COMP34711 Natural Language Processing - Coursework 2

1. Task 1 Distributional Semantics - Sparse Approach

1.1 PMI, PPMI, PPMI with smoothing

Pointwise Mutual Information (PMI) (1) and its derived functions- Pointwise Mutual Information (PPMI) and PPMI with smoothing are commonly used weighting functions for term-to-term matrices. PMI provides information about how often two events co-occur in contexts compared to how often they occur in a corpus overall. In (1) $P(w, c)$ is a measure of how often a keyword - ***w*** and a context word - ***c*** appeared in the same context, $P(w)$ and $P(c)$ are measures of how often the two words appear in the text overall. Because PMI uses a logarithmic operation to scale the cooccurrence ratio, PMI values can be in the range between negative and positive infinity. Negative PMI values imply that a key and context word are less likely to co-appear in a context than by chance. These values are generally accepted as unreliable (very large corpora greatly exceeding the size of the product review dataset are an exception). Because of that, a far more commonly used weighting function for term-to-term matrices is Positive Pointwise Mutual Information (PPMI), which nullifies all negative values. (2)

PMI and PPMI are generally biased towards prioritising co-appearances of keywords with rare context words. This bias could lead to disregarding the co-appearances with common context words in procedures such as word clustering based on cosine similarity. Because of that, we often apply smoothing on the context word probability by raising it by a power α in the range between 0 and 1. (3)

PMI and PPMI can be used to form sparse matrices for distributional semantics tasks such as clustering. A term-to-term matrix representing the PMI or PPMI score for a particular context word and keyword can be formed from a cooccurrence-based term-to-term matrix. We can label the PMI/PPMI score matrices as sparse since each cell representing a keyword-context word tuple, which has no occurrences, will contain 0. In my solution, I experimented with all three methods.

1.2 Experiments, fine-tuning hyperparameters

All the experiments were run for 20 iterations. The standard deviation and average accuracy across all of them is reported. Since every reverse word corpus generation is different and a new one happens per iteration, the results reported will slightly differ from the ones that will be generated when running the code.

1.2.1 Term-to-term context-cooccurrence matrix construction hyperparameters:

There are two main hyperparameters which affect the term-to-term cooccurrence matrix- the document cleaning type applied prior to the matrix construction and the context window length used to form the matrix. The size of the context window strongly affects what the matrix represents. Generally, shorter window sizes capture syntax and longer ones capture semantics. To experiment with longer windows, I ran tests with context windows, which are not based on a fixed offset of words before and after keywords, but on dynamic length language structures such as sentences and reviews. In my experiments (4), I achieved the best accuracy (86%) on the clustering task with the smallest context window possible - the word before and the word after the keyword. Smaller windows lead to a more syntactic representation, which would make it easier to partition based on the POS tag of the keyword. I also experimented with stop word removal and stemming. Applying either technique decreases accuracy with smaller contexts. This confirms the hypothesis highlighted above that smaller contexts reflect syntax in representation. (5)

1.2.2 PMI, PPMI, PPMI with smoothing comparison

Despite the theoretical advantage of PPMI over PMI, on our dataset, PMI and PPMI perform very similarly, there are even cases of running the experiments where PMI outperforms PPMI. This could showcase that the negative values actually do provide useful information for clustering or that there are very few negative

values in our PMI matrix. PPMI with smoothing outperforms the other two methods. This improvement is, however, negligible. The three methods usually have an accuracy range of 3%. (6)

The best accuracy achieved was with a context window of one word, PPMI with smoothing, no stemming, no stopword removal and with agglomerative clustering with complete linkage. The accuracy is 86% and the standard deviation is 0.03.

2. Task 1 Distributional Semantics - Dense Approach

2.1 Word2Vec Skip-grams

The Skip-grams technique is an unsupervised learning approach for finding word embeddings. The Skip-gram technique finds word embeddings by training a classifier to find the likelihood of a certain context word to appear given a keyword. The Skip-gram model architecture consists of an embedding layer - a layer which matches each word (represented by a scalar) to a vector of values, a linear layer, and a softmax function to convert the linear layer output to a probability distribution. The final probability distribution produced by the model represents how likely it is to have an occurrence of each context word, given the input keyword. The weights of the embedding layer of this classifier can be effectively used for word embeddings. The model resembles a linear regression with a softmax function applied to the output of the linear layer. Because of this, the Skip-gram model is a relatively flat mode, allowing for backpropagation to effectively be applied back to all layers, and has a relatively small number of weights. This makes it quick to train.

2.2 Experiments, fine-tuning hyperparameters

All the experiments were run for 5 iterations. The standard deviation and average accuracy across all of them are reported in the code. Since every reverse word corpus generation is different and a new one happens per iteration, the results reported will slightly differ from the ones that will be generated when running the code.

2.2.1 Skip-gram context window size

Similarly to sparse encodings, here the smallest possible context window size - the word before and the word after the keyword - lead to the best accuracy. (7) The reasons for this window size performing the best are the same as with the sparse approach.

2.2.2 Embedding length

My experiments showed that word embeddings of lengths 200, 250, 300 provide the best accuracy for a context window of 1 preceding word. (9) Shorter word embeddings are less explicit about the properties of keywords. Longer embeddings than 300 perhaps require more extensive training data so as to achieve optimal performance. A potential further test for this hypothesis is to investigate how longer word embeddings perform with larger window contexts as these lead to more training data.

2.2.3 Stopword removal and stemming

As with sparse approaches, here accuracy drops when stopword removal is applied. This showcases again that stopwords provide valuable information about a keyword's syntax, which is omitted when stop words removal is applied. Stemming did not influence the accuracy. (8)

The best accuracy achieved was with a context window of one word, embedding length of 300, no stemming, no stopword removal and agglomerative clustering with complete linkage. The accuracy is 91% and the standard deviation is 0.03. With these hyperparameters, we benefit from syntactic representation being more suitable for the task and from the explicitness of longer embeddings.

3. Task 2 Sentiment Analysis

3.1 Approach description

Many different neural network model architectures have been shown to be suitable for sentiment analysis tasks. I decided to implement a Convolutional Neural Network (CNN). An advantage of this approach in comparison to the LSTM-based approach is that CNNs do not suffer from the limited attention information bottleneck problems that occur with LSTMs. Because of that, generally, for most sentiment analysis tasks, CNN models tend to outperform LSTM models.

CNNs are designed to handle computer vision tasks. Due to the fact that in machine learning tasks, words are represented as embeddings, text can be represented as a two-dimensional matrix consisting of concatenations of word embeddings. On such a representation we can use convolutional layers of size [n x size of word embedding] to target n-grams with our convolutional layer.

In my neural network implementation, I use an embedding layer to represent the sentiment text as a list of dense vectors. This list is ingested by three CNN layers, with filter sizes of [3 x embedding size], [4 x embedding size] and [5 x embedding size]. This allows the neural network to infer on all tri-gram, four-grams and five-grams in each sentence. Following that, separate pooling layers are applied for each convolution. The output dimensions of the convolutional layers are dependent on the sentiment length. The pooling layers ensure that the output from all convolutional layers has the same consistent length no matter the text size. A fully connected layer and a sigmoid layer process this information to output a scalar in the range [0,1] to describe whether the model believes a text is negative or positive has a negative or positive sentiment (Refer to (10) for a diagram). Each inference value less than 0.5 is considered negative and each inference greater than or equal to 0.5 is considered positive.

3.2 Experiments, fine-tuning parameters

3.2.1 Document cleaning

Stopword removal had a negative impact on accuracy. The reason for that is that the nltk stopwords list includes tokens indicating negation such as “not”. Stemming also negatively affected the accuracy.

3.2.2 Dropout

Handling overfitting is a difficult task with this dataset. It is often the case that accuracy of more than 95% is reached on the training data, while the evaluation accuracy is close to 70%. The model is struggling to generalise its inferences. To reduce the extent of overfitting I introduced dropout to the input of the fully connected layer. Dropout is not applied to the testing data. This led to a 2% accuracy improvement (11).

3.2.3 Removing texts with mixed sentiments

Certain texts have both negative and positive tags. There are close to 50 such sentiments in the corpus. I experimented with removing them since they can be considered noisy data in binary classification. This improved accuracy by 2%. (12)

3.2.4 Embedding size and number of filters per convolutional layer

An embedding size of 300 and a filter count of 100 led to the best testing accuracy. A possible reason why CNNs with more parameters did not outperform smaller ones is the lack of enough training data. It is possible that the more complex models were overfitting for the limited data available.

The final hyperparameters selected were a dropout rate of 0.85, embedding length of 300, 100 CNN filters, applying mixed sentiment cleaning and not applying stemming or word cleaning. The model is trained for 30 epochs and has an accuracy of 74%. With these hyperperameters we benefit from the better generalisation dropout provides, the lack of ‘confusing’ mixed samples and from more explicit embeddings.

Appendix

Formulas:

$$PMI(w,c) = log_2 \frac{P(w,c)}{P(w)P(c)}$$

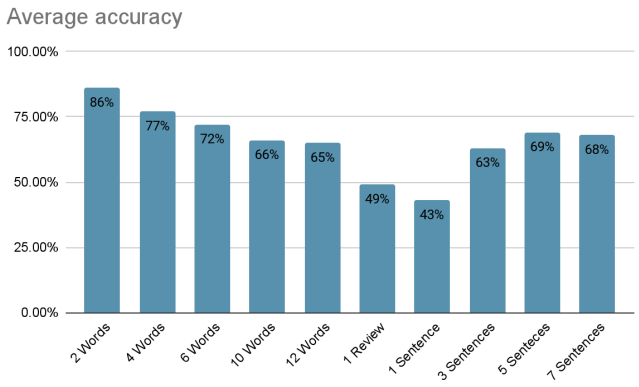
$$PPMI(w,c) = max\left(\frac{P(w,c)}{P(w)P(c)}, 0\right)$$

$$PPMI_\alpha(w,c) = max\left(log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0\right), \text{ where } P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

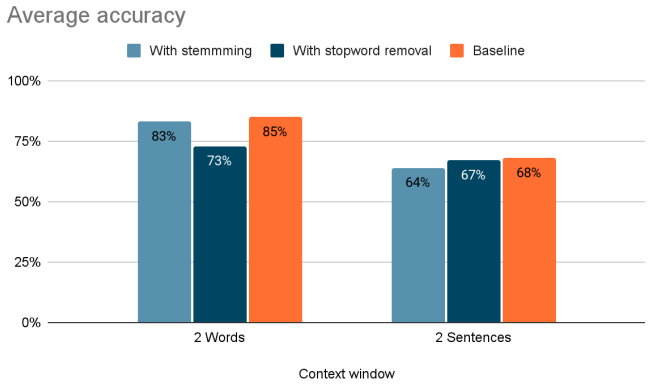
Experiment results:

Task 1- Approach 1:

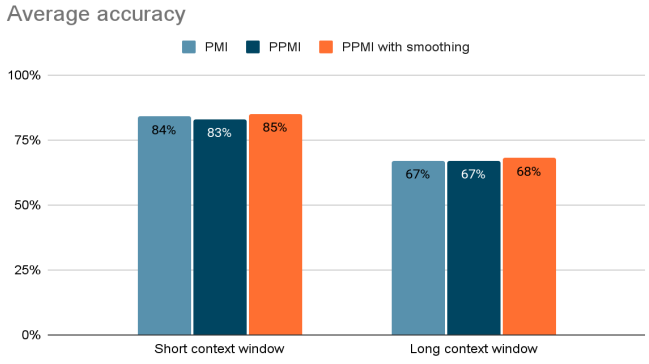
Effect of context window size:



Stemming and stop word removal

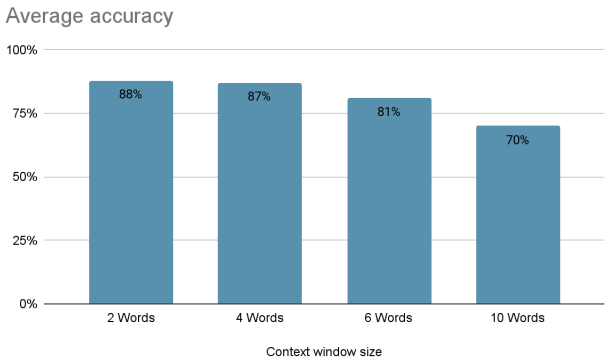


PPMI vs PPMI vs PPMI with smoothing

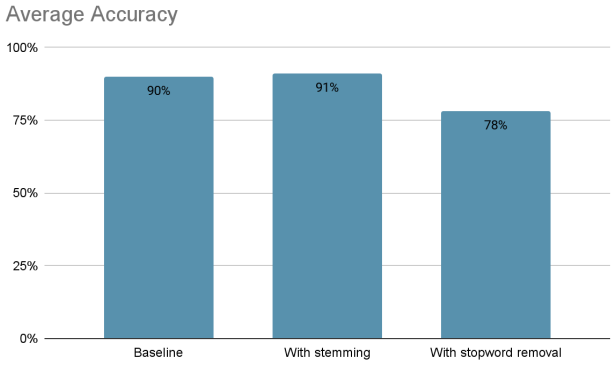


Task 1- Approach 2

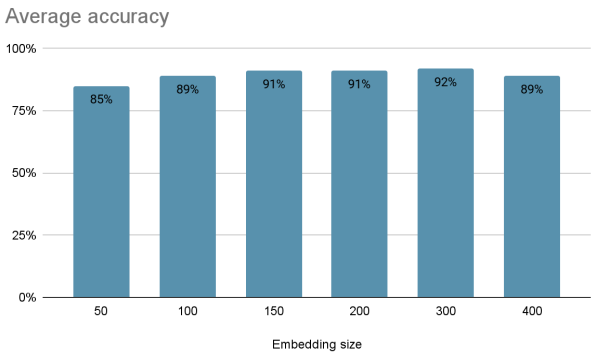
Effect of context window size:



Stemming and stop word removal:

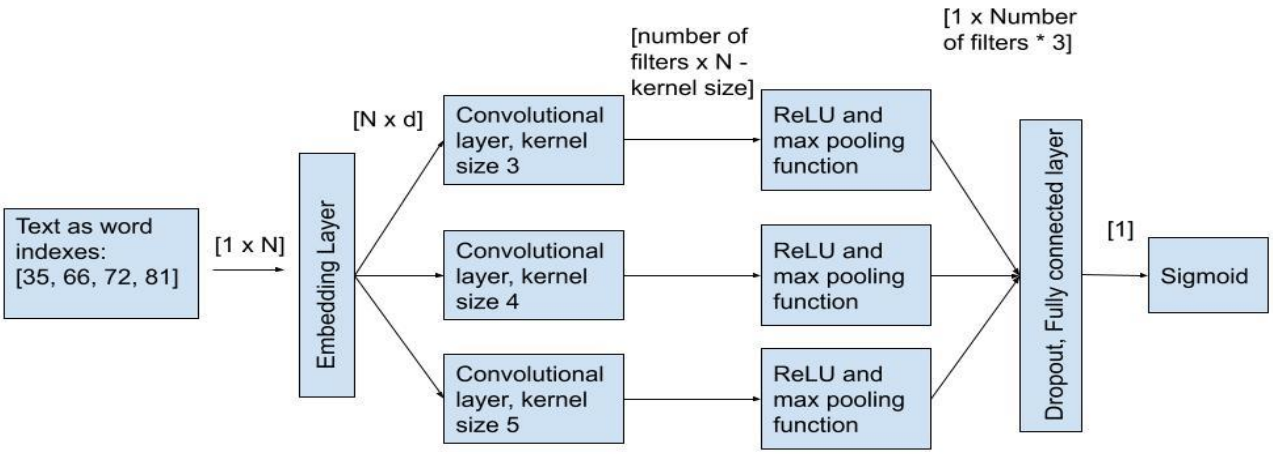


Embedding dimensions:

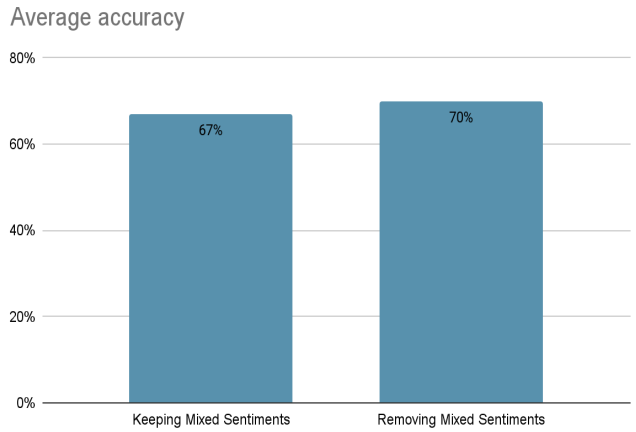


Task 2

Model summary



(11) Effect of mixed sentiment omission



(12) Effect of dropout rate