Instructions for NLP Practical (Units of Assessment)[*] SVM-based Sentiment Detection of Reviews (Part 2)

Simone Teufel (Lead demonstrator Guy Aglionby)
sht25@cl.cam.ac.uk; ga384@cl.cam.ac.uk

This is the second part of the NLP practical, where we will use a better document representation created by doc2vec, give you some ideas for how to perform error analysis and interpretation of the results, and introduce a more powerful statistical test. You will write up these results in Report 2, with a new word limit of 1000 words (changed from previous information I gave you).

In the first part of the practical (separate document!) you coded two baseline systems that operate with bags of words. In particular, you should have the following by now:

- code for NB classifier
- code for feature treatment (bigrams etc)
- code for Porter stemming
- code for performing n-fold crossvalidation
- code for performing the sign test (with ties treatment as described)
- installation of SVMlight

1 Validation Corpus

We will now introduce the use of a validation corpus. This was mentioned in MLRD, but you have not done such experiments yet. The validation corpus's purpose is to allow you to set (i.e., learn/try out) parameters of any kind necessary by your ML algorithm, without danger of overfitting. The rule is that no part of the validation corpus can be used for training or testing.

Even just setting a feature frequency cutoff for SVM BOW requires the use of a validation corpus (arguably, you should have used on last time already if you – against instructions – did some empirical setting of the feature cutoffs, as they are parameters of your ML system). Practically, please designate 10% (the first fold in stratified Round-Robin cross-validation) for this purpose.

The standard way to use the validation corpus is with a 10-10-80 split, where you set your parameters by training on the 80% training split, choose the best on comparing results on the validation split, then discard the validation split and test the best system, i.e, only once, on the test data that has been left alone until then.

Because we want comparability of cross-validated results with Pang et al, however, this is not an option. You should therefore do the following:

- set parameter to one setting, train on the entire 90%
- measure on validation corpus
- Move on to next parameter setting and measure again

[*]This part of the practical is based on a practical designed by Helen Yannakoudakis.

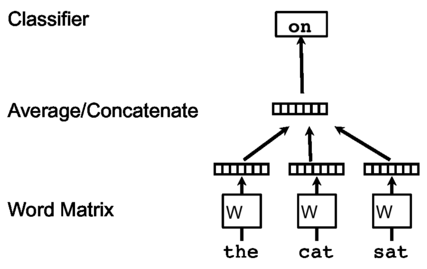

Figure 1: word2vec embeddings, trained via prediction.

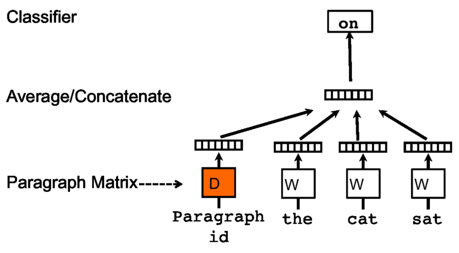

Figure 2: doc2vec, DM architecture

- Repeat until you know which parameter worked best on the validation corpus; set your final system to this.
- Then discard all trained models – it would not be fair to use these as they include the test data.
- But you are now allowed to run a normal crossvalidation on the 90% (with slightly smaller splits), using the hyperparameters you set with the validation corpus.

You still haven't tested on training with this method, but arguably some information about the test data has very indirectly sneaked into your parameters. Researchers differ as to whether they consider this overfitting or not, as it's probably a very small effect.

2 Doc2vec for Sentiment Analysis

Mikolov et al. (2013) presented the first approach to use Deep Learning (neural networks with at least one hidden layer) to NLP. They introduced the ideas of skipgrams and word2vec to create a more compact vector space representation where dimensions don't correspond to context words, but are no longer interpretable. This has often been called neural word embeddings. The approach can be used for language modelling (predicting the next word, given a context), for classification and many more NLP applications.

Based on word2vec, Le and Mikolov (2014) introduced doc2vec, which is able to learn embeddings for *sequences* of words. It is agnostic to granularity, meaning that the vectors that are created could represent a sequence of any length: a sentence, a paragraph, or even an entire document. The output of doc2vec is a document embedding, a new type of vector that has been shown to be effective for various/some tasks, including sentiment analysis.

Quick recap from lecture 9 on the distributed representation of words and prediction in word2vec (Figure 1): the CBOW model (continuous bags of words) learns to predict the target word, given some context words.

There are two possible architectures in doc2vec: the distributed memory (dm) architecture[1]and the distributed bag of words (dbow) architecture. DM architecture is shown in Figure 2 and DBOW architecture in Figure 3.

[1]Sometimes referred to as DMPV (distributed memory of paragraph vector).

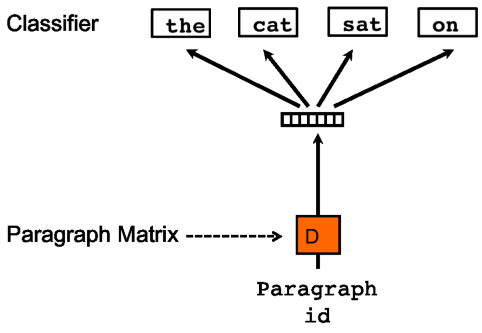

Figure 3: doc2vec, DBOW architecture

In the DM architecture, a paragraph token is added to the word2vec situation, and each paragraph is mapped to a unique vector. The paragraph vector now also contributes to the prediction task. The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. It is shared across all contexts from the same paragraph[2], and acts as a "memory" of context/topic.

The learning (constraint satisfaction) satisfies the following equation [3]:

Optimisation objective:

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t|w_{t-k},\ldots,w_{t+k})$$

Softmax output layer:

$$p(w_t|w_{t-k},\ldots,w_{t+k}) = \frac{\exp y_{w_t}}{\sum_i \exp y_i}; y = b + U\,h(w_{t-k},\ldots,w_{t+k};W)$$

In the DBOW model, alternatively, paragraph vectors are trained to predict words in a window (no word order); similar to Skip-gram model.

The relevant level of granularity is the document (review). Train various doc2vec models using the IMDB movie review database to learn document-level embeddings. For training, we use word vectors, weights, paragraph vectors (seen paragraphs). At test time, paragraph vectors are inferred by gradient descending while all else is kept fixed (word vectors, weights). The paragraph vectors received this way can be used directly as a document representation for supervised ML (here, the SVM classifier).

This is a database of 100,000 movie reviews you should use for training doc2vec:

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

For our task, you should use the `gensim` python doc2vec library. There are a number of parameters in doc2vec that can be set:

- Training algorithm (dm, dbow)
- The size of the feature vectors (e.g., 100 dimensions good enough for us)
- Number of iterations / epochs (e.g., 10 or 20)
- Context window
- Hierarchical softmax (faster version) …

[2]Please note that the paragraph vector is shared across all contexts from the same paragraph, but not across paragraphs, whereas word matrix W is shared across paragraphs.

[3]Images and formulas from Le and Mikolov (2014), though note that are there are inaccuracies: 1. figure is confusing as it does not take into account context from both sides. 2. The formula should not include the target word in the conditional. 3. In the U matrix, rows are words (in our vocab for softmax); columns features; h is the column vector and b and y are column vectors (latter size of vocab, so each cell number per word that indicates how likely, though this has not yet been converted to a probability yet)

Please familiarise yourself with the packages, train different doc2vec models, and implement a word embedding-based SVM classifier. 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. Observe the relative performance wrt. to the traditional, flat BOW representation.

3 A more powerful statistical test: Permutation test

From now on, you should use the permutation test for paired samples (we always have paired samples in this setting, as two systems are run on identical data). Like any paired test, the permutation test tests whether the population mean of the two conditions is different (H_1) or the same (H_0). Like the sign test, the permutation test is non-parametric, which means that there are no assumptions about the distribution in your underlying data. There is nothing wrong with the sign test, but it does not have a high power. The permutation test has a higher power than the sign test. You can also use the permutation test on real-valued results (here: just binary).

For an explanation of what is meant by the power of a statistical test, consider the two values α and β.

$$\alpha = P(\text{Type I Error}) = P(\text{Reject} H_0 | H_0 \text{is True})$$

α is the probability of a false positive (significance level), ie, the situation where the test declares a difference where there is really none.

$$\beta = P(\text{Type II Error}) = P(\text{Do Not Reject } H_0 | H_1 \text{ is True})$$

β in turn is the probability of a false negative, i.e., when a test declares no difference where there really is one. 1-β is called the power of the test.

The permutation test's probability of not discovering a real difference is therefore lower than the sign test's. This is a good property, because it also means that the test is more economical with data.

The core assumption of the permutation test is that if the measured difference d in mean M between systems A and B is a fluke, i.e., if there is no real difference between them (because they come from one and the same distribution), it should not matter how many times I randomly **swap** the two results. Say we have n paired results of System A and B. There are 2^n ways of flipping or not flipping the n pairs of results, i.e., 2^n permutations (hence the name of the test). These permutations operate row-wise. In other words, we create resamplings of these permutations in the following way: for each paired observation in the original runs, a_i and b_i, a coin is flipped. If it comes up heads (call this "1"), then swap the score for b_i with a_i. Otherwise, leave the pair unchanged. Now calculate the means M of the two system under this permutation, calculate their difference in means, and then do it 2^n times for all permutations. Count how many of the 2^n differences are as high as the difference in the unpermuted version. Call this number s.

There is a final twist: if due to a high number n you cannot do all exponentially many resamplings, then you should test a large enough random subset of cardinality R. This version of the test is called *Monte Carlo Permutation Test.* If $R = 2^n$, the test is referred to as the exact permutation test.

The probability of observing the difference between the original runs by chance approximated by:

$$p = \frac{s+1}{R+1} \quad (1)$$

Figure **??** shows a simple example of the Permutation test with real-valued results.[4]

As there are 6 items, exhaustive resampling would mean 64 permutations (one of these is shown in the table). Out of these, only 2 are equal or larger than the observed real difference

[4]Example due to Yiannos Stathopolous; values are MAPs (mean average precisions), where each item is a query in an Information Retrieval (IR) setting.

	Original		One permutation		
	System A	System B	Coin Toss	Permuted A	Permuted B
Item 1	0.01	0.1	1	0.1	0.01
Item 2	0.03	0.15	0	0.03	0.15
Item 3	0.05	0.2	0	0.05	0.2
Item 4	0.01	0.08	1	0.08	0.01
Item 5	0.04	0.3	0	0.04	0.3
Item 6	0.02	0.4	1	0.4	0.02
Observed MAP	0.0267	0.205		0.117	0.105
Absolute Observed Difference	0.178		0.0017		

Figure 4: Example: one out of 2^6 possible permutations: 100101.

in MAP, 0.178. The p-value$= \frac{2}{64} = 0.0462$ means that we can reject the null hypothesis at confidence level $\alpha = 0.05$.

For this practical, please implement the Monte Carlo Permutation test and use it from now on for all statistical testing where it's applicable. You can play around with its relative performance with the sign test, but you should not find a case where the sign test declares a difference and the Permutation test does not. If they contradict each other, because the Permutation test is stronger, you should always believe it. Use $R = 5000$.

4 Experimentation

The goal of this practical is to perform good science. This means that getting high numerical results isn't everything. You should aim for:

- an interesting research question
- sound methodology
- insightful analysis (something non-obvious)

Because doc2vec's behaviour is mathematically too complex to trace in detail, and because the representations it produces are not directly interpretable, you might want to find out what the model is really doing. You can perform some targeted/selected experimentation along the following lines:

- Perform an error analysis – which documents does the system get wrong most catastropically (i.e, despite high confidence)? Can you see patterns in the errors? What do you think the reason for systematic errors is? Which manipulation could improve the system?
- You could also simulate a deployment test with recent films – either downloaded from IMDB, or write your own and swap with a colleague.
- Or you could aim to explain what the model is really doing, like Lau and Baldwin (2016), and Li et al. (2015):
  - Are meaningfully similar documents close to each other? (Tip (advanced): there is visualisation SW for interpretability, typically taking the form of heat maps)

5 Some 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`

6 Some relevant 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.

7 How to write this up

Please refer to the far more contentful slides on this topic on the website. But a few hints right away: Pretend this is not a class assignment but your own idea/research question. Direct this not at the instructor (me), but at a general reader who has no knowledge of what you have done beyond what you tell them. Assume some knowledge of computational linguistic background (but if you use real technical terminology, define it first.) Don't explain obvious things (or risk looking naive), but give enough detail for an expert. After describing your data, dataset handling and methodology, describe your numerical results (clearly separated from methods). Only then do you analyse your numerical results: what is a source of errors? Interpretability of doc2vec space?