OS

September 12, 2023

[1]:
```python
import sys
## import all the packages needed
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
```

[2]:
```python
## read NHANES dataset
df = pd.read_csv('/Users/zhiyi/Desktop/for Yupei/OS.csv')
```

[3]:
```python
## find all the distinct values of os
df.os.value_counts()
```

[3]:
```
0    16745
1     4792
Name: os, dtype: int64
```

[4]:
```python
## data preparation
# exclude null values and NA
df = df[df.os.notnull()]
# check os
df.os.describe()
```

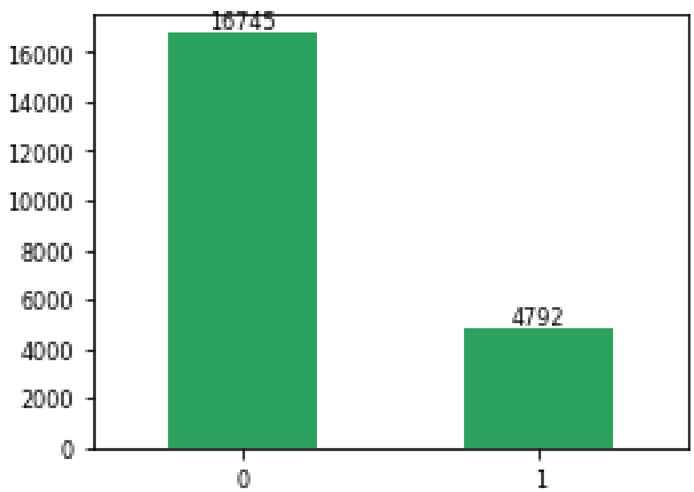
[4]:
```
count    21537.000000
mean         0.222501
std          0.415935
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: os, dtype: float64
```

[5]:
```python
# exclude non-numeric values
d = df.select_dtypes(['number'])
```

```
# exclude columns that have over 50% NaN
d = d.dropna(thresh = 0.5*len(d), axis =1)

print(len(d.columns), 'columns are left')
```

21 columns are left

[6]:
```
## plot the distribution of values of response variable
vals = d.os.value_counts()

plt.figure(figsize=(4,3))
plt.rc('font', size=8)

ax = vals.plot.bar(rot=0, color='#2ca25f')
for i in range(len(vals)):
    ax.annotate(vals[i], xy=[vals.index[i], vals[i]], ha='center', va='bottom')
```



[7]:
```
# replace NA with most frequent values
from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(strategy='most_frequent')

## show the complete dataset
d = pd.DataFrame(imp_mode.fit_transform(d), columns=d.columns)
d
```

[7]:

	os	gender	race	age	size	marry	income	site	grade	kind	…	N	\
0	1	2	2	3	3	1	3	1	5	1	…	3	
1	0	1	1	3	1	3	3	2	2	1	…	2	
2	0	2	1	3	3	2	3	2	2	1	…	1	
3	0	1	3	3	1	3	3	1	5	1	…	1	

```
4      0      1     2    3    3     2     3    1     2    1  …  1
…     ..     …    …   …    …    …    …    …    …    …   …  ..
21532  0      1     1    2    1     2     1    1     5    1  …  3
21533  0      2     1    2    3     3     2    1     5    1  …  3
21534  1      2     1    3    3     3     2    1     5    1  …  1
21535  1      2     1    3    2     3     2    2     2    1  …  1
21536  1      1     2    3    3     2     2    2     5    1  …  3

       surgery_pri  RX_Summ  radiate  chem  CEA  bone  brain  lung  group
0                0        0        0     0    1     0      0     1      3
1                1        0        0     0    0     0      0     0      1
2                0        0        0     1    1     0      0     0      3
3                0        0        0     0    1     0      0     1      3
4                0        0        0     1    2     0      0     1      3
…              …       …       …    …   …    …     …    …     …
21532            0        0        1     1    1     0      0     0      3
21533            1        0        0     0    2     0      0     0      3
21534            0        0        0     0    2     0      0     0      3
21535            1        0        0     0    1     0      0     1      1
21536            0        0        0     0    1     0      0     1      3

[21537 rows x 21 columns]
```

[8]:
```python
## separate predictors and responses
X = d.loc[:, d.columns != 'os']
y = d.os
print('X shape:', X.shape)
print('y shape:', y.shape)
```

```
X shape: (21537, 20)
y shape: (21537,)
```

[9]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=12)
```

[10]:
```python
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

model = XGBClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
def confusion(y_test, y_pred):
    conf = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['True[0]',␣
 ↪'True[1]'], columns=['Predict[0]', 'Predict[1]'])
    print('Confusion Matrix:')
    print(conf)
    return conf

confusion(y_test, y_pred)
```

```
Accuracy: 80.97%
Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        3094         278
True[1]         542         394
```

[10]:

	Predict[0]	Predict[1]
True[0]	3094	278
True[1]	542	394

[11]:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=12)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
X_test_sm, y_test_sm = smote.fit_resample(X_test, y_test)

X_train_sm = pd.DataFrame(X_train_sm, columns=X.columns)
X_test_sm = pd.DataFrame(X_test_sm, columns=X.columns)

print(y_train_sm.value_counts())
print(y_test_sm.value_counts())
```

```
1    13373
0    13373
Name: os, dtype: int64
0    3372
1    3372
Name: os, dtype: int64
```

[12]:

```
# After oversampling, the classification result is more reasonable.
model = XGBClassifier()
model.fit(X_train_sm, y_train_sm)
y_pred_sm = model.predict(X_test_sm)

accuracy = accuracy_score(y_test_sm, y_pred_sm)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
conf = pd.DataFrame(confusion_matrix(y_test_sm, y_pred_sm), index=['True[0]',␣
 ↪'True[1]'], columns=['Predict[0]', 'Predict[1]'])
```

```
conf
```

Accuracy: 76.90%

[12]:

	Predict[0]	Predict[1]
True[0]	2704	668
True[1]	890	2482

[13]:
```
X_scale = d.loc[:, d.columns != 'os']
y = d.os
```

[14]:
```
## min-max scaling
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
X = pd.DataFrame(minmax.fit_transform(X_scale), columns=X_scale.columns)
X
```

[14]:

	gender	race	age	size	marry	income	site	grade	kind	T	N	\
0	1.0	0.5	1.0	1.0	0.0	1.0	0.0	1.00	0.0	1.0	1.0	
1	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.25	0.0	0.5	0.5	
2	1.0	0.0	1.0	1.0	0.5	1.0	1.0	0.25	0.0	0.5	0.0	
3	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.00	0.0	0.5	0.0	
4	0.0	0.5	1.0	1.0	0.5	1.0	0.0	0.25	0.0	0.0	0.0	
…	…	…	…	…	…	…	…	…	…	…	…	…
21532	0.0	0.0	0.5	0.0	0.5	0.0	0.0	1.00	0.0	1.0	1.0	
21533	1.0	0.0	0.5	1.0	1.0	0.5	0.0	1.00	0.0	1.0	1.0	
21534	1.0	0.0	1.0	1.0	1.0	0.5	0.0	1.00	0.0	1.0	0.0	
21535	1.0	0.0	1.0	0.5	1.0	0.5	1.0	0.25	0.0	0.5	0.0	
21536	0.0	0.5	1.0	1.0	0.5	0.5	1.0	1.00	0.0	1.0	1.0	

	surgery_pri	RX_Summ	radiate	chem	CEA	bone	brain	lung	group
0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	1.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.5	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	1.0
4	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.5	1.0
…	…	…	…	…	…	…	…	…	…
21532	0.0	0.0	1.0	1.0	0.5	0.0	0.0	0.0	1.0
21533	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
21534	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
21535	1.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0
21536	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	1.0

[21537 rows x 20 columns]

[15]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=12)
```

[16]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=12)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
#X_test_sm, y_test_sm = smote.fit_sample(X_test, y_test)

X_train_sm = pd.DataFrame(X_train_sm, columns=X.columns)
#X_test_sm = pd.DataFrame(X_test_sm, columns=X.columns)
```

[17]:
```
print('X train shape: ',X_train_sm.shape)
print('y train values: \n', y_train_sm.value_counts())
print()
print('X test shape: ',X_test_sm.shape)
print('y test values: \n', y_test_sm.value_counts())
```

```
X train shape:  (26746, 20)
y train values:
 1    13373
0    13373
Name: os, dtype: int64

X test shape:  (6744, 20)
y test values:
 0    3372
1    3372
Name: os, dtype: int64
```

[18]:
```
mscore=[]
```

[19]:
```
from sklearn.linear_model import LogisticRegression
from time import process_time

clf = LogisticRegression(max_iter=100, solver='lbfgs', class_weight='balanced',␣
↪random_state=12)

start = process_time()
clf.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

clf_prediction_proba = clf.predict_proba(X_test)[:, 1]

y_pred = clf.predict(X_test)
```

```
print('Accuracy Score:', clf.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['Logistic Regression', clf.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
```

```
0.18343899999999999
Accuracy Score: 0.7674094707520891
Prediction: [1 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.91      0.78      0.84      3372
           1       0.48      0.71      0.57       936

    accuracy                           0.77      4308
   macro avg       0.69      0.75      0.71      4308
weighted avg       0.81      0.77      0.78      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        2640         732
True[1]         270         666
```

[19]:

	Predict[0]	Predict[1]
True[0]	2640	732
True[1]	270	666

[20]:

```
from sklearn.ensemble import RandomForestClassifier
from time import process_time

rnd_clf = RandomForestClassifier(n_estimators=100, criterion='gini',␣
 ↪random_state=12)

start = process_time()
model = rnd_clf.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

rf_prediction_proba = rnd_clf.predict_proba(X_test)[:, 1]

y_pred = rnd_clf.predict(X_test)

print('Accuracy Score:', rnd_clf.score(X_test, y_test))
print('Prediction:', y_pred)
```

```python
mscore.append(['Random Forest', rnd_clf.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
print(pd.DataFrame({'Variable':X.columns,
              'Importance':model.feature_importances_}).
 ↪sort_values('Importance', ascending=False))
```

```
3.0654910000000015
Accuracy Score: 0.7964252553389044
Prediction: [0 0 0 … 0 0 0]
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      3372
           1       0.54      0.45      0.49       936

    accuracy                           0.80      4308
   macro avg       0.70      0.67      0.68      4308
weighted avg       0.79      0.80      0.79      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        3009         363
True[1]         514         422
       Variable  Importance
14         chem    0.200109
7         grade    0.091992
2           age    0.079690
4         marry    0.071066
5        income    0.065997
3          size    0.060174
15          CEA    0.057551
18         lung    0.054610
9             T    0.050166
1          race    0.042649
10            N    0.036257
19        group    0.034877
0        gender    0.031644
6          site    0.029710
16         bone    0.021557
11  surgery_pri    0.016935
8          kind    0.016564
12      RX_Summ    0.015127
13      radiate    0.013905
17        brain    0.009419
```

[21]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from time import process_time

bagging = BaggingClassifier(base_estimator=␣
↪DecisionTreeClassifier(random_state=12), max_samples = 1.0, max_features = 1.
↪0,
                            bootstrap = True, bootstrap_features = False,␣
↪random_state=12)

start = process_time()
bagging.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

bg_prediction_proba = bagging.predict_proba(X_test)[:, 1]

y_pred = bagging.predict(X_test)

print('Accuracy Score:', bagging.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['Bagging DecisionTree', bagging.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
```

```
0.8341029999999989
Accuracy Score: 0.7748375116063139
Prediction: [0 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      3372
           1       0.48      0.40      0.44       936

    accuracy                           0.77      4308
   macro avg       0.66      0.64      0.65      4308
weighted avg       0.76      0.77      0.77      4308

Confusion Matrix:
        Predict[0]  Predict[1]
True[0]       2964         408
True[1]        562         374
```

[21]:

	Predict[0]	Predict[1]
True[0]	2964	408
True[1]	562	374

[22]:
```python
from sklearn.neighbors import KNeighborsClassifier
from time import process_time

knn = KNeighborsClassifier()

start = process_time()
knn.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

knn_prediction_proba = knn.predict_proba(X_test)[:, 1]

y_pred = knn.predict(X_test)

print('Accuracy Score:', knn.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['KNN', knn.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
```

```
0.008089999999999264
Accuracy Score: 0.692200557103064
Prediction: [1 0 1 … 0 1 0]
              precision    recall  f1-score   support

           0       0.88      0.70      0.78      3372
           1       0.38      0.65      0.48       936

    accuracy                           0.69      4308
   macro avg       0.63      0.68      0.63      4308
weighted avg       0.77      0.69      0.72      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        2373         999
True[1]         327         609
```

[22]:

	Predict[0]	Predict[1]
True[0]	2373	999
True[1]	327	609

[23]:
```python
from sklearn.naive_bayes import GaussianNB
from time import process_time

gnb = GaussianNB()
```

```
start = process_time()
gnb.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

gnb_prediction_proba = gnb.predict_proba(X_test)[:, 1]

y_pred = gnb.predict(X_test)

print('Accuracy Score:', gnb.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['Gaussian Naive Bayes', gnb.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
```

```
0.020607999999999574
Accuracy Score: 0.6952181987000928
Prediction: [1 0 1 … 1 1 0]
              precision    recall  f1-score   support

           0       0.90      0.68      0.78      3372
           1       0.39      0.74      0.51       936

    accuracy                           0.70      4308
   macro avg       0.65      0.71      0.65      4308
weighted avg       0.79      0.70      0.72      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        2304        1068
True[1]         245         691
```

[23]:

	Predict[0]	Predict[1]
True[0]	2304	1068
True[1]	245	691

[24]:

```
from sklearn.ensemble import GradientBoostingClassifier
from time import process_time

gbc = GradientBoostingClassifier(learning_rate=0.1, n_estimators=100,␣
 ↪random_state=12)

start = process_time()
model = gbc.fit(X_train_sm, y_train_sm)
```

```python
end = process_time()
print(end - start)

gbc_prediction_proba = gbc.predict_proba(X_test)[:, 1]

y_pred = gbc.predict(X_test)

print('Accuracy Score:', gbc.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['Gradient Boosting', gbc.score(X_test, y_test)])

print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
print(pd.DataFrame({'Variable':X.columns,
              'Importance':model.feature_importances_}).
 ↪sort_values('Importance', ascending=False))
```

```
3.0097549999999984
Accuracy Score: 0.8177808727948004
Prediction: [0 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.88      0.89      0.88      3372
           1       0.58      0.56      0.57       936

    accuracy                           0.82      4308
   macro avg       0.73      0.72      0.73      4308
weighted avg       0.82      0.82      0.82      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        2999         373
True[1]         412         524
       Variable  Importance
14         chem    0.433234
7         grade    0.106647
2           age    0.082341
4         marry    0.056565
9             T    0.048238
18         lung    0.046718
5        income    0.043684
15          CEA    0.040733
3          size    0.034911
11  surgery_pri    0.029539
19        group    0.028804
16         bone    0.015489
```

```
1          race    0.013821
12       RX_Summ   0.005227
17         brain   0.003429
6           site   0.003307
13       radiate   0.003261
8           kind   0.002079
10             N   0.001972
0         gender   0.000000
```

[25]:

```python
from sklearn.ensemble import AdaBoostClassifier
from time import process_time

ada = AdaBoostClassifier(learning_rate=1.0, n_estimators=50, random_state=12)

start = process_time()
model = ada.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

ada_prediction_proba = ada.predict_proba(X_test)[:, 1]

y_pred = ada.predict(X_test)

print('Accuracy Score:', ada.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['Adaptive Boosting', ada.score(X_test, y_test)])

# from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
print(pd.DataFrame({'Variable':X.columns,
              'Importance':model.feature_importances_}).
 ↪sort_values('Importance', ascending=False))
```

```
0.9511830000000003
Accuracy Score: 0.8033890436397401
Prediction: [1 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.89      0.86      0.87      3372
           1       0.54      0.61      0.58       936

    accuracy                           0.80      4308
   macro avg       0.72      0.74      0.72      4308
weighted avg       0.81      0.80      0.81      4308

Confusion Matrix:
```

```
          Predict[0]  Predict[1]
True[0]         2886         486
True[1]          361         575
       Variable  Importance
4         marry        0.20
5        income        0.14
15          CEA        0.12
18         lung        0.12
1          race        0.10
2           age        0.08
3          size        0.06
7         grade        0.04
16         bone        0.04
10            N        0.02
6          site        0.02
11  surgery_pri        0.02
12      RX_Summ        0.02
14         chem        0.02
17        brain        0.00
0        gender        0.00
13      radiate        0.00
9             T        0.00
8          kind        0.00
19        group        0.00
```

[26]:

```
from sklearn.svm import SVC
from time import process_time

svm_clf = SVC(kernel='rbf', gamma='scale', probability=True, random_state=12)

start = process_time()
svm_clf.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

svm_prediction_proba = svm_clf.predict_proba(X_test)[:, 1]

y_pred = svm_clf.predict(X_test)

print('Accuracy Score:', svm_clf.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['SVM', svm_clf.score(X_test, y_test)])

# from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
```

```
316.346303
Accuracy Score: 0.7704271123491179
Prediction: [1 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.91      0.79      0.84      3372
           1       0.48      0.70      0.57       936

    accuracy                           0.77      4308
   macro avg       0.69      0.75      0.71      4308
weighted avg       0.81      0.77      0.78      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        2660         712
True[1]         277         659
```

[26]:

	Predict[0]	Predict[1]
True[0]	2660	712
True[1]	277	659

[27]:

```
from xgboost import XGBClassifier
from time import process_time

xgbc = XGBClassifier(eta=0.3, max_depth=6, random_state=12)

start = process_time()
model = xgbc.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)

xgbc_prediction_proba = xgbc.predict_proba(X_test)[:, 1]

y_pred = xgbc.predict(X_test)

print('Accuracy Score:', xgbc.score(X_test, y_test))
print('Prediction:', y_pred)

mscore.append(['XGBoost', xgbc.score(X_test, y_test)])

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
confusion(y_test, y_pred)
print(pd.DataFrame({'Variable':X.columns,
              'Importance':model.feature_importances_}).
 ↪sort_values('Importance', ascending=False))
```

```
8.831576000000041
```

```
Accuracy Score: 0.8161559888579387
Prediction: [0 0 0 … 0 1 0]
              precision    recall  f1-score   support

           0       0.86      0.91      0.89      3372
           1       0.60      0.47      0.53       936

    accuracy                           0.82      4308
   macro avg       0.73      0.69      0.71      4308
weighted avg       0.80      0.82      0.81      4308

Confusion Matrix:
         Predict[0]  Predict[1]
True[0]        3076         296
True[1]         496         440
       Variable  Importance
14         chem    0.360615
11  surgery_pri    0.078392
2           age    0.067002
18         lung    0.055987
7         grade    0.048411
4         marry    0.046379
5        income    0.045417
15          CEA    0.044645
3          size    0.041301
9             T    0.040480
19        group    0.035866
1          race    0.023866
16         bone    0.023047
12      RX_Summ    0.019596
17        brain    0.014875
8          kind    0.012955
13      radiate    0.012584
10            N    0.011776
6          site    0.010218
0        gender    0.006586
```

[28]:

```python
from sklearn.neural_network import MLPClassifier
from time import process_time

mlp = MLPClassifier(hidden_layer_sizes=(100,), solver='adam', shuffle=True, tol= 0.0001, random_state=12)

start = process_time()
mlp.fit(X_train_sm, y_train_sm)
end = process_time()
print(end - start)
```

```
mlp_prediction_proba = mlp.predict_proba(X_test)[:, 1]

mlp_pred_diabetes = mlp.predict(X_test)

print('Accuracy Score:', mlp.score(X_test, y_test))
print('Prediction:', mlp_pred_diabetes)
print("parameter: ", mlp.get_params())

mscore.append(['MLP', mlp.score(X_test, y_test)])
```

```
52.18636399999997
Accuracy Score: 0.7432683379758589
Prediction: [1 0 1 … 1 1 0]
parameter:  {'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto',
'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08,
'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'learning_rate_init':
0.001, 'max_fun': 15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change':
10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': 12, 'shuffle':
True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose':
False, 'warm_start': False}
```

```
/Users/zhiyi/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

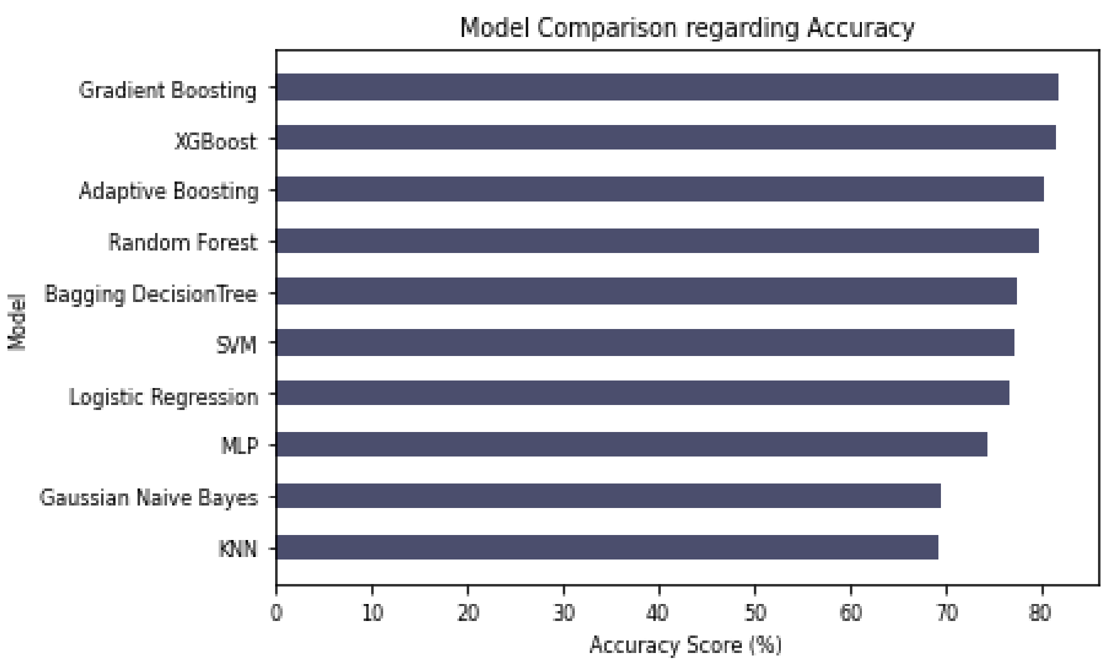
[29]:
```
mscore.sort(key=lambda x: x[1], reverse=True)
mscore
```

[29]:
```
[['Gradient Boosting', 0.8177808727948004],
 ['XGBoost', 0.8161559888579387],
 ['Adaptive Boosting', 0.8033890436397401],
 ['Random Forest', 0.7964252553389044],
 ['Bagging DecisionTree', 0.7748375116063139],
 ['SVM', 0.7704271123491179],
 ['Logistic Regression', 0.7674094707520891],
 ['MLP', 0.7432683379758589],
 ['Gaussian Naive Bayes', 0.6952181987000928],
 ['KNN', 0.692200557103064]]
```

[30]:
```
model = list(i[0] for i in mscore)
score = list(round(i[1]*100,2) for i in mscore)

print('Accuracy Score: \n')
for m,s in zip(model, score):
    print(f'{m}: {s}%')
```

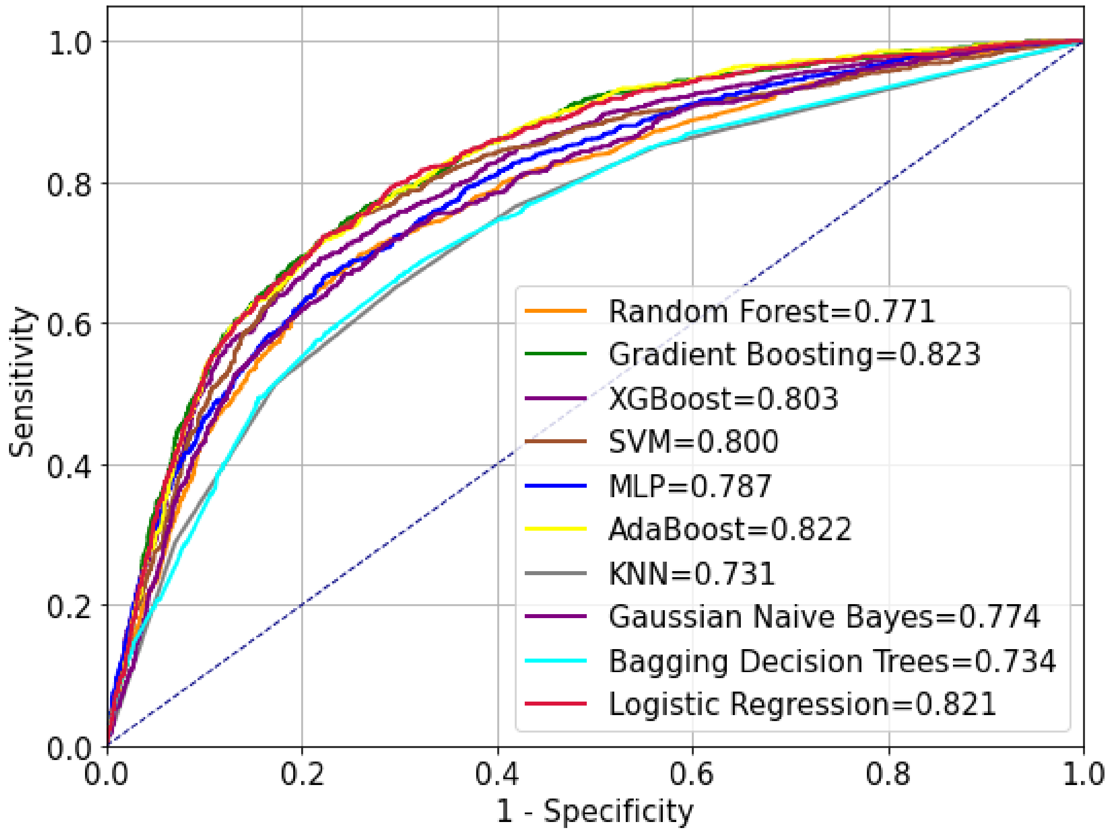
```python
# creating horizontal bar
plt.barh(model, score, height = 0.5, color='#4B4E6D')

plt.xlabel("Accuracy Score (%)")
plt.ylabel("Model")
plt.title("Model Comparison regarding Accuracy")
plt.gca().invert_yaxis()
plt.rc('font', size=9)
plt.show()
```

```
Accuracy Score:

Gradient Boosting: 81.78%
XGBoost: 81.62%
Adaptive Boosting: 80.34%
Random Forest: 79.64%
Bagging DecisionTree: 77.48%
SVM: 77.04%
Logistic Regression: 76.74%
MLP: 74.33%
Gaussian Naive Bayes: 69.52%
KNN: 69.22%
```



[31]:
```
rf_prediction_proba = rnd_clf.predict_proba(X_test)[:, 1]
def roc_curve_and_score(y_test, pred_proba):
    fpr, tpr, _ = roc_curve(y_test.ravel(), pred_proba.ravel())
    roc_auc = roc_auc_score(y_test.ravel(), pred_proba.ravel())
    return fpr, tpr, roc_auc

from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib
import matplotlib.pyplot as plt
plt.figure(figsize=(9, 7))
matplotlib.rcParams.update({'font.size': 15})
plt.grid()
fpr, tpr, roc_auc = roc_curve_and_score(y_test, rf_prediction_proba)
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label='Random Forest={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, gbc_prediction_proba)
plt.plot(fpr, tpr, color='green', lw=2,
         label='Gradient Boosting={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, xgbc_prediction_proba)
plt.plot(fpr, tpr, color='purple', lw=2,
         label='XGBoost={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, svm_prediction_proba)
plt.plot(fpr, tpr, color='sienna', lw=2,
         label='SVM={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, mlp_prediction_proba)
plt.plot(fpr, tpr, color='blue', lw=2,
         label='MLP={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, ada_prediction_proba)
plt.plot(fpr, tpr, color='yellow', lw=2,
         label='AdaBoost={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, knn_prediction_proba)
plt.plot(fpr, tpr, color='grey', lw=2,
         label='KNN={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, gnb_prediction_proba)
plt.plot(fpr, tpr, color='purple', lw=2,
         label='Gaussian Naive Bayes={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, bg_prediction_proba)
plt.plot(fpr, tpr, color='cyan', lw=2,
         label='Bagging Decision Trees={0:.3f}'.format(roc_auc))
fpr, tpr, roc_auc = roc_curve_and_score(y_test, clf_prediction_proba)
plt.plot(fpr, tpr, color='crimson', lw=2,
         label='Logistic Regression={0:.3f}'.format(roc_auc))

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.legend(loc="lower right")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('1 - Specificity')
plt.ylabel('Sensitivity')
plt.show()
```



[ ]: