Statistical Methods for Machine Learning

Exam

Troels Thomsen - qvw203

6. April 2015

Department of Computer Science
University of Copenhagen

Contents

1 Photometric Redshift Estimation

1.1 Question 1 - Linear Regression

For linear regression we decided to use both the Maximum Likelyhood approach (ML), the Maximum A Posteriori approach (MAP), and Support Vector Regression (SVR) with a linear kernel.
For ML and MAP we used our own implementation from the course exercises, but for SVR we used the python Sci-kit machine learning library[1].

To determine the best model, we compare the results for each model, using the mean-squared-error (MSE).
From the following test results we conclude that the model which fits our data best is Maximum Likelyhood, since we could not find an α which provided a smaller MSE for Maximum A Posteriori, thus yielding MAP only as good as ML.

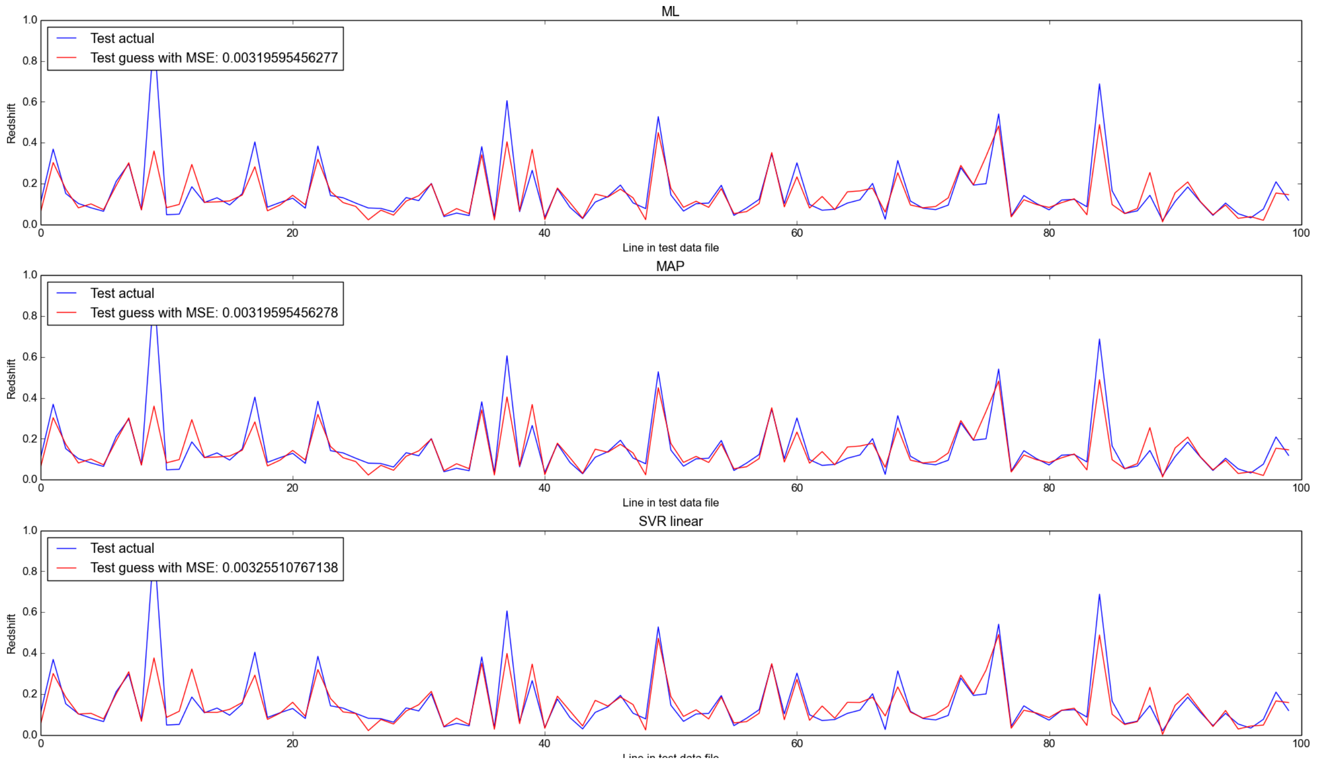

Figure 1: Chosen linear models - ML (Top), MAP (Middle), SVR Linear Kernel (Bottom).

[1]`http://scikit-learn.org/`

1.1.1 ML results

- Training mean-squared error: 0.00281098722063
- Test mean-squared error: 0.00319595456277

1.1.2 MAP results

Through grid-search we found the best α to be 0, and the best β to be 1.
We searched β from 1 to 100, and α from 0 to 1 with a step of 0.01.

- Training mean-squared error: 0.00281098722063
- Test mean-squared error: 0.00319595456278

As expected, we find our MSE for MAP to be equal to our MSE for ML, since our α is 0.

1.1.3 SVR linear kernel results

To discover the best parameters for SVR with linear kernel we used cross validation on γ and C. We searched through γ and C incrementing by an order of magnitude in range $[-3, 3]$, starting with γ and C values of 0.1 and 10 respectively.

```python
    gammas = [0.1*10**i for i in range(-3, 4)]
    cs = [10*10**i for i in range(-3, 4)]
```

Figure 2: Range of γ and C to use for cross validation.

We found the best γ to be 0.0001 and best C to be 100.

- Training mean-squared error: 0.00293831151397
- Training mean-squared error: 0.00325510767138

1.2 Question 2 - Non-linear Regression

For non-linear regression we decided to use Support Vector Regression with a Radial Basis Function (RBF) kernel and an polynomial kernel.

Again we compute and compare the MSE of our results, in order to determine the performance of our non-linear models.

In order to find the parameters for SVR, we performed cross validation on γ and C in the same manner as in section 1.1.3. We found the best γ to be 0.01 and the best C to be 10.

It is worth noting that we are using the same γ and C parameters for the polynomial kernel, as for the RBF kernel. This is due to the fact that we did not get a cross validation result for the polynomial kernel, even after five hours of runtime. We still decided to use the γ and C found for RBF, since they provided better results than the default γ and C values for the polynomial kernel.

As shown by the MSE of 0.00295284171625, we actually find that Support Vector Regression with a Radial Basis Function kernel outperforms our Maximum Likelyhood approach (although very slightly).

1.2.1 SVR RBF kernel results

- SVR rbf mean-squared training error: 0.00227026743696
- SVR rbf mean-squared test error: 0.00295284171625

1.2.2 SVR Polynomial kernel results

- SVR polynomial mean-squared training error: 0.00267447888801
- SVR polynomial mean-squared test error: 0.00426904768726

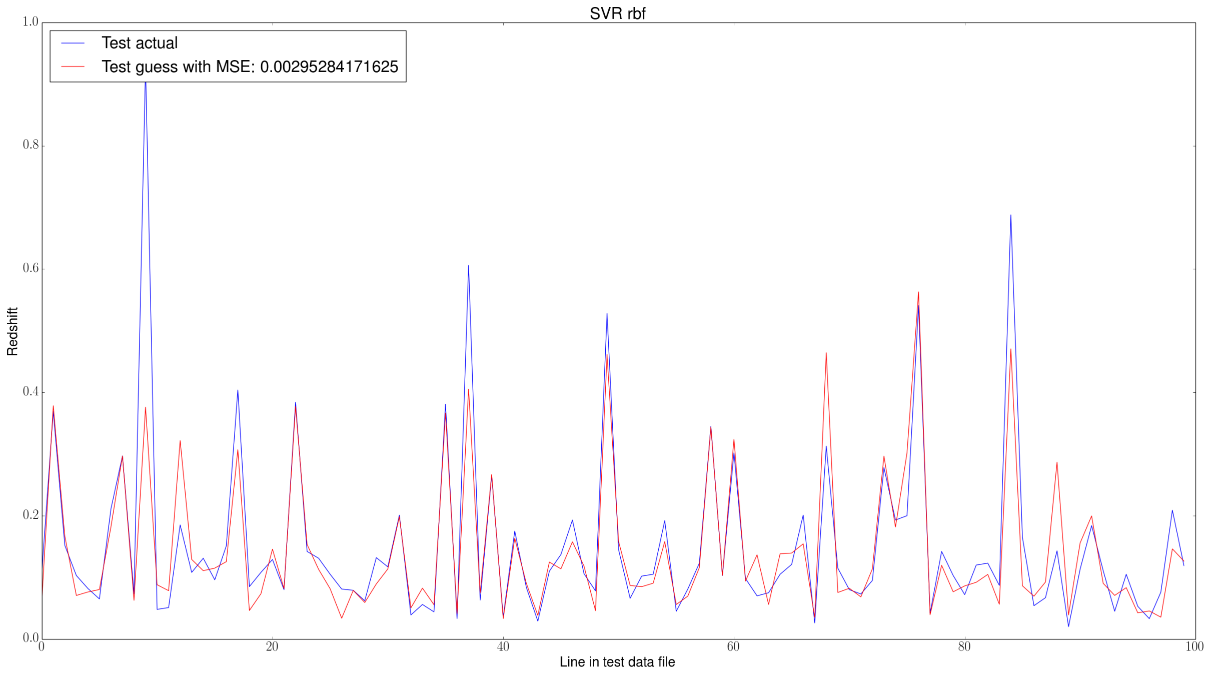

Figure 3: SVR rbf kernel model measured against the test data, for the first 100 test data points.

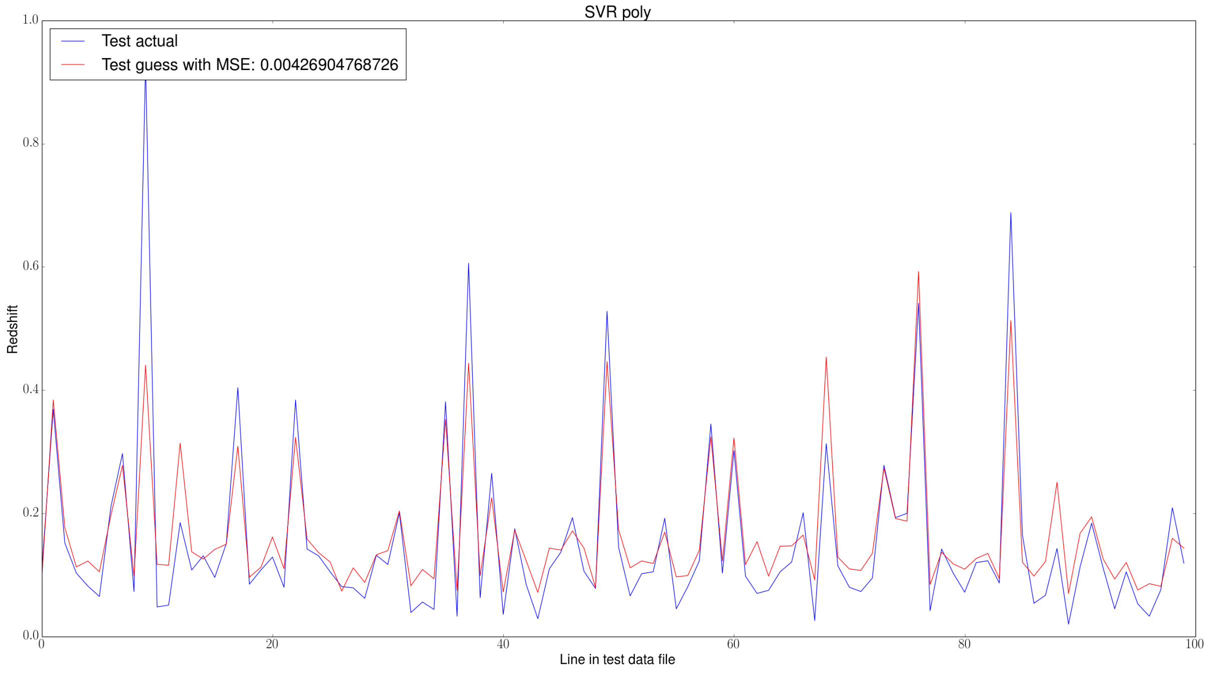

Figure 4: SVR polynomial kernel model measured against the test data, for the first 100 test data points.

1.3 Question 7 - Overfitting

In the case of photometric redshift estimation, we found that the following overfitting problems would be likely to apply.

1. *Traditional overfitting*: Train a complex predictor on too-few examples.
   In the case of redshift estimation we are modelling based on 10 parameters, with 2500 training samples. If we had only a hundred or a few hundred samples, such a complex predictor might have been easily overfitted in a model with several degrees of freedom.

2. *Parameter tweak overfitting*: Use a learning algorithm with many parameters. Choose the parameters based on the test set performance.
   In any case it is a common mistake to mix test and training data while building the model, and is likely to cause overfitting.
   In our case we are choosing our parameters for Support Vector Regression on the training set with k-fold cross validation on the training set.

3. *Brittle measures*: leave-one-out cross validation
   In the case of our Support Vector Regression, we used k-fold cross validation to choose our model parameters γ and C. There might have been a chance of overfitting the γ and C parameters during cross validation, had we used leave-one-out cross validation. This might have introduced a bias in our model towards a subset of our training set.

   It does not seem as if the parameters for our Radial Basis Function kernel SVR model were biased. In both the case of training and test data mean-squared-error, the model performed just slightly better than Maximum Likelyhood.

2 Cybercrime Detection

2.1 Question 3 - Binary Classification

For binary classification we chose to use Linear Discriminant Analysis as our linear model, and K-Nearest Neighbour classification as our non-linear model.

2.1.1 LDA results

- Training accuracy for LDA: 0.9921875
- Test accuracy for LDA: 0.9875

For training and test sensitivity and specificity we got the following results for our LDA model.

- Training
  - Sensitivity: 0.990625
  - Specificity: 0.99375
- Test
  - Sensitivity: 1.0
  - Specificity: 0.975

We see that for the test-set we are able to classify all of the good-guys correctly, whereas we are unable to correctly classify all of the bad-guys. Since we most likely could not always classify both correctly, this is our best-case scenario. We do not wish to classify good-guys as bad-guys, since we would then anger our legitimate customers.
Classifying a few bad-guys as good-guys is much better, since we might be able to use other measures to further prevent them from doing bad-guy stuff.

2.1.2 KNN results

For our KNN classifier we ran cross validation in order to discover the best k. We used a fold of 5 and a max_k of 25. We found the best k to be 1.

- Training accuracy for KNN: 1.0
- Test accuracy for KNN: 0.98125

For training and test sensitivity and specificity we got the following results for our 1-NN model.

- Training
  - Sensitivity: 1.0
  - Specificity: 1.0
- Test
  - Sensitivity: 0.9875
  - Specificity: 0.975

We are able to classify good-guys slightly worse than with our LDA classifier, and classify bad-guys with the same precision. This makes KNN slightly less preferable to our LDA model, since we with KNN are going to exclude some legitimate customers from our service.

2.2 Question 4 - Principal Component Analysis

In order to compute the eigenvalues and eigenvectors we used the numpy.linalg.eig library. We then use the eigenvectors with the smallest eigenvalues to define W and then scale our datapoints accordingly.

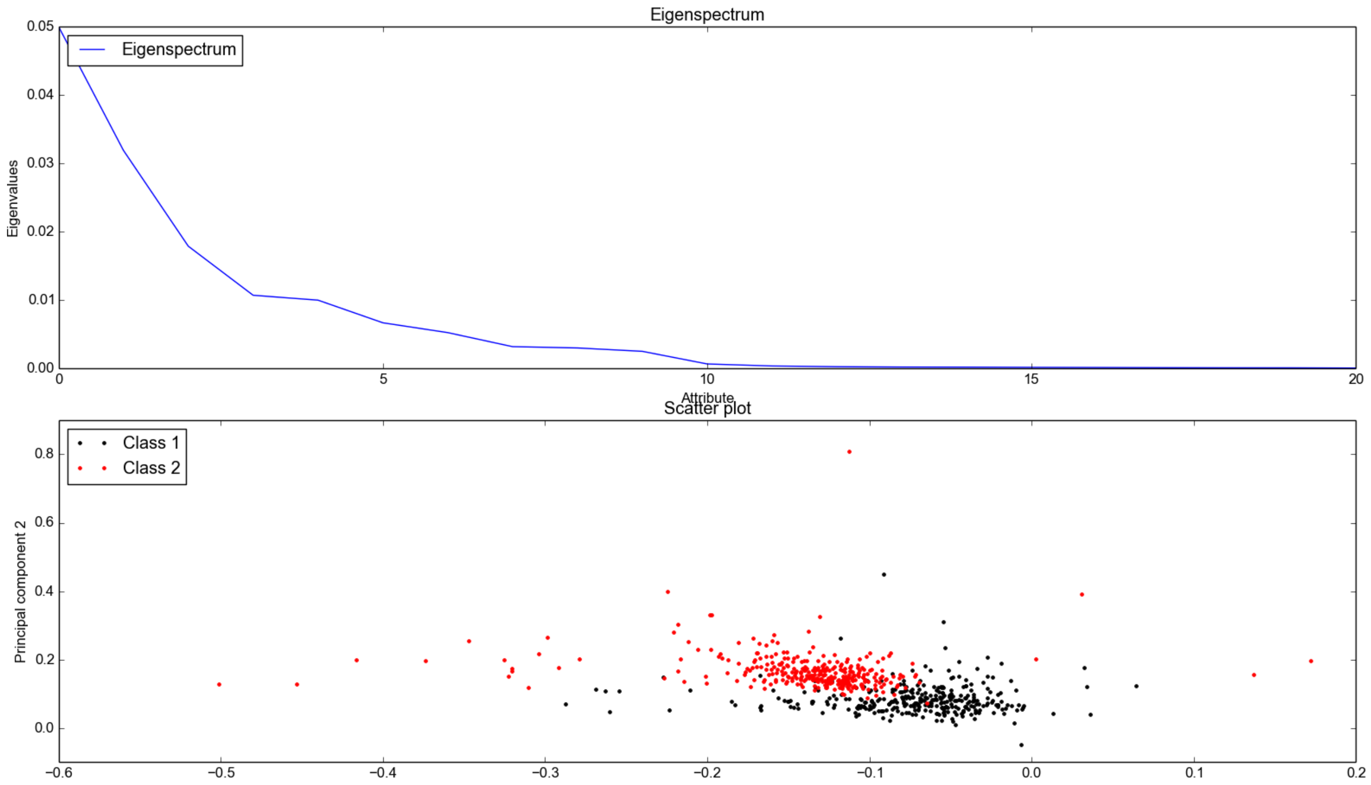

Figure 5: Eigenspectrum for all components (Top). Principal components coloured by class (Bottom).

2.3 Question 5 - Clustering

We notice that the cluster centers fit quite well with the two classes. In order to find the centers we used the same approach as with PCA.

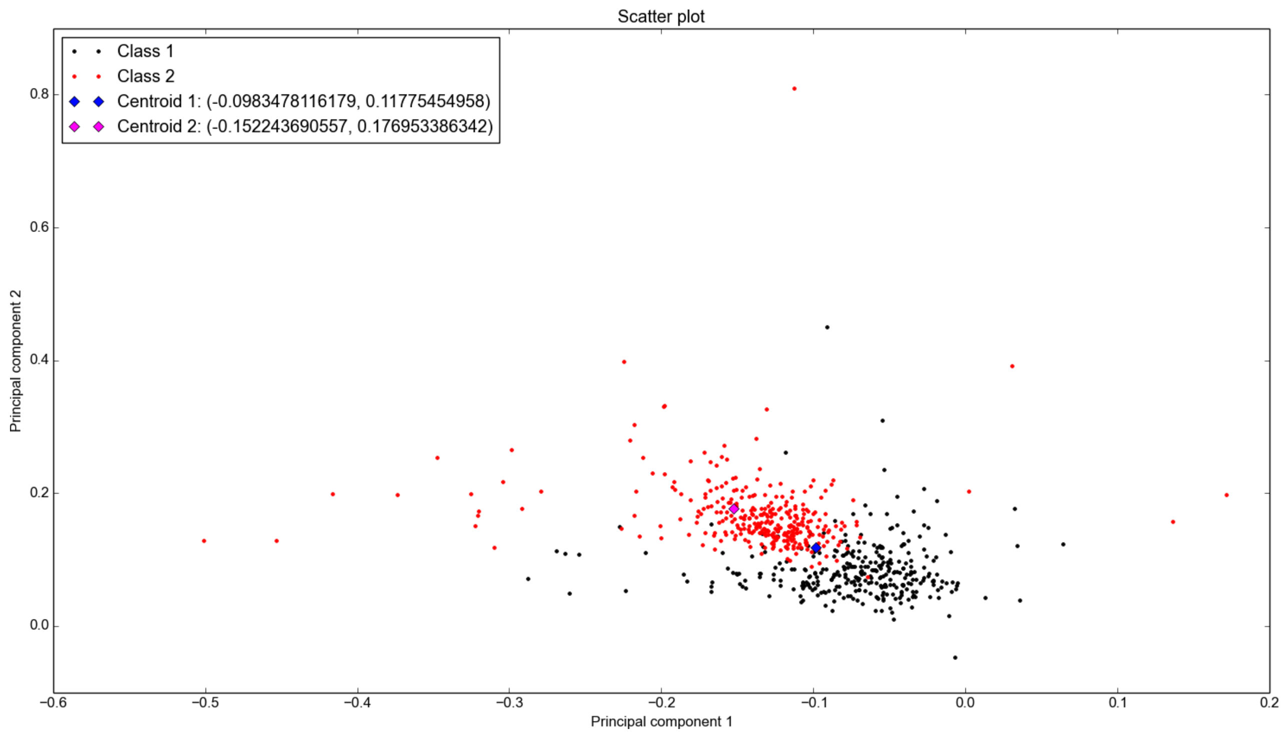

Figure 6: Principal components coloured by class along with the two cluster centers.

2.4 Question 6 - Multi-Class classification

We used K-Nearest-Neighbour and Linear Discriminant Analysis as our models, since we had already implemented support for higher dimensions during the course assignments. In order to determine the best k we used the same cross validation approach as in section 2.1.2.

The implementations used for both LDA and KNN was our own from the course exercises, which rely on the python numpy library for matrix operations, and on the python scipy library for calculating distances between points in higher dimensions for K-Nearest-Neighbour.

KNN classifier best k:1
KNN training accuracy: 1.0
KNN test accuracy: 0.9075

Training accuracy for LDA: 0.928125
Test accuracy for LDA: 0.9175

These results are slightly worse than our binary classification results, but considering the fact that we can classify 4 classes instead of 2, it is most likely also more useful.