Instructions for MLMI13 Practical: Sentiment Detection of Reviews

Kate Knill, Adian Liusie, Kevin Heffernan and Milica Gašić
`kate.knill@eng.cam.ac.uk al826@cam.ac.uk`

Michaelmas 2021

This practical[1] concerns sentiment classification of movie reviews. It aims to give you a grounding in a range of NLP approaches. You will use a sentiment lexicon and also a machine learning approach based on Bag-of-Word features, a stemmer and a POS-tagger to decide if a random unseen movie review is positive or negative.

You will be provided with partial solution code for this assignment. The GitHub repository containing this code can be found here `https://github.com/adianliusie/MLMI13`. Included in the repository you will find 1000 positive and 1000 negative movie reviews in `data/reviews` (whole set in `POS/NEG.zip,tar.gz` and individual files in `POS,NEG/*.txt`). This data set will be used for most of the assignments. To prepare yourself for this practical, you should have a look at a few of these reviews to understand the difficulties of the task, and think about how one might go about classifying them. You will need to install `NLTK` from `https://www.nltk.org` and the Python library `gensim` `https://pypi.org/project/gensim/`.

Your task will be to understand the provided code base and fill in the missing code blocks marked `TODO`.

Advice: Please read through the entire instruction sheet and familiarise yourself with all requirements before you start coding or otherwise solving the tasks. Writing clean, modular code can make the difference between solving the assignment in a matter of hours, and taking days to run all experiments.

Assessment: Write a report of up to 2000 words, excluding references, on what you have done as a stand alone document. Due on **Tuesday 7th December at 12:00 noon**. (Note: mention of the word limit always means that you should state the actual number of words you used.)

1 Part One: Baseline and Essentials - 60% of the marks

How could one automatically classify movie reviews according to their sentiment? Your task in Part One is to establish two baselines that have been commonly used - by implementting and evaluating several NLP methods on this task.

1.1 Symbolic approach - sentiment lexicon

If we had access to a sentiment lexicon, then there are ways to solve the problem without using Machine Learning. One might simply look up every open-class word in the lexicon, and compute a binary score S_{binary} by counting how many words match either a positive, or a negative word entry in the sentiment lexicon $SLex$.

$$S_{binary}(w_1 w_2 \dots w_n) = \sum_{i=1}^{n} sgn(SLex[w_i]) \tag{1}$$

[1] These practical notes are based on previous L90 practical texts written by Simone Teufel, Adrian Scoica and Yiannos Stathopoulos, and by Andreas Vlachos from the Dept. of Computer Science and Technology

If the sentiment lexicon also has information about the magnitude of sentiment (eg. "*excellent*" would have a higher magnitude than "*good*"), we could take a more fine-grained approach by adding up all sentiment scores, and deciding the polarity of the movie review using the sign of the weighted score $S_{weighted}$.

$$S_{weighted}(w_1 w_2 \dots w_n) = \sum_{i=1}^{n} SLex[w_i] \tag{2}$$

Your first task is to implement these approaches using the sentiment lexicon in `data/sent_lexicon`, which was taken from the following work:

> *Theresa Wilson, Janyce Wiebe, and Paul Homann (2005). Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. Proc. of HLT-EMNLP-2005.*

Their lexicon also records two possible magnitudes of sentiment (*weak* and *strong*), so you can implement both the binary and the weighted solutions.

1.2 Answering questions in statistically significant ways

Having implemented both lexicon methods above, consider answering the following question:

> (Q0.1) Does using the magnitude improve results?

Oftentimes, answering questions like this about the performance of different signals and/or algorithms by simply looking at the output numbers is not enough. When dealing with natural language or human ratings, it's safe to assume that there are infinitely many possible instances that could be used for training and testing, of which the ones we actually train and test on are a tiny sample. Thus, it is possible that observed differences in the reported performance are really just noise. There exist statistical methods which can be used to check for consistency (*statistical significance*) in the results, and one of the simplest such tests is the sign test. We can now add rigour to our answer by appending the following question in conjunction with the original one:

> (Q0.2) Is the performance difference between the two methods statistically significant?

Apply the sign test to answer questions (Q0). The sign test is described in Siegel and Castellan (1986)[2], page 80. For the purposes of this paired test, please treat ties by adding half a point to either side; since we are working with integers when computing significance using the binomial distribution, please round up the final two counts to the nearest integer. You can quickly verify the correctness of your sign test code using a free online tool.

From now on, report all differences between systems[3] using the sign test.

1.3 Machine Learning using Bags of Words representation

Your second task is to program a Machine Learning approach that operates on a simple Bag-of-Words (BoW) representation of the text data, as described in Pang et al. (2002).

> *Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan (2002).* Thumbs up? Sentiment Classification using Machine Learning Techniques. *Proceedings of EMNLP.*

Bo Pang et al. were the "inventors" of the movie review sentiment classification task, and the above paper was one of the first papers on the topic. In this approach, the only features we will consider are the words in the text themselves (or short sequences of these words), without any other sources of information. The BoW model is a popular way of representing text information as vectors (or points in space), making it easy to apply classical Machine Learning algorithms on NLP tasks.

[2] Siegel and Castellan, Nonparametric Statistics for the Behavioural Sciences, McGraw-Hill

[3] You can think about a change that you apply to one system as a new system.

1.3.1 Writing your own classifier

Write your own code to implement the Naive Bayes (NB) classifier.[4] As a reminder, the Naive Bayes classifier works according to the following equation:

$$\hat{c} = \arg\max_{c \in C} P(c|\bar{f}) = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(f_i|c)$$

where $C = \{\text{POS}, \text{NEG}\}$ is the set of possible classes, $\hat{c} \in C$ is the most probable class, and $\bar{f}$ is the feature vector. Remember that we use the log of these probabilities when making a prediction:

$$\hat{c} = \arg\max_{c \in C} \{logP(c) + \sum_{i=1}^{n} logP(f_i|c)\}$$

You can find more details about Naive Bayes here:

`http://web.stanford.edu/~jurafsky/slp3/4.pdf`

and pseudocode here:

`http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html`

The data in `data/reviews` contains the text of the reviews, where each document `*.txt` is one review. Your algorithm should read in the text, store the words and their frequencies in an appropriate data structure that allows for easy computation of the probabilities used in the Naive Bayes algorithm, and then make predictions for new instrances.

> (Q1.0) Train your classifier on files cv000–cv899 from both the `/POS` and the `/NEG` directories, and test it on the remaining files cv900–cv999. Report results using simple classification accuracy as your evaluation metric[5].

Smoothing

> The presence of words in the test dataset that haven't been seen during training can cause probabilities in the Naive Bayes classifier to be 0, thus making that particular test instance undecidable. The standard way to mitigate this effect (as well as to give more clout to rare words) is to use smoothing, in which the probability fraction
>
> $$\frac{f(w)}{N_{words}} = \frac{count(w_i, c)}{\sum\limits_{w \in V} count(w, c)}$$
>
> for a word w_i becomes
>
> $$\frac{f(w) + smoothing(w)}{N_{words} + \sum\limits_{w \in V} smoothing(w)} = \frac{count(w_i, c) + smoothing(w_i)}{\sum\limits_{w \in V} count(w, c) + \sum\limits_{w \in V} smoothing(w)}$$
>
> In your case, you are dividing the frequency for word w with the sum of frequencies for all words in the vocabulary, with added smoothing.
>
> (Q2.0) Implement Laplace feature smoothing ($smoothing(\cdot) = \kappa$, constant for all words) in your Naive Bayes classifier's code, and report the impact on performance.
> (Q2.1) Is the difference between Q2 and Q1 statistically significant?

[4] This section and the next aim to put you a position to replicate Pang et al., Naive Bayes results. However, the numerical results will differ from theirs, as they used different data.

[5] **Voluntary thought experiment**: Is accuracy an equally appropriate evaluation metric in a situation where 90% of your data instances are of positive movie reviews? If you are curious about this, you can simulate this scenario by keeping the positive reviews data unchanged, but only using negative reviews cv000–cv089 for training, and cv900–cv909 for testing. Calculate the classification accuracy, and explain what changed. (This is a voluntary exercise; if you decide to do it, please don't report it)

1.3.2 Cross-validation

A serious danger in using Machine Learning on small datasets, with many iterations of slightly different versions of the algorithms, is that we end up with Type III errors, also called the "testing hypotheses suggested by the data" errors. This type of error occurs when we make repeated improvements to our classifiers by playing with features and their processing, but we don't get a fresh, never-before seen test dataset every time. Thus, we risk developing a classifier that's better and better on our data, but worse and worse at generalizing to new, never-before seen data.

A simple method to guard against Type III errors is to use cross-validation. In N-fold cross-validation, we divide the data into N distinct chunks/folds. Then, we repeat the experiment N times, each time holding out one of the chunks for testing, training our classifier on the remaining N - 1 data chunks, and reporting performance on the held-out chunk. We can use different strategies for dividing the data:

- Consecutive splitting:

  cv000–cv099 = Split 1
  cv100–cv199 = Split 2
  . . .

- Round-robin splitting (mod 10):

  cv000, cv010, cv020,. . . = Split 1
  cv001, cv011, cv021,. . . = Split 2
  . . .

- Random sampling/splitting: Not used here (but you may choose to split this way in a non-educational situation)

> (Q3.0) Write the code to implement 10-fold cross-validation for your Naive Bayes classifier from and compute the 10 accuracies. Report the final performance, which is the average of the performances per fold.
> (Q3.1) If all splits perform equally well, this is a good sign. Write code to calculate and report the variance across the cross-validation sets.

Please report all future results using 10-fold Round-Robin cross-validation, unless told to use the held-out test set.

1.3.3 Features, overfitting and the curse of dimensionality

In the Bag-of-Words model, ideally we would like each distinct word in the text to be mapped to its own dimension in the output vector representation. However, real world text is messy, and we need to decide on what we consider to be a word. For example, is "`word`" different from "`Word`", from "`word,`", or from "`words`"? Too strict a definition, and the number of features explodes, while our algorithm fails to learn anything generalisable. Too lax, and we risk destroying our learning signal. In the following section, you will learn about confronting the feature sparsity and the overfitting problems as they occur in NLP classification tasks.

A touch of linguistics

> (Q4.0) Taking a step further, you can use stemming to hash different inflections of a word to the same feature in the BoW vector space. How does the performance of your classifier change when you use stemming on your training and test datasets[6]?
> (Q4.1) Is the difference from the results obtained at (Q3) statistically significant?
> (Q4.2) What happens to the number of features (i.e. the size of the vocabulary) when using stemming as opposed to (Q3)? Give actual numbers. You can use the held-out test set to determine these.

[6] Please use the Porter stemming algorithm; code is available at `http://tartarus.org/martin/PorterStemmer` or in NLTK.

Putting [some] word order back in

> (Q5.0) A simple way of retaining some of the word order information when using BoW representations is to use bigrams or trigrams as features. Retrain your classifier from (Q3) using bigrams or trigrams as features, and report accuracy and statistical significance in comparison to the experiment at (Q3).
> (Q5.1) How many features does the BoW model have to take into account now? How does this number compare (e.g. linear, square, cubed, exponential) to the number of features at (Q3)? Use the held-out test set once again for this.

1.3.4 Feature independence, and comparing Naive Bayes with SVM

Though simple to understand, implement, and debug, one major problem with the Naive Bayes classifier is that its performance deteriorates (becomes skewed) when it is being used with features which are not independent (i.e., are correlated). Another popular classifier that doesn't scale as well to big data, and is not as simple to debug as Naive Bayes, but that doesn't assume feature independence is the Support Vector Machine (SVM) classifier.[7]

> (Q6.0) Classify the BoW features from Q3 with a SVM classifier and compare the results to the Naive Bayes classifier from Q3.
> There are a number of available SVM classifiers or you can write your own with numpy. For example, the SVM classifier provided by scikit-learn or SVM Light.
> If using SVM Light[8] you will need to write the code to print out your BoW features from (Q3) in the SVM Light format and then run the classifier.

More linguistics

Now add in part-of-speech features. You will find the movie review dataset has already been tokenised and POS-tagged for you. These files end in `*.tag`. Try to replicate what Pang et al. were doing:

> (Q7.0) Replace your features with word+POS features, and report performance with the SVM. Does this help? Why?
> (Q7.1) Discard all closed-class words from your data (keep only nouns, verbs, adjectives and adverbs), and report performance. Does this help? Why

2 Part Two: extension - 40% of the marks

Now your task is to improve over the baseline systems using doc2vec (or Paragraph Vectors. These were proposed by Le and Mikolov (2014), who extended the learning of embeddings from words (word2vec) to sequences of words:

> Le, Q. and Mikolov, T. (2014). *Distributed representations of sentences and documents.* Proceedings of ICML.

Train various doc2vec models using the IMDB movie review database[9] This is a database of 100,000 movie reviews and can be found here:

> http://ai.stanford.edu/∼amaas/data/sentiment/

To train your models use Gensim[10]. Ideas for training different models include choosing the training algorithm, the way the context word vectors are combined, and the dimensionality of the resulting feature vectors.

[7] You can find more detaails about SVMs in Chapter 7 here: `http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf`

[8] `http://svmlight.joachims.org/`

[9] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

[10] `https://radimrehurek.com/gensim/models/doc2vec.html`

(Q8.0) Infer/generate document vectors/embeddings for each review in your train and test set. Report performance using the document embeddings with an SVM and compare them to your Naive Bayes classifier (Q3). Do you achieve a significant result?
(Q8.1) Perform an analysis of the doc2vec approach to sentiment classification. For example, see Lau and Baldwin (2016) and Li et al (2015). Are similar documents and/or words close to each other in doc2vec space? Are document embeddings close in space to their most critical content words? [11]

Lau, J. H. and Baldwin, T. (2016). *An empirical evaluation of doc2vec with practical insights into document embedding generation.* In Proceedings of the 1st Workshop on Representation Learning for NLP.

Dai, A. M., Olah, C., and Le, Q. V. (2015). *Document embedding with paragraph vectors.* arXiv preprint arXiv:1507.07998.

Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2015). *Visualizing and understanding neural models in NLP.* In Proceedings of NAACL.

For your interest You may have heard about BERT (Bidirectional Encoder Representations from Transformers). There are a number of Google Colab pages which allow you to train and run a BERT-based movie review sentiment classification task. For example:
using a Google movie review database
https://curiousily.com/posts/sentiment-analysis-with-bert-and-hugging-face-using-pytorch-and-python/
and using the IMDB movie review database
https://stackabuse.com/text-classification-with-bert-tokenizer-and-tf-2-0-in-python/.

[11] Useful for visualisation: t-SNE https://lvdmaaten.github.io/tsne/ and embedding projector https://projector.tensorflow.org.