Homework 1 (Support Vector Machines)

Using the data here, build a series of SVM models as described. Make sure that data is **z score appropriately** before fitting your models. Due to the number of models you're fitting with GridSearch, this make take a while to run.

- A. 💻 Load your data and do a 80/20 Train Test Split
- B. 💻 Use sklearn to build an SVM model. Use `GridSearchCV` to choose the `kernel` (choose from `linear` and `rbf`), `C` (choose from `[0.001, 0.01, 1, 5, 25, 50]`), and `gamma` (choose from `[0.001,0.01, 0.1, 0.5, 1,2,5]`). Print out the train and test accuracies and ROC/AUCs, and plot the train and test confusion matrices.
- C. ✍ What hyperparameters did GridSearch choose? (in a TEXT CELL)
- D. 💻 Use sklearn to build a Logistic Regression on the same data with the same train/test set. Print out the train and test accuracies and ROC/AUCs, and plot the train and test confusion matrices. (CPSC 392 code)
- E. 💻 Use sklearn to build a KNearest Neighbors model on the same data with the same train/test set (use GridSearch to choose `n_neighbors`).Print out the train and test accuracies and ROC/AUCs, and plot the train and test confusion matrices. (CPSC 392 code)
- F. ✍ Discuss in detail how your models performed based on the metrics you printed. Write this discussion as if you are presenting your results to a CEO/Stakeholder (in a TEXT CELL)
- G. ✍ Compare the performance of your model to the Logistic Regression and KNN model, and provide a justification for which model you want to use "in production" (in a TEXT CELL)

```
import warnings
warnings.filterwarnings('ignore')

# data and plotting
import pandas as pd
import numpy as np
from plotnine import *
import matplotlib.pyplot as plt

# preprocessing
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.model_selection import train_test_split

# metrics
from sklearn.metrics import accuracy_score, plot_confusion_matrix,
roc_auc_score, recall_score

# models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.svm import SVC

# pipelines
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.model_selection import GridSearchCV
```

A.

```
data =
pd.read_csv('https://raw.githubusercontent.com/cmparlettpelleriti/
CPSC393ParlettPelleriti/main/Data/hw1.csv')

data.isnull().sum()
```

```
X1       0
X2       0
X3       0
X4       0
X5       0
X6       0
X7       0
X8       0
Group    0
dtype: int64
```

```
data.head()
```

```
         X1        X2        X3        X4        X5        X6
X7  \
0 -0.604285 -0.610629  0.026014  0.019710  0.406532  0.678796
0.524360
1 -0.111772 -1.125178  0.744157  0.078315  0.088176  0.891009
0.018460
2 -0.916802  1.965494  0.150022  0.388770  0.179276  0.064449
0.159279
3 -0.280479  0.920669  0.208949  0.940153  0.854437  0.688172
0.365126
4  1.856025  1.043214  0.167088  0.207002  0.979049  0.641019
0.628764

         X8 Group
0  0.404739     B
1  0.536511     B
2  0.951204     B
3  0.985259     A
4  0.045912     B
```

```
data.shape
```

```
(1000, 9)
```

```
X = data.filter(like='X')
y = data['Group']

predictors = X.columns

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

predictors
```

```
Index(['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8'],
dtype='object')
```

B.

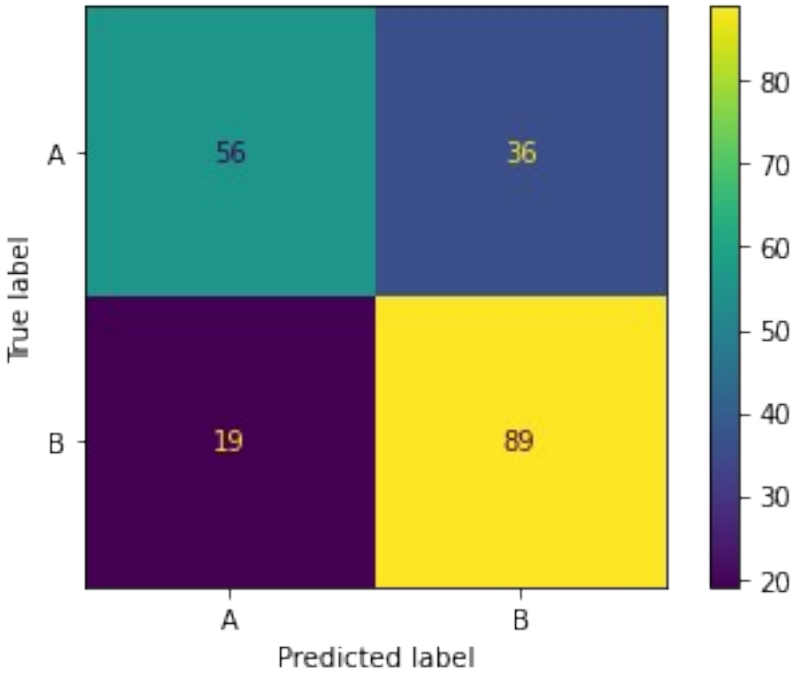
```
# build parts of pipeline
z = make_column_transformer((StandardScaler(), predictors),
                            remainder = 'passthrough')

svm = SVC(probability=True, random_state = 42)

# build pipeline
pipe = make_pipeline(z, svm)
#print(pipe.get_params().keys())
# parameters dict

params = {'svc__C': [0.001, 0.01, 1, 5, 25, 50],
          'svc__gamma': [0.001,0.01, 0.1, 0.5, 1,2,5],
          'svc__kernel': ['rbf', 'linear'],
          'svc__random_state': [42]}
# grid search
grid = GridSearchCV(pipe, params,
                    scoring = 'accuracy',
                    cv = 5, refit = True)
# fit and check
grid.fit(X_train,y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',

ColumnTransformer(remainder='passthrough',

transformers=[('standardscaler',

StandardScaler(),

Index(['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8'],
dtype='object'))])),
                                       ('svc',
```

```
                                        SVC(probability=True,
                                            random_state=42))]),
             param_grid={'svc__C': [0.001, 0.01, 1, 5, 25, 50],
                         'svc__gamma': [0.001, 0.01, 0.1, 0.5, 1, 2,
5],
                         'svc__kernel': ['rbf', 'linear'],
                         'svc__random_state': [42]},
             scoring='accuracy')

y_pred_train = grid.predict(X_train)
y_pred_test = grid.predict(X_test)

print("Train Acc: ", accuracy_score(y_train, y_pred_train))
testAccSVM = ("Test Acc: ", accuracy_score(y_test, y_pred_test))
print(testAccSVM)

y_pred_train_p = grid.predict_proba(X_train)[:,1]
y_pred_test_p = grid.predict_proba(X_test)[:,1]

print("Train AUC: ", roc_auc_score(y_train, y_pred_train_p))
testAucSVM = ("Test AUC: ", roc_auc_score(y_test, y_pred_test_p))
print(testAucSVM)
```

```
Train Acc:  0.79625
('Test Acc: ', 0.725)
Train AUC:  0.8741463350630777
('Test AUC: ', 0.7831119162640903)
```

```
plot_confusion_matrix(grid, X_test, y_test)
plt.show()
```



C.

```
grid.best_params_
```

```
{'svc__C': 50,
 'svc__gamma': 0.01,
 'svc__kernel': 'rbf',
 'svc__random_state': 42}
```

The best parameters that the grid search chose for SVC were using an rbf kernel, a c of 50, and a gamma value of 0.01

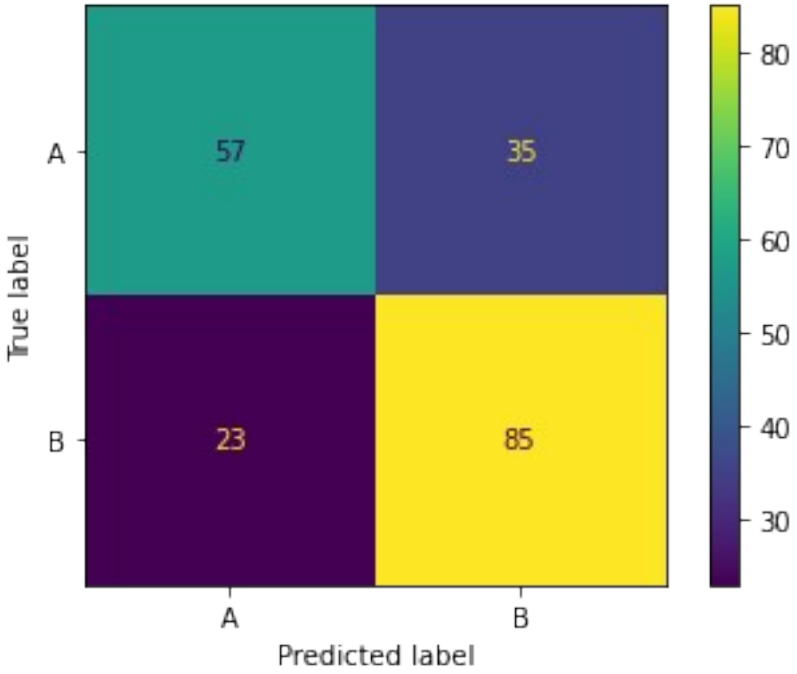
D.

```
zscore = StandardScaler()

zscore.fit(X_train)
X_train[predictors] = zscore.transform(X_train)
X_test[predictors] = zscore.transform(X_test)
myLogit = LogisticRegression(penalty = "none", random_state = 42)
#create
myLogit.fit(X_train,y_train) #fit

LogisticRegression(penalty='none', random_state=42)

y_pred_train = myLogit.predict(X_train)
y_pred_test = myLogit.predict(X_test)
```

```
print("Train Acc: ", accuracy_score(y_train, y_pred_train))
testAccLog = ("Test Acc: ", accuracy_score(y_test, y_pred_test))
print(testAccLog)


y_pred_train_p = myLogit.predict_proba(X_train)[:,1]
y_pred_test_p = myLogit.predict_proba(X_test)[:,1]

print("Train AUC: ", roc_auc_score(y_train, y_pred_train_p))
testAucLog = ("Test AUC: ", roc_auc_score(y_test, y_pred_test_p))
print(testAucLog)
```

```
Train Acc:  0.77375
('Test Acc: ', 0.71)
Train AUC:  0.8590181787169276
('Test AUC: ', 0.7816022544283414)
```

```
plot_confusion_matrix(myLogit, X_test, y_test)
plt.show()
```



E.

```
# build parts of pipeline
zknn = make_column_transformer((StandardScaler(), predictors),
                               remainder = 'passthrough')

knn = KNeighborsClassifier()
```

```
# build pipeline
pipeknn = make_pipeline(zknn, knn)
#print(pipeknn.get_params().keys())
# parameters dict

paramsknn = {'kneighborsclassifier__n_neighbors': range(2,100)
             #'kneighborsclassifier__random_state': [42]
            }
# grid search
gridknn = GridSearchCV(pipeknn, paramsknn,
                   scoring = 'accuracy',
                   cv = 5, refit = True)
# fit and check
gridknn.fit(X_train,y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',

ColumnTransformer(remainder='passthrough',

transformers=[('standardscaler',

StandardScaler(),

Index(['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8'],
dtype='object'))])),
                                       ('kneighborsclassifier',
                                        KNeighborsClassifier())]),
             param_grid={'kneighborsclassifier__n_neighbors': range(2,
100)},
             scoring='accuracy')
```

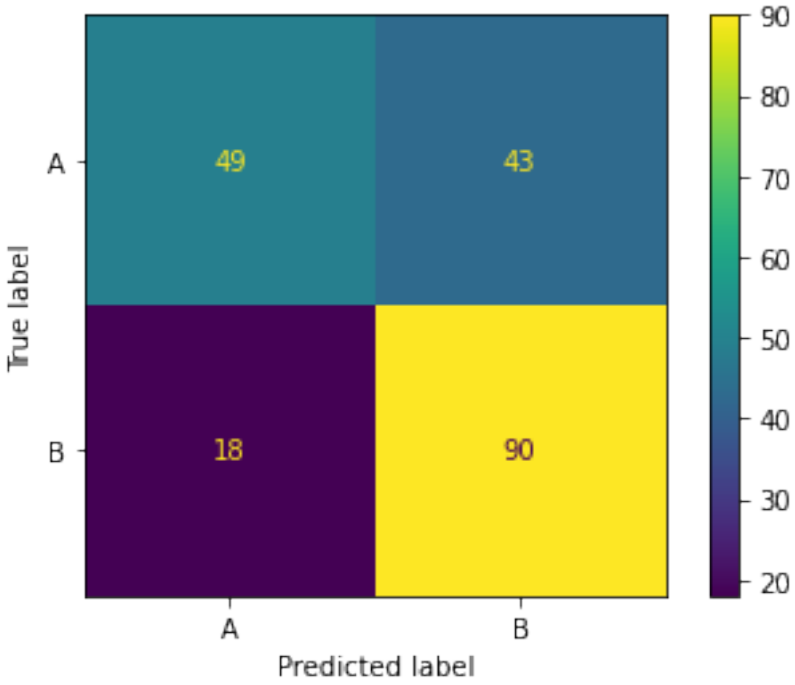
```
gridknn.best_params_
```

```
{'kneighborsclassifier__n_neighbors': 31}
```

```
y_pred_train = gridknn.predict(X_train)
y_pred_test = gridknn.predict(X_test)

print("Train Acc: ", accuracy_score(y_train, y_pred_train))
testAccKNN = ("Test Acc: ", accuracy_score(y_test, y_pred_test))
print(testAccKNN)

y_pred_train_p = gridknn.predict_proba(X_train)[:,1]
y_pred_test_p = gridknn.predict_proba(X_test)[:,1]

print("Train AUC: ", roc_auc_score(y_train, y_pred_train_p))
testAucKNN = ("Test AUC: ", roc_auc_score(y_test, y_pred_test_p))
print(testAucKNN)
```

```
Train Acc:  0.77875
('Test Acc: ', 0.695)
Train AUC:  0.8600940753521265
('Test AUC: ', 0.7810990338164251)
```

```
plot_confusion_matrix(gridknn, X_test, y_test)
plt.show()
```



F.

```
print("SVM: ", (testAccSVM))
print("Logistic: ", testAccLog)
print("KNN: ", testAccKNN)
```

```
SVM:  ('Test Acc: ', 0.725)
Logistic:  ('Test Acc: ', 0.71)
KNN:  ('Test Acc: ', 0.695)
```

```
print("SVM: ", testAucSVM)
print("Logistic: ", testAucLog)
print("KNN: ", testAucKNN)
```

```
SVM:  ('Test AUC: ', 0.7831119162640903)
Logistic:  ('Test AUC: ', 0.7816022544283414)
KNN:  ('Test AUC: ', 0.7810990338164251)
```

The above print statements display how our three models perform on unseen data, an indication of how the models will perform in production. There are two metrics we used to measure the model performance, accuracy and AUC. Accuracy being a percentage of how

often the model predicted the correct classification. AUC being a better indication overall since it is a relationship between how often the model predicts positively correctly and how often it predicts negatively correctly.

All of the models performed pretty well based off of both metrics. A value above 0.7 is great for both AUC and accuracy, which all models surpassed with the exception of KNN on accuracy. However, it was only by a very small margin so it should still be considered pretty good.

Both accuracy and AUC placed the model in the same order of rankings. The worst being K nearest neighbor, middle performance by Logistic model, and the best performance from Support Vector Classification.

One thing to note is that all models performed a decent amount better on the data that is was trained on than the testing data. This is an indication of slight overfitting. Severely overfit models are not generalizable or suitable on new data and should not be put into production. However, there was only one model where it was severe enough to worry about, the KNN model.

G.

As stated before, the SVC model outperformed the other models. It would be the best of the three models since it is generalizable onto unseen data and performs pretty well.

KNN should not be implemented since it was much more overfit onto training data than the other models. It being put into production is not a good idea. Logistic model is not the best idea either just because it doesn't perform as well as SVC and would provide less accurate predictions for the business.