XGBoost

As a last ditch effort, let's try XGBoost

```
import numpy as np
import pandas as pd

from scipy.stats import uniform, randint

# These are a godsend
# https://scikit-learn.org/stable/modules/model_evaluation.html
from sklearn.metrics import auc, accuracy_score, confusion_matrix, mean_squared_error

# these are also a godsend
# https://scikit-learn.org/stable/model_selection.html
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, RandomizedSearchCV,

import xgboost as xgb
```

Load data

```
# TF V2
TRAINING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-d
TESTING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-de
EVAL_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis

# Load data for random forest + NN ensemble evaluation
def load_data(path):
  data = pd.read_csv(path).to_numpy()
  X = data[:,1:]
  y = data[:,0]
  y[y == -1] = 0
  return X,y

%time X_train,y_train = load_data(TRAINING_PATH_TF)
%time X_test, y_test = load_data(TESTING_PATH_TF)
%time X_eval, y_eval = load_data(EVAL_PATH_TF)
# %time Xm,ym = load_data(TRAINING_PATH)
# %time Xm_val, ym_val = load_data(TESTING_PATH)
```

```
CPU times: user 1min 5s, sys: 6.19 s, total: 1min 11s
Wall time: 1min 13s
CPU times: user 8.65 s, sys: 246 ms, total: 8.9 s
Wall time: 9.09 s
CPU times: user 19.5 s, sys: 370 ms, total: 19.8 s
Wall time: 20.3 s
```

```
# Feat VEC
# NN feature vecs
# Feature vector paths
```

```
# feature vector paths
X_TRAIN = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/
y_TRAIN = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/

X_TEST = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/e
y_TEST = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/e

X_EVAL = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/e
y_EVAL = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions/e

X_train = np.load(X_TRAIN)
y_train = np.load(y_TRAIN)
X_test = np.load(X_TEST)
y_test = np.load(y_TEST)
X_eval = np.load(X_EVAL)
y_eval = np.load(y_EVAL)

def load_data_np(X,y):
  y[y==0] = -1
  raw_data = np.append(y,X,axis=1)
  bias = np.ones((X.shape[0],1))
  X = np.append(X,bias,axis=1)
  y = np.ravel(y)
  return X,y

X_train, y_train = load_data_np(X_train,y_train)
X_test,y_test = load_data_np(X_test,y_test)
X_eval,y_eval = load_data_np(X_eval,y_eval)
```

Training XGBoost

https://www.kaggle.com/stuarthallows/using-xgboost-with-scikit-learn

This is still getting about what the most basic perceptron algorith gets.... a little wacky imo..

```
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
%time xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_train)

print(accuracy_score(y_train, y_pred))
```

```
CPU times: user 5min 52s, sys: 3.02 s, total: 5min 55s
Wall time: 5min 55s
0.8454285714285714
```

```
y_pred = xgb_model.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.8395555555555556
```

```python
# if more than one evaluation metric are given the last one is used for early stopping
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42, eval_metric="auc")

%time xgb_model.fit(X, y, early_stopping_rounds=5, eval_set=[(X_val, y_val)])

y_pred = xgb_model.predict(X_val)

accuracy_score(y_val, y_pred)
```

```
[0]     validation_0-auc:0.849465
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]     validation_0-auc:0.850305
[2]     validation_0-auc:0.869562
[3]     validation_0-auc:0.874523
[4]     validation_0-auc:0.878177
[5]     validation_0-auc:0.88519
[6]     validation_0-auc:0.885503
[7]     validation_0-auc:0.887853
[8]     validation_0-auc:0.888591
[9]     validation_0-auc:0.889354
[10]    validation_0-auc:0.888923
[11]    validation_0-auc:0.889387
[12]    validation_0-auc:0.893736
[13]    validation_0-auc:0.893381
[14]    validation_0-auc:0.896269
[15]    validation_0-auc:0.897974
[16]    validation_0-auc:0.898069
[17]    validation_0-auc:0.899338
[18]    validation_0-auc:0.901077
[19]    validation_0-auc:0.901985
[20]    validation_0-auc:0.902484
[21]    validation_0-auc:0.902278
[22]    validation_0-auc:0.902294
[23]    validation_0-auc:0.902451
[24]    validation_0-auc:0.903214
[25]    validation_0-auc:0.904819
[26]    validation_0-auc:0.905079
[27]    validation_0-auc:0.906403
[28]    validation_0-auc:0.907313
[29]    validation_0-auc:0.909438
[30]    validation_0-auc:0.909114
[31]    validation_0-auc:0.90943
[32]    validation_0-auc:0.90997
[33]    validation_0-auc:0.911729
[34]    validation_0-auc:0.912402
[35]    validation_0-auc:0.91243
[36]    validation_0-auc:0.912636
[37]    validation_0-auc:0.91259
[38]    validation_0-auc:0.913091
[39]    validation_0-auc:0.913242
[40]    validation_0-auc:0.91332
[41]    validation_0-auc:0.913273
[42]    validation_0-auc:0.913265
[43]    validation_0-auc:0.913722
[44]    validation_0-auc:0.913805
[45]    validation_0-auc:0.914501
[46]    validation_0-auc:0.914936
[47]    validation_0-auc:0.914952
[48]    validation_0-auc:0.915367
[49]    validation_0-auc:0.915528
```

```
[50]	validation_0-auc:0.915895
[51]	validation_0-auc:0.916255
[52]	validation_0-auc:0.916952
[53]	validation_0-auc:0.917385
[54]	validation_0-auc:0.917594
[55]	validation_0-auc:0.917833
[56]	validation_0-auc:0.917886
[57]	validation_0-auc:0.91803
```

More sophisticated hyper parameter search

```
xgb_model = xgb.XGBClassifier(objective="binary:logistic")

params = {
    "colsample_bytree": uniform(0.7, 0.3),
    "gamma": uniform(0, 0.5),
    "learning_rate": uniform(0.03, 0.3), # default 0.1
    "max_depth": randint(2, 6), # default 3
    "n_estimators": randint(100, 150), # default 100
    "subsample": uniform(0.6, 0.4)
}

search = RandomizedSearchCV(xgb_model, param_distributions=params, random_state=42, n_iter=10,

search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:  4.1min finished
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call
last)
<ipython-input-18-7cfa8a37dbf0> in <module>()
     14 search.fit(X_train, y_train)
     15
---> 16 report_best_scores(search.cv_results_, 1)

NameError: name 'report_best_scores' is not defined
```

```
def report_best_scores(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
report_best_scores(search.cv_results_, 1)
```

```
Model with rank: 1
Mean validation score: 0.896 (std: 0.002)
Parameters: {'colsample_bytree': 0.8835558684167137, 'gamma': 0.06974693032602092, 'learr
```

Train on these params

For feat vecs: Parameters: {'colsample_bytree': 0.8835558684167137, 'gamma': 0.06974693032602092, 'learning_rate': 0.11764339456056544, 'max_depth': 5, 'n_estimators': 114, 'subsample': 0.7824279936868144}

```
# if more than one evaluation metric are given the last one is used for early stopping
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42, colsample_bytree=0
                              gamma=0.06974693032602092, learning_rate=0.11764339456056544,
                              max_depth=5,n_estimators=114,subsample=0.7824279, eval_metric="a

%time xgb_model.fit(X_train, y_train, early_stopping_rounds=5, eval_set=[(X_test, y_test)])

y_pred = xgb_model.predict(X_test)

accuracy_score(y_test, y_pred)
```

```
[0]     validation_0-auc:0.922966
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]     validation_0-auc:0.923672
[2]     validation_0-auc:0.923446
[3]     validation_0-auc:0.924609
[4]     validation_0-auc:0.924868
[5]     validation_0-auc:0.924608
[6]     validation_0-auc:0.925357
[7]     validation_0-auc:0.925657
[8]     validation_0-auc:0.926147
[9]     validation_0-auc:0.925938
[10]    validation_0-auc:0.925844
[11]    validation_0-auc:0.925917
[12]    validation_0-auc:0.925995
[13]    validation_0-auc:0.926154
[14]    validation_0-auc:0.926022
[15]    validation_0-auc:0.926023
[16]    validation_0-auc:0.926052
[17]    validation_0-auc:0.926022
[18]    validation_0-auc:0.92585
Stopping. Best iteration:
[13]    validation_0-auc:0.926154

CPU times: user 3.22 s, sys: 18 ms, total: 3.23 s
Wall time: 3.23 s
0.8422222222222222
```

Run on eval and output csvs

```
# eval ids
```

```
def load_ids(file_path):
  with open(file_path) as f:
    raw_data = [int(line.split()[0]) for line in f]
  # print(raw_data)
  return raw_data
EVAL_IDS = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions
eval_ids = np.reshape(np.array(load_ids(EVAL_IDS),dtype=np.int32),(X_eval.shape[0],1))

preds = xgb_model.predict(X_eval)
preds[preds == -1] = 0

predictions = np.reshape(preds,(X_eval.shape[0],1))
print(predictions.shape)
print(predictions)
eval_out = np.hstack((eval_ids,predictions))
print(eval_out.shape)
print(eval_out)
eval_df = pd.DataFrame(data = eval_out,index = None,columns=['example_id','label'])
save_to_path = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis
eval_df.to_csv(path_or_buf=save_to_path,index=False)
```

```
(5250, 1)
[[1.]
 [0.]
 [1.]
 ...
 [1.]
 [0.]
 [0.]]
(5250, 2)
[[0.000e+00 1.000e+00]
 [1.000e+00 0.000e+00]
 [2.000e+00 1.000e+00]
 ...
 [5.247e+03 1.000e+00]
 [5.248e+03 0.000e+00]
 [5.249e+03 0.000e+00]]
```

Graphing feature importance... cool!

```
# requires graphviz and python-graphviz conda packages
import graphviz


xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42, eval_metric="auc")

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

xgb_model.fit(X_train, y_train, early_stopping_rounds=10, eval_set=[(X_test, y_test)], verbose

xgb.plot_importance(xgb_model)

# plot the output tree via matplotlib, specifying the ordinal number of the target tree
# xgb.plot_tree(xgb_model, num_trees=xgb_model.best_iteration)
```

```
# converts the target tree to a graphviz instance
xgb.to_graphviz(xgb_model, num_trees=xgb_model.best_iteration)
```