First year project 3, Spring 2022
Natural Language Processing
Christian Hardmeier

In this project, you will learn how to work with natural language data. You will learn

- what makes natural language different from other types of data,
- how to prepare text data for automatic processing,
- how to annotate data for supervised classification,
- how to train and run a classifier for a basic NLP task, and
- how to use n-gram language models for data augmentation.

We use the TweetEval corpus, a collection of 7 datasets for different classification tasks based on social media posts:

Binary Classification Tasks

- Irony Detection – 2 labels: *irony, not irony*
- Hate Speech Detection – 2 labels: *hateful, not hateful*
- Offensive Language Identification – 2 labels: *offensive, not offensive*

Multiclass Classification Tasks

- Emotion Recognition – 4 labels: *anger, joy, sadness, optimism*
- Emoji Prediction – 20 labels: 😍, 😂, . . . , 🌲, 📷, 😜
- Sentiment Analysis – 3 labels: *positive, neutral, negative*
- Stance Detection – 5 different target topics (*abortion, atheism, climate change, feminism, Hillary Clinton* and 3 labels: *favour, neutral, against*)

The corpus can be found here: `https://github.com/cardiffnlp/tweeteval`
The datasets come with predefined splits into *training*, *validation* and *test* sets that you can use in your work.

Both binary and multiclass classification problems occur frequently in natural language processing. For your project, you should select ***one** of the binary and **one** of the multiclass problems* to work on.

1 Preprocessing

Using regular expressions, implement a tokeniser to split the input texts into meaningful tokens. Your tokeniser should be a script that takes as input the data as distributed in the dataset and outputs the tokenised text, one output line per input line, with spaces between tokens.

- Use the *training* set of one of the tasks you've chosen to work on. Set aside a part of the *training* set for evaluating the tokeniser so you don't have to touch the *validation* data for that.
- Start by taking a subset of the *training* data, look through it and discuss in your group what a good tokenisation of this subset *should* look like. Then design your initial Python implementation to match this ideal tokenisation, and run it over the portion of the *training* set that you didn't hold out for tokeniser evaluation. Keep an eye on infrequently occurring tokens in the output that look like tokenisation errors.
- Once you're done, compare your tokeniser's output with the baseline tokenisation you get from the social media tokeniser in the NLTK library (`nltk.tokenize.TweetTokenizer`), using the data you've set aside for this purpose.
- Consider using the `difflib` package in Python, or the `diff` utility in the Unix shell, to compare the output of the two tokenisers efficiently.

2 Characterising Your Data

Characterise the training sets of the two tasks you've chosen in terms of elementary corpus statistics:

- Corpus size, vocabulary size, type/token ratio.
- What are the most frequent tokens?
- What types of tokens occur only once, or 2 or 3 times?
- Are there any noticeable differences between your two datasets?
- Are the corpus statistics consistent with Zipf's law? (no formal test needed, but a plot would be helpful)

3 Manual Annotation and Inter-Annotator Agreement

Choose one of your two datasets. For this subtask, the *emoji prediction* dataset doesn't make sense, so if that is one of your choices, pick the other one for this part of the assignment.

In the README file in the TweetEval corpus repository, there are links to research papers from the SemEval workshop, describing how each of the datasets was created and annotated. Locate the one that belongs to the dataset you've picked for this subtask and find the passages that describe in detail how the labels for the dataset were created. Read these passages carefully.

Next, select a random sample of 100 tweets from the *training* set. Working *independently from each other* and *without consulting the labels published in the TweetEval corpus*, each member of your group should now go manually through this sample and label them according to the same scheme.

Report on the inter-annotator agreement, including the agreement with the published labels, and discuss what phenomena in the data caused the biggest problems for inter-annotator agreement.

4 Automatic Prediction

Finally, use *scikit-learn* to train a classifier for the automatic prediction of the labels in the two datasets you have chosen. During the lessons, we have not had time to discuss machine learning techniques and classification methods in detail, so in this exercise you will be using library implementations as “black box” methods.

Run all classification experiments on *both* of the tasks you've chosen (one binary and one multi-class task). Evaluate your different classifiers on the *validation* set and report relevant evaluation metrics (accuracy, precision/recall/F-score).

As a baseline, start with the `sklearn.linear_model.SGDClassifier` in a *logistic regression* configuration (`loss='log'`) using *bag of words* features. Then run additional experiments trying to improve your initial scores by any means you can think of. Try out at least 4 different methods. Usually you will need to run several experiments for each methods to test different parameter values. Here are some settings in *scikit-learn* that you could experiment with:

- Additional preprocessing options: n-gram features, lowercasing, stop word lists (see options to `sklearn.feature_extraction.text.CountVectorizer`).
- Count transformations (`sklearn.feature_extraction.text.TfIdfTransformer`).
- The classification loss (`loss` parameter to `SGDClassifier`).

- The regularisation strength (`alpha` parameter to `SGDClassifier` – try varying it in exponentially spaced steps).
- Different classifiers (e.g., `sklearn.ensemble.RandomForestClassifier` or `sklearn.naive_bayes.MultinomialNB`.
- Anything else discussed during the lessons, or implemented in *scikit-learn*.

For the systems that achieves the highest accuracy on the *validation* set, run the evaluation on the *test* set and report your results.

5 Data Augmentation

Run at least one experiment with an augmented dataset that includes additional data selected based on training data perplexity under a language model trained the task you've selected. Possible sources of extra data include

- the training sets of the other tasks,
- the Broad Twitter Dataset (`https://github.com/GateNLP/broad_twitter_corpus`),
- or any other dataset you can think of.

To obtain labels for the augmented data, you could try fully automatic annotation, or (full or partial) human validation of labels.

6 Hand-in

You are expected to hand in:

- `report.pdf` – A project report.
- `code.zip` – One zip file containing the commented Python code underlying your report. This should run without errors and be completely self-contained. Please do include the datasets of (only) the two tasks you've chosen to work with, including the version preprocessed by the tokeniser you developed, as well as the data you've annotated yourselves (Section 3).

The project report should be between 4 and 6 pages including figures (with 11pt font size and about 1.5cm margins) and should consist of the following sections:

- **Introduction** – Providing context and motivation for the problem. What are the main ideas you pursued, and why does your research provide value?

- **Data and Preprocessing** – Describe the datasets and explain the tasks you selected for your project. Briefly describe your preprocessing procedure and the main difficulties you encountered. Present data statistics and compare between your two datasets if it makes sense to do so.
- **Annotation** – Present the results of your annotation quality check, including inter-annotator agreement figures, and discuss the most important sources of disagreement, if there was any.
- **Classification** – Briefly describe your classification experiments and report and discuss their results.
- **Conclusion and Future Work** – Summarise the main lessons you've learnt in this project and discuss how your work could be improved and extended.

Your submission should include at least one plot, illustrating your investigation of Zipf's law, and at least one table, summarising all your experimental results in a manner that allows easy comparison of different experiments. There is no upper limit on the number of figures/tables, but every figure or table you include should be there for a good reason and discussed in the main text.

When you write your report, you may assume that your target audience consists of data scientists who have a similar level of training as yourself, but not precisely the same background. They might, for instance, have followed a similar course of study at a different university. You should be prepared to defend any presentational choices you make based on this scenario.