% test). You can downsample the data to 10% of the full dataset if needed.

8.2 Train a DecisionTreeClassifier, RandomForestClassifier, AdaboostClassifier, LGBMClassifier, XGBoostClassifier, and CatBoost on the data.

On a first pass, use the classifiers out of the box, with no parameter modification. As a second pass, use crossvalidation on the following parameters:

- n_estimators
- min_samples_leaf
- max_depth
- max_leaf_nodes
- max_features

Make sure that you use the sklearn-like interfaces:

- DecisionTreeClassifier, RandomForestClassifier, AdaboostClassifier given by sklearn
- XGBClassifier can be accessed with `from xgboost import XGBClassifier`
- LGBMClassifier can be accessed with `from lightgbm.sklearn import LGBMClassifier`

- CatBoostClassifier can be accessed with `from catboost import CatBoostClassifier`

8.3 Time both training (.fit method) and inference (.predict method), and show classification accuracy for all classifiers. For this dataset, substract 1 to your array of labels so that the label format plays nicely with CatBoost. Comment on the results.

8.4 Let's now play with a high-dimensional dataset. Load the Faces in The Wild dataset with `datasets.fetch_lfw_people(return_X_y=True, min_faces_per_person=20)`. Split the data into train and test sets (30% test).

8.5 Again, train all classifiers enumerated above and report training and inference times. Comment on the results.

8.6 How did the high dimensionality affect each classifier?

Answers

Question 7

7.1 Describe the main differences between bagging and adaptative boosting.

Criteria	Bagging	Boosting
What data is each individual model trained on?	The training data with weightings given by a multinomial distribution with n trials and n classes	The training data with samples classified incorrectly by the previous classifier weighted higher
How is the ensemble created?	Averaging or majority vote	A linear combination with better performing models weighted higher
Can it be fit in parallel?	Yes	No
Bias/Variance trade-off?	Increases bias	Increases variance

7.2 Why do we use the word "gradient" in gradient boosting?

Gradient boosting can be thought of as gradient descent through the prediction space or function space. For example, if our loss function is the mean squared error $L(y, \bar{y}) = 1/n \sum_i (y_i - \bar{y}_i))^2$, then $\partial L / \partial \bar{y}_i \propto \bar{y}_i - y_i$. When we fit our weak classifier to the residuals and add it to the previous model, we are essentially moving our prediction in the direction of the gradient of the loss function with respect to our previous predictions.

This post is very clear on this (could be a good resource for next year): [https://explained.ai/gradient-boosting/descent.html (https://explained.ai/gradient-boosting/descent.html)](https://explained.ai/gradient-boosting/descent.html)

7.3 Describe three improvements of XGBoost over the conventional implementation of Boosted

Trees.

1. It can be run in parallel
2. Can handle arbitrary differentiable loss functions (makes regularization easier)
3. Incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data

Question 8

Here, we will compare some of the top ensemble methods for classification. We will look at AdaBoost, XGBoost, LGBM and CatBoost.

- To install XGBoost, run `pip3 install xgboost`.
- To install LGBM, run `pip install lightgbm`
- To install CatBoost, run `conda -c conda-forge install catboost` if using conda, or `pip install catboost` if not.

We will be using a different dataset than what we're used to, so as to test the capabilities of these advanced classifiers. We will be playing with the Forest Cover Type dataset, a classification dataset where observations from 30mx30m patches of forest are associated with the type of tree that grows there. We will be trying to predict the primary species of those patches based on 54 predictors, e.g. elevation, slope, distance to water, etc.

Here are the main predictors of the dataset:

- Elevation
- Aspect
- Slope
- Horizontal_Distance_To_Hydrology
- Vertical_Distance_To_Hydrology
- Hillshade_9am
- Hillshade_Noon
- Hillshade_3pm
- Horizontal_Distance_To_Fire_Points
- Wilderness_Area (one-hot encoded, 4 binary columns)
- Soil_Type (one-hot encoded, 40 binary columns)

Response: Cover_Type (7 types), integer, 1 to 7

For more details on the dataset, visit [http://archive.ics.uci.edu/ml/datasets/Covertype (http://archive.ics.uci.edu/ml/datasets/Covertype)](http://archive.ics.uci.edu/ml/datasets/Covertype)

8.1 Import the coverage type dataset from sklearn.datasets with `datasets.fetch_covtype`. Use return_X_y=True and split the data into train and test sets (30% test). You can downsample the data to 10% of the full dataset if needed.

In [3]:
```
data = pd.read_csv('./covtype.csv')
data = data.sample(frac = 0.1)
X_train, X_test, y_train, y_test = train_test_split(data.drop('Cover_Type',
```

8.2 Train a DecisionTreeClassifier, RandomForestClassifier, AdaboostClassifier, LGBMClassifier, XGBoostClassifier, and CatBoost on the data.

On a first pass, use the classifiers out of the box, with no parameter modification. As a second pass, use crossvalidation on the following parameters:

- n_estimators
- min_samples_leaf
- max_depth
- max_leaf_nodes
- max_features

Make sure that you use the sklearn-like interfaces:

- DecisionTreeClassifier, RandomForestClassifier, AdaboostClassifier given by sklearn
- XGBClassifier can be accessed with `from xgboost import XGBClassifier`
- LGBMClassifier can be accessed with `from lightgbm.sklearn import LGBMClassifier`
- CatBoostClassifier can be accessed with `from catboost import CatBoostClassifier`

In [4]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from lightgbm.sklearn import LGBMClassifier
from sklearn.model_selection import GridSearchCV
# I'm sorry, I couldn't get catboost to install on my machine.
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/lightg
bm/__init__.py:46: UserWarning: Starting from version 2.2.1, the library
file in distribution wheels for macOS is built by the Apple Clang (Xcode_
8.3.1) compiler.
This means that in case of installing LightGBM from PyPI via the ``pip in
stall lightgbm`` command, you don't need to install the gcc compiler anym
ore.
Instead of that, you need to install the OpenMP library, which is require
d for running LightGBM on the system with the Apple Clang compiler.
You can install the OpenMP library by the following command: ``brew insta
ll libomp``.
  "You can install the OpenMP library by the following command: ``brew in
stall libomp``.", UserWarning)
```

In [5]:
```
models = [DecisionTreeClassifier,
          RandomForestClassifier,
          AdaBoostClassifier,
          XGBClassifier,
          LGBMClassifier,
          ]
trained_vanilla_models = []
for m in models:
    print(m)
    trained_vanilla_models.append(m().fit(X_train, y_train))
```

```
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
<class 'sklearn.ensemble.forest.RandomForestClassifier'>
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/sklear
n/ensemble/forest.py:248: FutureWarning: The default value of n_estimator
s will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
<class 'sklearn.ensemble.weight_boosting.AdaBoostClassifier'>
<class 'xgboost.sklearn.XGBClassifier'>
<class 'lightgbm.sklearn.LGBMClassifier'>
```

In [6]:
```
trained_vanilla_models
```

Out[6]:
```
[DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=No
ne,
             max_features=None, max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, presort=False, random_state=No
ne,
             splitter='best'),
 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
             max_depth=None, max_features='auto', max_leaf_nodes=None,
             min_impurity_decrease=0.0, min_impurity_split=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
             oob_score=False, random_state=None, verbose=0,
             warm_start=False),
 AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
           learning_rate=1.0, n_estimators=50, random_state=None),
 XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
        colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
        max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
        n_jobs=1, nthread=None, objective='multi:softprob', random_state=
0,
        reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
        silent=True, subsample=1),
 LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree
=1.0,
         importance_type='split', learning_rate=0.1, max_depth=-1,
         min_child_samples=20, min_child_weight=0.001, min_split_gain=0.
0,
         n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
         random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
         subsample=1.0, subsample_for_bin=200000, subsample_freq=0)]
```

In [8]:
```
param_grid = {
    'n_estimators': [200, 300],
    'min_samples_leaf': [1,5],
    'max_depth': [1,5],
    'max_leaf_nodes': [8,16],
    'max_features': [0.5,1]
}

models = [DecisionTreeClassifier,
          RandomForestClassifier,
          AdaBoostClassifier,
          XGBClassifier,
          LGBMClassifier,
         ]

trained_cv_models = []
```

In [14]:
```
for m in models:
    print(m)
    try:
        model = m(n_jobs = 8)
        print('running multiple jobs')
    except TypeError:
        model = m()
    params = {k : param_grid[k] for k in param_grid if k in model.get_param
    m_cv = GridSearchCV(model, params, cv = 3)
    trained_cv_models.append(m_cv.fit(X_train, y_train))
```

```
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
<class 'sklearn.ensemble.forest.RandomForestClassifier'>
running multiple jobs
<class 'sklearn.ensemble.weight_boosting.AdaBoostClassifier'>
<class 'xgboost.sklearn.XGBClassifier'>
running multiple jobs
<class 'lightgbm.sklearn.LGBMClassifier'>
running multiple jobs
```

In [15]:
```
trained_cv_models
```

Out[15]:
```
[GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=DecisionTreeClassifier(class_weight=None, criterion='gi
ni', max_depth=None,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, presort=False, random_state=No
ne,
              splitter='best'),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'min_samples_leaf': [1, 5], 'max_depth': [1, 5], 'max
_leaf_nodes': [8, 16], 'max_features': [0.5, 1]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=DecisionTreeClassifier(class_weight=None, criterion='gi
ni', max_depth=None,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, presort=False, random_state=No
ne,
              splitter='best'),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'min_samples_leaf': [1, 5], 'max_depth': [1, 5], 'max
_leaf_nodes': [8, 16], 'max_features': [0.5, 1]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=DecisionTreeClassifier(class_weight=None, criterion='gi
ni', max_depth=None,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, presort=False, random_state=No
ne,
              splitter='best'),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'min_samples_leaf': [1, 5], 'max_depth': [1, 5], 'max
_leaf_nodes': [8, 16], 'max_features': [0.5, 1]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=RandomForestClassifier(bootstrap=True, class_weight=Non
e, criterion='gini',
              max_depth=None, max_features='auto', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=8,
              oob_score=False, random_state=None, verbose=0,
              warm_start=False),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'n_estimators': [200, 300], 'min_samples_leaf': [1,
5], 'max_depth': [1, 5], 'max_leaf_nodes': [8, 16], 'max_features': [0.5,
1]},
```

```
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=AdaBoostClassifier(algorithm='SAMME.R', base_estimator=
None,
           learning_rate=1.0, n_estimators=50, random_state=None),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'n_estimators': [200, 300]}, pre_dispatch='2*n_jobs',
        refit=True, return_train_score='warn', scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsamp
le_bylevel=1,
        colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
        max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
        n_jobs=8, nthread=None, objective='binary:logistic', random_state
=0,
        reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
        silent=True, subsample=1),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'n_estimators': [200, 300], 'max_depth': [1, 5]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0),
 GridSearchCV(cv=3, error_score='raise-deprecating',
        estimator=LGBMClassifier(boosting_type='gbdt', class_weight=None,
colsample_bytree=1.0,
         importance_type='split', learning_rate=0.1, max_depth=-1,
         min_child_samples=20, min_child_weight=0.001, min_split_gain=0.
0,
         n_estimators=100, n_jobs=8, num_leaves=31, objective=None,
         random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
         subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
        fit_params=None, iid='warn', n_jobs=None,
        param_grid={'n_estimators': [200, 300], 'max_depth': [1, 5]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring=None, verbose=0)]
```

your answer here

8.3 Time both training (.fit method) and inference (.predict method), and show classification accuracy for all classifiers. For this dataset, substract 1 to your array of labels so that the label format plays nicely with CatBoost. Comment on the results.

In [16]:
```
import timeit
```

In [17]:
```
models = [DecisionTreeClassifier,
          RandomForestClassifier,
          AdaBoostClassifier,
          XGBClassifier,
          LGBMClassifier,
          ]

trained_vanilla_models = []
time = []
for m in models:
    print(m)
    start_time = timeit.default_timer()
    trained_vanilla_models.append(m().fit(X_train, y_train))
    elapsed = timeit.default_timer() - start_time
    time.append(elapsed)
```

```
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
<class 'sklearn.ensemble.forest.RandomForestClassifier'>

/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/sklear
n/ensemble/forest.py:248: FutureWarning: The default value of n_estimator
s will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

<class 'sklearn.ensemble.weight_boosting.AdaBoostClassifier'>
<class 'xgboost.sklearn.XGBClassifier'>
<class 'lightgbm.sklearn.LGBMClassifier'>
```

In [18]:
```
print("training time")
print(dict(zip(models,time)))
```

Out[18]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.5296440760139376,
 sklearn.ensemble.forest.RandomForestClassifier: 0.622468160931021,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 1.611194389872253,
 xgboost.sklearn.XGBClassifier: 68.43641737685539,
 lightgbm.sklearn.LGBMClassifier: 3.465232075890526}
```

In [21]:
```
from sklearn.metrics import accuracy_score
```

In [22]:
```
train_accs = []
test_accs = []
time_predict_train = []
time_predict_test = []

for m in trained_vanilla_models:
    print(m)

    # predict train
    start_time = timeit.default_timer()
    y_train_pred = m.predict(X_train)
    elapsed = timeit.default_timer() - start_time
    time_predict_train.append(elapsed)
    train_accs.append(accuracy_score(y_train, y_train_pred))

    # predict test
    start_time = timeit.default_timer()
    y_test_pred = m.predict(X_test)
    elapsed = timeit.default_timer() - start_time
    time_predict_test.append(elapsed)
    test_accs.append(accuracy_score(y_test, y_test_pred))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=Non
e,
            splitter='best')
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=
0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=
1.0,
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
        n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [24]:
```
print("pred time training")
dict(zip(models,time_predict_train))
```

```
pred time training
```

Out[24]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.0161871500313282,
 sklearn.ensemble.forest.RandomForestClassifier: 0.11870791390538216,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.2457608159165829
4,
 xgboost.sklearn.XGBClassifier: 0.9766346090473235,
 lightgbm.sklearn.LGBMClassifier: 0.5447103709448129}
```

In [25]:
```
print("training accuracy")
dict(zip(models, train_accs))
```

```
training accuracy
```

Out[25]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 1.0,
 sklearn.ensemble.forest.RandomForestClassifier: 0.9937791984263585,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.6329727071551512,
 xgboost.sklearn.XGBClassifier: 0.7517580526186378,
 lightgbm.sklearn.LGBMClassifier: 0.8636587164986477}
```

In [26]:
```
print("pred time testing")
dict(zip(models,time_predict_test))
```

```
pred time testing
```

Out[26]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.011219922918826342,
 sklearn.ensemble.forest.RandomForestClassifier: 0.03834013803862035,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.1012372169643640
5,
 xgboost.sklearn.XGBClassifier: 0.4385811679530889,
 lightgbm.sklearn.LGBMClassifier: 0.20654238597489893}
```

In [27]:
```
print("test accuracy")
dict(zip(models, test_accs))
```

```
test accuracy
```

Out[27]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.820434857437898,
 sklearn.ensemble.forest.RandomForestClassifier: 0.8527910045321554,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.6299695944007803,
 xgboost.sklearn.XGBClassifier: 0.7465435144283173,
 lightgbm.sklearn.LGBMClassifier: 0.8239343698009294}
```

your answer here

The boosting methods are slower to train, particularly xgboost. Boosting methods are also slower in predictions. Out-of-the-box, Random forest had the best test set accuracy.

8.4 Let's now play with a high-dimensional dataset. Load the Faces in The Wild dataset with `datasets.fetch_lfw_people(return_X_y=True, min_faces_per_person=20)` . Split the data into train and test sets (30% test).

In [28]:
```
data = datasets.fetch_lfw_people(return_X_y=True, min_faces_per_person=20)
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
 (https://ndownloader.figshare.com/files/5976012)
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
 (https://ndownloader.figshare.com/files/5976009)
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
 (https://ndownloader.figshare.com/files/5976006)
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/597
6015 (https://ndownloader.figshare.com/files/5976015)
```

In [39]:
```
X_train, X_test, y_train, y_test = train_test_split(data[0], data[1], test_
```

your answer here

8.5 Again, train all classifiers enumerated above and report training and inference times. Comment on the results.

In [42]:
```
models = [DecisionTreeClassifier,
          RandomForestClassifier,
          AdaBoostClassifier,
          XGBClassifier,
          LGBMClassifier,
          ]

trained_vanilla_models = []
time = []
for m in models:
    print(m)
    try:
        start_time = timeit.default_timer()
        trained_vanilla_models.append(m(n_estimators = 50).fit(X_train, y_t
        elapsed = timeit.default_timer() - start_time
    except TypeError:
        start_time = timeit.default_timer()
        trained_vanilla_models.append(m().fit(X_train, y_train))
        elapsed = timeit.default_timer() - start_time
    time.append(elapsed)
```

```
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
<class 'sklearn.ensemble.forest.RandomForestClassifier'>
<class 'sklearn.ensemble.weight_boosting.AdaBoostClassifier'>
<class 'xgboost.sklearn.XGBClassifier'>
<class 'lightgbm.sklearn.LGBMClassifier'>
```

In [44]:
```
print("training time")
dict(zip(models,time))
```

```
training time
```

Out[44]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 9.668490819865838,
 sklearn.ensemble.forest.RandomForestClassifier: 5.625482650008053,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 24.437815509038046,
 xgboost.sklearn.XGBClassifier: 957.7722799461335,
 lightgbm.sklearn.LGBMClassifier: 594.304289652966}
```

In [45]:
```
train_accs = []
test_accs = []
time_predict_train = []
time_predict_test = []

for m in trained_vanilla_models:
    print(m)

    # predict train
    start_time = timeit.default_timer()
    y_train_pred = m.predict(X_train)
    elapsed = timeit.default_timer() - start_time
    time_predict_train.append(elapsed)
    train_accs.append(accuracy_score(y_train, y_train_pred))

    # predict test
    start_time = timeit.default_timer()
    y_test_pred = m.predict(X_test)
    elapsed = timeit.default_timer() - start_time
    time_predict_test.append(elapsed)
    test_accs.append(accuracy_score(y_test, y_test_pred))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=Non
e,
            splitter='best')
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=50,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=
0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=
1.0,
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
        n_estimators=50, n_jobs=-1, num_leaves=31, objective=None,
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [50]:
```
print("pred time training")
dict(zip(models,time_predict_train))
```

```
pred time training
```

Out[50]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.0058713448233902454,
 sklearn.ensemble.forest.RandomForestClassifier: 0.11207350599579513,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.1746476301923394
2,
 xgboost.sklearn.XGBClassifier: 5.322163850069046,
 lightgbm.sklearn.LGBMClassifier: 0.4039787990041077}
```

In [51]:
```
print("training accuracy")
dict(zip(models, train_accs))
```

```
training accuracy
```

Out[51]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 1.0,
 sklearn.ensemble.forest.RandomForestClassifier: 1.0,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.1980151228733459
4,
 xgboost.sklearn.XGBClassifier: 0.999054820415879,
 lightgbm.sklearn.LGBMClassifier: 1.0}
```

In [52]:
```
print("pred time testing")
dict(zip(models,time_predict_test))
```

```
pred time testing
```

Out[52]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.0022696589585393667,
 sklearn.ensemble.forest.RandomForestClassifier: 0.02217451692558825,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.0613770550116896
6,
 xgboost.sklearn.XGBClassifier: 2.262235410977155,
 lightgbm.sklearn.LGBMClassifier: 0.18916428904049098}
```

In [53]:
```
print("test accuracy")
dict(zip(models, test_accs))
```

```
test accuracy
```

Out[53]:
```
{sklearn.tree.tree.DecisionTreeClassifier: 0.19625137816979052,
 sklearn.ensemble.forest.RandomForestClassifier: 0.3539140022050717,
 sklearn.ensemble.weight_boosting.AdaBoostClassifier: 0.1786108048511576
6,
 xgboost.sklearn.XGBClassifier: 0.40352811466372657,
 lightgbm.sklearn.LGBMClassifier: 0.38257993384785005}
```

your answer here **Comments:**

- The boosting methods are slower to train than the others
- XGBoost has the slowest prediction time (this is strange because one of the advantages of xgboost is that it's fast...I think I'm not using it correctly) I've read that XGBoost isn't great with multiclass classification, so maybe that's what's going on

- XGBoost performs best out of all the classifiers, withg lightGBM and random forest being 2nd and 3rd best, respectively.

8.6 How did the high dimensionality affect each classifier?

All of the classifiers got worse, with xgboost and lightgbm being the most resilient.