Statistical Methods for Machine Learning

Exam Assignment: Stars and Fraud

Mariuta Nicolae (rqt629) April 7, 2015

Contents

Question 1 – linear regression

I have chosen to use Maximum Likelihood solution much like as I did in the second assignment of the course because I already have a good implementation of it and I understood the algorithm very well so it was easy to make the implementation. I have chosen to make the implantation in Matlab and I used mostly the functions that we have written for that assignment.

I solved in question by going through the following steps:

- At first, I read the data from both files and did not normalize because it has no effect on this linear model
- I created the design matrix corresponding to variable Φ (according to Eq. 3.16 from Bishop book) which is needed in the algorithm for building the model by inserting a column of 1 corresponding to the bias parameter and adding the 10 columns corresponding to the first 10 parameters in the train dataset.
- I calculated the model by applying the Eq. 3.15(from Bishop book): $w_{ML} = (\Phi^t\Phi)^{-1}\Phi^t\boldsymbol{t}$ where Φ is the design matrix and $\boldsymbol{t}$ is the matrix with column 11 corresponding to target values of redshift from the training data. This calculation is done in the function *my_wML* I obtained the following model where the first parameter is the biass:

$$w_{ML} = \begin{pmatrix} -0.8807 \\ -0.0060 \\ 0.1097 \\ 0.2784 \\ -0.0018 \\ 0.0094 \\ -0.0018 \\ 0.0094 \\ -0.0086 \\ 0.0041 \\ -0.1075 \\ -0.0098 \\ 0.0408 \end{pmatrix}$$

- By using the model obtained previously, I applied the test data to the linear model by using the function *my_lmfunction* (that I have implemented for the assignment) which takes as input the first 10 columns of test data and the model to make the matrix multiplication. The result is a column matrix containing the prediction for the redshift of each galaxy in test.
- By using the function *my_ms* which was also implemented for the assignment the mean-squared-error is calculated by computing the equation:

$$MS = \frac{1}{N}\sum_{n=1}^{N}(t_n - y(x_n, w))^2$$

The resulting mean-squared-error for train data is $MS = 0.0028$.
The resulting mean-squared-error for test data is $MS = 0.0032$.

In order to also have a visual estimation on how the model behaves, I have also created a plot as in assignment 2 to visualize the difference between the redshift that was calculated for the test set and the prediction obtained by using the linear model. I have only plotted the values for galaxies 100-200 because it was hard to visualize if I was plotting for all the 2500 galaxies.

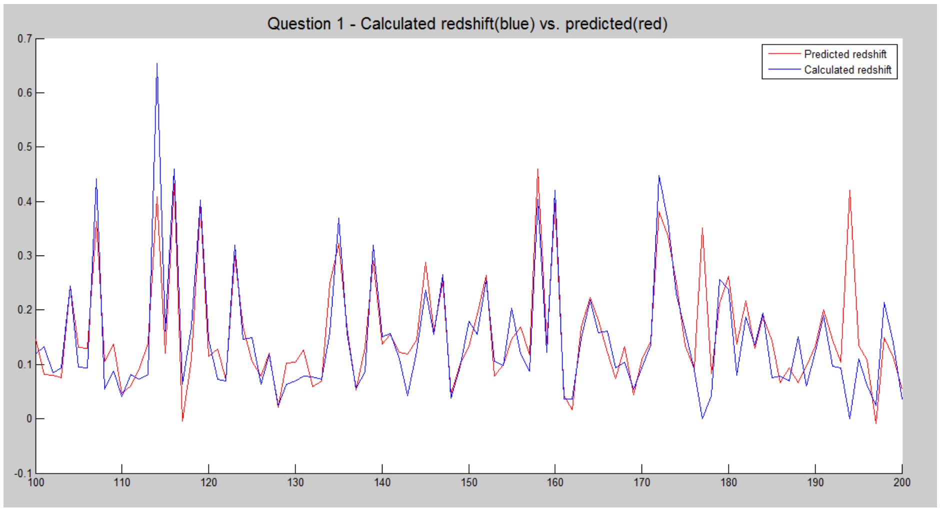

Figure 1. The plotting of the redshift from the test set and the redshift predicted by the linear model that I have obtained for the test data

Both the RMS and the plot show that the model obtained using the linear regression are quite good and the algorithm I used provided good generalization. The good generalization is achieved because the train dataset was quite large and the patterns used cover very well the behavior of the regression function.

Question 2 – nonlinear regression

For this question I have begun with the implementation of k-nearest neighbor because it is an easier non-linear algorithm and I wanted to see faster the results of a non-linear approach for this problem, but the results were not very good. After that I implemented the Random Forests which provided better accuracy than the linear model. I have decided to also make a short description of the knn approach and the results and will discuss in more detail the implementation of the Random Forests.

Method 1 : knn

For this approach I have done basic implementation of k-nearest neighbor by averaging the nearest neighbors in order to assign the redshift value to the test pattern. I started with reading the data from the files and normalized it. In order to find the best k I have chosen to apply the 5-fold cross validation, in order to obtain god generalization of the model, and I have obtained the best accuracy for k=15. By using this value I obtained mean-squared error 0.0375 for the train set and 0.0385 for the test dataset.

Unfortunately, the results of this approach were not better than those obtained by the linear regression. Perhaps with some improvements of the algorithm (like making a weighted average of the nearest neighbors by considering the distance and also consider the nearest neighbors in a radius and also find the best radius by using cross validation), the results could get better but since the mean-squared error is larger than the one I obtained for the maximum likelihood, I have decided to also try another algorithm.

Method 2: Random forests

As a second method for trying to obtain a better prediction for redshift method, I have chosen to use random forests. I have chosen this method mostly because it was not in the assignments, and I have read in many documentations that is one of the most efficient methods in Machine Learning and was excited to try something new, especially that the algorithm for random forest is not difficult.

I have also implemented this in Matlab and I have implemented on my own the algorithm for building the random forest and calculate the final results but I have chosen to use the Matlab implementation *fitrtree* for Regression Threes for several reasons: I do not have good knowledge about how to create a tree data structure in Matlab and certainly it would have taken a lot of time to make it work properly, and also the algorithms are optimized in Matlab for building the trees and calculate prediction.

I have implemented the algorithm by following the steps:

- I read data from files and decided not to make any normalization because, since one feature is never compared in magnitude to other features, the means and deviations do not matter.
- I have applied the 5-fold cross validation and I have chosen the numbers: 100,200 and 300 of trees. Perhaps I should have chosen more values but the computation time is very long and I wanted to get some results to check if the random forest may provide better predictions than the linear model
- At each step I created the random forest by implementing the algorithm shown in the course: the trees are created using the bootstrap sampling, and using the Matlab function *fitrtree* that creates objects of type *RegressionTree*. For each tree that was created I have provided the parameters of the patterns and target values, and also set off the *Prune* which is not used for random forests.
- After creating the trees predictions are calculated for the split from train data which is used as test. The method *predicts* returns prediction value for the specific *RegressionTree* and the input pattern from test. As I did for k-means I took the averaged sum of predictions from each tree and decided the final prediction. After this step the mean-squared error between predictions and redshift column from the patterns is calculated and averaged between the sections of the cross validation.
- At the end I used the number of trees that provided the best mean-squared error and used it to calculate predictions for train and test data and see final results

The cross validation shown that there I not much difference in result between the numbers 100, 200 and 300 of trees but for 300 trees the random forest provided slightly better mean squared error. It is proven the theory that random forests provide better accuracy for larger number of trees but at some point the increase in accuracy is very small. I obtained a mean-squared error of 0.00072 for train data and 0.0012 for test data.

As a conclusion, the random forest performed better than the linear model and perhaps with more tuning of the model I could have obtained better results, because there are several parameters which can be changed and using cross validation it can be decided which provides the best results: the minimum size of terminal nodes and maximum number of terminal nodes. Unfortunately, because of the very long computation time and also long time I needed to understand the Matlab libraries for *RegressionTree* and how to make them work in my Random Forest, I did not have enough time to also check which values for these parameters provide the best results at cross validation but. However, the large number of trees used for the final model provides very good generalization even with standard parameters for the construction of the regression tree and it has proven that Random Forest is a powerful algorithm which provides good results, even if it quite easy to understand but it comes with the drawback of long computation time.

Question 3 – binary classification

For solving this question I have decided to use a linear SVM and a non-linear SVM mostly to have some diversity in the algorithms I use for the questions in the exam and also because I understood very well how to implement them when working at the assignment. I implemented both in Matlab using the LIBSVM for the implementation of SVM. For both implementations, I shuffled the train data before applying cross-validations because the patterns are ordered according to the labels and I wanted to have almost same number of patterns with each label into each split so I can obtain better generalization.

Linear method: linear SVM

For the implementation of linear SVM I have used the following function from the library:

- *svmtrain* to build the model with the following parameters:
  - columns with parameters of train data and columns with classification for each pattern
  - kernel function chosen to be *'linear'* in order to have a linear SVM
  - "*boxtconstraint*" set to be the C so I can specify the regularization parameter
  - "*autoscale*" set to be "*false*" prevents LIBSVM to do any normalization of data because I do that before using this function
- *svmclassify* which takes as input the model that was created previously and the test block to calculate predictions

For the model selection I have started by normalizing the data, applied 5-fold cross validation to find the best value of the regularization parameter C between the values $= [0.01, 0.1, 1, 10, 100, 1000, 10000]$.

After finding the optimal value of the parameter, I have calculated the accuracy for both: train and data set.

The regularization parameter that offered the best accuracy during the cross-validation is C = 0.01 and used this value to build the model and calculated: accuracy for the train data (0.9938); accuracy for the test set(0.9875); sensitivity for the train set(0.9938); specificity for the train set(0.9906); sensitivity for the test set(1) and specificity for the test set(0.9705).

Non-linear method: SVM with non-linear kernel

For the implementation of non-linear SVM I have used same function from LIBSVM with the difference that at *svmtrain* the kernel function option I have chosen the kernel that I created: I have decided to use same Gaussian kernel as in the assignment 3: $k(x, z) = \exp\left(-\gamma||x - z||^2\right)$.

For the model selection process I have followed the same steps as building the linear model, with the difference that at cross validation, I applied the 5-fold cross validation for all the combinations between the regularization parameter and the kernel parameter with the following values:
$C = [0.01, 0.1, 1, 10, 100, 1000, 10000]$ and $\gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]$.

The regularization parameter that offered the best accuracy during the cross-validation is C = 10 and the best Gaussian parameter is $\gamma =$ 0.001. Using these values I built the model and calculated: accuracy for the train data (0. 9953); accuracy for the test set(0.9875); sensitivity for the train set(0. 9969); specificity for the train set(0. 9938); sensitivity for the test set(1) and specificity for the test set(0.9750).

After implementing both models, I can conclude that the accuracies are very good for both train and test data with little higher values on the non-linear SVM. Regarding the sensitivity it is very important that it had value 1 on the test set for both: linear and non-linear SVM, because the application will not detect the real user of the account as a hacker and it will not happen that someone cannot access his account because of a classification error. On top of this, a the specificity is very good which shows that it is also a very high chance to detect a hacker which tries to enter the system. So, the models I obtained can be very efficiently be used in cybercrime with good detection of hackers but also it does not interrupt the activity of normal users of the accounts and this makes them useful in the application scenario.

Question 4 – principal component analysis

For solving this question I have also used Matlab, I normalized the data, to ensure that each parameter has same weight into making the Principal Component Analysis, and calculated: the 21x21 covariance matrix by using the Matlab built-in *cov()* function and then the eigenvectors and eigenvalues by using the Matlab built-in *eig()* function by having as input the resulting covariance matrix. Because the eigenvalues that result from the Matlab built-in function are sorted ascending, I have sorted the principal components descending and plotted the eigenspectrum.

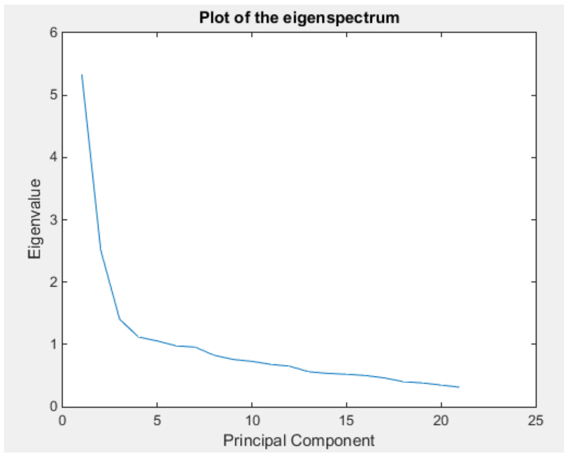

Figure 2 Eigenspectrum for the train data from file keystrokesTrainTwoClass.csv

In the next stage, I used the Matlab built-in function *eigs()* to get returned the eigenvalues and eigenvectors first two principal components and I have multiplied the transposed eigenvectors for those with the data matrix to transform the 21-dimensional data into 2-dimensional space.

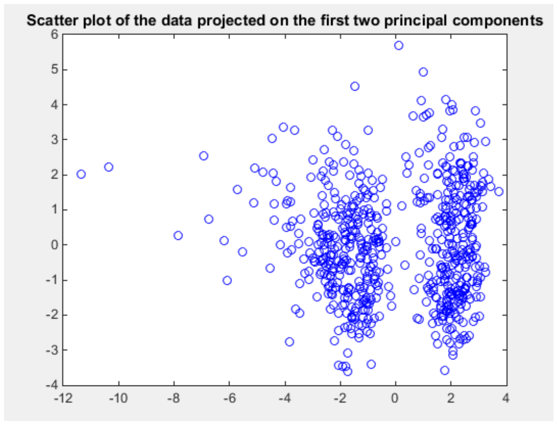

Figure 3. Scatter plot of the data projected on the first two principal components

The scatter plot shows that there are two concentrations of patterns which the most likely correspond to the two types of patterns from data that was used for classification at previous question. Also, from a non-supervised machine learning perspective, it is clear that there are two distinctive types of patterns. Now it is easy to understand the high accuracies for the binary classification algorithms that I have used: the patterns from each type are very well grouped together, so it is easy, for example, to find a linear or non-linear separator for the train points.

Question 5 – clustering

For the implementation of this unsupervised machine learning technique, I have also used Matlab and implemented the code on my own because the algorithm is quite easy and I have already done it few times in the past for some projects.

I have implemented the algorithm much as it is described in the book but with some small changes to make it easier to implement:

- I have shuffled the train set so I pick the initial centroids randomly
- I have implemented a stop condition but I also limited the number of runs of the centroids and clusters recalculation to 100 and break the operation when the centroid is no longer moving. I have found that the algorithm stops after 7-8 iterations but it could take longer for a lot larger datasets and it could be possible that after a lot of runs the centroids and clusters are no longer changing very much but the program does not stop running.
- Other than these I implemented the algorithm as described at the course: calculate clusters at each step by assigning each point to the cluster with closest centroid, calculate mean and update centroids position.

The centroids that I obtained are: Centroid 1 (1.4859, -0.6817, -1.4105, -1.4426, 0.0971, -0.6631, -0.1486, -0.2890, -1.1756, -0.5306, 0.2657, -0.9728, -1.5941, 4.8012, -2.2838, -0.4202, 0.1597, -0.4160, 1.1314, 0.0653, -0.8793) and Centroid 2 (0.2174,-0.1379,-0.4537, 2.1711, -1.5622, 0.1503, 1.3849 , 0.1585, 0.4546, 0.3622 , 1.2053, 0.7023, 0.0865, 0.7966 , -0.5790, -0.2398, -0.1525, 0.7014, 0.4249, 1.0444, -0.1460).

After running the program I have also ran the operations from Question 4 to display scatter data by first principal components and transformed the points in both clusters to 2-dimensional space. After this, I displayed centroids with 2 different colors and I also displayed the points from each cluster with 2 different colors to understand better the meaning of the clusters.

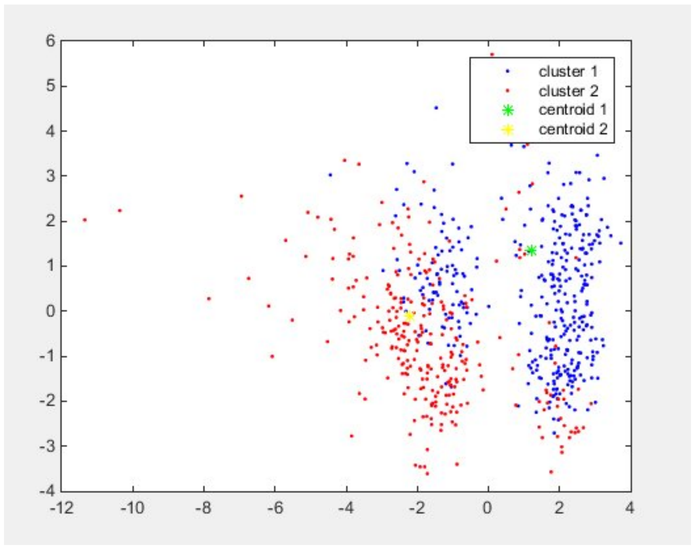

Figure 4 Plot the results for calculating the 2 clusters of the training data by using k-means

The clusters that I obtained and plotted show that the points from each cluster are grouped around its corresponding centroid and each cluster has mostly points from each group of points that, as noticed at previous question, they should correspond to the two types of patterns. The clusters interfere a little bit and especially the centroid to the right may not seem to be right in the middle of the centroid because the k-means clustering was performed on untransformed data, in its original 21-dimensional space and that is why some points appear not to be assigned to closest centroid. And the centroid on the left is somehow to the left of the group because there are some points far to the left which alter the mean a little bit. I have also tried to perform k-means clustering after applying the transformation to 2-dimensional space and the clusters appeared to be more meaningful.

Question 6 – multi-class classification

For this question I have decided to use LDA as linear classifier and k-nearest neighbor as non-linear classifier, just as it is suggested. Since the scatter plot and clustering from previous exercises shown good split of the two types of patterns, I guess it will be almost the same at this dataset and then these two classifiers should perform quite well. On top of that, I already had most of the code written since the assignments and wanted to solve the task easier. I have also done these implementations in Matlab as I did for the rest of the exam.

LDA – Linear classifier

I have implemented this classifier just as I did for the assignment without using many Matlab built-in functions.

I have the function *my_lda* which takes as input the train and test data and returns error rate on both. Unfortunately, the function is not well parameterized and only built to work for this exercise, so I calculated into separate variables the means and covariance for each the five types of patterns. The model is built and the prediction values are calculated for each class of patterns, put together and the accuracy for train and test set is returned.

The train error obtained is 0.0719 and the test error is 0.0825 which are quite low. As expected, it looks like even when the patterns are divided into these 5 classes, they are well grouped and provide good prediction accuracy using linear classifier. It could also happen that to be harder to identify 5 types of users instead of 2 but still in this application, with this dataset the performance is still good.

K-nearest neighbor – Non-linear classifier

Since the 5 types of patterns are well grouped, I used the knn being quite confident that I will still obtain good performance. For this I also used most of my code from the assignment so I did not use many Matlab built-in functions.

I have started by reading data and normalizing it because unlike the linear classifier, a higher mean or standard deviation for some parameters can lead to bad results of the algorithm. Since I used 5-fold cross validation technique, I shuffled the train set in order to find the k that provides the best accuracy. I had to do that because the patterns are sorted according to the label number and I wanted to have almost equal number of types of patterns at each split during cross-validation. For the cross validation I have decided to use 5-fold cross validation and obtained the best performance for k = 3.

After performing the cross validation, I used the value for k that provided the best results and obtain error rate of 0.0175 for train set and 0.0400 for test set which is better performance than for using linear classifier, perhaps because at the border between the groups of patterns the k-nearest neighbor does better classification of patterns and the separation created by the linear classifier does not split very well the patterns with different labels.

Question 7 – overfitting

I have decided to discuss about the following types of overfitting: Traditional Overfitting, Parameter Tweak Overfitting and Old Datasets, mostly because I have seen these to be more relevant and easier to describe in the context of redshift estimation.

1. Traditional overfitting

In the case of using too few examples for training a complex predictor, it may happen that it will not offer a good generalization and the error for the test sets will be higher.

For example, by using our dataset with all the 2500 patterns to build the linear regression model I obtained the mean-squared error 0.0032 for the test set but if I create the model by using only the first 500 examples I get higher MS: 0.0038. This happens because the fewer examples for training the model do not cover very well the behavior of regression and the model is not built same accurately.

The same thing could also happen for using other machine learning algorithms to solve this problem: if creating a complex neural network and train it to offer a good accuracy on train set will result into overfitting if we use a very low number of examples and the behavior of model for test set will result into very high mean-squared error because the weights into the nodes of neural network that will result from training will not offer a good generalization. And in the situation of using k-nearest neighbor approach it will happen that too few examples will take into consideration totally wrong values of redshift because it may happen that a test pattern will not have neighbors with somehow similar parameters.

If it is not possible to have access to a larger dataset, then perhaps it will be a good idea to do some changes as the input patterns so to obtain more training examples: since it is a measurement of objects in space, then rotations, scaling of the initial object will obtain new sets parameters and then, by calculating the redshift for the newly obtained parameters should provide more examples for obtaining a more accurate predictor.

As a conclusion, training a smaller dataset for a complex predictor it is important to have a larger dataset and try to get as many examples as possible in order to obtain a good model.

2. Parameter Tweak Overfitting

This is obviously a bad overfitting method because, as it was shown while working at the questions regarding the redshift dataset, the train mean-squared error is usually better than the MS on test set, which is normal because the test set may contain more patterns similar to those few which provided higher mean-squared error while choosing the parameters during cross validation and perhaps also the train set does not have enough patterns to describe very well the regression of the redshift. But it does not mean the model provides better generalization if we decide the parameters of training algorithm based on the accuracy on the test set. A lot better strategy could be to acquire more datasets before considering the test set. With better variety of training patterns, it will happen that the model to be more accurate and the test set will also provide lower mean-squared error that in currently does and the overfitting is avoided.

3. Old datasets

As it is described by John Langford, the old datasets have very well optimized model for obtaining good accuracy but it does not mean that they are very effective into calculating redshift for new data obtained through spectroscopy. This happens because old dataset may not cover very well the behavior of redshift according to the input parameters. It is very possible that during the old measurements, there were considered only some types of objects for a specific purpose and the dataset does contain a large variety of patterns and some parts of the regression function do not have examples for training accurately.

Another reason would be that in many domains the techniques for obtaining the features evolve, by using more accurate devices and algorithms and perhaps the photometric data about the galaxies is more accurate and calibrated differently. This will mean that the models obtained by using old dataset are not even being correct and that is why it could be a better strategy to put more weight into the new measurement of Doppler Effect in order to estimate the redshift.

On top of this, if creating algorithms based on old dataset, the feedback provided by different websites and articles on those datasets will lead to temptation to do little tricks like parameter tweak in order to obtain better accuracy on the same old test sets but the generalization will no longer be as good for newer patterns.

As a conclusion, after discussing these three types of overfitting, I can say that it is always good to take a large dataset into consideration, design the algorithm without considering the feedback from the performance on the test set, and also put more weight into new datasets because those could provide better examples to build a more accurate predictor.