Instructions for L90 Practical: Sentiment Detection of Reviews[*]

Kevin Heffernan
kh562@cam.ac.uk

Helen Yannakoudakis
hy260@cam.ac.uk

This practical concerns sentiment classification of movie reviews. Your first task is to use a sentiment lexicon and a machine learning approach based on bag-of-word features, a stemmer and a POS tagger. For the first task, please do not use any other packages than those described below. Your second task is to improve over the two baseline systems using document embeddings and perform an error analysis on the strengths and weaknesses of the approach. You must use the MPhil machines for these tasks. We provide your own personal VM on these machines.

You will find 1000 positive and 1000 negative movie reviews in `/usr/groups/mphil/L90/data/{POS,NEG}/*.txt`. To prepare yourself for this practical, you should have a look at a few of these texts to understand the difficulties of the task (how might one go about classifying the texts?); you will write code that decides whether a random unseen movie review is positive or negative, and two reports in the form of a scientific article that describe the results you achieved in the two tasks.

Please also make sure you have read the following paper:

> 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. The first version of your sentiment classifier will do something similar to Bo Pang's system. If you have questions about it, we should resolve them in our first demonstrated practical.

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.

Note: Please include a pointer to your working code on the Mphil machines (your account).

1 A quick note on installing packages in the MPhil machines

You can install packages by downloading the .tar file in your *home folder* and then installing the packages from there (while setting your path variables as needed). An alternative would be to do the following:

1. Go to `https://pip.pypa.io/en/stable/installing/` and download `get-pip.py`
2. Run `python get-pip.py --user`
3. Then to install a package (e.g., `scipy`) run `python -m pip install --user scipy`

[*]Adapted from L90 practical notes by Simone Teufel, Adrian Scoica, and Yiannos Stathopoulos.

2 Part One: Baseline and Essentials

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

2.1 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 $SLex$.

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

If the sentiment lexicon also has information about the magnitude of sentiment (e.g., *"excellent"* would have 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 $S_{weighted}$.

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

Your first task is to implement these approaches using the sentiment lexicon in `/usr/groups/mphil/L90/resources/sent_lexicon`, which was taken from the following work:

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

Their lexicon also records two possible magnitudes of sentiment (*weak* and *strong*), so you can implement both the binary and the weighted solutions (please use a switch in your program). For the weighted solution, you can choose the weights intuitively *once* before running the experiment.

2.2 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 rigorosity 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)[1], page 80 (scans of the relevant pages are available in the L90 directory `/usr/groups/mphil/L90/resources/`). As presented in the slides, the sign test is based on the binomial distribution. Count all cases when System 1 is better than System 2, when System 2

[1]Siegel and Castellan, Nonparametric Statistics for the behavioural sciences, McGraw-Hill.

is better than System 1, and when they are the same. Call these numbers $Plus$, $Minus$ and $Null$ respectively. The sign test returns the probability that the null hypothesis is true. This probability is called the p-value and it can be calculate for the two-sided sign test using the following formula (we multiply by two because this is a two-sided sign test and tests for the significance of differences in either direction):

$$2 \sum_{i=0}^{k} \binom{N}{i} q^i (1-q)^{N-i}$$

where $N = 2\lceil \frac{Null}{2} \rceil + Plus + Minus$ is the total number of cases, and $k = \lceil \frac{Null}{2} \rceil + \min\{Plus, Minus\}$ is the number of cases with the less common sign. In this experiment, $q = 0.5$. Here, we treat ties by adding half a point to either side, rounding up to the nearest integer if necessary. You can quickly verify the correctness of your sign test code using a free online tool.[2].

From now on, report all differences between systems[3] using the sign test. In your reports, you should report statistical test results in an appropriate form – if there are several different methods (i.e., systems) to compare, tests can only be applied to pairs of them at a time. This creates a triangular matrix of test results in the general case. When reporting these pair-wise differences, you should summarise trends to avoid redundancy.

2.3 Machine Learning using Bags of Words representations

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). In this approach, the only features we will consider are the words in the text themselves, without bringing in external 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. However, the BoW representation is also very crude, since it discards all information related to word order and grammatical structure in the original text.

2.3.1 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:

https://web.stanford.edu/ jurafsky/slp3/6.pdf

and pseudocode here:

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

[2]For example `https://www.graphpad.com/quickcalcs/binomial1.cfm`

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

[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.

You may use whichever programming language you prefer (C++, Python, or Java being the most popular ones), but you must write the Naive Bayes training and prediction code from scratch. You will not be given credit for using off-the-shelf Machine Learning libraries such as **mlpack** (C++), **scikit** (Python), **Weka** (Java), etc.

The data in `/usr/groups/mphil/L90/data-tagged/{POS,NEG}/*.tag` contains the text of the reviews, where each document is one review. You will find the text has already been tokenised for you. 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 instances.

> (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.
>
> (Q1.1) [Optional. Even if you do this, please don't report it.] Would you consider accuracy to also be a good way to evaluate your classifier in a situation where 90% of your data instances are of positive movie reviews? 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.

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 $\frac{count(w_i,c)}{\sum\limits_{w \in V} count(w,c)}$ for a word w_i becomes $\frac{count(w_i,c)+smoothing(w_i)}{\sum\limits_{w \in V} count(w,c) + \sum\limits_{w \in V} smoothing(w)}$
>
> (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?

2.3.2 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 Q2 and compute the 10 accuracies. Report the final performance, which is the average of the performances per fold.

If all splits perform equally well, this is a good sign.

> (Q3.1) Write code to calculate and report variance, in addition to the final performance.

Please report all future results using 10-fold cross-validation now (unless told to use the held-out test set).

YOU HAVE NOW REACHED THE MINIMAL REQUIREMENT FOR THE BASELINE SYSTEM.

2.3.3 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?[5]
>
> (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 training set to determine these.

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 training set once again for this.

2.3.4 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

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

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.[6]

> (Q6.0) Write the code to print out your BoW features from (Q3) in SVM Light format.[7]
>
> (Q6.1) Download and use the SVM Light implementation[8] on our dataset. Compare the classification performance of the SVM classifier to that of the Naive Bayes classifier from (Q3) and report the numbers.

More linguistics

Now add in part-of-speech features. You will find the movie review dataset has already been POS-tagged for you. 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?

3 Part Two: extension

Now your task is to improve over the baseline systems using doc2vec, and perform an error analysis on the strengths and weaknesses of the approach.[9] Doc2vec (or Paragraph Vectors), proposed by Le and Mikolov (2014), extend 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 to learn document-level embeddings.[10] This is a database of 100,000 movie reviews and can be found here:

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

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. You will also find pre-trained doc2vec models here `/usr/groups/mphil/L90/models/`, which can help you verify your implementation.[11]

> (Q8.0) Use the trained doc2vec model to infer/generate document vectors/embeddings for each review in the train and test sets you used in Part One. Now report performance using the document embeddings with an SVM.[12]
>
> (Q8.1) [Please don't report this] Compare your performance to the one you obtained with your Naive Bayes classifier from Part One (Q3). Do you achieve a significant result?

[6] You can find more details 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`

[7] Described in detail on `http://svmlight.joachims.org/`

[8] SVM Light is available for download at `http://svmlight.joachims.org/`

[9] Use the 4,000 word limit to describe the extension system only.

[10] You can use the gensim python library: `https://radimrehurek.com/gensim/models/doc2vec.html`

[11] Trained using the gensim python library.

[12] You can find more details 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`

Now inspect the model(s), examine the results, and perform an in-depth insightful analysis (something non-obvious) of the doc2vec approach to sentiment classification. Some suggestions are presented below:

(Q8.2) Are meaningfully similar documents close to each other?

(Q8.3) Are meaningfully similar words close to each other?

(Q8.4) What happens when you use pre-trained word embeddings?

(Q8.5) Are document embeddings close in space to their most critical content words?

(Q8.6) Do inferred embeddings (at a finer level of granularity perhaps) capture local compositionality?

(Q8.7) Which dimensions contribute the most to the classification decision?

(Q8.8) Are there categories of instances for which the doc2vec vectors perform better?

(Q8.9) What information do document embeddings capture? E.g., do they capture differences in genres?

Useful resources

Doc2vec:

- `https://radimrehurek.com/gensim/models/doc2vec.html`
- `https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-IMDB.ipynb`
- `https://github.com/jhlau/doc2vec`

Scikit:

- `http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`

TensorFlow:

- `https://www.tensorflow.org/programmers_guide/embedding`
- `http://projector.tensorflow.org/`

t-SNE:

- `https://lvdmaaten.github.io/tsne/`

MALLET:

- `http://mallet.cs.umass.edu/topics.php`

And papers

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.