Instructions for L90 Practical*
SVM-based Sentiment Detection of Reviews (Part 2)

Simone Teufel (Lead demonstrator Paula Czarnowska)
sht25@cl.cam.ac.uk

This is the second part of the L90 practical, where we will use a better Machine Learning algorithm, and a better document representation created by doc2vec.

In the first part of the practical (separate document!) you coded a baseline system that operates 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 performing n-fold crossvalidation
- code for performing the sign test (with ties treatment as described)

1 Validation Corpus

We will now introduce the use of a validation corpus. 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.

Please from now on 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
- 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.

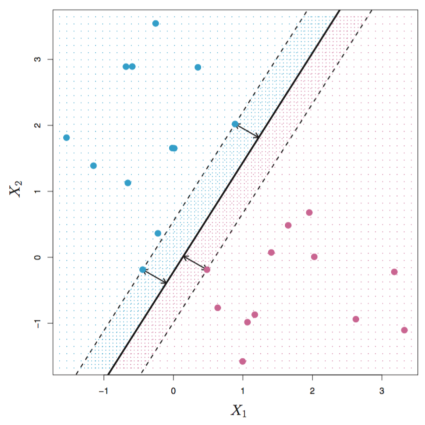
*This part of the practical is based on a practical designed by Helen Yannakoudakis.

2 Support Vector Machines; Bag of Words/ngram representation

SVM is a generalisation of simple maximal margin classifier and support vector classifier, both of which require that classes are separable by linear boundary. 3 Support Vector Machines are the extension of support vector classi

er to non-linear boundaries, but they support only two classes (extensions to multi-class is possible).

A hyperplane in p-dimensions is a at (p-1)-dimensional affine subspace. In 2 dimensions a hyperplane is a line; in 3 dimensions, a hyperplane is a plane. We can now compute the distance between data points and various hyperplane, out of which we select the one that creates the largest margin (best separation) between the two classes. Support vectors are data points lying on the margin. Classification of a test point depends on which side of this hyperplane it falls on. The points closest to the margin are the most difficult to classify. The size of the margin is standardly interpreted as the SVM's confidence in the classification.



source: ISLR Fig 9.3

We recommend SVMLight as implementation for the practical.

3 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).

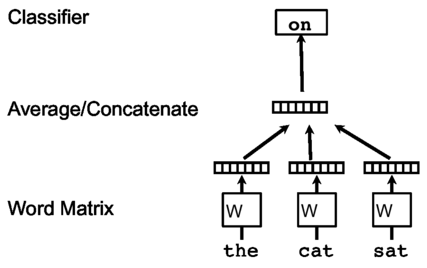


Figure 1: word2vec embeddings, trained via prediction.

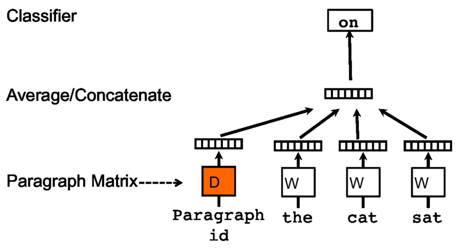


Figure 2: doc2vec, DM architecture

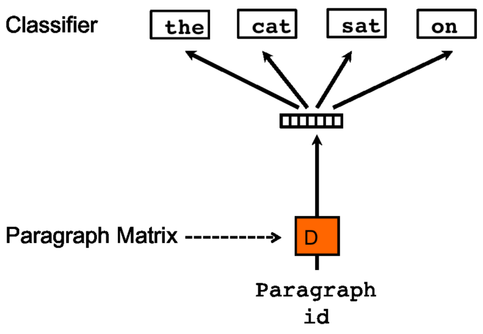


Figure 3: doc2vec, DBOW architecture

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).

For our task, you should use the `gensim` python doc2vec library. Use this dataset of 100,000 reviews as training data:

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

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

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) ...

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.

4 Some useful resources

Doc2vec:

- `https://radimrehurek.com/gensim/models/doc2vec.html`

[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)

- 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

5 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.