Code organization

Inside of the .zip file, there are two folders, “unigrams” and “bigrams”, corresponding to problem1 and problem2 separately.

- unigrams
  - train.py
    command: python train.py -f 1 -r 1 -e 100 -s 100

    f stands for frequency_threshold
    r stands for learning_rate
    e stands for epoch_limit
    s stands for batch_size

    save trained model to “model.txt”

  - develop.py
    command: python develop.py
    Results of different hyper parameter combinations are saved in “develop_result.txt”

  - test.py
    command: python test.py
    load the model from “model.txt” saved by train.py

- bigrams
  - train_bigram.py
    command: python train_bigram.py -f 1 -r 1 -e 100 -s 100

    save trained model to “model_bigram.txt”

  - develop_bigram.py
    command: python develop_bigram.py
    Results of different hyper parameter combinations are saved in “develop_result_bigram.txt”

  - test_bigram.py
    command: python test_bigram.py
    load the model from “model_bigram.txt” saved by train_bigram.py

python version: python3

Problem 1

1. What were your best hyper parameters according to the analysis on the development partition?

   Tune among parameters:

   frequency_threshold = [1, 10, 100, 200, 300]
   learning_rate = [0.01, 0.1, 1, 10, 100]
   epoch_limit = [1, 50, 100, 200, 300]
   batch_size = [1, 100, 500, 800]

   Best hyper parameters:

   frequency_threshold = 1
   learning_rate = 1
   epoch_limit = 100
   batch_size = 100

   Best accuracy on the development partition = 97.76% (precision: 95.17%, recall: 88.46%)

2. What was the classification accuracy of your algorithm on the test partition, using the best hyper parameters from the development partition?

   Accuracy on the test partition = 97.76% (precision: 91.67%, recall: 91.03%)

Problem 2

1. What were your best hyper parameters according to the analysis on the development partition?

   Tune among parameters:

   frequency_threshold = [1, 10, 100, 200, 300]
   learning_rate = [0.01, 0.1, 1, 10, 100]
   epoch_limit = [1, 50, 100, 200, 300]
   batch_size = [1, 100, 500, 800, 1000]

   Best hyper parameters:

   frequency_threshold = 1
   learning_rate = 10
   epoch_limit = 50
   batch_size = 100

   Best accuracy on the development partition = 97.85% (precision: 96.48%, recall: 87.82%)

2. What was the classification accuracy of your algorithm on the test partition, using the best hyper parameters from the development partition?

Accuracy on the test partition = 97.58% (precision: 92.75%, recall: 88.28%)

Problem 3

How does classification accuracy change for the different filtering thresholds? Include in your report pointers to where the functionality to filter features is implemented.

This functionality is implemented in train_bigram.py and inside the function of TrainData_preprocess (datafile, frequency). For each word or bi-word (contiguous sequences of two words), there is a counter to count its frequency.

When the hyperparameter frequency is given, only the words and bi-words with frequency higher than the parameter are selected as features. This hyperparameter is chosen among [1, 10, 100, 200, 300], specified in develop_bigram.py.

From my observation, accuracy decreases with increasing filtering thresholds.

Here are several results on development partition when changing frequency_threshold, while maintaining other three parameters:

f	R	E	s	Accuracy	Precision	Recall
1	10	50	100	97.85%	96.48%	87.82%
10	10	50	100	96.68%	87.42%	89.10%
100	10	50	100	91.30%	64.97%	82.05%
200	10	50	100	88.34%	56.25%	75.00%
300	10	50	100	89.06%	60.90%	60.90%

f	R	e	s	Accuracy	Precision	Recall
1	1	200	500	97.67%	93.92%	89.10%
10	1	200	500	97.04%	90.73%	87.82%
100	1	200	500	93.45%	75.78%	78.21%
200	1	200	500	89.15%	59.46%	70.51%
300	1	200	500	88.79%	58.29%	69.87%

I think the reason is that higher filer threshold means less features are used, which is not enough in this case since the training dataset is small and lots of words in test data not even appear in training data.

Problem 4

What are the top 20 features associated with the spam class, and the top 20 features associated with the ham class? For this analysis, you may use the best model you obtained as a result of the first three problems. What do you observe in this analysis? Do you believe your algorithm is overfitting, or not?

Top 20 features associated with the spam class：

'txt' 'text' 'chat' 'service' '18' 'uk' 'www' '150p' 'sale' 'but no'

'xxx' 'sexy' 'sms' 'this is' 'what you' 'free' 'â£1' 'mobile' 'lucy' 'reply'

Top 20 features associated with the ham class：

'nice' 'thanx' 'yes' 'anything' 'check' 'its' 'where' 'k' 'lor' 'your

call' 'da' 'that' 'me' 'i' 'he' 'then' 'yup' 'okie' 'my' 'ok'

I do not think my algorithm is overfitting because I tune the hyper parameters on development partition, and my accuracy on test partition and accuracy on development partition is very close.

Problem 5

Paper: Exploiting Domain Knowledge via Grouped Weight Sharing with Application to Text Categorization http://aclanthology.coli.uni-saarland.de/pdf/P/P17/P17-2024.pdf

1. What task is the paper addressing?

A fundamental advantage of neural models for NLP is their ability to learn representations from scratch. However, in practice this often means ignoring existing external linguistic resources, e.g., WordNet or domain specific ontologies such as the Unified Medical Language System (UMLS). This paper addresses how to exploit known relationships between words when training neural models for NLP tasks.

2. What is the approach implemented?

The authors use weight sharing as a flexible mechanism for incorporating prior knowledge into neural models.
They view the partial parameter sharing induced by feature hashing as a flexible mechanism for tying together network node weights that they believe to be similar a priori. In effect, this acts as a regularizer that constrains the model to learn weights that agree with the domain knowledge codified in external resources like ontologies.

3. What are the main results and how do they compare with other approaches in the same space?

Their aim is not necessarily to achieve state-of-art results on any given dataset, but rather to evaluate the proposed method for incorporating external linguistic resources into neural models via weight sharing. They have therefore compared to baselines that enable them to assess this.

They present the results on seven diverse classification tasks (three sentiment and four biomedical), which show that their method consistently improves performance over (1) baselines that fail to capitalize on domain knowledge, and (2) an approach that uses retrofitting (Faruqui et al., 2014) as a preprocessing step to encode domain knowledge prior to training.

Generally, this work improves text classification performance vs. model variants which do not exploit external resources and vs. an approach based on retrofitting prior to training.

This method improves performance compared to all relevant baselines (including an approach that also exploits external knowledge via retrofitting) in six of seven cases.

4. What are the limitations of the proposed work?

I think the limitation is inherent in the weight sharing method. It shares weight among related words, but would not able to distinguish weather the words are positively related or negatively related.