# **Artificial Intelligence and Machine Learning Fundamentals**

**Activity 6**: Stock Price Prediction with Quadratic and Cubic Linear Polynomial Regression with Multiple Variables

This section will discuss how to perform linear, polynomial, and support vector regression with scikit-learn. We will also learn to predict the best fit model for a given task. We will be assuming that you are a software engineer at a financial institution and your employer wants to know whether linear regression, or support vector regression is a better fit for predicting stock prices. You will have to load all data of the S&P 500 from a data source. Then build a regressor using linear regression, cubic polynomial linear regression, and a support vector regression with a polynomial kernel of degree 3. Then separate training and test data. Plot the test labels and the prediction results and compare them with the y=x line. And finally, compare how well the three models score.

Let's load the S&P 500 index data using Quandl, then prepare the data for prediction.

You can read the process in the Predicting the Future section of the topic Linear Regression with Multiple Variables.

import quandl

import numpy as np

from sklearn import preprocessing

from sklearn import model\_selection

from sklearn import linear\_model

from sklearn.preprocessing import PolynomialFeatures

from matplotlib import pyplot as plot

from sklearn import svm

data\_frame = quandl.get("YALE/SPCOMP")

data\_frame[['Long Interest Rate', 'Real Price',

'Real Dividend', 'Cyclically Adjusted PE Ratio']]

data\_frame.fillna(-100, inplace=True)

# We shift the price data to be predicted 20 years forward

data\_frame['Real Price Label'] = data\_frame['RealPrice'].shift(-240)

# Then exclude the label column from the features

features = np.array(data\_frame.drop('Real Price Label', 1))

# We scale before dropping the last 240 rows from the features

scaled\_features = preprocessing.scale(features)

# Save the last 240 rows before dropping them

scaled\_features\_latest240 = scaled\_features[-240:]

# Exclude the last 240 rows from the data used for #

# modelbuilding

scaled\_features = scaled\_features[:-240]

# Now we can drop the last 240 rows from the data frame

data\_frame.dropna(inplace=True)

# Then build the labels from the remaining data

label = np.array(data\_frame['Real Price Label'])

# The rest of the model building stays

(features\_train,

features\_test,

label\_train,

label\_test

) = model\_selection.train\_test\_split(

scaled\_features,

label,

test\_size=0.1

)

Let's first use a polynomial of degree 1 for the evaluation of the model and for the prediction. We are still recreating the main example from the second topic.

model = linear\_model.LinearRegression()

model.fit(features\_train, label\_train)

model.score(features\_test, label\_test)

1. The output is as follows:

0.8978136465083912

1. The output always depends on the test data, so the values may differ after each run.

label\_predicted = model.predict(features\_test)

plot.plot(

label\_test, label\_predicted, 'o',

[0, 3000], [0, 3000]



The closer the dots are to the y=x line, the less error the model works with.

It is now time to perform a linear multiple regression with quadratic polynomials. The only change is in the Linear Regression model

poly\_regressor = PolynomialFeatures(degree=3)

poly\_scaled\_features = poly\_regressor.fit\_transform(scaled\_features)

(poly\_features\_train,

poly\_features\_test,

poly\_label\_train,

poly\_label\_test) = model\_selection.train\_test\_split(

poly\_scaled\_features,

label,

test\_size=0.1)

model = linear\_model.LinearRegression()

model.fit(poly\_features\_train, poly\_label\_train)

print('Polynomial model score: ', model.score(

poly\_features\_test, poly\_label\_test))

print('\n')

poly\_label\_predicted = model.predict(poly\_features\_test)

plot.plot(

poly\_label\_test, poly\_label\_predicted, 'o',

[0, 3000], [0, 3000]

)

The model is performing surprisingly well on test data. Therefore, we can already suspect our polynomials are overfitting for scenarios used in training and testing. We will now perform a Support Vector regression with a polynomial kernel of degree 3.

model = svm.SVR(kernel='poly')

model.fit(features\_train, label\_train)

label\_predicted = model.predict(features\_test)

plot.plot(

label\_test, label\_predicted, 'o',

[0,3000], [0,3000]

)



model.score(features\_test, label\_test)

The output will be **0.06388628722032952**.

We will now perform a Support Vector regression with a polynomial kernel of degree 3.