Data Analysis and Knowledge Discovery

Exercise Work 2

Tatu Seppä-Lassila

December 14, 2015

1 Task 1: Wine Quality Prediction with Linear Regression

In this task, scikit-learn's LinearRegression method was used. It takes two arguments: input variables and output variables. Therefore, input variables and output variable (quality) was separated from the data to their own variables. After that LinearRegression was called and model was fitted using fit method:

```
lm = LinearRegression()
lm.fit(x,y)
```

After fitting, intercept and coefficients can be obtained and those can be combined to weight vector:

```
# Get intercept and coefficients
coefs = lm.coef_
intercept = lm.intercept_
# Combine them to get the weight vector
weightVector = np.hstack((np.array(intercept), np.array(coefs)))
```

Result is the following weight vector presented here in two rows to save horizontal space:

$$\begin{bmatrix} 150.19284 & 0.06552 & -1.86318 & 0.02209 & 0.08148 & -0.24728 \\ 0.00373 & -0.00029 & -150.28418 & 0.68634 & 0.63148 & 0.19348 \end{bmatrix}$$

I assume the method provides the best weight vector for the samples automatically.

After the initial setup, quality can be predicted with ”predict” method. The method takes sample with input variables as its argument.

For calculating out-of-sample and in- sample error vectors, the data was splitted to five parts using numpy’s ”array_split” method. In out-of-sample error calculation, each data part is iterated through and predicted quality is subtracted from actual quality. The result is added to the error vector of the data part. So in the end five error vectors are constructed.
For calculating in-sample error vectors, ”combinations” method from Python’s itertools was used to calculate combinations in which one data part is excluded from the rest of the five data parts. With five data parts, there are five combinations. After calculating combinations, four data parts in each combination was combined together. Finally, the abovementioned procedure was used to calculate error values and constructing error vectors.
The out-of-sample and in-sample error vectors were concatenated and error histograms were plotted using concatenated error vectors.

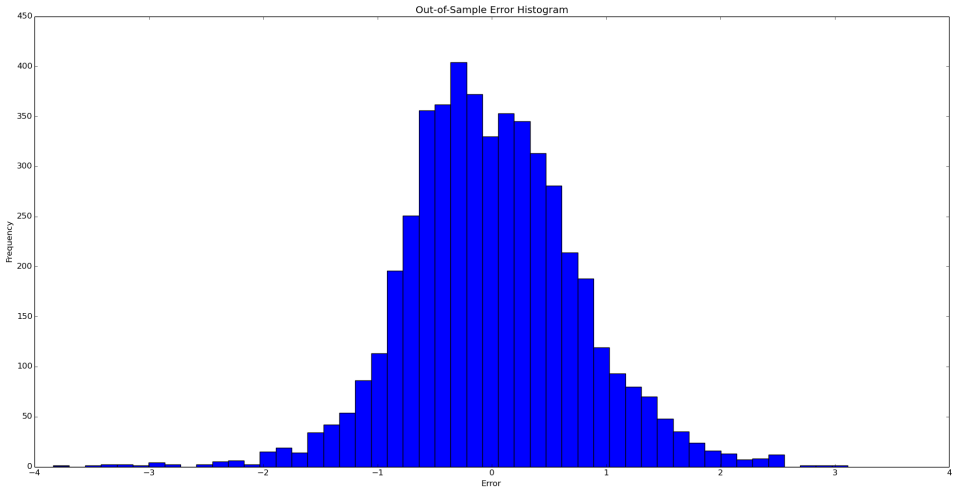

Figure 1: Out-of-sample error histogram plotted with matplotlib

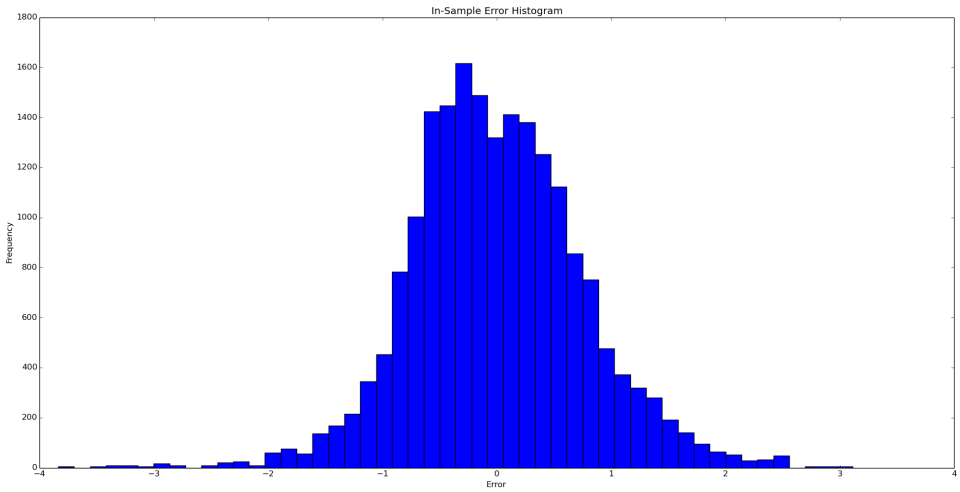

Figure 2: In-sample error histogram plotted with matplotlib

Figures 1 and 2 show the out-of-sample and in-sample error histograms. They look similar, but Figure 2 has higher frequencies, because the concatenated error vector has more elements.

The average error and variance for the out-of-sample and in-sample are reported in the following table:

	average error	**variance**
out-of-sample	9.83496755747e-13	0.563154062989
in-sample	9.83614260773e-13	0.563154062989

2 Task 2: Wine Quality Prediction with K Nearest Neighbors

In the second task, scikit-learn's KNeighborRegressor method was used to predict wine quality with k nearest neighbors. Usage of method is similar to method used in the first task. The number of neighbors is passed as an argument. The method has optional arguments like algorithm and weights. The weights argument defaults to uniform weights. The API does not specify default algorithm, but I suppose auto is used and the method tries to decide the most appropriate algorithm based on the values passed to fit method. The following method was written to setup model to calculate k nearest neighbors for asked k=1..8:

```
# Calculates model with K nearest neighbors, returns the model
def calculateKNearestNeighborsModel(data, numberOfNeighbors):
  # Select input variables as x and typecast to numpy array
  x = np.array(data.iloc[0:,0:11])
  # Select output variable (quality) as y and typecast to numpy array
  y = np.array(data.quality)
  neighbors = KNeighborsRegressor(n_neighbors=numberOfNeighbors)
  neighbors.fit(x, y)
  return neighbors
```

Now prediction can be done by passing a sample as argument for the ”predict method”. If k >1, average of k nearest neighbors quality attribute is returned. Calculating out-of-sample and in-sample error vector was done similarly than in Task 1.

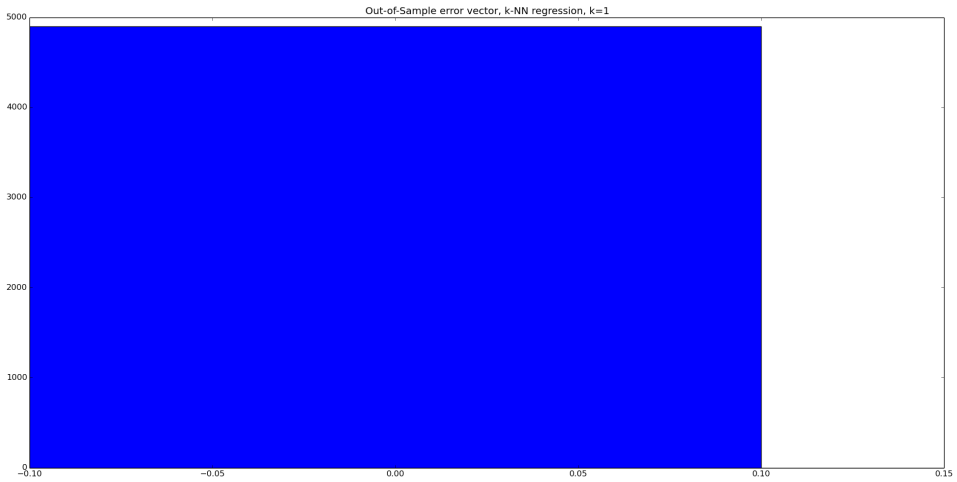

Figure 3: Out-of-sample error histogram, k=1. Plotted with matplotlib

Figure 3 shows the out-of-sample histogram for k=1. It seems a bit weird. This happens, because the query set matches the training set. Hence, the nearest neighbor of each point is the point itself, at distance of zero. Therefore, the concatenated error vectors are full of zeros. The in-sample error histogram is similar with higher frequencies. The figure is omitted from this report. The data could have been split to two parts, so the training set and query set would have been different.

To decide k best, out-of-sample mean errors for k=1..8 were printed:

```
Out-of-sample mean error when neighbors 1 : 0.0
Out-of-sample mean error when neighbors 2 : -0.0134748877093
Out-of-sample mean error when neighbors 3 : -0.0101401932762
Out-of-sample mean error when neighbors 4 : -0.0114842792977
Out-of-sample mean error when neighbors 5 : -0.0145773785218
Out-of-sample mean error when neighbors 6 : -0.0162991697291
Out-of-sample mean error when neighbors 7 : -0.0147290439246
Out-of-sample mean error when neighbors 8 : -0.014623315639
```

k=1 was excluded for the abovementioned reason. So, k best is k=3.

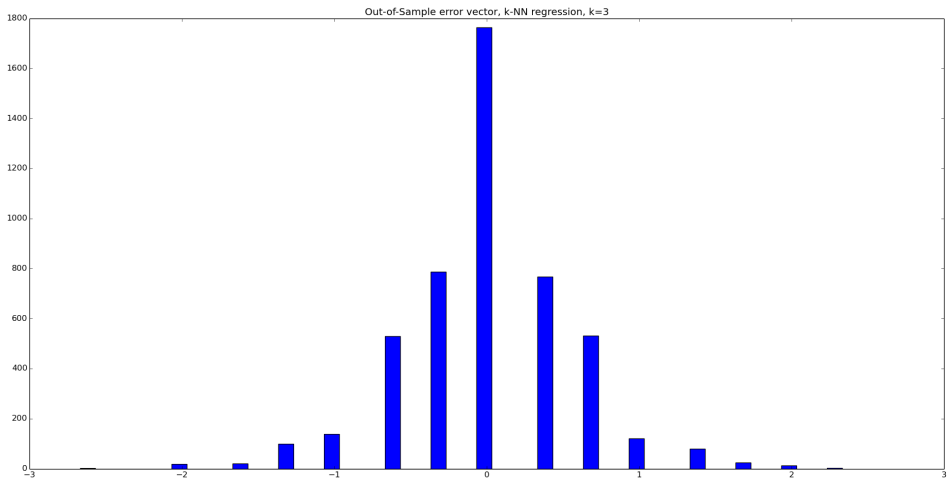

Figure 4: Out-of-sample error histogram, k=3. Plotted with matplotlib

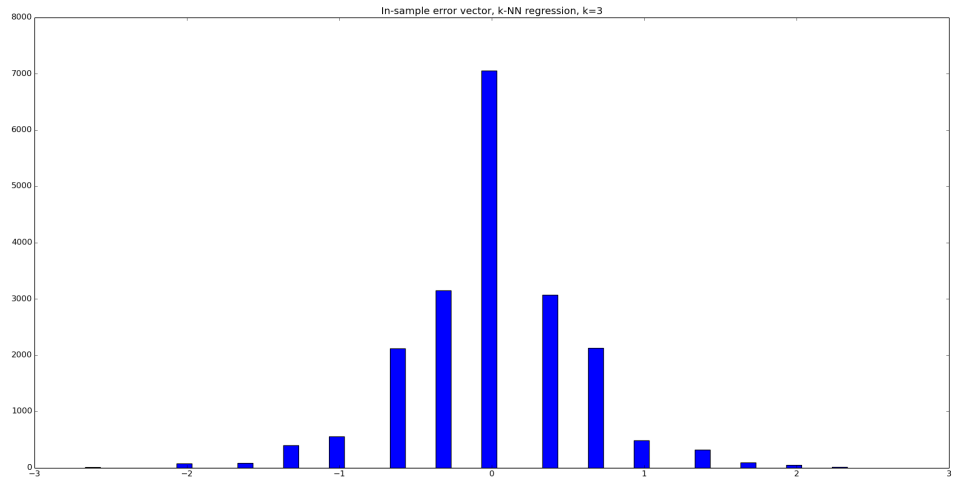

Figure 5: In-sample error histogram, k=3. Plotted with matplotlib

Figures 4 and 5 show the error histograms for k=3. The histograms seem similar, but in-sample histogram has greater frequencies.

I also experimented a bit with scikit-learn NearestNeighbor method:

```
x = np.array(data.iloc[0:,0:11])
y = np.array(data.quality)

nbrs = NearestNeighbors(n_neighbors=1, algorithm='ball_tree').fit(data)

distances, indices = nbrs.kneighbors(data)

for i in range(0, len(y)):
  print('Actual quality: ' + str(y[i]) + ' predicted quality: ' + str(y[
   indices[i][0]]))
  print(y[i] == y[indices[i][0]])
```

The method returns distances and indices of nearest neighbors for all data points. Index refers to the index in the training data. So, if k >1 the average needs to be calculated for nearest neighbors to predict quality. Given this, it was easier to use KNeighborRegressor for this task.