Exercise 6 - Feature Selection

First name: Brian

Last name: Schweigler

Matriculation number: 16-102-071

(1) Take the titanic dataset and using all attributes to predict the class `Survived' (convert age and fare into classes ; exclude names from the attribute list)

(a) Choose Three classifiers and evaluate their performance using all attributes;

We will use the same classifiers as in last series, Decision Tree, KNN, Naive Bayes

In [1]:

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from mlxtend.evaluate import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFE, SelectKBest, f_classif
from sklearn.decomposition import PCA

df = pd.read_csv("data/titanic.csv", index_col='Name')
pd.set_option('display.max_colwidth', None)
le = preprocessing.LabelEncoder()
bins = [0, 4, 18, 65, 100]
labels = ['Infant', 'Child', 'Adult', 'Elderly']
labels = [1, 2, 3, 4]
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)
df["AgeGroup"] = le.fit_transform(df["AgeGroup"])
df['FareGroup'] = pd.qcut(x=df['Fare'], q=4)
df["FareGroup"] = le.fit_transform(df["FareGroup"])
df["Survived"] = le.fit_transform(df["Survived"])
df["Sex"] = le.fit_transform(df["Sex"])
print(df.head(2))

models = {
    'Decision Tree': DecisionTreeClassifier(),
    'KNN': KNeighborsClassifier(metric='manhattan'),
    'Naive Bayes': GaussianNB(),
}
```

```
                                                     Survived  Pclass  Sex  \
Name
Mr. Owen Harris Braund                                      0       3    1
Mrs. John Bradley (Florence Briggs Thayer) Cumings          1       1    0

                                                      Age  \
Name
Mr. Owen Harris Braund                               22.0
Mrs. John Bradley (Florence Briggs Thayer) Cumings   38.0

                                                     Siblings/Spouses Aboard  \
Name
Mr. Owen Harris Braund                                                     1
Mrs. John Bradley (Florence Briggs Thayer) Cumings                         1

                                                     Parents/Children Aboard  \
```

```
Name
Mr. Owen Harris Braund                                          0
Mrs. John Bradley (Florence Briggs Thayer) Cumings              0

                                                Fare  AgeGroup  \
Name
Mr. Owen Harris Braund                        7.2500         2
Mrs. John Bradley (Florence Briggs Thayer) Cumings  71.2833         2

                                              FareGroup
Name
Mr. Owen Harris Braund                                0
Mrs. John Bradley (Florence Briggs Thayer) Cumings    3
```

In [2]:

```python
all_features = ['Pclass', 'Sex', 'Siblings/Spouses Aboard',
                'Parents/Children Aboard', 'AgeGroup', 'FareGroup']


def kfold_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=5, shuffle=True, random_state=6)
    accuracy = np.empty(kf.n_splits)
    precision = np.empty(kf.n_splits)
    recall = np.empty(kf.n_splits)
    f1 = np.empty(kf.n_splits)

    i = 0
    for train_index, test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        model.fit(x_train, y_train)
        prediction = model.predict(x_test)

        accuracy[i] = accuracy_score(prediction, y_test)
        precision[i] = precision_score(prediction, y_test)
        recall[i] = recall_score(prediction, y_test)
        f1[i] = f1_score(prediction, y_test)
        i += 1

    print(model)
    return accuracy, precision, recall, f1


for name, model in models.items():
    a, p, r, f = kfold_eval(model, df[all_features], df["Survived"])
    data = {
        'Fold': [1, 2, 3, 4, 5],
        'Accuracy': a,
        'Precision': p,
        'Recall': r,
        'F1-Score': f,
    }

    scores = pd.DataFrame(data).set_index('Fold')
    display(scores)
```

```
DecisionTreeClassifier()
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.814607	0.661765	0.818182	0.731707
2	0.803371	0.625000	0.849057	0.720000
3	0.774011	0.716418	0.695652	0.705882
4	0.745763	0.614286	0.704918	0.656489

Fold	Accuracy	Precision	Recall	F1-Score
5	0.858757	0.738462	0.857143	0.793388

```
KNeighborsClassifier(metric='manhattan')
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.842697	0.691176	0.870370	0.770492
2	0.837079	0.666667	0.905660	0.768000
3	0.813559	0.731343	0.765625	0.748092
4	0.796610	0.642857	0.803571	0.714286
5	0.841808	0.707692	0.836364	0.766667

```
GaussianNB()
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.764045	0.691176	0.691176	0.691176
2	0.735955	0.722222	0.658228	0.688742
3	0.768362	0.820896	0.654762	0.728477
4	0.728814	0.828571	0.617021	0.707317
5	0.807910	0.830769	0.701299	0.760563

(b) Define a feature selection method and use it on all the classifiers ;

Our approach will be using PCA

In [3]:

```python
def pca_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=5, shuffle=True, random_state=6)
    pca = PCA(n_components=4)
    fit = pca.fit(x)

    print("Explained Variance: %s" % fit.explained_variance_ratio_, " \n")
    print(fit.components_)
    accuracy = np.empty(kf.n_splits)
    precision = np.empty(kf.n_splits)
    recall = np.empty(kf.n_splits)
    f1 = np.empty(kf.n_splits)

    i = 0
    for train_index, test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        x_train_pca = fit.transform(x_train)
        x_test_pca = fit.transform(x_test)

        model.fit(x_train_pca, y_train)
        prediction = model.predict(x_test_pca)

        accuracy[i] = accuracy_score(prediction, y_test)
        precision[i] = precision_score(prediction, y_test)
        recall[i] = recall_score(prediction, y_test)
        f1[i] = f1_score(prediction, y_test)
        i += 1
```

```
    print(model)
    return accuracy, precision, recall, f1


for name, model in models.items():
    a, p, r, f = pca_eval(model, df[all_features], df["Survived"])
    data = {
        'Fold': [1, 2, 3, 4, 5],
        'Accuracy': a,
        'Precision': p,
        'Recall': r,
        'F1-Score': f,
    }

    scores = pd.DataFrame(data).set_index('Fold')
    display(scores)
```

```
Explained Variance: [0.48313985 0.27595039 0.11107641 0.04733949]

[[-0.27269815 -0.09305713  0.54242269  0.33034372 -0.11331966  0.70766192]
 [ 0.60450815  0.01781941  0.63204615  0.23097823 -0.18033632 -0.3858729 ]
 [ 0.17721424 -0.17735289 -0.4875729   0.8300098  -0.10135234  0.01500562]
 [ 0.00589098  0.94848776 -0.01035563  0.22062434  0.21703746  0.06669835]]
DecisionTreeClassifier()
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.820225	0.647059	0.846154	0.733333
2	0.792135	0.583333	0.857143	0.694215
3	0.762712	0.701493	0.681159	0.691176
4	0.768362	0.657143	0.730159	0.691729
5	0.830508	0.661538	0.843137	0.741379

```
Explained Variance: [0.48313985 0.27595039 0.11107641 0.04733949]

[[-0.27269815 -0.09305713  0.54242269  0.33034372 -0.11331966  0.70766192]
 [ 0.60450815  0.01781941  0.63204615  0.23097823 -0.18033632 -0.3858729 ]
 [ 0.17721424 -0.17735289 -0.4875729   0.8300098  -0.10135234  0.01500562]
 [ 0.00589098  0.94848776 -0.01035563  0.22062434  0.21703746  0.06669835]]
KNeighborsClassifier(metric='manhattan')
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.837079	0.691176	0.854545	0.764228
2	0.797753	0.597222	0.860000	0.704918
3	0.779661	0.746269	0.694444	0.719424
4	0.774011	0.757143	0.697368	0.726027
5	0.830508	0.738462	0.786885	0.761905

```
Explained Variance: [0.48313985 0.27595039 0.11107641 0.04733949]

[[-0.27269815 -0.09305713  0.54242269  0.33034372 -0.11331966  0.70766192]
 [ 0.60450815  0.01781941  0.63204615  0.23097823 -0.18033632 -0.3858729 ]
 [ 0.17721424 -0.17735289 -0.4875729   0.8300098  -0.10135234  0.01500562]
 [ 0.00589098  0.94848776 -0.01035563  0.22062434  0.21703746  0.06669835]]
GaussianNB()
```

Fold	Accuracy	Precision	Recall	F1-Score

Fold	Accuracy	Precision	Recall	F1-Score
1	0.831461	0.705882	0.827586	0.761905
2	0.769663	0.583333	0.792453	0.672000
3	0.774011	0.671642	0.714286	0.692308
4	0.751412	0.657143	0.696970	0.676471
5	0.836158	0.692308	0.833333	0.756303

(c) Compare the classifiers and explain the differences observed;

PCA is a data reduction technique, that uses linear algebra to compress the dataset. Thus, we have "less" features, as we compress the features into the principal components.

Performance in terms of accuracy, precision and recall is quite similar for the classiffiers, except for GaussianNB; this seems to slightly prefer the non-PCA approach.

Generally, it seems to be a good approach, but is a bit overkill for such a small dataset. It will be interesting to see if there are larger differences in performance for the stock dataset.

(2) Build Decision tree model with your selected stock / market index using all attributes to predict `daily returns'(decision). ('daily returns' must frst be converted into a decision class that will be used as the target(label), all other attributes must be grouped into classes)

(a) Choose Three feature selection methods to evaluate the model;

For the first part, we can follow last week's implementation:

In [4]:

```
stock_df = pd.read_csv("data/Nasdaq.csv", index_col='Date')
print(stock_df.head(3))

daily_return = np.empty(stock_df['Close'].shape)
#  From Slides: Daily return (r): Difference in percentage between
#  price at time t+1 and time t
daily_return[0] = float('NaN')  # The first
daily_return[1:] = np.ediff1d(stock_df['Close']) / stock_df['Close'][:-1]
stock_df.insert(loc=len(stock_df.columns), column='Daily Return', value=daily_return)

binary = (daily_return > 0).astype(float)
stock_df.insert(loc=len(stock_df.columns), column='Binary Decision', value=binary)
stock_df["Binary Decision"] = le.fit_transform(stock_df["Binary Decision"])

stock_df['Rolling Mean 5'] = stock_df['Close'].rolling(5).mean()
stock_df['Rolling Mean 10'] = stock_df['Close'].rolling(10).mean()
stock_df['Rolling Mean 20'] = stock_df['Close'].rolling(20).mean()
stock_df['Rolling Mean 50'] = stock_df['Close'].rolling(50).mean()
stock_df['Rolling Mean 200'] = stock_df['Close'].rolling(200).mean()
stock_df = stock_df.fillna(0)  # NAs replaced with zero

stock_models = {
    'Decision Tree': DecisionTreeClassifier(),
}
print(stock_df.tail(2))
```

```
                  Open        High         Low       Close   Adj Close  Volume
Date
1971-02-05  100.000000  100.000000  100.000000  100.000000  100.000000       0
1971-02-08  100.839996  100.839996  100.839996  100.839996  100.839996       0
1971-02-09  100.760002  100.760002  100.760002  100.760002  100.760002       0
```

```
                  Open           High            Low          Close  \
Date
2021-09-20  14758.139648  14841.820312  14530.070312  14713.900391
2021-09-21  14803.400391  14847.027344  14696.467773  14779.216797

               Adj Close      Volume  Daily Return  Binary Decision  \
Date
2021-09-20  14713.900391  4860630000     -0.021940                0
2021-09-21  14779.216797  3083208000      0.004439                1

            Rolling Mean 5  Rolling Mean 10  Rolling Mean 20  Rolling Mean 50  \
Date
2021-09-20    15027.816016     15126.937012     15143.909961     14875.705195
2021-09-21    14976.107422     15067.425684     15135.738281     14876.624727

            Rolling Mean 200
Date
2021-09-20      13856.711787
2021-09-21      13868.721973
```

With the Rolling Mean, we already have a sort of grouping, thus I vouch to leave the values ungrouped and just work with the Rolling Mean, especially as it is what was also used in prior exercises.

In [5]:

```python
stock_df["Class Daily Return"] = pd.qcut(stock_df["Daily Return"], q=3)
stock_df["Class Daily Return"], _ = stock_df["Class Daily Return"].factorize(sort=True)

all_stock_features = ['Volume', 'Rolling Mean 5', 'Rolling Mean 10',
                      'Rolling Mean 20', 'Rolling Mean 50', 'Rolling Mean 200']
```

In [6]:

```python
def stock_pca_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=10)
    pca = PCA(n_components=3)
    fit = pca.fit(x)

    print("Explained Variance: %s" % fit.explained_variance_ratio_, " \n")
    print(fit.components_)
    accuracy = np.empty(kf.n_splits)
    precision = np.empty(kf.n_splits)
    recall = np.empty(kf.n_splits)
    f1 = np.empty(kf.n_splits)

    i = 0
    for train_index, test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        x_train_pca = fit.transform(x_train)
        x_test_pca = fit.transform(x_test)

        model.fit(x_train_pca, y_train)
        prediction = model.predict(x_test_pca)

        accuracy[i] = accuracy_score(prediction, y_test)
        precision[i] = precision_score(prediction, y_test,
                                       average='macro', zero_division=0)
        recall[i] = recall_score(prediction, y_test,
                                 average='macro', zero_division=0)
        f1[i] = f1_score(prediction, y_test,
                         average='macro', zero_division=0)
        i += 1

    print(str(model) + " Using PCA")
    return accuracy, precision, recall, f1


def stock_rfe_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=10)
```

```python
    accuracy = np.empty(kf.n_splits)
    precision = np.empty(kf.n_splits)
    recall = np.empty(kf.n_splits)
    f1 = np.empty(kf.n_splits)
    rfe = RFE(model, n_features_to_select=3)

    i = 0
    for train_index, test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        fit = rfe.fit(x_train, y_train)
        x_train_rfe = fit.transform(x_train)
        x_test_rfe = fit.transform(x_test)

        # print("Num features: %d" % fit.n_features_)
        # print("Feature Ranking: %s" % fit.ranking_)

        model.fit(x_train_rfe, y_train)
        prediction = model.predict(x_test_rfe)

        accuracy[i] = accuracy_score(prediction, y_test)
        precision[i] = precision_score(prediction, y_test,
                                       average='macro', zero_division=0)
        recall[i] = recall_score(prediction, y_test,
                                 average='macro', zero_division=0)
        f1[i] = f1_score(prediction, y_test,
                         average='macro', zero_division=0)
        i += 1

    print(str(model) + " Using RFE")
    return accuracy, precision, recall, f1


# Selects the 3 best scoring features as input
def stock_kbest_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=10)
    accuracy = np.empty(kf.n_splits)
    precision = np.empty(kf.n_splits)
    recall = np.empty(kf.n_splits)
    f1 = np.empty(kf.n_splits)
    best_features = SelectKBest(score_func=f_classif, k=3)

    i = 0
    for train_index, test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        fit = best_features.fit(x_train, y_train)
        # df_scores = pd.DataFrame(fit.scores_)
        # df_columns = pd.DataFrame(x_train.columns)
        # feature_scores = pd.concat([df_columns, df_scores], axis=1)
        # feature_scores.columns = ['Feature_Name', 'Score']
        # print(feature_scores.nlargest(3, 'Score'))

        x_train_kbest = fit.transform(x_train)
        x_test_kbest = fit.transform(x_test)

        model.fit(x_train_kbest, y_train)
        prediction = model.predict(x_test_kbest)

        accuracy[i] = accuracy_score(prediction, y_test)
        precision[i] = precision_score(prediction, y_test,
                                       average='macro', zero_division=0)
        recall[i] = recall_score(prediction, y_test,
                                 average='macro', zero_division=0)
        f1[i] = f1_score(prediction, y_test,
                         average='macro', zero_division=0)
        i += 1
```

```python
    print(str(model) + " Using k=3 best features")
    return accuracy, precision, recall, f1
```

In [7]:

```python
for name, model in stock_models.items():
    a, p, r, f = stock_pca_eval(model, stock_df[all_stock_features],
                                stock_df["Class Daily Return"])
    stock_data = {
        'Fold': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'Accuracy': a,
        'Precision': p,
        'Recall': r,
        'F1-Score': f,
    }

    stock_scores = pd.DataFrame(stock_data).set_index('Fold')
    display(stock_scores)
```

```
Explained Variance: [1.00000000e+00 7.77331260e-12 4.02766284e-14]

[[ 1.00000000e+00  1.96057382e-06  1.95750602e-06  1.95030502e-06
   1.92540289e-06  1.80813391e-06]
 [-4.29553087e-06  4.61669966e-01  4.59950931e-01  4.56591956e-01
   4.48108035e-01  4.07466513e-01]
 [ 6.53639040e-08  3.33271455e-01  3.01747927e-01  2.10877506e-01
  -8.22324056e-02 -8.64087284e-01]]
DecisionTreeClassifier() Using PCA
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.318292	0.344197	0.331186	0.288834
2	0.393231	0.362061	0.361201	0.358971
3	0.361290	0.370021	0.369138	0.361444
4	0.332258	0.342702	0.341717	0.326632
5	0.337097	0.345062	0.344109	0.318398
6	0.342742	0.351657	0.350400	0.329881
7	0.343548	0.347136	0.344881	0.341826
8	0.370968	0.363832	0.363164	0.363314
9	0.343548	0.346188	0.350882	0.339140
10	0.353226	0.354133	0.336953	0.272066

In [8]:

```python
for name, model in stock_models.items():
    a, p, r, f = stock_rfe_eval(model, stock_df[all_stock_features],
                                stock_df["Class Daily Return"])
    stock_data = {
        'Fold': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'Accuracy': a,
        'Precision': p,
        'Recall': r,
        'F1-Score': f,
    }

    stock_scores = pd.DataFrame(stock_data).set_index('Fold')
    display(stock_scores)
```

```
DecisionTreeClassifier() Using RFE
```

	Accuracy	Precision	Recall	F1-Score

Fold	Accuracy	Precision	Recall	F1-Score
Fold				
1	0.305399	0.335865	0.338626	0.260303
2	0.346495	0.328387	0.339926	0.321607
3	0.346774	0.351313	0.343261	0.339635
4	0.300806	0.328797	0.311409	0.256077
5	0.325806	0.330587	0.334277	0.210185
6	0.353226	0.339394	0.338944	0.329983
7	0.347581	0.348216	0.344796	0.340891
8	0.388710	0.382545	0.382062	0.381041
9	0.336290	0.339469	0.372021	0.301421
10	0.359677	0.335537	0.342048	0.185986

In [9]:

```
for name, model in stock_models.items():
    a, p, r, f = stock_kbest_eval(model, stock_df[all_stock_features],
                                  stock_df["Class Daily Return"])
    stock_data = {
        'Fold': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'Accuracy': a,
        'Precision': p,
        'Recall': r,
        'F1-Score': f,
    }

    stock_scores = pd.DataFrame(stock_data).set_index('Fold')
    display(stock_scores)
```

```
DecisionTreeClassifier() Using k=3 best features
```

	Accuracy	Precision	Recall	F1-Score
Fold				
1	0.349718	0.365259	0.359046	0.305377
2	0.356164	0.352839	0.344614	0.337216
3	0.377419	0.338125	0.330384	0.300806
4	0.370968	0.391240	0.399634	0.368657
5	0.319355	0.338600	0.356970	0.240909
6	0.366935	0.317761	0.318583	0.317901
7	0.327419	0.317344	0.320953	0.318205
8	0.337903	0.317179	0.320793	0.306351
9	0.296774	0.301389	0.300846	0.277635
10	0.228226	0.236339	0.276374	0.188258

(b) Compare the feature selection methods and explain the differences observed;

PCA is the only one that seems to perform best in fold 10, so it seems that the intuition was right that PCA might be a good choice.

Generally, they all exhibit only small differences in performance, thus all of them could be used.