Assignment DSM ML 6

```
In this assignment students need to predict whether a person makes over 50K per year
or not from classic adult dataset using XGBoost. The description of the dataset is as
follows:

Data Set Information:
Extraction was done by Barry Becker from the 1994 Census database. A set of
reasonably clean records was extracted using the following conditions: ((AAGE>16) &&
(AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Attribute Information:
Listing of attributes:
>50K, <=50K.
age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov,
Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc,
9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed,
Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial,
Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing,
Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany,
Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras,
Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France,
Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala,
Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong,
Holand-Netherlands.
```

In [1]:

```
#Import libraries:
import pandas as pd
import numpy as np
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn import cross_validation, metrics   #Additional scklearn functions
from sklearn.grid_search import GridSearchCV   #Perforing grid search

import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 4
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of t
he model_selection module into which all the refactored classes and function
s are moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: Depre
cationWarning: This module was deprecated in version 0.18 in favor of the mo
del_selection module into which all the refactored classes and functions are
moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [2]:

```

train_set =pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adul

test_set = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adul

```

In [3]:

```
col_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
              'marital_status','occupation','relationship', 'race', 'sex',
              'capital_gain', 'capital_loss', 'hours_per_week',
'native_country', 'wage_class']
train_set.columns = col_labels
test_set.columns = col_labels
```

In [4]:

```
train_set.head()
```

Out[4]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife

In [5]:

```
test_set.head()
```

Out[5]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband
4	18	?	103497	Some-college	10	Never-married	?	Own-child

In [6]:

```
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age               32561 non-null int64
workclass         32561 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education_num     32561 non-null int64
marital_status    32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital_gain      32561 non-null int64
capital_loss      32561 non-null int64
hours_per_week    32561 non-null int64
native_country    32561 non-null object
wage_class        32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [7]:

```
train_set.replace(' ?', np.nan).dropna().shape
```

Out[7]:

```
(30162, 15)
```

In [8]:

```
test_set.replace(' ?', np.nan).dropna().shape
```

Out[8]:

```
(15060, 15)
```

In [9]:

```
train_no_missing = train_set.replace(' ?', np.nan).dropna()
test_no_missing = test_set.replace(' ?', np.nan).dropna()
```

In [10]:

```
train_no_missing.head()
```

Out[10]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife

In [11]:

```
test_no_missing.head()
```

Out[11]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family

In [12]:

```
test_no_missing['wage_class'] = test_no_missing.wage_class.replace({' <=50K.': ' <=50K', '
```

In [13]:

```
#Checking the unique values from each set, we can see if they now match.

test_no_missing.wage_class.unique()
```

Out[13]:

```
array([' <=50K', ' >50K'], dtype=object)
```

In [14]:

```
train_no_missing.wage_class.unique()
```

Out[14]:

```
array([' <=50K', ' >50K'], dtype=object)
```

Applying Ordinal Encoding to Categoricals

In [15]:

```
#First, combine them together into a single dataset.
```

In [16]:

```
combined_set = pd.concat([train_no_missing, test_no_missing], axis = 0) # Stacks them verti
```

In [17]:

```
combined_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45222 entries, 0 to 16280
Data columns (total 15 columns):
age               45222 non-null int64
workclass         45222 non-null object
fnlwgt            45222 non-null int64
education         45222 non-null object
education_num     45222 non-null int64
marital_status    45222 non-null object
occupation        45222 non-null object
relationship      45222 non-null object
race              45222 non-null object
sex               45222 non-null object
capital_gain      45222 non-null int64
capital_loss      45222 non-null int64
hours_per_week    45222 non-null int64
native_country    45222 non-null object
wage_class        45222 non-null object
dtypes: int64(6), object(9)
memory usage: 5.5+ MB
```

In [18]:

```
##Next, if the feature is not already numerical, we need to encode it as one. We can use pa
```

In [19]:

```
for feature in combined_set.columns: # Loop through all columns in the dataframe
    if combined_set[feature].dtype == 'object': # Only apply for columns with categorical s
        combined_set[feature] = pd.Categorical(combined_set[feature]).codes # Replace strir
```

In [20]:

```
combined_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45222 entries, 0 to 16280
Data columns (total 15 columns):
age               45222 non-null int64
workclass         45222 non-null int8
fnlwgt            45222 non-null int64
education         45222 non-null int8
education_num     45222 non-null int64
marital_status    45222 non-null int8
occupation        45222 non-null int8
relationship      45222 non-null int8
race              45222 non-null int8
sex               45222 non-null int8
capital_gain      45222 non-null int64
capital_loss      45222 non-null int64
hours_per_week    45222 non-null int64
native_country    45222 non-null int8
wage_class        45222 non-null int8
dtypes: int64(6), int8(9)
memory usage: 2.8 MB
```

In [21]:

```
combined_set.head()
```

Out[21]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	39	5	77516	9	13	4	0	1
1	50	4	83311	9	13	2	3	0
2	38	2	215646	11	9	0	5	1
3	53	2	234721	1	7	2	5	0
4	28	2	338409	9	13	2	9	5

In [22]:

```
# Now split these back into their original train/test sizes
```

In [23]:

```
final_train = combined_set[:train_no_missing.shape[0]] # Up to the last initial training se
final_test = combined_set[train_no_missing.shape[0]:] # Past the last initial training set
```

In [24]:

```
#We can now finally start playing around with XGBoost.
```

Initial Model Setup and Grid Search

In [25]:

```
y_train = final_train.pop('wage_class')
y_test = final_test.pop('wage_class')
```

In [26]:

```
#Now import the libraries we will need to do grid search for XGBoost.
```

In [27]:

```
import xgboost as xgb
from sklearn.grid_search import GridSearchCV
```

In [28]:

```
cv_params = {'max_depth': [3,5,7], 'min_child_weight': [1,3,5]}
ind_params = {'learning_rate': 0.1, 'n_estimators': 1000, 'seed':0, 'subsample': 0.8, 'cols
             'objective': 'binary:logistic'}
optimized_GBM = GridSearchCV(xgb.XGBClassifier(**ind_params),
                            cv_params,
                             scoring = 'accuracy', cv = 5, n_jobs = -1)
# Optimize for accuracy since that is the metric used in the Adult Data Set notation
```

In [29]:

```
# Run our grid search with 5-fold cross-validation and see which parameters perform the bes
```

In [30]:

```
optimized_GBM.fit(final_train, y_train)
```

Out[30]:

```
GridSearchCV(cv=5, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=0, silent=True,
       subsample=0.8),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
```

In [31]:

```
optimized_GBM.grid_scores_
```

Out[31]:

```
[mean: 0.86712, std: 0.00225, params: {'max_depth': 3, 'min_child_weight':
1},
 mean: 0.86659, std: 0.00339, params: {'max_depth': 3, 'min_child_weight':
3},
 mean: 0.86659, std: 0.00295, params: {'max_depth': 3, 'min_child_weight':
5},
 mean: 0.86214, std: 0.00197, params: {'max_depth': 5, 'min_child_weight':
1},
 mean: 0.86161, std: 0.00143, params: {'max_depth': 5, 'min_child_weight':
3},
 mean: 0.86208, std: 0.00236, params: {'max_depth': 5, 'min_child_weight':
5},
 mean: 0.85651, std: 0.00183, params: {'max_depth': 7, 'min_child_weight':
1},
 mean: 0.85575, std: 0.00246, params: {'max_depth': 7, 'min_child_weight':
3},
 mean: 0.85694, std: 0.00347, params: {'max_depth': 7, 'min_child_weight':
5}]
```

In [32]:

```
## Let's try optimizing some other hyperparameters now to see if we can beat a mean of 86.
## accuracy. This time, we will play around with subsampling along with lowering the learn
## rate to see if that helps.
```

In [33]:

```
cv_params = {'learning_rate': [0.1, 0.01], 'subsample': [0.7,0.8,0.9]}
ind_params = {'n_estimators': 1000, 'seed':0, 'colsample_bytree': 0.8,
             'objective': 'binary:logistic', 'max_depth': 3, 'min_child_weight': 1}


optimized_GBM = GridSearchCV(xgb.XGBClassifier(**ind_params),
                            cv_params,
                             scoring = 'accuracy', cv = 5, n_jobs = -1)
optimized_GBM.fit(final_train, y_train)
```

Out[33]:

```
GridSearchCV(cv=5, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=0, silent=True,
       subsample=1),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'learning_rate': [0.1, 0.01], 'subsample': [0.7, 0.8, 0.
9]},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
```

In [34]:

```
optimized_GBM.grid_scores_
```

Out[34]:

```
[mean: 0.86622, std: 0.00198, params: {'learning_rate': 0.1, 'subsample': 0.
7},
 mean: 0.86712, std: 0.00225, params: {'learning_rate': 0.1, 'subsample': 0.
8},
 mean: 0.86758, std: 0.00299, params: {'learning_rate': 0.1, 'subsample': 0.
9},
 mean: 0.86052, std: 0.00290, params: {'learning_rate': 0.01, 'subsample':
0.7},
 mean: 0.86029, std: 0.00297, params: {'learning_rate': 0.01, 'subsample':
0.8},
 mean: 0.86025, std: 0.00341, params: {'learning_rate': 0.01, 'subsample':
0.9}]
```

we have just got bit of improvement in accuracy at 86.77%

we can try to optimize a little further by utilizing XGBoost’s built-in cv which allows early stopping to prevent overfitting.

```
**Early stopping CV**</br>
Based on the CV testing performed earlier, we want to utilize the following parameters:

Learning_rate (eta) = 0.1
Subsample, colsample_bytree = 0.8
Max_depth = 3
Min_child_weight = 1
```

In [37]:

```
## To increase the performance of XGBoost’s speed through many iterations of the training s
## and since we are using only XGBoost’s API and not sklearn’s anymore, we can create
## a DMatrix. This sorts the data initially to optimize for XGBoost when it builds trees,
## making the algorithm more efficient. This is especially helpful when you have a very lar
##number of training examples. To create a DMatrix:
```

In [38]:

```
xgdmat = xgb.DMatrix(final_train, y_train) # Create our DMatrix to make XGBoost more effici
```

In [39]:

```
our_params = {'eta': 0.1, 'seed':0, 'subsample': 0.8, 'colsample_bytree': 0.8,
             'objective': 'binary:logistic', 'max_depth':3, 'min_child_weight':1}
# Grid Search CV optimized settings

cv_xgb = xgb.cv(params = our_params, dtrain = xgdmat, num_boost_round = 3000, nfold = 5,
                metrics = ['error'], # Make sure you enter metrics inside a list or you may
                early_stopping_rounds = 100) # Look for early stopping that minimizes error
```

```
[00:45:50] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:45:51] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
```

In [40]:

```
cv_xgb.tail(5)
```

Out[40]:

	train-error-mean	train-error-std	test-error-mean	test-error-std
489	0.115584	0.001395	0.129998	0.004691
490	0.115617	0.001414	0.129965	0.004479
491	0.115501	0.001397	0.129965	0.004494
492	0.115501	0.001483	0.129964	0.004581
493	0.115385	0.001528	0.129932	0.004657

In [41]:

```
our_params = {'eta': 0.1, 'seed':0, 'subsample': 0.8, 'colsample_bytree': 0.8,
              'objective': 'binary:logistic', 'max_depth':3, 'min_child_weight':1}

final_gb = xgb.train(our_params, xgdmat, num_boost_round = 432)
```

```
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[00:47:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
```

In [42]:

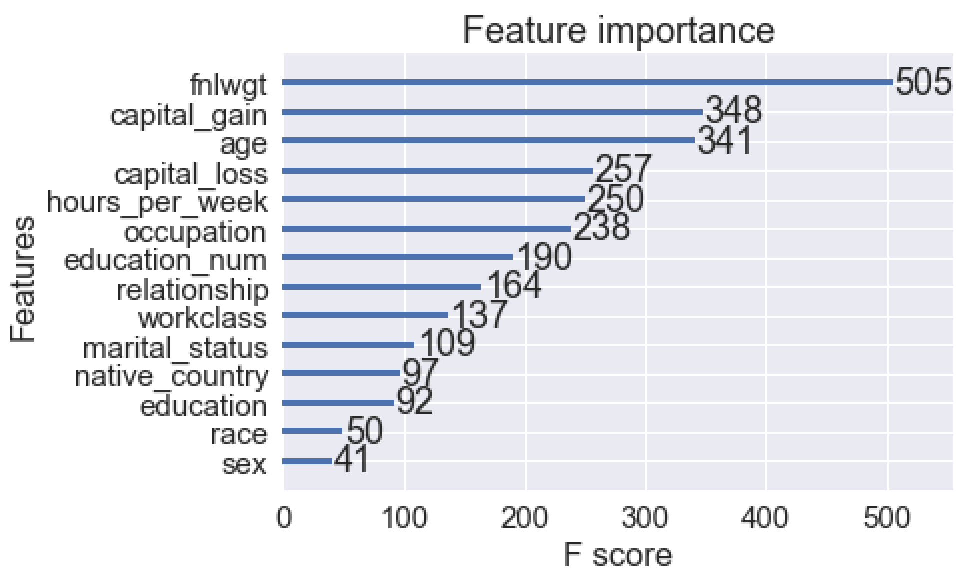
```
%matplotlib inline
import seaborn as sns
sns.set(font_scale = 1.5)
```

In [43]:

```
xgb.plot_importance(final_gb)
```

Out[43]:

<matplotlib.axes._subplots.AxesSubplot at 0x535aba8>



In [44]:

```
importances = final_gb.get_fscore()
importances
```

Out[44]:

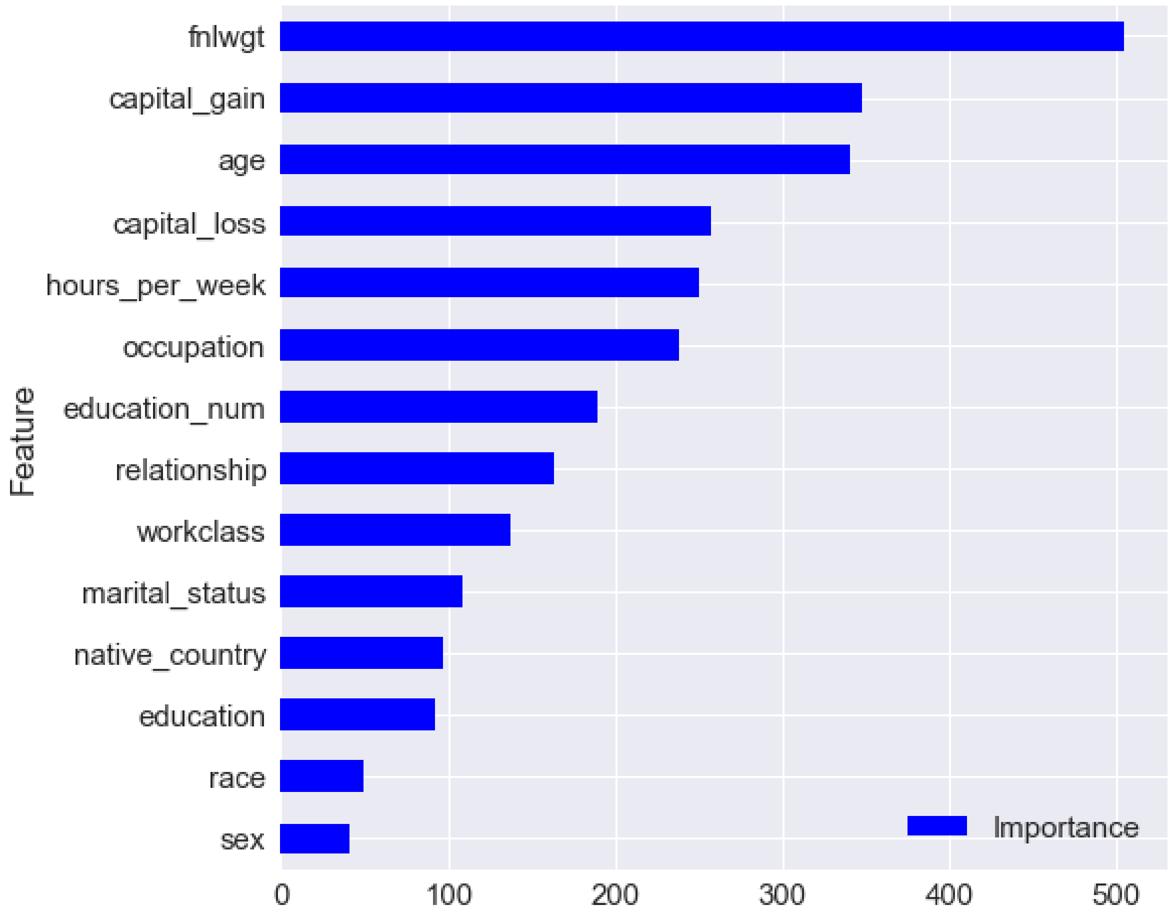
```
{'relationship': 164,
 'capital_gain': 348,
 'education': 92,
 'fnlwgt': 505,
 'marital_status': 109,
 'education_num': 190,
 'capital_loss': 257,
 'age': 341,
 'hours_per_week': 250,
 'occupation': 238,
 'workclass': 137,
 'sex': 41,
 'native_country': 97,
 'race': 50}
```

In [46]:

```
importance_frame = pd.DataFrame({'Importance': list(importances.values()), 'Feature': list(
importance_frame.sort_values(by = 'Importance', inplace = True)
importance_frame.plot(kind = 'barh', x = 'Feature', figsize = (8,8), color = 'blue')
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xbc8bbe0>
```



Analyzing Performance on Test Data

In [47]:

```
testdmat = xgb.DMatrix(final_test)
```

In [48]:

```
## let’s use sklearn’s accuracy metric to see how well we did on the test set.
```

In [49]:

```
from sklearn.metrics import accuracy_score
y_pred = final_gb.predict(testdmat) # Predict using our testdmat
y_pred
```

Out[49]:

```
array([0.00279659, 0.20289436, 0.2911482 , ..., 0.84031725, 0.12937884,
       0.774844  ], dtype=float32)
```

Setting probability t0 0.5

In [51]:

```
y_pred[y_pred > 0.5] = 1
y_pred[y_pred <= 0.5] = 0
y_pred
```

Out[51]:

```
array([0., 0., 0., ..., 1., 0., 1.], dtype=float32)
```

calculate accuracy

In [53]:

```
accuracy_score(y_pred, y_test), 1-accuracy_score(y_pred, y_test)
```

Out[53]:

```
(0.8685258964143426, 0.13147410358565736)
```

In [54]:

```
##Our final accuracy is 86.86%, or a 13.13% error rate. We beat our goal by a whole percent
```