Exercise 1 - Language Identification

From Shallow to Deep :-)

Learning goals

This exercise consists of two parts: the first part aims at deepening your understanding of linear models. The second part will already target a simple kind of multi-layered network; the multilayer perceptron (MLP). Don't worry if you don't know anything about MLPs when you read this. We will cover all you need to know to do the second part of this exercise next week in class and in the tutorial session. By completing this exercise you should …

- … understand linear models, implement them and use them for multiclass classification tasks
- … be able to implement all kind of machine learning models in scikit-learn.
- … construct your first MLP and compare it to the performance of linear models.
- … get acquainted to Google's Colab.
- … have your first experience with TensorFlow.

Please keep in mind that you can always consult and use the exercise forum if you get stuck (note that we have a separate forum for the exercises).

Deliverables

Please hand in your code separately for part one and two. Hand in one zipped folder with all your files containing code and output you have written. Name your files in the following fashion:

- ex01_name_surname_part1_softmax.py
- ex01_name_surname_part1_softmax.txt
- ex01_name_surname_part1_sk-models.py
- ex01_name_surname_part1_sk-models.txt
- ex01_name_surname_part2_mlp.py
- ex01_name_surname_part2_mlp.txt

The .py files contain your well documented code, while the .txt files are text files containing the desired output. Please put all the six files in a folder, zip it, and upload it to OLAT. Don't include any data. Make sure, however, that your code is easy to run. We assume that the data is in the same folder as the scripts.

Deadline

Deadline for Exercise 1 is **15.10.2018, 12:00 (MESZ)**.

Data

For both parts, you will work with the same data. Our goal is to classify the language of Twitter messages. This is an extension of Goldberg's problem as described in chapter 2. On the one hand, we will work with many more languages than just six. On the other hand, the text segments we need to classify are much shorter. Download the data from the material folder in the exercise section of OLAT. The folder contains a file called "hydrated.json" and a file called "uniformly_sampled.tsv". The former contains the Tweets along with the Tweet-IDs, while the latter contains the labels along with the Tweet-IDs. Since the json-file is a recent download from Twitter based on the Tweet-IDs in the tsv-file, it might be that certain Tweets might have been deleted in the meantime. Therefore, your first step is to prepare the data and match the language codes to the Tweets. Inspect the data and make sure classes are more or less uniformly distributed. Balance out the data (you might get rid of a language or two). Then, create a training set and a test set. Use 80% of the data for training and 20% for testing. This part of the code should also go into ex01_name_surname_part1_softmax.py. Of course it is forbidden to peek into the test data ;-).

Part 1 - Language identification with softmax classifier

This is an extension of Goldberg's language identification problem. In tutorial session 2, we've shown you some code which implements an SVC. Now it's your turn to turn this into a softmax classifier!

1. Adapt the code presented in tutorial session 2 so it works with the Twitter data and calculate the accuracy on the test set. We call this "SVC".

2. Implement a language classifier with softmax output transformation and categorical cross-entropy loss. Calculate the accuracy on the test set. We call this “SMC”.
3. Write all this code and documentation in ex01_name_surname_part1_softmax.py
4. Compare the the output from the SVC and SMC. Is there any difference in the performance. If yes, why might that be? What additional information do we get from the SMC? Summarise the outputs from SVC and SVM, as well as the answers to these questions in ex01_name_surname_part1_softmax.txt

Additional help: Since we’re doing hard classification, you can use formula 2.22 on page 28 in Goldberg [2017]. As a basis, you can use the code presented in tutorial session 2, which you find on GitHub. You’re free to take whatever you want from there and you might change the code in any way you want. Be aware that we might want to use different preprocessing steps, since we are dealing with totally different data. For the first part of part 1, we still use bigrams as features. For a good overview on linear classifiers, see here.

So far, so good. You’ve written your first classifier, almost from scratch. In reality, however, this is rarely the case. There have been many bright minds who have compiled whole libraries for us, so all we need to do is write four lines of codes to train a model. Scikit-learn does exactly that for us! In the following, you will train several models in sklearn to solve the task above.

5. Create a suitable pipeline in sklearn to preprocess the data. Think about extending the feature space. What other features could you use to determine the language. You’re supposed not only to use bigrams for this task.
6. Train the following classifiers: Linear Support Vector Classifier, Multinomial Naïve Bayes, and a Decision Tree Classifier.
7. In order to find the optimal model, use sklearn’s GridSearchCV. Put all your code in ex01_name_surname_part1_sk-models.py.
8. Compare the outputs of the best models for the three classifiers. Which classifier scores highest on the test set? Do you have an idea, why this might be? What is the advantage of grid search cross validation? Summarise your answers in ex01_name_surname_part1_sk-models.txt.

Part 2 - Your first MLP

Let’s see if you can beat your best linear model you’ve trained with sklearn with an MLP.

1. Since you’ve prepared all your data in sklearn, train a MLP classifier. You can also use grid search cross validation in this case. The code of this part of the exercise goes into ex01_name_surname_part2_mlp.py.
2. It is also possible to model a MLP classifier in TensorFlow. Go ahead and try this in Google’s Colab. You should be able to reuse your sklearn feature processing pipeline. However, you won’t be able to use grid search in order to determine which model scores best. Try at least five different configurations (meaning number of layers and depth of layers). Note down all the configurations you’ve made and report the accuracy of the best model. Share the Colab notebook with Phillip (click on “Share” in the upper right corner and share with phillip.stroebel@gmail.com).
3. Comment on the performance of the MLP classifier trained in sklearn and the one trained using Colab. Are there any differences? If yes, where could they come from? What are the differences between the MLP classifier and the linear models you’ve trained. Put your comments and accuracy scores in ex01_name_surname_part2_mlp.txt.

If you need help on that, Raschka’s [2015] chapter 2 provides an introduction to MLPs. The Google Machine Learning Crash Course also offers good material. Alternatively, you might also use keras to model the MLP in Colab.