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. The intuition here being that 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. However, having a v Unlike Perceptron, the SVC can also classify non-linear distributions by creating kernel functions, as we have seen in chapter 2.1. We have simply run the linear SVC, which can arguably be described as a conceptual extension of the Perceptron. With such a comparison we also expect them to perform somewhat similar.

The reason why I included the descriptions of the algorithms is quite simple: I believe that one of the biggest strengths of

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. Any re-structuring of the data of course affects 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 *SVC*[4] is a SVM implementation that is based off the LIBSVM [1] library. 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, meaning the class with most votes among the subsets is chosen. This is in contrast to the one-vs-all implementation, where each class is matched against the rest of the classes. 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. `scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.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.