eXtreme Gradient Boosted Trees

YoungWoong Cho

July 2020

In this project, xTreme gradient boosted trees algorithm is used to train the model. Among several parameters that can be manually chosen(eta, gamma, lambda, alpha, max_depth and more), only lambda is optimized in this project for simplicity.

Dataset : Breast Cancer Wisconsin

Source : https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Features : ID Number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses

Label : Class

0.1 Preliminary works

1. Import libraries

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
```

2. Get a dataset

[ ]:
```
from google.colab import files
data_to_load = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving breast-cancer-wisconsin.csv to breast-cancer-wisconsin.csv
```

[ ]:
```
data = pd.read_csv("breast-cancer-wisconsin.csv").dropna()
data.columns = ['ID Number', 'Clump Thickness', 'Uniformity of Cell Size',␣
 ↪'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size'␣
 ↪, 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class']
data['Class'] = data['Class'].replace(2,0).replace(4,1) #Replace 2(benign) and␣
 ↪4(malignant) into 0(benign) and 1(malignant)
```

3. Create train, test and validation sets

[ ]:
```python
X = data.iloc[:, 1:-1]
y = data.iloc[:, -1]

train_X, test_X, train_Y, test_Y = train_test_split(X, y, test_size=0.2,␣
↪random_state=1)
val_X, test_X, val_Y, test_Y = train_test_split(test_X, test_Y, test_size=0.5,␣
↪random_state=1)

print("Data length: %d, Training data length: %d, Validation data length: %d,␣
↪Testing data length: %d" %(len(X), len(train_X), len(val_X), len(test_X)))

# Create DMatrix
dtrain = xgb.DMatrix(train_X, label=train_Y)
dval = xgb.DMatrix(val_X, label=val_Y)
dtest = xgb.DMatrix(test_X, label=test_Y)
```

```
Data length: 682, Training data length: 545, Validation data length: 68, Testing
data length: 69
```

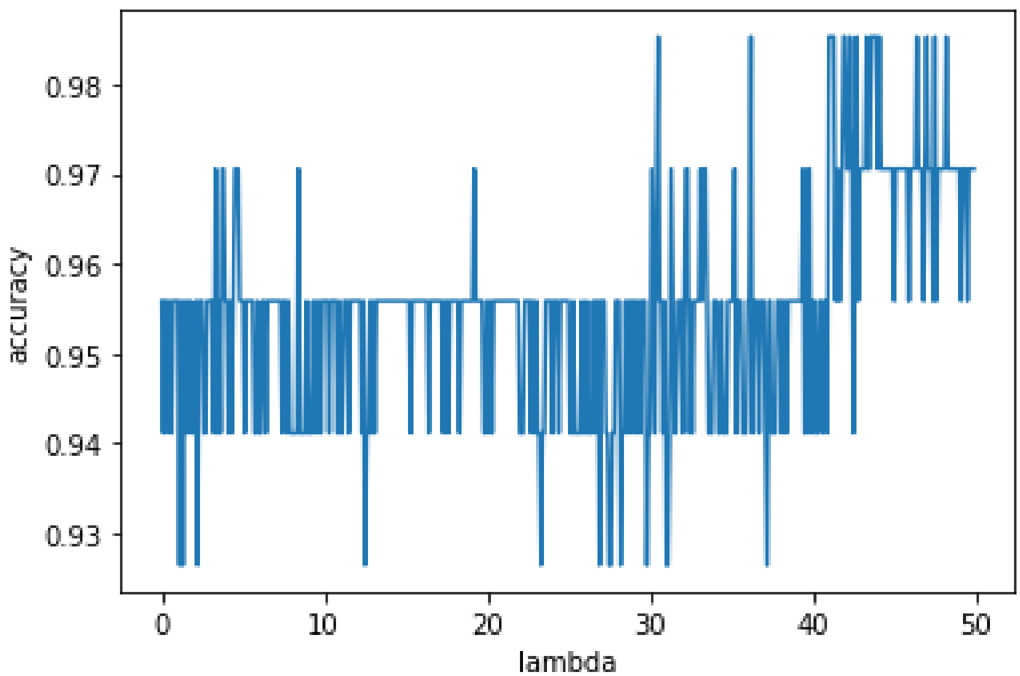
4. Define functions for optimal parameter selection

[ ]:
```python
# Chose best lambda
def val_accuracy_lamb(dtrain, dval, lamb, num_round):
  """
  Input:
    dtrain: training DMatrix
    dval: validation DMatrix
    eta: learning rate
    lamb: L2 penalizing tuning parameter
  Output:
    accuracy score between predicted and true label of validation dataset
  """
  param = {'booster':'gbtree', 'max_depth':2, 'lambda':lamb, 'objective':'binary:
↪hinge'}
  num = num_round
  bst = xgb.train(param, dtrain, num)
  preds = bst.predict(dval)
  return accuracy_score(val_Y, preds)
```

[ ]:
```python
lambs = np.arange(0, 50, 0.1)

num_round = 50

accuracy = []

#for eta in etas:
```

```
for lamb in lambs:
  accuracy.append(val_accuracy_lamb(dtrain, dval, lamb, num_round))
```

[ ]:
```
plt.xlabel('lambda')
plt.ylabel('accuracy')
plt.plot(lambs,np.asarray(accuracy))
plt.show()

print("Best lambda: %.2f with accuracy: %.2f%%" %(lambs[accuracy.
 ↪index(max(accuracy))],max(accuracy)*100))
```

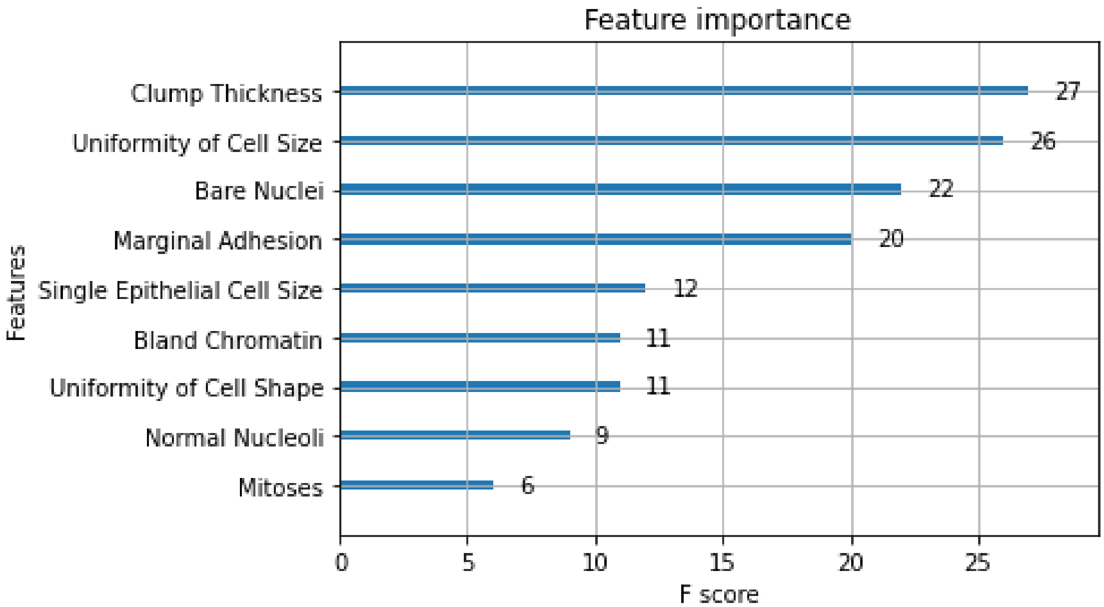



```
Best lambda: 30.50 with accuracy: 98.53%
```

0.2 XGBoost algorithm

[ ]:
```
# Train the algorithm with the optimal lambda value and plot the feature␣
 ↪importance
param = {'booster':'gbtree', 'max_depth':2, 'lambda':lambs[accuracy.
 ↪index(max(accuracy))], 'objective':'binary:hinge'}
num = 50
bst = xgb.train(param, dtrain, num)
xgb.plot_importance(bst)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f21a468fcf8>
```



[ ]:
```
# Calculate the accuracy of this model using test dataset
preds = bst.predict(dtest)
accuracy = accuracy_score(test_Y, preds)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 94.20%
```