Applied Machine Learning - EDAN95
Lab Session 5 - Fall 2018 and 2019

Pyspark Clustering (Fall 2018)
Naive Bayesian Classifiers (NBC) - GNB
Nearest Centroid Classifier (NCC) - MNIST

Hicham Mohamad, hi8826mo-s
hsmo@kth.se

December 7, 2018

1 Lab 5 - Part 1: Word Counting and Embedding Clustering

In this lab 5, we Learn how to read, transform and process text data with Pyspark. Then, we preprocess and create a suitable dataset for clustering, use KMeans from sklearn and cluster 10 000 words to 200 clusters, and write a function which displays words nearby.

1.1 Outline of the lab

- You will first solve a few exercises on Spark to learn how to write basic commands.
- You will then apply Spark to extract the 10,000 most frequent words in the English Wikipedia.
- As this corpus is very large, you will use 1
- You will finally cluster these words into 100 groups using their GloVe100 representation.
- As clustering program, you will use KMeans from sklearn.

2 Lab 5 - Part 2: Bayesian Classifiers (Fall 2019)

In this lab session you will

- explore some programming concepts in Python, SciKitLearn, and Numpy to implement several **statistical/Bayesian classifiers** and compare them against each other and with one provided in SciKitLearn,
- get acquainted with another version of the MNIST dataset, and
- explore the effects of specific phenomena in the data on the classification results, also depending on the choice of **discrete** vs **continuous** implementation of the models.

2.1 Background and Tools

A rather intuitive and greedy (supervised) method to classification is the **K-Nearest Neighbour** classification. This can be based on a **brute force** method, in which the entire (training) data set is searched for the (k) example(s) that is (are) closest to the sample one wants to classify. Similar to this is the **Nearest Centroid Classifier** (NCC) that minimises the distance between the sample to the centroids (means) of the clusters formed in feature space by the examples belonging to the different classes. In the

supervised case, it is quite obvious how to find the **centroid** for a class - it is simply the "point" derived by computing the mean for each attribute value for the examples belonging to the given class.

Slightly more sophisticated is then to go from applying the distance to the class-centroids in a direct comparison to interpreting them in terms of a **probability distribution** (density function). This can be done with **Naive Bayesian Classifiers** (NBCs), as discussed in the lecture. You will for this lab session implement several variants of the mentioned classifiers, and you will do that on both the extremely simplified version of the MNIST dataset provided in SciKitLearn (`datasets.digits`), and on a slightly more elaborate but still preprocessed version of the MNIST data provided together with a bit of a code skeleton to read in and visualise the data HERE (in `Handout_NaiveBayes.zip`).

[If you are interested and willing to do a bit more file I/O handling, another, even larger and more realistic, version of the MNIST data set can be found on Yann LeCun's site (`http://yann.lecun.com/exdb/mnist/`).]

2.2 Your task

You will implement an NCC, a discrete (count-based) NBC, and a Gaussian NBC and run and compare them on different versions of the data set(s) explained in point 2 as follows:

2.2.1 Classifiers implementation

Classifiers you should implement and make sure that they work with the data sets indicated

1. Make use of the provided **Gaussian NB Classifier** (`sklearn.naive_bayes GaussianNB`) for all data sets as a comparison. It is already implemented in the handout for the `MNIST_Light` set (see below).
2. Implement your own **Nearest Centroid Classifier (NCC)**: The NCC `fit` method should simply compute the mean values over the attribute values of the examples for each class. `Prediction` is then done by finding the `argmin` over the distances from the class centroids for each sample. This classifier should be run on all three variants of data sets, see below.
3. Implement a **Naive Bayesian Classifier** (NBC) based on discrete (statistical) values (i.e., counts of examples falling into the different classes and attribute value groups) both for the **priors** and for the **conditional probabilities**. Run this on the two `SciKitLearn digits` data sets. It should also work with the (non-normalised) `MNIST_Light` set, but it will probably take a (very long) while and not give anything interesting really...
4. Implement your own **Gaussian Naive Bayesian Classifier (GNB)** (assume priors for the classes based on counts and Gaussian distributions for the conditional probabilities) and make sure it works for all three data sets. You will most likely encounter problems due to the **edge pixels** having value 0.0 in practically ALL images in ALL data sets. A workaround is to add an **epsilon** to the variance. Why is that still working?

2.2.2 Data sets

Data sets and which classifiers to apply

1. **SciKitLearn digits**: just load it and use it as is. Use 70% of the data for training and 30% for testing. Run all of the four classifiers and compare and discuss the results (ideally with your lab partner ;-)
2. **SciKitLearn digits summarised**: reduce the data set to only contain three values for the attributes, e.g., 0 for 'dark', 1 for 'grey' and 2 for 'light', with dark, grey and light corresponding to the values suggested in lab 2 (decision trees). Split again into 70% training and 30% test data.

   Run all four classifiers on this set and compare the results. Why are they so different from those for the original data in particular for the NBC? Why do they decrease in accuracy for the GNB?
3. **MNIST_Light**: load the `MINST_Light` data set (see above), inspect the contents (data and specifically the target) carefully and split the set into 70% training and 30% test data if you do not use

the code in the handout (based on MNIST.py) as is. If you use the provided code to retrieve the data set, the three-dimensional (20x20-pixels per image) arrays will have been reshaped to be of the same two dimensions as given in the digits set (one flattened array per image) and the values for the attributes (pixels) will have been normalised from [0 ... 255] to [0.0 ... 1.0]. If you do not normalise them, you can observe a problem with the GNB. What is this problem and why is it solved by **normalisation** of the data?

To pass this lab session task you should get all classifiers to run and deliver reasonable results, i.e., the lowest **accuracy** will probably be observed for the discrete NBC on the original `digits` data (0.58 in my case) while the accuracy for the NCC and GNB should end up above 0.8 (or not too far from that) and thus reach a level comparable to or even higher than that of the SciKitLearn GNB with default (no) parameters for the `MNIST_Light` set (0.66 in one of my own tests). (Bonus for the interested: Inspect the *documentation of the SciKitLearn implementation* and try to find an explanation for this **difference**.)

3 Appendix

References

[1] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[2] Yann Lecun, The MNIST Database of handwritten digits: `http://yann.lecun.com/exdb/mnist/`

[3] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`