Exercise 5 - Evaluation

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' with Decision tree, KNN, Naive Bayes classifiers. (convert age and fare into classes ; exclude names from the attribute list) Determine:

(a) Accuracy of the classifiers with 5-fold CV

(b) Calculate theirs Precision, Recall and F1-score

First some imports and preprocessing

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

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(3))

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
Miss. Laina Heikkinen                                      1       3    0

                                                     Age  \
Name
Mr. Owen Harris Braund                              22.0
Mrs. John Bradley (Florence Briggs Thayer) Cumings  38.0
Miss. Laina Heikkinen                               26.0

                                                    Siblings/Spouses Aboard  \
Name
Mr. Owen Harris Braund                                                    1
```

```
Mrs. John Bradley (Florence Briggs Thayer) Cumings                     1
Miss. Laina Heikkinen                                                   0

                                                    Parents/Children Aboard  \
Name
Mr. Owen Harris Braund                                                  0
Mrs. John Bradley (Florence Briggs Thayer) Cumings                     0
Miss. Laina Heikkinen                                                   0

                                                       Fare  AgeGroup  \
Name
Mr. Owen Harris Braund                              7.2500         2
Mrs. John Bradley (Florence Briggs Thayer) Cumings  71.2833         2
Miss. Laina Heikkinen                               7.9250         2

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

In [2]:

```
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()
```

	Accuracy	Precision	Recall	F1-Score
Fold				

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

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

	Accuracy	Precision	Recall	F1-Score
Fold				
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()
```

	Accuracy	Precision	Recall	F1-Score
Fold				
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

It seems like for the decision Tree and KNeighbors, 3 fold would suffice, in terms of accuracy.

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

(a) Accuracy of the classifiers with 10-fold CV

(b) Calculate their Precision, Recall and F1-score

First simply import the dataset and set up the values:

In [3]:
```python
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"])
```

```python
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(),
    'KNN': KNeighborsClassifier(metric='manhattan'),
    'Naive Bayes': GaussianNB(),
}
print(stock_df.tail(3))
```

```
                  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
2021-09-17  15163.360352  15166.559570  14998.730469  15043.969727
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
2021-09-17  15043.969727  6682650000     -0.009086                0
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
2021-09-17    15106.151953     15191.898926     15143.947949     14875.465586
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
2021-09-17      13844.889136
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 [4]:

```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']


def stock_kfold_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)

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


for name, model in stock_models.items():
    a, p, r, f = stock_kfold_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()
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.331185	0.347599	0.329873	0.317819
2	0.308622	0.316172	0.322944	0.307801
3	0.343548	0.357075	0.359593	0.325359
4	0.305645	0.331909	0.324319	0.297232
5	0.304839	0.325341	0.295058	0.204497
6	0.304032	0.324861	0.322175	0.296573
7	0.344355	0.342394	0.345174	0.334500
8	0.377419	0.372596	0.371232	0.371268
9	0.300000	0.305284	0.301397	0.271422
10	0.277419	0.289440	0.277566	0.241323

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

Fold	Accuracy	Precision	Recall	F1-Score
1	0.338437	0.359269	0.346339	0.310099
2	0.352941	0.341236	0.343763	0.339733
3	0.368548	0.321497	0.302754	0.268633
4	0.358065	0.341657	0.338960	0.337458
5	0.334677	0.340029	0.341090	0.331863
6	0.325806	0.336826	0.335054	0.305033
7	0.334677	0.326823	0.324489	0.319987
8	0.334677	0.325243	0.324334	0.318713

Fold	Accuracy	Precision	Recall	F1-Score
9	0.308871	0.310049	0.316524	0.299645
10	0.345968	0.359740	0.379808	0.319568

```
GaussianNB()
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.359388	0.333333	0.119796	0.176250
2	0.439162	0.333333	0.146387	0.203434
3	0.402419	0.333333	0.134140	0.191298
4	0.426613	0.333333	0.142204	0.199359
5	0.364516	0.333333	0.121505	0.178093
6	0.252419	0.388310	0.410156	0.242221
7	0.352419	0.347758	0.214705	0.247343
8	0.376613	0.343797	0.273010	0.211165
9	0.317742	0.333310	0.220640	0.173877
10	0.303226	0.331054	0.348902	0.162319

In the stock_df case, 2-fold works well enough with GaussianNB, 3-fold with Kneighbors and Decision Tree prefers 7 (or 8) fold.