Natural Language Processing

Assignment 2

Fatemeh Nadi 810101285

April 28, 2023

Problem 1

Please see this file: "CA2_NLP_Q1.ipynb"
Code and explanation are provided

Our dataset has 34916 positive comments and 34564 negative comments, and we want to train a naive bayes model to detect semantics on unseen data.
After splitting the original dataframe into two dataframes based on 'label/$_i d' column, randomly sample the$ $Start preprocessing the dataset now.$

Remove stop words.

Remove punctuation, digits, Latin characters, extra spaces.

Replace Arabic to Persian.

Normalize the comment.

Tokenize words.

Lemmatize the words.

Part A: TF-IDF

$$W_{i,j} = tr_{i,j} \times log\left(\frac{N}{df_i}\right)$$

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

TF: (Number of repetitions of word in a document) / (of words in a document)

```python
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)

    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)

    return tfDict
```

✓ 0.0s

IDF: Log[(Number of documents) / (Number of documents containing the word)]

```python
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))

    return idfDict
```

Take a look at the report:

	precision	recall	f1-score	support
0	0.57	0.83	0.68	355
1	0.66	0.36	0.46	340
accuracy			0.60	695
macro avg	0.62	0.59	0.57	695
weighted avg	0.62	0.60	0.57	695

Part B: PPMI

$$PMI\left(t_1, t_2\right) = log\left[\frac{p\left(t_1, t_2\right)}{p\left(t_1\right) \times p\left(t_2\right)}\right]$$

Create a dictionary that holds the word frequencies for each sentence after creating a list of unique words in the sentences.
Create a matrix with the dimension of $len(comments) \times len(VOCAB)$, in which each row represents the frequency of that word in all comments.

In order to reach word-matrix, we must calculate $|VOCAB| \times |VOCAB|$ Hence, if previous matrix is M, $M^T M$.
The matrix shows how often each pair of words occurs together.

Now we can calculate $p\left(t_1, t_2\right), p\left(t_1\right), p\left(t_2\right)$, and each word should be embedded in a vector.
For each comment, create an embedding based on the average of the embedding of words in the sentence.

```
for i in range(0,x.shape[0]-1):
    for j in range(0,y.shape[1]-1):
        res = x[i][j]/(x[i][-1]*x[-1][j])
        if res !=0:
            pmi = np.log(res)
            #print(pmi)
            if pmi>0:
                y[i][j] = pmi
            else:
                y[i][j] = 0
        else:
            y[i][j] = 0

ppmi_df = pd.DataFrame(y, columns=words, index=words)
ppmi_df
```

Take a look at the report:

	precision	recall	f1-score	support
0	0.87	0.55	0.68	325
1	0.70	0.93	0.80	365
accuracy			0.75	690
macro avg	0.78	0.74	0.74	690
weighted avg	0.78	0.75	0.74	690

Conclusion

It can be said that PPMI is generally more effective than TFIDF, and that TFIDF has a higher score on the positive class and PPMI has a higher score on the negative class.
The TFIDF approach has a higher recall in the positive class and a higher precision in the negative class, and PPMI has the opposite result.

Problem 2

Please see this file: ”CA2_NLP_Q2.ipynb”
Code and explanation are provided

A.

Skip-gram is used to predict the context word for a given target word. It's reverse of CBOW algorithm.
Follow these steps to preprocess data:

- Convert input text to lower case
- Expand contractions in input text
- Removes all punctuation from a string
- Remove the number from the input text
- Remove leading, trailing, and (optionally) duplicate whitespace
- Normalize the data.
- Remove the stop words
- Delete some unneeded data

Let's take a look at the SkipGram model now.

This algorithm aims to obtain a 100D vector for each word in the corpus.
For the initial part, the embedding size was set to 100 and initializes the weights of the model randomly, we want to close these vector to contex.

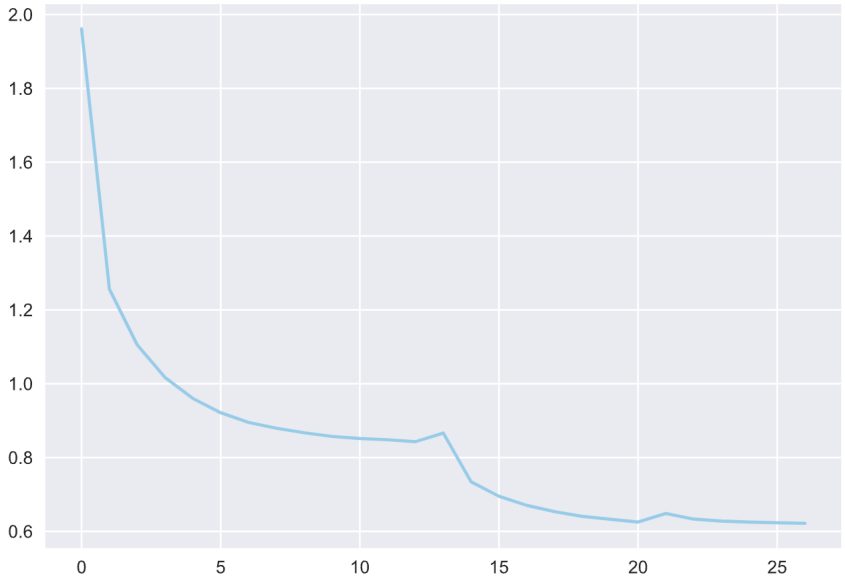
Based on a center word and a context word, the Skip-Gram model predicts a context word. Using a negative log-likelihood model, the model minimizes the negative log-likelihood of the target word and its context words. A gradient loss is calculated based on the parameters of the model using the method calculate_gradients.

The ‘get_negative_samples‘ method returns a list of negative samples, which are words that are not in the context of the target word.
Negative samples are generated randomly from the vocabulary excluding context

words. The num_ns parameter specifies the number of negative samples to generate, and the internal multiplication of the center vector of word is maximized.

Positive samples are those words that occur within a certain window size around the center word. For each center word, the algorithm generates one positive sample for each context word within the window.

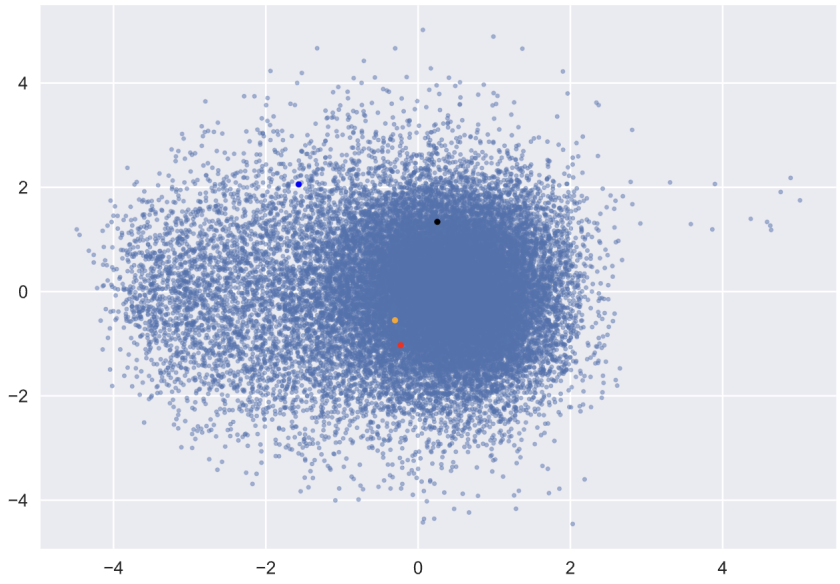
After building our model, train it for 13 epochs with 0.1 learning rate, 8 epochs with 0.01 learning rate, and 6 epochs with 0.001 learning rate.
The loss reached 0.62, so let's look at the cure:



B.

```
a = [
final_model_pca[model.word2index['queen'] - model.word2index['woman']],    # black
final_model_pca[model.word2index['king'] - model.word2index['man']],       # blue
final_model_pca[model.word2index['brother'] - model.word2index['sister']], # red
final_model_pca[model.word2index['uncle'] - model.word2index['aunt']]      # orange
]
a = np.array(a)
```

and result:



A brother-sister vector is closed to an uncle-aunt vector, which represents a relationship in 100 dimensions.

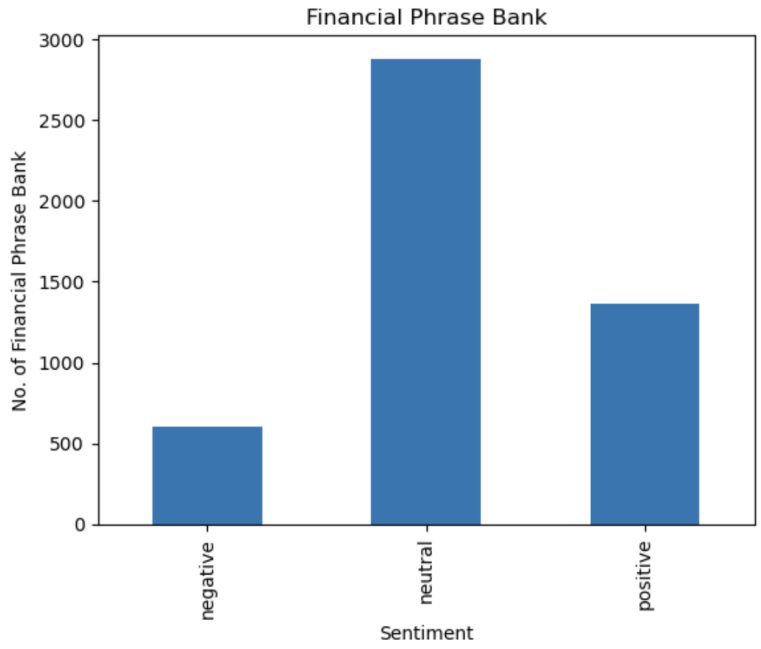
King vector - man vector is not closed to queen vector - woman vector, perhaps because when we reduce dimension, this relationship is far from each other, or because the model didn't train well, or because our corpus is not suitable for such a representation.

Problem 3

Please see this file: ”CA2_NLP_Q3.ipynb”
Code and explanation are provided

A.

Let’s start by looking at the distribution of each class in this dataset.



Follow these steps to preprocess data:

- Convert input text to lower case
- Expand contractions in input text
- Removes all punctuation from a string
- Remove the number from the input text
- Remove leading, trailing, and (optionally) duplicate whitespace
- Normalize the data.
- Remove the stop words

Then download glove and unzip it in Notebook to use for embedding.

For this project, I use 300d vector embedding for each word, so I load all words' embeddings into matrix embeddings and then find their embeddings using those. By using the text_to_embeddings function, we get each sentence and split it into its words. The embedding matrix is used to find embedded words, average them, and save those embeddings as features in the Embeddings column.

Divide all data into train and test parts. We will train our model based on these features.
Take a look at the report:

	precision	recall	f1-score	support
negative	0.83	0.50	0.62	58
neutral	0.76	0.89	0.82	281
positive	0.67	0.56	0.61	146
accuracy			0.74	485
macro avg	0.75	0.65	0.69	485
weighted avg	0.74	0.74	0.73	485

We can see that all scores with the highest accuracy belong to the neutral class, and then we can see that negative scores are better.
It has an upper presicion.

B.

Since the neutral class has more data, it is clearly better trained and has a higher test accuracy.
Due to the neutral label on most of the data, the recall reaches close to 90%.
However, the recall of the rest of the classes is very low and far from neutral.

C.

	precision	recall	f1-score	support
negative	0.31	0.69	0.43	58
neutral	0.75	0.79	0.77	281
positive	0.56	0.22	0.32	146
accuracy			0.61	485
macro avg	0.54	0.57	0.50	485
weighted avg	0.64	0.61	0.59	485

Naive bayes outperforms logistic regression in general, since it considers data independence and estimates class probabilities based on maximum likelihood.
In imbalanced datasets, Naive Bayes is less sensitive to distribution of the minority class because of this assumption of independence.

In contrast, logistic regression uses a logistic function to estimate the class probabilities without assuming independence between the features. A logistic regression model may be more sensitive to imbalanced data, and may be biased towards the majority, resulting in poor performance for minorities.
In this model, neutral class data have a lower F1 than LR data.