Task 1: Distributional Semantics

Method Description

- **Text cleaning & pre-processing** - As the task is concerned with product reviews, the text data contained a large variety of (semantically) unhelpful characters and symbols such as brackets, encoding strings (e.g., \u00F4), currency expressions, numbers, and punctuation. As such, they were removed for the purpose of the task. Additionally, all the sentiment scores and details (e.g., [u], [p], etc.) were excluded so as not to allow them (e.g., `u`, `p`) be recognised as single tokens. Finally, regarding the cleaning, we expand the contractions (i.e., apostrophes) which are prevalent in online reviews based on predefined set of rules. Once the text was tokenized (with NLTK's recommended word tokenizer), stop words are discarded because 1) this will reduce the document size and 2) it will not affect the distributional semantics as stop words are ubiquitous (hence they are not a feature). Furthermore, stemming is applied (Snowball Stemmer as a more aggressive method) to facilitate the construction of representations – allowing different word forms (i.e., `batteries` and `battery`) to be transformed into a single base form (***approach novelty 1***). This is helpful as it 1) accounts for wordcount of top words with multiple forms throughout the text, 2) ensures our top words are unique (otherwise, we cannot easily classify them based on a window of neighbours). However, we could use lemmatization just as well (worth further experimentation). An avenue to improve my tokenization would be to have an external knowledge base on multi-word expressions (e.g., 'iPhone 12') to effectively calculate statistics on them (maybe a few might have made it into top 50).
- **Distributional semantic representations** – After cleaning & pre-processing the text data, we partition it to a sentence level (i.e., split by '##' and '\n'). The selected method to represent the sentences is word2vec as an alternative to the sparse one-hot encoding (***approach novelty 2***). It is also the state-of-the-art method allowing modelling words based on its neighbours (impossible with bag-of-words which destroys order). Furthermore, we do not make any assumptions on whether the model should be CBOW or skip-gram (both are experimented with).

Result Analysis

- **Hyperparameters** – we are evaluating four hyperparameters – 1) number of epochs *[5, 10, 30, 50, 75, 100],* 2) word2vec dimensions *[100, 164, 200, 300]* as the product review dataset is quite small, 3) window sizes of *[2, 3, 4, 5]*, 4) CBOW or Skip-gram (***approach novelty 3***). Each combination is evaluated 10 times. Regarding the clustering algorithm, we use the "Elkan" variation of K-means which are more efficient on datasets with well-defined clusters (***approach novelty 4***), such as our case as we ensure the top 50 words are unique (stemming).
- **Hyperparameter tuning** – The analysis is split based on the last hyperparameter (i.e., CBOW or Skip-gram). While both methods result in a similar best accuracy (from the experiments), they do so under different training conditions – 30 epochs are enough for the skip-gram (in fact more than 30, hurt accuracy possibly due to over-fitting), whereas CBOW requires at least 20 more epochs to reach the same level of accuracy (*Figure 1, 2*). Additionally, while the latter also displays slight differences in accuracy between feature dimensions, skip-gram does not (*Figure 3, 4*). Regarding the window size, it does not seem to significantly affect the accuracy in either of the models (*Figure 5, 6*).

  After performing the experimentation, we have picked the following model as one of the best:
  - Skip-gram trained for 30 epochs with an accuracy of 82% and
  - 100 feature dimensions with a window size of 2

  Due to the small dataset, the minimal hyperparameters that maximise accuracy, the better.

Task 2: Neural Network for Classifying Product Reviews

Method Description

- **Text cleaning & pre-processing** – Though, we are mostly reusing the text cleaning methods from the distributional semantics task, there are key differences. Firstly, we ensure all provided quantitative sentiments are in unified format (e.g., `[+]` and `[1]` both become `[+1]`)), whereas all qualitative ones are excluded (e.g., [u], [p], etc.). We introduce a **Sentiment Extractor** to parse review-level and sentence-level (in case when the provided review separation is missing). Thus, we maintain a data structure containing each review, its **sentiment score** (i.e., ***the sum of sentence-level sentiment scores***) and its **binary sentiment** (be it Positive, Negative or Neutral). We insist on this binary score as Neutral reviews are not included in the training of our Deep Learning (DL) model. These scores are derived from the total integer sentiment score per review, and we provide an example on how to calculate them here:

  If *`install[-2], router[+1]`* are the sentence-level scores, then the total sentiment score will be *`-2+1 = -1`* ⇔ Negative sentiment.
- **Model design & training** - Once the data is successfully split into two binary classes, we could further partition it into train and test datasets. The analysis below will be predominantly based on **imbalanced data** (positive-negative data has an approximate ratio of 5:2). Finally, we run an experiment with **stratified data** to try and address this imperfection.

  Once the data has been cleaned and partitioned, it is tokenized and encoded with the default LSTM approach (because my custom self-built Index-Based Encoding was finicky to pass as input to the network). We select to implement a bidirectional LSTM model as it 1) offers better predictions and 2) is slightly more advanced Deep Learning approach compared to earlier RNNs like normal LSTMs [1] due to its left-to-right and right-to-left modelling. Our architecture extends this with typical classification layers known from experience with Convolutional Neural Networks (CNNs) – (global) max pooling to reduce latent signal number, a dense fully-connected layer to mix the signals together, and a random dropout for regularization prior to final binary prediction.

Result Analysis

- **Hyperparameters & tuning** – Initially, we identified the following hyperparameters: LSTM dimension ***[32**, 64, 128, 180, 256]*, Fully-connected final layer size ***[10**, 50, 100]*, dropout rate *[0.01, **0.1**, 0.3]* and training epochs (up to 10). After multiple experiments we concluded for our architecture made of: Embedding, Bidirectional LSTM, GlobalMaxPool1D, Dense, Dropout, Dense(1) layers, that none of the internal parameters significantly affect the accuracy. However, we also observed that, overall, training beyond **third epoch (experiment 1)** did not result in positive change in validation score, which could be caused by over-fitting. Therefore, we relaxed training time by cutting epochs to maximum of three and exploring all LSTM dimensions (**experiment 2**), which confirmed that **our minimal hyperparameters were sufficient**.

  Finally, we compared our normal K-Fold cross validation with the Stratified version which ensures distributional consistency in imbalanced data in train-test partitioning (**experiment 3**). While in principle this is sensible and correct, it did not seem to display any improvement in the **test accuracy (71%)**, which I think is due to the significantly small size of the negative samples.
- **Discussion on Optimizations**
  1. Employ word2vec for text representation when passing text to the DL's input layer.
  2. Our architecture can be enhanced with a convolutional layer to extract text features and reduce dimensionality of data [2].
  3. Embed the sentiment word scores as word weights to our review-level representations for an easier differentiation between positive and negative words.
  4. Implement a self-attention mechanism or a transformer-based architecture, although our training data is quite small.

Appendix

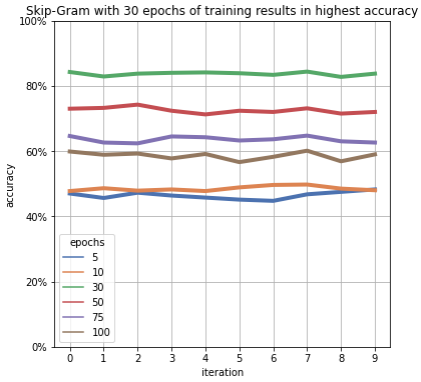

Figure 1. Skip-gram best accuracy.

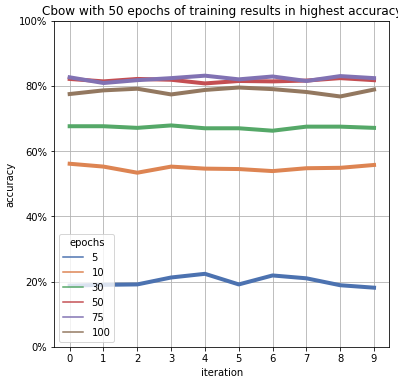

Figure 2. CBOW best accuracy

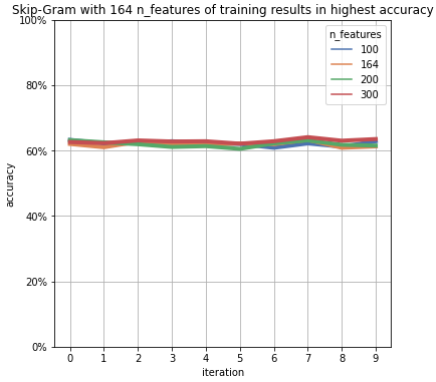

Figure 3. Skip-gram dimensions

Figure 4. CBOW dimensions.

Figure 5. CBOW best accuracy

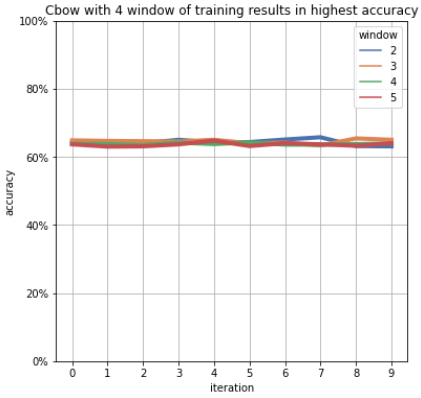

Figure 6. Skip-gram dimensions

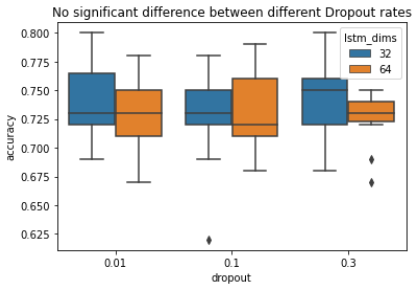

Figure 7 Experiment 1. Dropout rate.

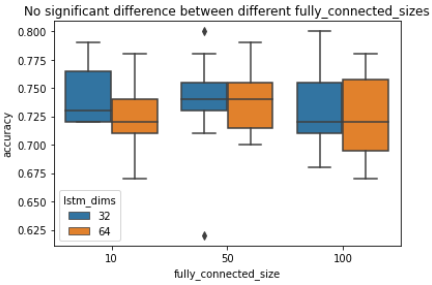

Figure 8 Experiment 2. Fully connected layer.

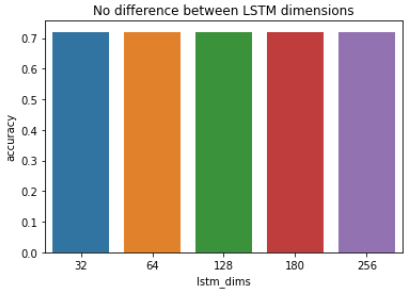

Figure 9 Experiment 2. LSTM dims

References:

[1] - S. Siami-Namini, N. Tavakoli and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3285-3292, doi: 10.1109/BigData47090.2019.9005997.

[2] - Gang Liu, Jiabao Guo, “Bidirectional LSTM with attention mechanism and convolutional layer for text classification”, Neurocomputing, Volume 337, 2019, Pages 325-338, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2019.01.078 .