xgboost

February 5, 2020

Table of Contents

0.1

pima-indians-diabetes 8

[1]:
```
import pandas as pd
```

[2]:
```
data_dir = '../../../../Workspace/text_resources/datasets/pima-indians-diabetes.
↪csv'
```

[6]:
```
data_df = pd.read_csv(data_dir)
data_df.head()
```

[6]:
```
   pregnants  Plasma_glucose_concentration  blood_pressure  \
0          6                           148              72
1          1                            85              66
2          8                           183              64
3          1                            89              66
4          0                           137              40
```

```
   Triceps_skin_fold_thickness  serum_insulin   BMI  \
0                           35              0  33.6
1                           29              0  26.6
2                            0              0  23.3
3                           23             94  28.1
4                           35            168  43.1

   Diabetes_pedigree_function  Age  Target
0                       0.627   50       1
1                       0.351   31       0
2                       0.672   32       1
3                       0.167   21       0
4                       2.288   33       1
```

This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within

ve years.

Fields description follow:

pregnants = Number of times pregnant

Plasma_glucose_concentration = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

blood_pressure = Diastolic blood pressure (mm Hg)

Triceps_skin_fold_thickness = Triceps skin fold thickness (mm)

serum_insulin = 2-Hour serum insulin (mu U/ml)

BMI = Body mass index (weight in kg/(height in m)^2)

Diabetes_pedigree_function = Diabetes pedigree function

Age = Age (years)

Target = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

[9]:
```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnants                       768 non-null int64
Plasma_glucose_concentration    768 non-null int64
blood_pressure                  768 non-null int64
Triceps_skin_fold_thickness     768 non-null int64
serum_insulin                   768 non-null int64
BMI                             768 non-null float64
Diabetes_pedigree_function      768 non-null float64
Age                             768 non-null int64
```

```
Target                          768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

0.2

[11]:
```
from sklearn.model_selection import train_test_split
```

[18]:
```
X = data_df.iloc[:, 0:8]
y = data_df.iloc[:, 8]
```

[19]:
```
import time
```

[21]:
```
train_size = 0.7
seed = int(time.time())
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪train_size=train_size, random_state=seed)
```

0.3 : +

[30]:
```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score
```

[23]:
```
#
# n_estimators 100
# max_depth  3
# learning_rate  0.1
# objective  logloss
model = XGBClassifier()
model.fit(X_train, y_train)
```

[23]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=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='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[26]:
```
y_pred = model.predict(X_test)
y_pred
```

[26]:
```
array([0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
```

```
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1,
0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0])
```

[32]:
```
acc = accuracy_score(y_test, y_pred)
acc
```

[32]: 0.7489177489177489

[36]:
```
f1 = f1_score(y_test, y_pred)
f1
```

[36]: 0.6419753086419754

0.4

[37]:
```
eval_set = [(X_test, y_test)]
model = XGBClassifier()
#
# +     100     error
model.fit(X_train, y_train, eval_set=eval_set)
```

```
[0]     validation_0-error:0.290043
[1]     validation_0-error:0.285714
[2]     validation_0-error:0.255411
[3]     validation_0-error:0.255411
[4]     validation_0-error:0.255411
[5]     validation_0-error:0.264069
[6]     validation_0-error:0.25974
[7]     validation_0-error:0.264069
[8]     validation_0-error:0.268398
[9]     validation_0-error:0.264069
[10]    validation_0-error:0.268398
[11]    validation_0-error:0.25974
[12]    validation_0-error:0.264069
[13]    validation_0-error:0.251082
[14]    validation_0-error:0.251082
[15]    validation_0-error:0.251082
[16]    validation_0-error:0.242424
[17]    validation_0-error:0.242424
[18]    validation_0-error:0.251082
```

```
[19]	validation_0-error:0.242424
[20]	validation_0-error:0.233766
[21]	validation_0-error:0.233766
[22]	validation_0-error:0.238095
[23]	validation_0-error:0.238095
[24]	validation_0-error:0.242424
[25]	validation_0-error:0.238095
[26]	validation_0-error:0.238095
[27]	validation_0-error:0.242424
[28]	validation_0-error:0.242424
[29]	validation_0-error:0.238095
[30]	validation_0-error:0.246753
[31]	validation_0-error:0.242424
[32]	validation_0-error:0.242424
[33]	validation_0-error:0.242424
[34]	validation_0-error:0.246753
[35]	validation_0-error:0.242424
[36]	validation_0-error:0.242424
[37]	validation_0-error:0.242424
[38]	validation_0-error:0.238095
[39]	validation_0-error:0.233766
[40]	validation_0-error:0.233766
[41]	validation_0-error:0.233766
[42]	validation_0-error:0.238095
[43]	validation_0-error:0.238095
[44]	validation_0-error:0.238095
[45]	validation_0-error:0.233766
[46]	validation_0-error:0.229437
[47]	validation_0-error:0.233766
[48]	validation_0-error:0.242424
[49]	validation_0-error:0.242424
[50]	validation_0-error:0.242424
[51]	validation_0-error:0.233766
[52]	validation_0-error:0.229437
[53]	validation_0-error:0.229437
[54]	validation_0-error:0.233766
[55]	validation_0-error:0.242424
[56]	validation_0-error:0.246753
[57]	validation_0-error:0.251082
[58]	validation_0-error:0.251082
[59]	validation_0-error:0.246753
[60]	validation_0-error:0.246753
[61]	validation_0-error:0.246753
[62]	validation_0-error:0.251082
[63]	validation_0-error:0.251082
[64]	validation_0-error:0.255411
[65]	validation_0-error:0.255411
[66]	validation_0-error:0.255411
```

```
[67]	validation_0-error:0.255411
[68]	validation_0-error:0.251082
[69]	validation_0-error:0.251082
[70]	validation_0-error:0.251082
[71]	validation_0-error:0.251082
[72]	validation_0-error:0.251082
[73]	validation_0-error:0.251082
[74]	validation_0-error:0.251082
[75]	validation_0-error:0.251082
[76]	validation_0-error:0.25974
[77]	validation_0-error:0.25974
[78]	validation_0-error:0.25974
[79]	validation_0-error:0.25974
[80]	validation_0-error:0.25974
[81]	validation_0-error:0.25974
[82]	validation_0-error:0.25974
[83]	validation_0-error:0.25974
[84]	validation_0-error:0.25974
[85]	validation_0-error:0.25974
[86]	validation_0-error:0.25974
[87]	validation_0-error:0.255411
[88]	validation_0-error:0.251082
[89]	validation_0-error:0.251082
[90]	validation_0-error:0.251082
[91]	validation_0-error:0.251082
[92]	validation_0-error:0.255411
[93]	validation_0-error:0.255411
[94]	validation_0-error:0.255411
[95]	validation_0-error:0.251082
[96]	validation_0-error:0.251082
[97]	validation_0-error:0.251082
[98]	validation_0-error:0.251082
[99]	validation_0-error:0.251082
```

[37]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=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='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

100 error

[42]:
```
eval_set = [(X_test, y_test)]
model = XGBClassifier()
model.fit(X_train, y_train, eval_set=eval_set, eval_metric="logloss")
```

```
[0]	validation_0-logloss:0.661799
[1]	validation_0-logloss:0.638594
[2]	validation_0-logloss:0.614165
[3]	validation_0-logloss:0.596266
[4]	validation_0-logloss:0.581508
[5]	validation_0-logloss:0.569146
[6]	validation_0-logloss:0.555601
[7]	validation_0-logloss:0.548347
[8]	validation_0-logloss:0.5407
[9]	validation_0-logloss:0.533629
[10]	validation_0-logloss:0.527112
[11]	validation_0-logloss:0.523614
[12]	validation_0-logloss:0.518502
[13]	validation_0-logloss:0.515161
[14]	validation_0-logloss:0.512742
[15]	validation_0-logloss:0.512369
[16]	validation_0-logloss:0.509281
[17]	validation_0-logloss:0.505474
[18]	validation_0-logloss:0.504088
[19]	validation_0-logloss:0.503247
[20]	validation_0-logloss:0.501015
[21]	validation_0-logloss:0.498152
[22]	validation_0-logloss:0.497204
[23]	validation_0-logloss:0.496385
[24]	validation_0-logloss:0.496324
[25]	validation_0-logloss:0.493388
[26]	validation_0-logloss:0.493843
[27]	validation_0-logloss:0.492685
[28]	validation_0-logloss:0.494117
[29]	validation_0-logloss:0.493605
[30]	validation_0-logloss:0.493431
[31]	validation_0-logloss:0.493073
[32]	validation_0-logloss:0.491477
[33]	validation_0-logloss:0.492211
[34]	validation_0-logloss:0.492007
[35]	validation_0-logloss:0.489295
[36]	validation_0-logloss:0.489829
[37]	validation_0-logloss:0.490244
[38]	validation_0-logloss:0.489837
[39]	validation_0-logloss:0.490538
[40]	validation_0-logloss:0.491024
[41]	validation_0-logloss:0.491747
[42]	validation_0-logloss:0.491212
[43]	validation_0-logloss:0.491711
[44]	validation_0-logloss:0.492628
[45]	validation_0-logloss:0.492661
[46]	validation_0-logloss:0.492572
[47]	validation_0-logloss:0.495079
```

```
[48]	validation_0-logloss:0.495767
[49]	validation_0-logloss:0.49626
[50]	validation_0-logloss:0.499096
[51]	validation_0-logloss:0.498114
[52]	validation_0-logloss:0.498173
[53]	validation_0-logloss:0.498107
[54]	validation_0-logloss:0.498787
[55]	validation_0-logloss:0.499905
[56]	validation_0-logloss:0.501356
[57]	validation_0-logloss:0.501158
[58]	validation_0-logloss:0.502634
[59]	validation_0-logloss:0.503527
[60]	validation_0-logloss:0.503598
[61]	validation_0-logloss:0.503445
[62]	validation_0-logloss:0.503634
[63]	validation_0-logloss:0.504414
[64]	validation_0-logloss:0.504526
[65]	validation_0-logloss:0.505095
[66]	validation_0-logloss:0.506114
[67]	validation_0-logloss:0.506205
[68]	validation_0-logloss:0.506217
[69]	validation_0-logloss:0.506044
[70]	validation_0-logloss:0.505599
[71]	validation_0-logloss:0.506433
[72]	validation_0-logloss:0.507073
[73]	validation_0-logloss:0.507484
[74]	validation_0-logloss:0.509158
[75]	validation_0-logloss:0.509098
[76]	validation_0-logloss:0.509252
[77]	validation_0-logloss:0.50795
[78]	validation_0-logloss:0.508793
[79]	validation_0-logloss:0.508761
[80]	validation_0-logloss:0.509851
[81]	validation_0-logloss:0.510714
[82]	validation_0-logloss:0.510132
[83]	validation_0-logloss:0.510497
[84]	validation_0-logloss:0.510498
[85]	validation_0-logloss:0.510762
[86]	validation_0-logloss:0.512082
[87]	validation_0-logloss:0.513018
[88]	validation_0-logloss:0.513076
[89]	validation_0-logloss:0.512327
[90]	validation_0-logloss:0.512549
[91]	validation_0-logloss:0.512883
[92]	validation_0-logloss:0.512052
[93]	validation_0-logloss:0.513347
[94]	validation_0-logloss:0.514289
[95]	validation_0-logloss:0.51473
```

```
[96]	validation_0-logloss:0.515996
[97]	validation_0-logloss:0.515763
[98]	validation_0-logloss:0.514425
[99]	validation_0-logloss:0.51542
```

[42]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=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='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

0.5 early_stop

[43]:
```python
eval_set = [(X_test, y_test)]
model = XGBClassifier()
#  early_stopping_rounds
#      early_stopping_rounds     error
model.fit(X_train, y_train, eval_set=eval_set, eval_metric="logloss", early_stopping_rounds=5)
```

```
[0]	validation_0-logloss:0.661799
Will train until validation_0-logloss hasn't improved in 5 rounds.
[1]	validation_0-logloss:0.638594
[2]	validation_0-logloss:0.614165
[3]	validation_0-logloss:0.596266
[4]	validation_0-logloss:0.581508
[5]	validation_0-logloss:0.569146
[6]	validation_0-logloss:0.555601
[7]	validation_0-logloss:0.548347
[8]	validation_0-logloss:0.5407
[9]	validation_0-logloss:0.533629
[10]	validation_0-logloss:0.527112
[11]	validation_0-logloss:0.523614
[12]	validation_0-logloss:0.518502
[13]	validation_0-logloss:0.515161
[14]	validation_0-logloss:0.512742
[15]	validation_0-logloss:0.512369
[16]	validation_0-logloss:0.509281
[17]	validation_0-logloss:0.505474
[18]	validation_0-logloss:0.504088
[19]	validation_0-logloss:0.503247
[20]	validation_0-logloss:0.501015
[21]	validation_0-logloss:0.498152
[22]	validation_0-logloss:0.497204
```

```
[23]	validation_0-logloss:0.496385
[24]	validation_0-logloss:0.496324
[25]	validation_0-logloss:0.493388
[26]	validation_0-logloss:0.493843
[27]	validation_0-logloss:0.492685
[28]	validation_0-logloss:0.494117
[29]	validation_0-logloss:0.493605
[30]	validation_0-logloss:0.493431
[31]	validation_0-logloss:0.493073
[32]	validation_0-logloss:0.491477
[33]	validation_0-logloss:0.492211
[34]	validation_0-logloss:0.492007
[35]	validation_0-logloss:0.489295
[36]	validation_0-logloss:0.489829
[37]	validation_0-logloss:0.490244
[38]	validation_0-logloss:0.489837
[39]	validation_0-logloss:0.490538
[40]	validation_0-logloss:0.491024
Stopping. Best iteration:
[35]	validation_0-logloss:0.489295
```

[43]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=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='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

logloss

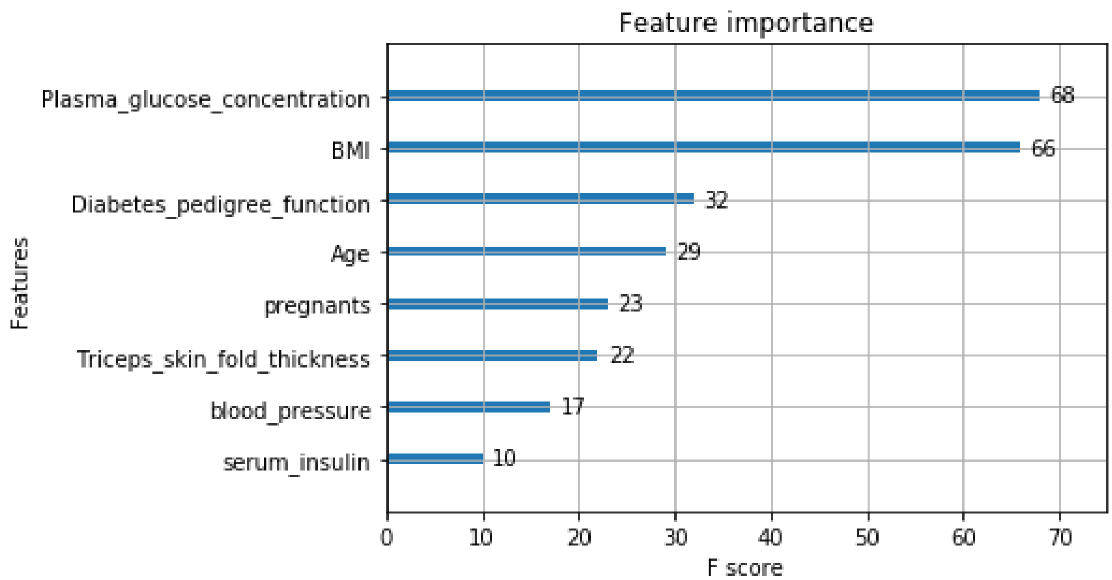
0.6

[44]:
```
from xgboost import plot_importance
```

[45]:
```
%matplotlib inline
```

[46]:
```
plot_importance(model)
```

[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e4f9e10>
```



0.7

XGBOOST + learning_rate(0.1): + n_estimators(100): + max_depth(3): + colsample_bytree(1): + colsample_bylevel(1): + colsamole_bynode(1): + subsample(1):

`colsample_bytree, colsample_bylevel, colsamole_bynode, subsample`

0.7.1

[47]:
```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

[49]:
```python
model = XGBClassifier()
learning_rates = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
param = dict(learning_rate=learning_rates)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=int(time.
 ↪time()))
grid_search = GridSearchCV(model, param_grid=param, scoring="neg_log_loss",␣
 ↪cv=kfold)
ret = grid_search.fit(X, y)
```

[51]:
```python
#
ret.best_score_, ret.best_params_
```

[51]: (-0.48532649107437464, {'learning_rate': 0.1})

[57]:
```
#     10
means = ret.cv_results_["mean_test_score"]
stds = ret.cv_results_["std_test_score"]
params = ret.cv_results_["params"]
for mean, std, param in zip(means, stds, params):
    print(mean, std, param)
```

```
-0.6897171731106937 0.00040455031056076216 {'learning_rate': 0.0001}
-0.6613499955274165 0.0035903914452080723 {'learning_rate': 0.001}
-0.530141705297865 0.020407630471569333 {'learning_rate': 0.01}
-0.48532649107437464 0.04189947381838934 {'learning_rate': 0.1}
-0.5348682376841983 0.05535405809252925 {'learning_rate': 0.2}
-0.5649503417167333 0.06831279591078589 {'learning_rate': 0.3}
```

0.7.2 subsample

[58]:
```
model = XGBClassifier()
subsample_ratio = [1, 0.9, 0.8, 0.7]
colsample_bytree_ratio = [1, 0.9, 0.8, 0.7]
param = dict(subsample=subsample_ratio, colsample_bytree=colsample_bytree_ratio)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=int(time.
 ↪time()))
grid_search = GridSearchCV(model, param_grid=param, scoring="neg_log_loss",␣
 ↪cv=kfold)
ret = grid_search.fit(X, y)
```

[59]:
```
#
ret.best_score_, ret.best_params_
```

[59]: (-0.48961818076228764, {'colsample_bytree': 0.7, 'subsample': 1})

[60]:
```
#     k
means = ret.cv_results_["mean_test_score"]
stds = ret.cv_results_["std_test_score"]
params = ret.cv_results_["params"]
for mean, std, param in zip(means, stds, params):
    print(mean, std, param)
```

```
-0.4911601440129137 0.07166828259954668 {'colsample_bytree': 1, 'subsample': 1}
-0.5008964773966605 0.08026046435687785 {'colsample_bytree': 1, 'subsample':
0.9}
-0.5036791476286453 0.08576088331871805 {'colsample_bytree': 1, 'subsample':
0.8}
-0.509751289907399 0.09280578362748748 {'colsample_bytree': 1, 'subsample': 0.7}
```

```
-0.4938803892143066 0.07408538527055748 {'colsample_bytree': 0.9, 'subsample':
1}
-0.49784539641192777 0.07932142213419963 {'colsample_bytree': 0.9, 'subsample':
0.9}
-0.49807773682338546 0.08210942595143153 {'colsample_bytree': 0.9, 'subsample':
0.8}
-0.5061238677996395 0.08982376566148907 {'colsample_bytree': 0.9, 'subsample':
0.7}
-0.4927833476416102 0.06782473665570085 {'colsample_bytree': 0.8, 'subsample':
1}
-0.4933515631443394 0.07927876748214507 {'colsample_bytree': 0.8, 'subsample':
0.9}
-0.5015384495694283 0.08551004472870154 {'colsample_bytree': 0.8, 'subsample':
0.8}
-0.5046200485812733 0.08461528014070543 {'colsample_bytree': 0.8, 'subsample':
0.7}
-0.48961818076228764 0.07460728597345921 {'colsample_bytree': 0.7, 'subsample':
1}
-0.4947519394612148 0.07510347017620993 {'colsample_bytree': 0.7, 'subsample':
0.9}
-0.5028556188814642 0.08665319459334589 {'colsample_bytree': 0.7, 'subsample':
0.8}
-0.496187441622169 0.0803519692567954 {'colsample_bytree': 0.7, 'subsample':
0.7}
```

0.8

- Kaggle xgboost
- xgboost sklearn API
-

```
[ ]:
```