Paper for the Data Mining Techniques course

Spring semester, Ac. Year 2018-19

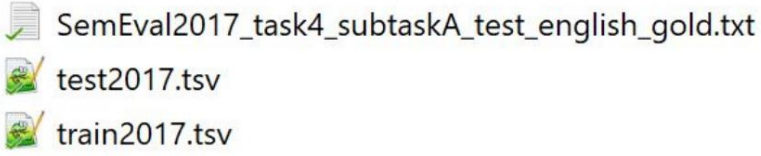
Sentiment Analysis: the process of computationally identifying and categorizing the opinions expressed in a piece of text, in order to determine whether the author's attitude towards a specific topic, product, etc. is positive, negative or neutral.

The data you will be working on comes from the annual SemEval (International Workshop on Semantic Evaluation) competition, specifically the 2017 competition on sentiment analysis in tweets.

Job data:

In eclass you will find a folder for your work. This folder has the following structure:

The **twitter_data** folder contains 3 files,



The file **train2017.tsv** contains the data you will use to train your models. The training data contains 28061 tweets labeled positive, negative or neutral.

test2017.tsv **contains** the data you will use to test your model and make a prediction. The test data contains 12284 tweets labeled UNKNOWN , as for this set of tweets your model needs to decide whether they express positive, negative, or neutral sentiment.

Finally, the *.gold.txt file contains the correct labels for the test2017 file. Gold labels may not be used in training your models under any circumstances. You can only use them to verify your results and specifically you can estimate how well your classifier is doing (for example you can use the F1 score https://scikit-learn.org/stable/modules/generated/sklearn. metrics.f1_score.html).

In the **lexica** folder we give you various emotional dictionaries that you can use for your work. As you will see, these dictionaries include for each word a continuous value in the interval [-1,1], which represents what in emotional analysis we call the "valence" of a word ***(means the intrinsic attractiveness/"good"-ness (positive valence) or averseness/"bad"-ness (negative valence) of an event, object, or situation).*** The dictionaries have been generated either by the evaluators' method or by machine learning methods. In the dictionaries you will find many of the words contained in the tweets. You can use any of these dictionaries you want, or even find others and combine them.

Wanted:

The work will be done with the Python programming language. In the relevant section in eclass you will find all the material related to python.

Data preprocessing and cleaning a. Cleaning the data, (we remove the symbols, such as hashtags, emoticons, emojis, links and stopwords from the training set) b. tokenization

c. stemming

For b,c you can use NLTK Python toolkit or any other library you like.

Analysis of the data.

You are invited to write some python commands that will help you "study" the data you are given and draw some conclusions. Some of the questions you can answer are the following:

What are the most common words in the entire data set? What are the most common words in the dataset for negative, positive, and neutral tweets, respectively?

You can present the above results with a Word cloud. Can you think of any other observations that emerge from the data? If possible present related graphs.

Vectorization - feature extraction

Follow the instructions we presented in the tutorial and prepare the attributes for each tweet using:

1. Bag-of-words
2. Tf-idf 3. word embeddings

Use Python's pickle library to store attributes in *.pkl files. This way the attributes don't have to be calculated from scratch every time you run your program, but you can just load them into memory using the corresponding ***load method.***

Adding features to the word vector

Using embeddings we get for each tweet a vector with 200-300 values (features). We can "add" additional features by expanding the table above. For this purpose you can use dictionaries that correspond to (continuous) sentiment values, in words. For each tweet look up its corresponding words in dictionaries and calculate an average "sentiment" value across

the tweet, per dictionary. After this step you will have for example a vector for each tweet of size [300 + N], where N is the number of dictionaries you used.
Bonus: Can you think of other features that could be added to the word vector by increasing the number of features? (such examples are: length of the tweet, maximum and minimum value of the valence of the words in each tweet, to divide the tweet into two parts and calculate an average value of valence in the first and the 2nd half, etc.)

We test classifiers (SVM, KNN, Round Robin Classification)

a. SVM
b. KNN

Test your classifiers with the features BOW, TFID, word embeddings and with the vector to which you have added the features from the dictionaries.

c. Round Robin Classification - **bonus :** Your original problem is a 3-class classification problem. But you can try the pairwise classification technique, as described in the post below. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.5074&rep=rep1&type=pdf

---

Round robin classification is a technique that divides the problem into individual binary problems. It then uses a classifier for each pair of classes. For this method you will use Nearest neighbor classifier.

The algorithm is as follows:

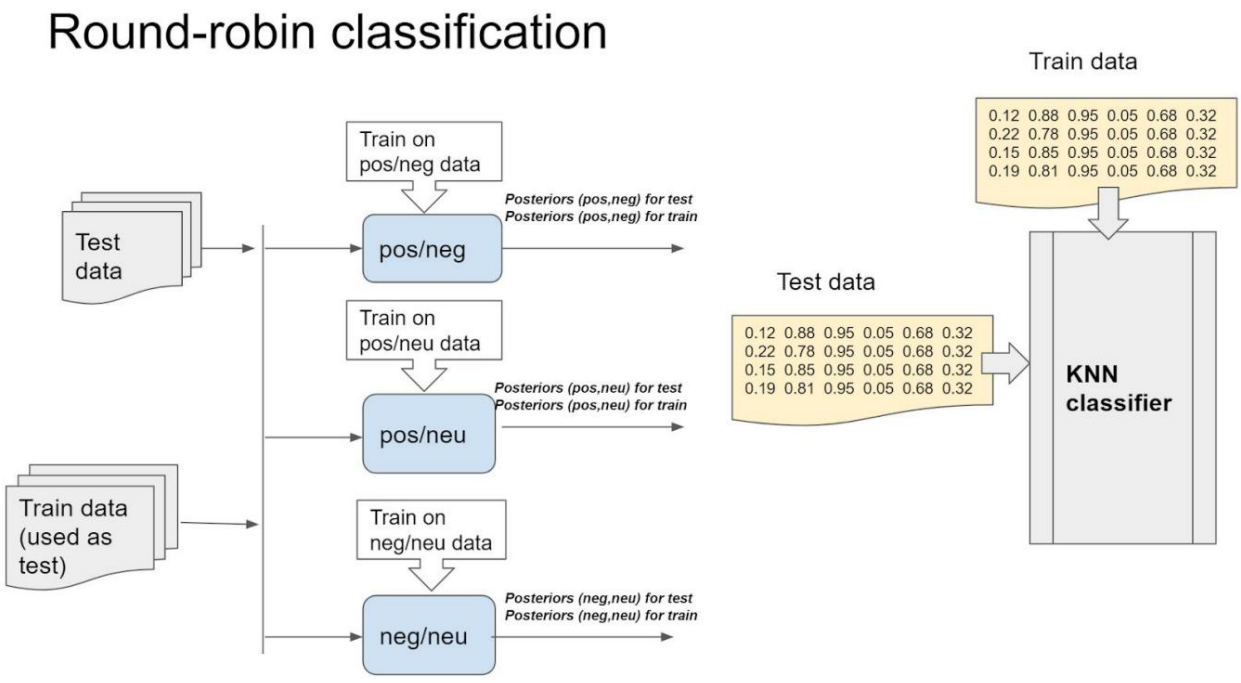
I. I convert the class **c** problem into **c(c-1) / 2** two problems
of classes, one for each set of classes {i, j}, i = 1 ... c-1, j = i + 1 ... c.
II. The binary classifier is trained with examples of classes i and j.
III. Examples of the remaining classes that do not belong to i, j are ignored.
IV. The results of individual classifiers are combined into a separate one
sorter.

In our problem, we have 3 categories, ***positive, negative, neutral.*** Therefore we will have 3*(3-1)/2 = 6 different problems, which are for example, positive-negative, negative-neutral, positive-neutral etc. Therefore we have 6 different classifiers. The output of each classifier can be either the class label, i.e. return positive, negative, neutral, or the posterior, i.e. return

assign a probability value to each class (see sklearn's predict_proba method) .

Let's take a simple example. The first classifier (positive-negative) will be trained (fit method) only on tweets that are positive or negative. You can use in this query any of the feature vectors you have saved in the previous queries. Then it will be tested (predict method) on all tweets of the train set. The output of the classifier that produces posterior probability has 2 values, for each tweet of its input. We will apply the predict method to all test tweets in this classifier. With the same procedure, each iterative classifier is tested not only on the test but also on the train data to get the corresponding feature vectors.

Schematically, the algorithm is shown below (in the last step the KNN classifier accepts a table with 6 features for each tweet, which come from the result (posterior) of the individual classifiers):



Presentation of results:

Draw a table showing your results for the different classifiers you used based on the different vectors

characteristics. Comment on your results (when is improvement seen, what do you think are the best features, etc.) .

Deliverable:

The work can be done individually or in groups **of 2 people.**

The job's **scr** folder is the folder you'll write your code in, and it's also the one you'll commit (ie you won't redeliver the dictionaries and training/test data). You will upload to eclass a folder of the format scr_sdixxxx. (where sdi is the ID of one of the people in the group).

Your code MUST include an **Ipython notebook** with which someone can run your work step-by-step. You can also have *.py files with your functions but the task must be run from a notebook. In the notebook, wherever you deem necessary, you can insert **visualizations** in the way we will explain in the tutorials (indicatively we mention the word cloud, word embedding visualization and of course you can also present your results in a nice way). **The notebook is also the complete reference** for your work (you won't hand in anything in doc, pdf), design it carefully, remember to write a step-by-step description of what your code does in each cell.