به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین شماره دو (سوال دوم و سوم)

نام و نام خانوادگی: امیرحسین شاه قلی

شماره دانشجویی: ۸۱۰۱۹۹۴۴۱

فروردین ماه ۱۴۰۳

Table of Contents

Answer of the question two

Answer of the first part:

Preprocessing Part: We discard the “article_link” column as it is not required for sarcasm detection. We convert the text to lowercase, remove punctuation and numbers, and filter out English stop-words. Next the text is split into individual words (Tokenization) and we reduce tokens to their dictionary form (lemmatization).

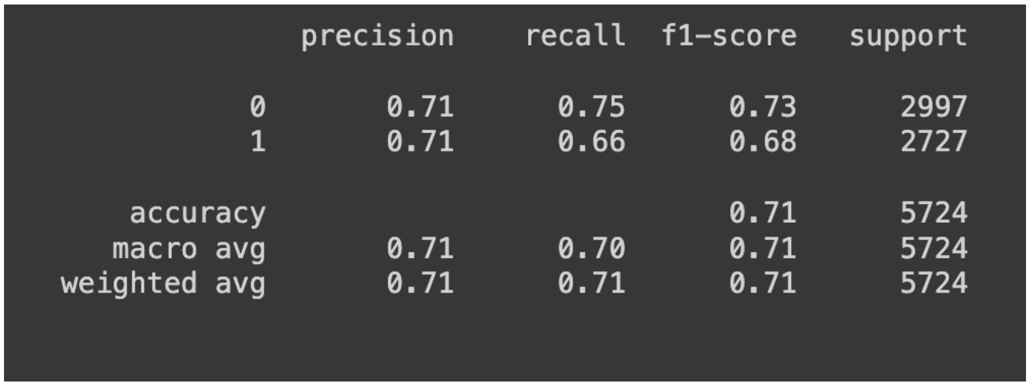
Answer of the second part:

First we load the GloVe embeddings from a pre-trained file. The embedding is a vector of 300 floating-point numbers (this is specific to the glove.6B.300d.txt version we are using). we parse each line to map words to their corresponding embeddings and store this in the embeddings_index dictionary. After that, using Keras Tokenizer, we convert the preprocessed headlines into sequences of integers, where each integer corresponds to a unique word. The 5000 parameter limits the tokenizer to the top 5,000 words. Then, we pad these sequences to ensure they are all of the same length, defined by maxlen. An embedding matrix is created where each row index corresponds to the integer representation of a word and the row content is the GloVe embedding for that word. If a word is not found in the GloVe data, its embedding row stays as zeros. For each padded headlined sequence, it looks up the embeddings for each word that is not zero and takes the mean of these word embeddings to get a single vector representation for the entire sequence.

Answer of the third part:

We use a Logistic Regression classifier to distinguish between sarcastic and non-sarcastic headlines. We utilize the SMOTE algorithm to generate synthetic samples in the feature space. The dataset is split into training 80% and testing 20% sets. We fit the LogisticRegression model to the training data and make predictions on the test set. The performance of the model is assessed by the following metrics: precision, recall, F1-score, and support.

Results:


```
              precision    recall  f1-score   support

           0       0.71      0.75      0.73      2997
           1       0.71      0.66      0.68      2727

    accuracy                           0.71      5724
   macro avg       0.71      0.70      0.71      5724
weighted avg       0.71      0.71      0.71      5724
```

Model achieved an overall accuracy of 0.71 in predicting whether a headline is sarcastic or not. Precision for both classes is 0.71. This means that when the model predicts a headline as sarcastic, it is correct about 71% of the time. Recall for the non-sarcastic headlines is 0.75, indicating that the model correctly identified 75% of the actual non-sarcastic headlines. Recall for the sarcastic headlines is slightly lower at 0.66, suggesting that it misses more sarcastic headlines compared to non-sarcastic ones. The F1-score, which is a balanced measure of precision and recall, is 0.73 for non-sarcastic headlines and 0.68 for sarcastic ones. This indicates a generally balanced performance for precision and recall, but with the model being slightly better at identifying non-sarcastic headlines.

Model demonstrates a good ability to classify headlines into sarcastic and non-sarcastic categories. The balanced precision and recall suggest that the model does not excessively favor either class. However, there is room for improvement, especially in the recall for sarcastic headlines, where the model can be optimized to capture more instances without significantly compromising precision.

Answer of the question three

Answer of the first part:

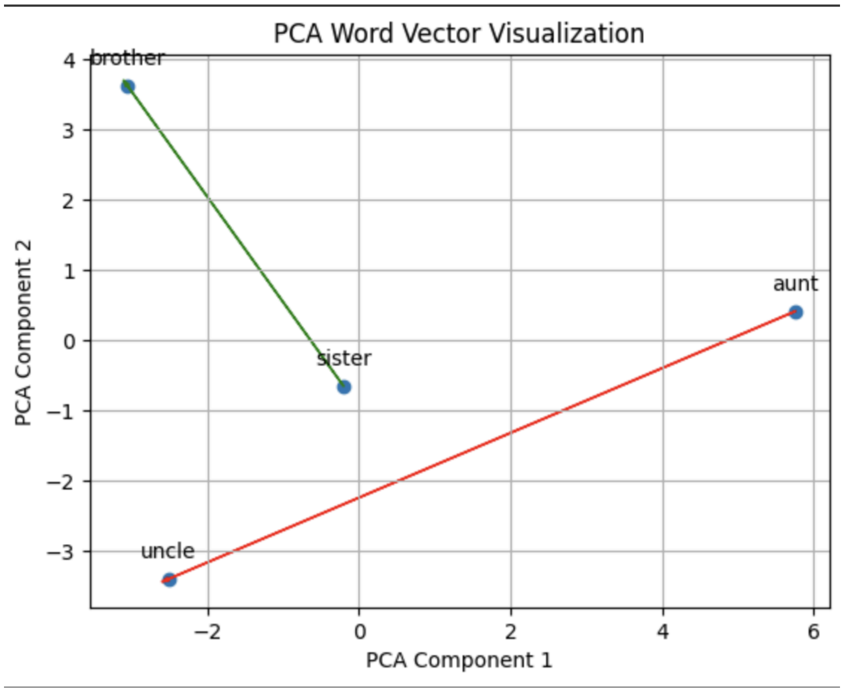
Preprocessing Part: We convert the text to lowercase, remove punctuation, and filter out English stop-words. Next the text is split into individual words (Tokenization) and we reduce tokens to their dictionary form (lemmatization). We continue by creating two matrices with 100 dimensions each, corresponding to our choice of the embedding size. These two matrices will serve as the initial representations for our words. One matrix is for the target words, and the other is for the context words. We process the text, extracting pairs of target words and their surrounding context words within a specified window size. For instance, with a window size of 5. For each pair of target and context words, we select additional negative words that are not present in the context. These words serve as negative samples and help the model learn which words are not typically found together. The number of negative samples can be set according to the model's needs, and in this case, we are using 4 negative samples for each positive pair. During training, we aim to adjust the word vectors in such a way that the model produces high scores for actual context words and low scores for negative samples. For a given target word, we take its vector from the first matrix and calculate its dot product with the vector of the context word from the second matrix. Simultaneously, we calculate the dot products with the vectors of the negative samples. We apply an objective function to these scores that encourages the model to output high probabilities for real context-target pairs and low probabilities for negative samples. This is typically done using the log-sigmoid function and involves maximizing the probability of the true context words and minimizing the probability of the negative samples. We use an optimization algorithm, such as stochastic gradient descent, to adjust the vectors in both matrices to minimize the loss function.After training, we may choose to represent each word by either the vector from the target matrix, the context matrix, or by an average of both. This gives us the final embeddings, which we can use to measure similarity between words, or as features for various natural language processing tasks.

Answer of the second part:

Initially, we normalized the word_vectors matrix. Each word vector is divided by its L2 norm (Euclidean norm), resulting in unit vectors. This normalization step ensures that the cosine similarity calculations are strictly comparing the directions of the vectors, rather than being influenced by their magnitudes. Next, we retrieve the embeddings for four specific words: "king," "man," "woman," and "queen." The operation "king - man + woman" aims to perform a gender transfer, effectively attempting to find a vector that represents the female counterpart to a "king." According to the famous analogy, if word embeddings have captured the relational concepts correctly, the result of this vector arithmetic should be a vector similar to the embedding for "queen." We calculate the result of this arithmetic operation and then compute the cosine similarity between this resulting vector and the actual embedding for "queen." Cosine similarity is a standard measure used to quantify the similarity between two vectors, with a value of 1 indicating perfect similarity, 0 indicating orthogonality, and -1 indicating complete dissimilarity.

Notice: The low cosine similarity score reported (approximately 0.23352) suggests that the resulting vector from "king - man + woman" is not very similar to the actual "queen" vector according to the model's embedding space.

Answer of the third part:



The PCA Word Vector Visualization shows the words “brother”, "sister”, "uncle”, and “aunt” plotted in a 2D space after applying PCA to their respective high-dimensional word vectors.And also, vectors representing the differences between the words “brother” to “sister” and “uncle” to “aunt” are drawn as arrows. The plot reveals that the direction of these difference vectors doesn't appear to be parallel, which suggests that in this embedding space, the gender relationship is not consistently captured across these word pairs; that is to say, the transformation that relates “sister” to “brother” is different from the one that relates “aunt” to “uncle” according to the embeddings trained on your corpus.