Exercise work – Part 2

Data Analysis and Knowledge Discovery

Name: Joonas Syysvirta

Student number: 502603

Wine set: Red wines

Preparing the data

I had the data well prepared from the first exercise work, with the feature data stored in an array of arrays (each feature's/attribute's values stored in separate arrays), and the output (quality) values stored in a separate array. Therefore, no extra data preparation was needed for this assignment.

2.1. Linear regression

For the linear regression I chose to calculate the intercept and the coefficients for the features using *scikit-learn*'s *LinearRegression* library.

Coefficients:

Fixed acidity: 0.0249905527

Volatile acidity: -1.08359026

Citric acid: -0.182563948

Residual sugar: 0.0163312698

Chlorides: -1.87422516

Free sulfur dioxide: 0.00436133331

Total sulfur dioxide: -0.00326457970

Density: -17.8811638

pH: -0.413653144

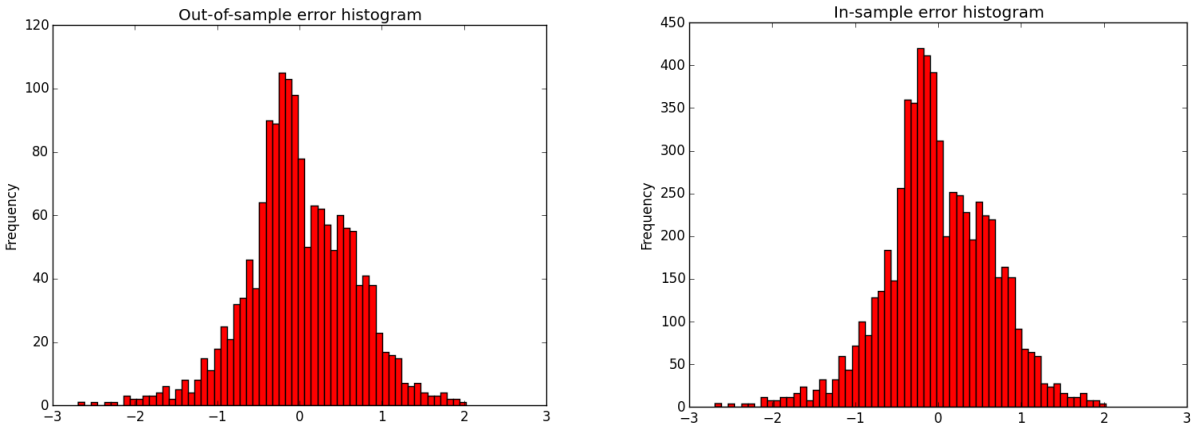
Sulphates: 0.916334413

Alcohol: 0.276197699

Intercept: 21.9652084

Now I could combine the intercept and the coefficients, by stacking them horizontally with *numpy*'s *hstack()*, which gives us the following weight vector $\boldsymbol{\beta}$ for the whole data**:**

[2.19652084e+01, 2.49905527e-02, -1.08359026e+00, -1.82563948e-01,
1.63312698e-02, -1.87422516e+00, 4.36133331e-03, -3.26457970e-03,
-1.78811638e+01, -4.13653144e-01, 9.16334413e-01, 2.76197699e-01]



Above: The absolute error histograms

To calculate the absolute prediction errors, I created one method that calculates the in-sample error vectors, and another method that calculates the out-sample error vectors for the data. Then I created a method that concatenates the vectors given to it, to get the assembled error vectors (one for in-sample and one for out-of-sample). To calculate the error vectors I chose to split the data into 5 parts. To do this I used *numpy's array_split()* method.

For the out-sample errors I did the following for each part separately:

I made the predictions for each sample using *scikit-learn*'s *LinearRegression's predict()* method. Now I could get the error for each sample by subtracting the prediction from the sample's *quality* value. I saved the errors to a list to be used later.

For the in-sample-errors I did the following for each part separately:

I subtracted the part from the whole data set, and then for each sample in the resulting data set I made the predictions, calculated the error by subtracting the prediction from the sample's *quality* value, and saved the errors to a list to be used later.

Now I could assemble the error vectors using my vector concatenation method, and plot the histograms by using the method I created in the first exercise work.

Average out-of-sample error (mean):

1.07758982753e-15

Average in-sample error (mean):

1.07536799283e-15

Variance of in-sample error:

0.416767167221

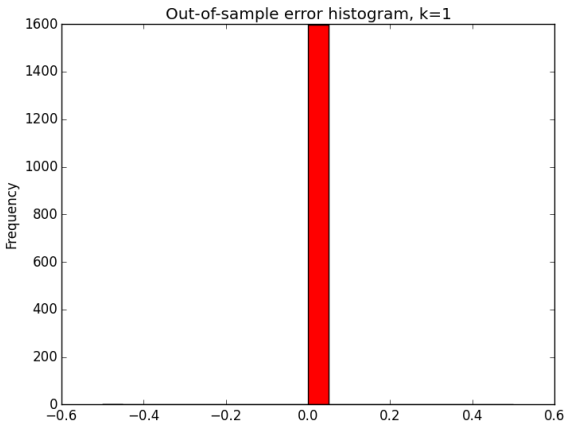
Variance of out-of-sample error:

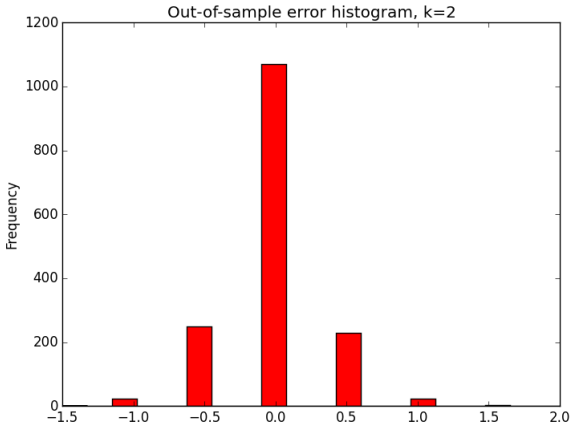
0.416767167221

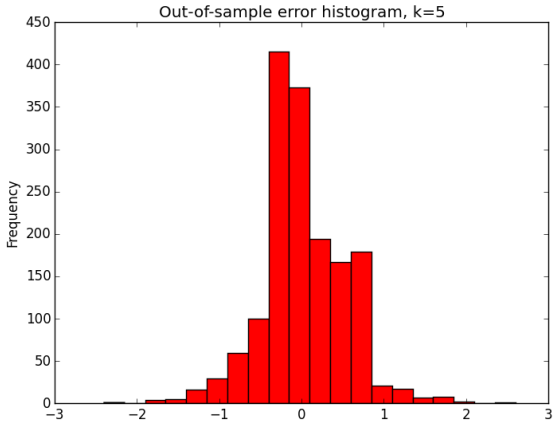
To get the average errors and the variances, I used *numpy's mean()* and *var()* methods, respectively.

2.2. k-NN regression

Out-of-sample:



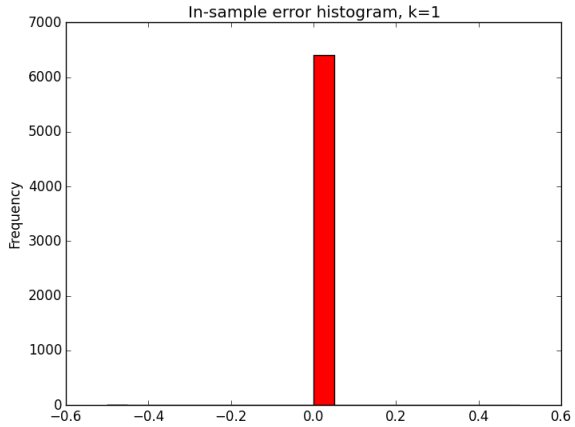


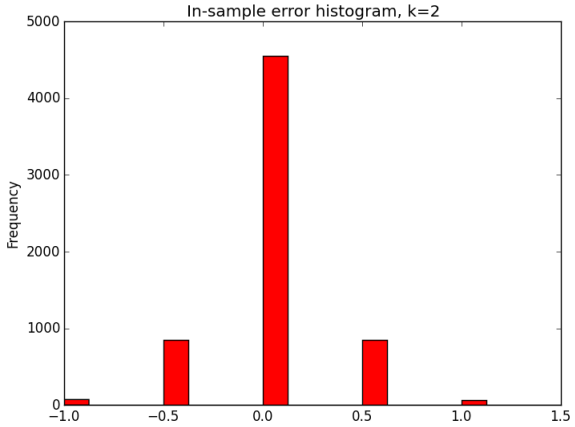


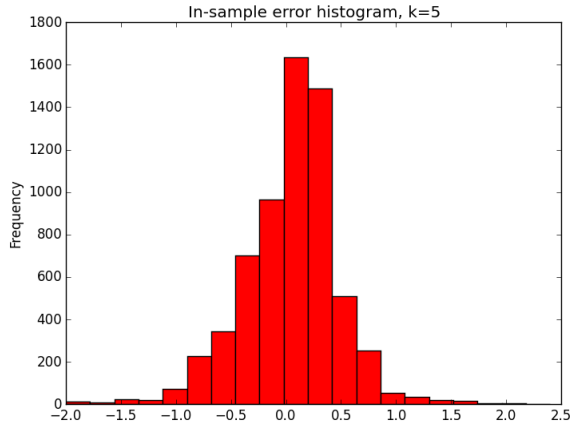
Since I used the same data set as the query set and the training set, the nearest neighbor of each data point is the point itself (with zero distance). So, I've included here the error histogram of the

actual nearest neighbors (k=2). The out-of-sample average error is smallest when k=5 (0.00237648530331).

In-sample:







Since the nearest neighbor of each data point is the point itself here (again I included the histogram anyway), I've included here the error histogram of the actual nearest neighbors (k=2), and for the $\mathbf{k_{best}}$, which is 5.

For the k-NN regression I created a function that calculates and returns the mean of the *quality* values of the k neighbors of a given sample in a data part. I edited the functions that calculate the in-sample and out-of-sample error vectors (the ones I created for 2.1.), to subtract the aforementioned mean from the samples's *quality* value. Otherwise the functions work as in 2.1. To

get the nearest neighbors I used *scikit-learn's NearestNeighbors* (with the ball-tree algorithm), and then used the indices to access these neighbors' and their *quality* values.

After using my concatenation method to assemble the error vectors for both the absolute errors, I calculated the mean of the out-of-sample error vectors for each k=1..8. The out-of-sample average error is smallest when k=5 (0.00237648530331), so $\mathbf{k_{best}}$ =5.

k	**Average out-of-sample error**
1	0.0
2	-0.00375234521576
3	0.00625390869293
4	0.00328330206379
5	0.00237648530331
6	0.00583698144674
7	0.00464576074332
8	0.0083646028768

k	**Average in-sample error**
1	0.0
2	-7.81738586617e-05
3	0.00359599749844
4	0.0093808630394
5	0.0121013133208
6	0.0128465707734
7	0.0117037434111
8	0.0120974046279