Dobre Bogdan - Mihai

**Geo-location of German Tweets**

In this document I will describe my approach for the geo-location of german tweets task.

The program has 2 modes: dev and test. The current mode is set at the beginning of each file in a variable called "mode". The dev mode is used to compute predictions for the validation data and calculate the mean absolute error and the mean squared error while the test mode is used to write the predictions for the test data in an output file called output.txt in the required format. This was the file used in the contest submissions.

The chosen feature set was words. The first step was cleaning the text of punctuation and other special characters. I also turned any uppercase letters in the text to lowercase. After that, I lemmatized and tokenized the words using the "de\_core\_news\_sm" module from the "spacy" library.

The function which does the tokenization and the lemmatization is called "text\_to\_tokens" and is used on the training, validation and test datasets.

The approach chosen for the machine learning part was using word embeddings together with SVM. Word embeddings are useful for getting meaningful numerical array representations for words. Using these numerical arrays, I trained the SVM model and used it to get predictions.

In order to get the word embeddings I used the word2vec algorithm. It is an algorithm which uses a neural network to learn word associations and to produce a numerical array that is meaningful for each word, encapsulating its meaning. I used the "Word2Vec" class from the "gensim" library which implements the algorithm. The hyperparameters for this model were:

* min\_count = 10; the minimum number of apparitions for a word to be considered
* window = 10; maximum distance between 2 correlated words in a sentence
* size = 300; the size of the numerical vectors
* alpha = 0.01; the learning rate of the model
* min\_alpha = 0.01; the value towards which the learning rate decreases; in this case the learning rate remains constant
* workers = 4; number of threads working to train the model

These hyperparameters were chosen after multiple experiments. I chose the combination that yielded the best results. The model is then trained for 100 epochs. This was the number of epochs after which I noticed that the accuracy was not improving significantly.

After the model is created, the train data is converted to word embeddings. I created a function called "text\_to\_coords" which return a word embedding for each tweet. To do this, it first cleans, tokenizes and lemmatizes the tweet using the "text\_to\_tokens" function. After that, it uses the model to get the word embeddings for each of the words that are known (known words appear in its vocabulary, so only the words which have a frequency bigger than 20 in the training data would end up being known). In order to get a word embedding for the entire tweet the function computes the average of the word embeddings generated for the known words and returns it. This results in a list of word embeddings for the training data. The program creates a similar list of word embeddings for the testing or validation data, depending on whether the program must do validation and compute errors or just output the predictions. The labels (meaning the coordinates) of the validation data and training data are converted to strings and are treated as strings for the entirety of the program. The predicted labels are also generated as strings and are converted to a pair of doubles in order to compute the mean absolute error and the mean squared error. The training word embeddings and the validation or testing word embeddings are then used for the SVM classifier.

The SVM classifier fits the training word embeddings and generates predictions for the test or validation word embeddings. The only hyperparameter used for the SVM is the "rbf" kernel as it was the only parameter which improved accuracy in my tests. I used the SVC class in the sklearn library to create this model. The resulting predictions are then written in an output file, in the required format. All the io operations are handled using the "csv" library.

In case of the validation data, the model, when trained on just the training data provided, returns the following mean absolute error and mean squared error: MAE = 0.8873541760530818, MSE = 1.5251011728783703.

The accuracy is expected to significantly increase in the testing case however, as the validation data is also used for training. However, the errors for this case can't be calculated without the correct labels for the testing data.