Datamining report
Databaser og datmining 2015

Jonathan Schrøder Holm Hansen // fdg890

April 9, 2015

Preliminary comments

This report is part of the solution to the final assignment for the datamining part of the course *Databases and datamining* at DIKU, spring 2015. The report presents the accomplished results, interprets when necessary, and introduces some thoughts on the chosen ways to solve the assignment. Solutions were computed using Python 3.4 and SQLite3. The entire deliverable consists of:

- This final report
- The Python scripts used to solve the assigned problems
- Command prompt prints of the compiled script in .txt
- README.txt listing used tools and programs

Photometric Redshift Estimation

1.2 Mean and sample variance

The sample mean of $Redshift\_Train\_Y$ was calculated using the library average function of SQLite3. This solution seemed straight forward and had only the disadvantage that the sample mean was not stored in a variable.

The biased sample variance of $Redshift\_Train\_Y$ was found in the same way as the sample mean using extended relational algebra statements in SQLite3. The choice to solve the problem without storing the sample mean in a variable forced me to copy it from print to the SQLite3 statement, which may in theory pose a possible loss of precision due to rounding of the printed sample mean. However the used data set holds three decimals so I decided that my computed 16-decimal solutions was sufficiently precise (se the command promt script). The sample mean and the biased sample variance are reported in figure 0.1.

Sample mean	**Biased sample variance**
0.1555464	0.0136722

Figure 0.1: Data: Redshift_Train_Y

1.3 Linear regression

Building the affine linear model for *Redshift_Train* required calculations that was impossible (or at least very difficult) in relational algebra why the data was imported to matrices using a simple for loop that reads one row of a data table at a time. With the data in matrices the package *numpy* (also used to create the matrices) enabled computation of the model parameters. First however a column of 1s was added to the *Redshift_Train_X*-matrice. The model parameters are reported in figure 0.2.

	Model parameters
w_1	-0.00598922
w_2	0.10965194
w_3	0.27844859
w_4	-0.00183018
w_5	0.00935054
vw_6	-0.00863705
w_7	0.00414825
w_8	-0.10749617
w_9	-0.0098086
w_10	0.04078572
w_11	-0.88069336

Figure 0.2: Data: Redshift_Train

The training error of the model could be found by computing the mean-squared error (MSE) of the model over *Redshift_Train*. First the sum-of-squares error, J, was computed using a for loop that proceeded row by row through tables X and Y. The result was divided by the number of elements in the set - 2500 in this case. The MSE is reported in figure 0.3.

Mean-squared error, MSE
0.00280929

Figure 0.3: Data: Redshift_Train

Figure 0.4 shows the MSE of *Redshift_Train* next to the biased sample variance. The MSE is significantly (informally (possibly formally as well)) lower than the biased sample varience. First of all this suggests that a linear model is justified. Our estimated dependent variables are closer to the actual data than there is variance within the data. However this also means that there is some variance in the dependent variables that our model does not capture. In other words our model parameters predicts dependant variables that on an average are closer to the regression line that it is actually the case. One reason for this could be that there is some randomness in the data or that some significant information is not captured by the data variables.

MSE	**Biased sample variance**
0.00280929	0.0136722

Figure 0.4: Data: Redshift_Train

To find the test error of the model it was run over the $Redshift\_Test$ data. This means that the model parameters from the $Redshift\_Train$-model was tested over a so far unused set of data. Figure 0.5 shows the mean-squared error of the model build on $Redshift\_Train$ over $Redshift\_Test$ and the biased sample variance of $Redshift\_Test$:

MSE	**Biased sample variance**
0.00319585	0.0147140

Figure 0.5: Model: Redshift_Train Data: Redshift_Test

The MSE is a little higher when our model is used over the $Test$ dataset. As it shows the biased sample variance is also a bit higher, which might account for some of the poorer performance of the model. Overall the model performs well. As before the MSE is lower than the biased sample variance when used on both datasets which is a good thing.

Cyberfraud

2.1 Classification

The nearest neighbour classifier (1NN) was build on two helping functions. The first, *distance*, simply computes the distance between two vectors using the numpy.linalg-function *norm* that gives the length of a vector.

The second helping function, *minDistance*, takes a matrix and a vector representing a datapoint and finds the vector/datapoint in the matrix that is closest to the given vector. This function uses *distance* and a for loop to run through the entire matrix.

The main function of the 1NN, *compareY*, uses *minDistance* to go vector by vector through one matrix and finding the nearest vector in another matrix. While doing this *compareY* also compares the dependent values of the two matrices and keeps count on the matches.
Finally the performance of the 1NN is translated to a percentage using simple calculations. The result is reported in figure 0.6:

Truecount	**Performance**
157	98.125 %

Figure 0.6: Performance of 1NN

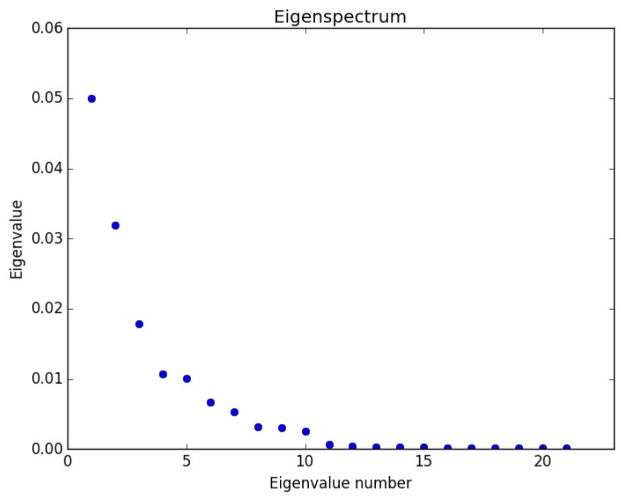
2.2 Dimensionality reduction and visualization

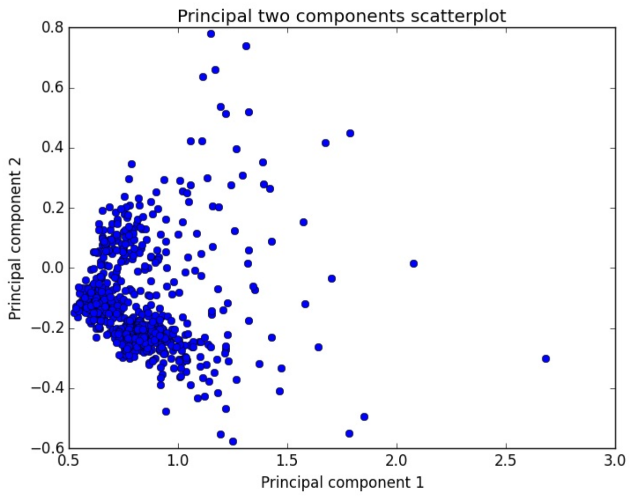
To conduct a principal component analysis two functions were written. First however the covariance matrix of $Keystroke\_Train\_X$ was computed using the numpy.cov library function. Likewise the eigenvectors and eigenvalues were computed using numpy.eigh. For some peculiar reason the eigenvalues were listed in increasing order. The first function written, *sumInc*, simply sums a specified number of values from a matrix from top down. The model function, PSA, computes how many eigenvalues are needed to reach a desired level of explained variance. Results are reported below in figure 0.7.

Componentcount	**Explained variance**
7	92.02 %

Figure 0.7: PCA analysis of components needed to reach 90% explained variance

The first of the following scatterplots shows the eigenvalues of the covariance matrix. Notice that the two principal components make up close to two thirds of the eigenvalue sum meaning that these two dimensions are vital in a dimension reduction.
The second scatterplot simply shows the data encoded to two dimensions.





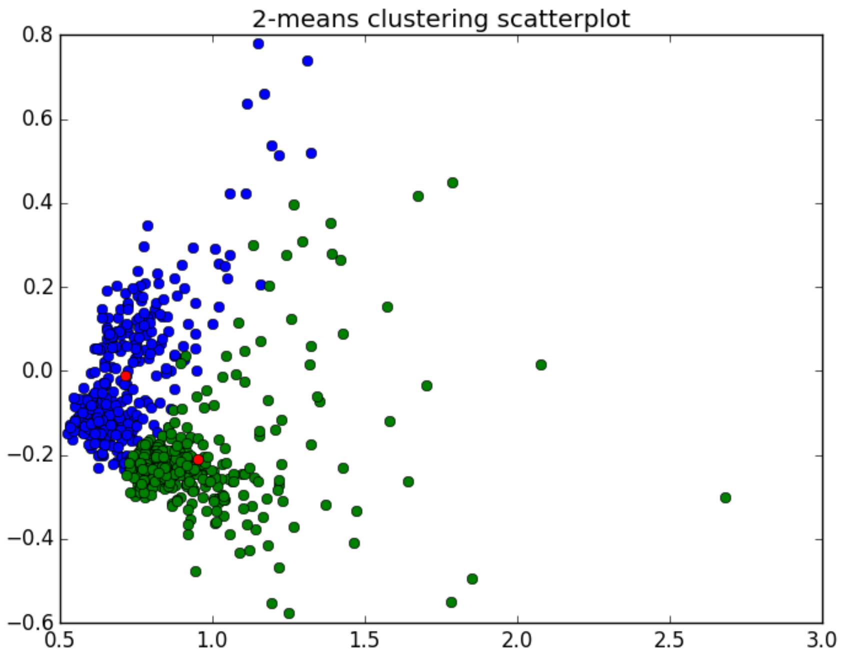
2.3 Clustering

To conduct the 2-means clustering a few functions were written. The first, *clusterAssign*, assigns a datapoints to one of two clusters for an entire dataset.
The main clustering function, *twoMeansCluster*, repeatedly assigns datapoints to one of two clusters and computes new cluster centroids to assign by until the new centroids are the same as in the previous iteration. Finally a function, *clusterDistortion*, computes the distortion measure of the found clusters. Figure 0.8 shows the results.

	Cluster n	**Distortion**
Cluster 1	304	22.6
Cluster 2	336	52.1

Figure 0.8: Results of 2-means clustering

The following is a scatterplot of the cluster data base shifted using the two principal components. Judging by appearance the clusters are meaningful. The boundary between the clusters is a clearly visible line and only few datapoints seem to "out of place".


2-means clustering scatterplot