Exercise 7 - Association Rules

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) Build a Support vector machines (SVM) model with:

(a) Linear kernel (b) Polynomial kernel (c) radial basis function (RBF) kernel (d) sigmoid kernel Show the Comparison of the performances.

In [1]:

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

import pandas as pd
import numpy as np
from sklearn import preprocessing
from mlxtend.evaluate import accuracy_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.svm import SVC
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(2))

models = {
    'Linear SVM Kernel': SVC(kernel='linear'),
    'Poly SVM Kernel': SVC(kernel='poly'),
    'Rbf SVM Kernel': SVC(kernel='rbf'),
    'Sigmoid SVM Kernel': SVC(kernel='sigmoid'),
}
```

```
                                                      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_svm_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(name)
    return accuracy, precision, recall, f1


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

Linear SVM Kernel

Fold	Accuracy	Precision	Recall	F1-Score
1	0.842697	0.705882	0.857143	0.774194
2	0.758427	0.625000	0.737705	0.676692
3	0.762712	0.701493	0.681159	0.691176
4	0.745763	0.685714	0.676056	0.680851
5	0.819209	0.692308	0.789474	0.737705

Poly SVM Kernel

Fold	Accuracy	Precision	Recall	F1-Score
1	0.848315	0.735294	0.847458	0.787402
2	0.792135	0.638889	0.807018	0.713178
3	0.785311	0.761194	0.698630	0.728571
4	0.768362	0.685714	0.716418	0.700730
5	0.841808	0.723077	0.824561	0.770492

Rbf SVM Kernel

Fold	Accuracy	Precision	Recall	F1-Score
1	0.865169	0.720588	0.907407	0.803279
2	0.803371	0.638889	0.836364	0.724409
3	0.796610	0.731343	0.731343	0.731343
4	0.774011	0.685714	0.727273	0.705882
5	0.853107	0.738462	0.842105	0.786885

```
Sigmoid SVM Kernel
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.539326	0.338235	0.383333	0.359375
2	0.589888	0.388889	0.491228	0.434109
3	0.587571	0.447761	0.454545	0.451128
4	0.502825	0.400000	0.378378	0.388889
5	0.468927	0.307692	0.289855	0.298507

Seems like the linear and polynomial models have the highest accuracy, while the Sigmoid model performs the worst. We expect a clear correlation between e.g. PClass and Survived, thus a Sigmoid shouldn't fit, which is confirmed.

Build Support vector machines (SVM) model with your selected stock / market index using all attributes to predict 'daily returns'(decision). ('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)

Explain how the following kernels affects the performance of the model. (a) Linear kernel (b) Polynomial kernel (c) radial basis function (RBF) kernel (d) sigmoid kernel Show the Comparison of the Performance of the Kernels

In [7]:

```
from sklearn.svm import LinearSVC

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 = {
    'Linear SVM Kernel': LinearSVC(dual=False),
    'Poly SVM Kernel': SVC(kernel='poly'),
    'Rbf SVM Kernel': SVC(kernel='rbf'),
    'Sigmoid SVM Kernel': SVC(kernel='sigmoid'),
}
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 [8]:

```
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 [9]:

```
def kfold_svm_stock_eval(model, x: pd.DataFrame, y: pd.DataFrame):
    kf = KFold(n_splits=5)
    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(name)
    return accuracy, precision, recall, f1
```

In [10]:

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

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

```
Linear SVM Kernel
```

Accuracy	Precision	Recall	F1-Score

Fold	Accuracy	Precision	Recall	F1-Score
1	0.398227	0.332787	0.216155	0.190712
2	0.304313	0.319782	0.215310	0.227100
3	0.385484	0.333333	0.128495	0.185487
4	0.364516	0.333333	0.121505	0.178093
5	0.317339	0.333333	0.105780	0.160596

```
Poly SVM Kernel
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.289802	0.333333	0.096601	0.149792
2	0.289399	0.333333	0.096466	0.149630
3	0.256452	0.333333	0.085484	0.136072
4	0.251210	0.333333	0.083770	0.133892
5	0.354435	0.365574	0.264213	0.262435

```
Rbf SVM Kernel
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.399033	0.333333	0.133011	0.190147
2	0.414349	0.333009	0.138284	0.195419
3	0.310081	0.374814	0.361534	0.238784
4	0.377823	0.341705	0.606177	0.265555
5	0.315726	0.331403	0.297484	0.166609

```
Sigmoid SVM Kernel
```

Fold	Accuracy	Precision	Recall	F1-Score
1	0.289802	0.333333	0.096601	0.149792
2	0.285369	0.323368	0.193278	0.225996
3	0.362097	0.335218	0.282369	0.213611
4	0.368952	0.331721	0.244251	0.272592
5	0.326613	0.325263	0.194355	0.220808

It seems like the poly kernel performs the worst, while rbf performs the best.

Sigmoid is slightly behind linear in performance.