UNIVERSITY OF COPENHAGEN

STATML EXAM

In a Galaxy Far, Far Away

Author:
Alexander WAHL-RASMUSSEN

KU Identity:
LGV740

March 31, 2014

1 Predicting the Specific Star Formation Rate

1.1 Linear Regression - Maximum Likelihood

1.2 Non-Linear Regression - Maximum A Posteriori

2 Stars vs. Galaxies

2.1 Binary Classification with Support Vector Machines

2.2 Principal Components Analysis

2.3 Clustering

2.4 Kernel Mean Classifier

Claim 1 This is really fun!

Proof. Since

$$a + b + c = d \tag{2}$$

it follows that equation (2) is a fun equation. □

3 Variable Stars

3.1 Multi-Class Classification

For this part I chose K-NearestNeighbor (KNN) as my non-linear classifier and Support Vector Machine with a linear kernel (SVM) as my linear classifier. There are several reasons for why I chose these classifier and why I consider a SVM with a linear kernel as a linear classifier, as shall now become evident. However, let us start by examining how the algorithms classify. Imagine a case, where we only have 3 samples and 2 categories, in a n-dimensional space, and the first sample belongs to the first class while the second sample belongs to the other category. We then want to classify the third sample based on its features compared to the classified samples' features. This means we first train the specific algorithm on the two samples and then run the algorithm on the last sample.

The *(K-)NN* will then try to find the nearest neighbour by calculating the distance between the test sample and all other training data points. The nearest neighbour will then be the training data point that has the shortest distance to the test data point. The test data point is then classified as having the same class as its nearest neighbour, because their features are most alike. The amount of nearest neighbors to be considered can be increased and this notated with a K - hence the name KNN, where the classification decision is based on majority voting between the neighbors. The distance metric itself can also vary, e.g. it can either be Euclidean, Manhattan, Hamming etc., where the three also are known as direct, absolute and binary distances.

A *Linear SVM* will instead try to create a linear spatial decision boundary (in this case a straight line). The data is then divided . This resembles the approach of the

Perceptron, but instead of trying to minimize the amount of errors, it creates the decision boundary at the maximum distance from the nearest data points, which is also known as the support vectors. So instead of finding *a* decision boundary it tries to the *the best* decision boundary. Intuitively this also means that the bigger the distance (or margin) is to the nearest support vectors, the lower amount of prediction errors should be. It should be noted that having a hard margin may not be ideal, since it can result in overfitting on the train set, which leads to bad generalizations. Unlike Perceptron, the linear SVM can also classify non-linear distributions by utilizing the "kernel trick" and projecting the distribution into an unknown, but well-defined, high-dimensional space. In this space the distribution will then become linearly seperable and the decision boundary can be located. The kernel itself can also be changed to a non-linear function such as we have seen in chapter 2.1. I have simply run the linear SVM, which can arguably be described as a conceptual extension of the Perceptron.

The reason why I included the descriptions of the algorithms is quite simple: I believe that one of the biggest strengths of both classifiers is how they classify. It makes sense that

3.1.1 Description of software used

For both classifiers I used implementaions from the Scikit-Learn Python library[3].

The *KNeighborsClassifier* is a KNN implementation that also incorporates an underlying data-structuring algorithm such as a KDTree, Ball Tree or 'brute force'[2]. The brute force search is the 'vanilla' KNN where every data point is iterated over, while both KDTree and Ball Tree partitions the data into trees, where the KNN algorithm then iterates over the tree nodes instead. The documentation does not give a clear explanation of how it specifically works, but such an approach seems intuitive. Any re-structuring of the data will of course affect the performance and the decision-making of the classifier. However, as I have a fairly limited knowledge of space-partitioning, I chose the beautiful setting of 'auto', also known as the black box approach, where the function itself chooses an appropriate algorithm parameter as to achieve the most efficient solution. The only parameter I then have to supply is the amount of neighbors K. Additionally, the function .score(X,y) returns the accuracy instead of the 1-0 Loss and this I manually convert with $Loss = 1 - Accuracy$.

The *SVC*[4] is a SVM implementation that is based off the LIBSVM [1] library. The mathematical formulation of the learning problem it tries to satisfy is explained in [4, 1.2.7.1. SVC], with the decision function being:

$$\operatorname{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + p\right) \tag{3}$$

where $y_i \alpha_i$ is the dual coefficients for the support vectors and p an individual constant term.

As the SVM is a binary classifier it employs a one-by-one strategy for multiclass classification. This means that the function creates one classifier per class-pair, mean-

ing the class with most votes among the subsets is chosen [1]. This is in contrast to the one-vs-all implementation, where each class is matched against the rest of the classes. I also chose the output to be deterministic and with the class weight set to 1, meaning each class has the same weight. Furthermore, the function .score(X,y) returns the accuracy which I then convert to a 0-1 loss. As I chose a linear kernel of the form $K(x, x')$ the only relevant hyperparameter I have to supply is C.

3.1.2 Results

Normalizing + crossvalidation for optimal hyperparameters.

3.2 Overfitting

References

[1] CHANG, C.-C., AND LIN, C.-J. LIBSVM. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`, March 2014.

[2] SCIKIT-LEARN. KNeighborsClassifier. `http://scikit-learn.org/stable/modules/neighbors.html`, March 2014.

[3] SCIKIT-LEARN. Machine Learning in Python. `http://scikit-learn.org`, March 2014.

[4] SCIKIT-LEARN. Support Vector Machines. `http://scikit-learn.org/stable/modules/svm.html`, March 2014.

[1] This means that for a case of n classes there will be a classifier for each pair: 0 vs 1, 0 vs 2, .. 0 vs n, .. n-1 vs n