CS5740: Assignment 1

https://github.com/cornell-cs5740-21sp/a1--group-7

Xingyue Dai xd86 · Shuo Han sh2439 · Hangyu Lin hl2357

1 Introduction (5pt)

Provided with a starter repository , our task for this project is to process provided datasets to implement and train two classification models from scratch, the Perceptron model and the multi-layer perceptron(MLP) model.

For each dataset, we developed multiple features for data preprocessing. Running both models on both datasets using developed features, we achieved 81% prediction accuracy on the test data for Newsgroups with MLP and 79% accuracy of prediction on the test data for Proper Names with MLP.

2 Features (20pt)

We developed Character n-grams, Bag of words, n-grams and mixed n-grams features for both our models. For each case of the feature, we created a word bank that includes all grams of specific length occurred the dataset. We ranked the grams in the word bank by occurrences and set a limit size (k) for the word bank, where k is an empirical choice. We ignored any unknown word/char that's absent in our word bank.

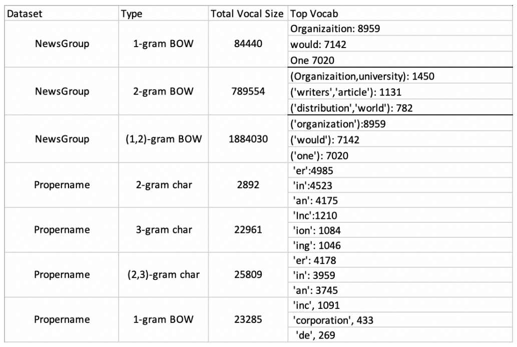
Character n-grams
We developed n-grams on character level for an input document $\bar{d}$. We experimented with several values of n in both classifiers.

Bag of words n-grams
To developed n-grams on word level, We firstly tokenized the document to a list of words. We took every adajacent n words and put them in the word bank. The word bank was built with descending order in terms of frequency. k is set to be the maximum size of the word bank.

Mixed n-grams
Mixed n-grams was built on top of n-grams features. We included a range of n-grams when building the word bank from the training data. For example, in bag of words (1, 2) - gram, we would include 1-gram (single word) and 2-gram (two adjacent words) for each index in a text. A count of occurrences of 1-gram and 2-gram was created at the same time. Similarly, we would keep the most frequent k mixed 1-gram and 2-gram words in the final word bank. We developed Mixed n-grams on both character level and word level.

3 Experimental Setup

Data (5pt) We used the provided two datasets: The Proper Names dataset which contains 28876 names, specifically 23121 names in the training dataset, 2893 names in development dataset and 2862 names in the test dataset, where each name belongs to one of five categorizations including person, place, movie, drug and company. The Newsgroup dataset contains 18846 newsgroup documents written on a variety of topics, specifically 9050 documents in the training dataset, 2263 documents in the development dataset and 7531 documents in the test dataset. Each document is of 20 Newsgroups including comp.graphics, comp.os.ms-windows.misc and etc. Below is a summarized statistics of our data which includes the vocabulary size and the top words for each experiment.



Dataset	Type	Total Vocal Size	Top Vocab
NewsGroup	1-gram BOW	84440	Organizaition: 8959
			would: 7142
			One 7020
NewsGroup	2-gram BOW	789554	(Organizaition,university): 1450
			('writers','article'): 1131
			('distribution','world'): 782
NewsGroup	(1,2)-gram BOW	1884030	('organization'):8959
			('would'): 7142
			('one'): 7020
Propername	2-gram char	2892	'er':4985
			'in':4523
			'an': 4175
Propername	3-gram char	22961	'Inc':1210
			'ion': 1084
			'ing': 1046
Propername	(2,3)-gram char	25809	'er': 4178
			'in': 3959
			'an': 3745
Propername	1-gram BOW	23285	'inc', 1091
			'corporation', 433
			'de', 269

Data Preprocessing (5pt) For data preprocessing, we created a function that takes one piece of data from the dataset at a time and convert it from a string to a list of strings. First, the function split the text by lines using the python built-in splitlines function, this is due to the fact that we noticed there are \r and \n in the newsgroup dataset. Then, we used regular expression to remove all the punctuation from the each line of that one piece of data. After that, for each line, we split the words by space, creating a list of words that contained some unwanted words. Next step is to turn each word into lower case and skip all the numeric strings or string with only 1 character. Lastly, we append words that are not in the stopwords set or the unwanted words set into the final output list for that piece of data.

Perceptron Implementation Details (5pt) In Perceptron model, we took 10% from the training data as validation data. The model was trained with varies learning rates. The model will terminate training if there are no errors in the training samples or the maximum number of iterations is reached. We saved the model based on the performance on the development data. Here we set the maximum iteration to be 30.

MLP Implementation Details (8pt) The MLP model has 2 hidden layers with 100 and 64 neurons respectively. After each hidden layer's linear transformation, we used ReLU as activation function, followed by a dropout with probability of 0.2. The dropout was turned off when evaluating. Similar to Perceptron, We took 5% from the training data as validation data. The model was trained for 100 epochs and evaluated on validation at each epoch. The model with best performance on validation dataset was saved and tested on the development data.

Lr	Batch	Hidden Layers	Epochs
0.0001	256	(100, 64)	100

Table 1: Hyperparameters of MLP

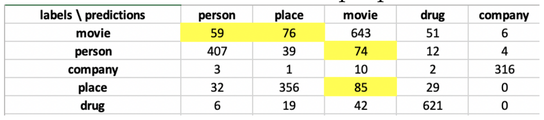
4 Results and Analysis

4.1 Proper Name Classification (12pt)

As we can see from the table, training both our perceptron and multi-layer perceptron models using the BOW feature didn't yield high accuracy. So we put our focus on the char-n gram feature. We experimented with multiple values of n including (2,3) and 3,4 with both our models. The best result we got from training the perceptron model is 79.54% accuracy of prediction on the development data using char (2,3) gram and the highest accuracy we got from training the the mlp model is 80.99% on the development data using char (2,3) gram as well. According to the leaderboard, our highst accuracies on the Leaderboard on the test dataset of Proper Names are 73.55% for Perceptron and 78.76% for mlp. **However, due to the mistakes of one of our team members who mistakenly submitted the wrong format of prediction files of the test dataset of Proper Names, our current test accuracies shown in the Leaderboard is 0.** Below is the confusion matrix for the propername dataset.

System	Accuracy
Development Results	
Perceptron (2,3)-gram	79.54
Perceptron 3-gram	76.79
Perceptron 4-gram	71.10
Perceptron BOW	70.60
MLP (2,3)-gram	80.99
MLP 3-gram	79.71
MLP 4-gram	75.5
MLP BOW	72.94
Test Results(highest in history)	
Perceptron	73.55
MLP	78.76

Table 2: Results of Proper Name Classification



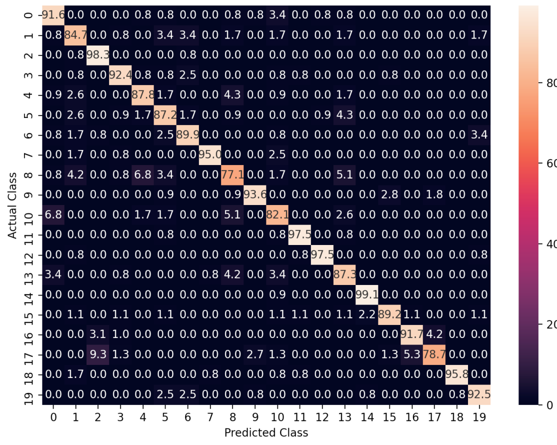
labels \ predictions	person	place	movie	drug	company
movie	59	76	643	51	6
person	407	39	74	12	4
company	3	1	10	2	316
place	32	356	85	29	0
drug	6	19	42	621	0

As we can see from the table above, the most frequently misclassified pairs are (movie, person)and (movie,place) which are marked in yellow. One possible explanation is that, since a movie title could be anything(random), it might contain features that are specificly held by person and place.

4.2 Newsgroup Classification (12pt)

As shown in the table, we trained both our perceptron and multi-layer perceptron models using a combination of different ranges of n-gram range BOW/character feature and word bank size. First, we experimented with n-gram BOW vs. character, the results for Perceptron model were

very close at around 84% empirically speaking. However, n-gram character did not perform as well as the n-gram BOW for the MLP model. The top occurred were ”ing”, ”tion”, etc. Intuitively, the character chunks provides no meaning or help the prediction much. It made more sense for us to continue our experiments with Bag of Words only, as the rest of the results from the table used BOW by default. Then, we adjusted the word bank size, we found that 25,000 to 30,000 is a good number. We also tried to skip all the headers of the emails, but the results were 5% worse than if we kept them.
Our best model which we used to predict the test dataset for Perceptron model was using mixed 1-2 grams Bag of Words, word bank size of 25,000, with binary vectors. It achieved a test prediction of 73.03% on the leaderboard. For the MLP model, the best setting was mixed 1-3 grams Bag of Word, word bank size of 30,000, with binary vectors. We were able to get 81.61% on the test dataset.
The confusion matrix shows that comp.sys.ibm.pc.hardware class performed the worse. It is often being mistaken as another comp class.



To conclude, our model did pretty well for the complexity of them. Newsgroup dataset is more difficult to preprocess than propername dataset. We were able to predict at a 81.61% accuracy rate with an unseen dataset. We were satisfied with our accomplishments.

4.3 Batching Benchmarking (4pt)

We tested the speed of our MLP model with different batch sizes on one Nvidia Tesla T4 GPU offered by Google Colab. We randomly selected 1000 samples from Newsgroup and trained the model for 100 epochs. Table 4 shows the wall-time seconds of different batch sizes:

System	Accuracy
Development Results	
Perceptron 3-5 gram character, Bank Size 25000	84.00
Perceptron, Bank Size 25000, Count	82.85
Perceptron, Bank Size 25000	84.66
MLP, 3-5 gram character, Bank Size 25000	82.00
MLP, Bank Size 20000	91.16
MLP 1-3 gram, Bank Size 20000	90.27
Test Results	
Perceptron Bank Size 25000	73.03
MLP 1-3 gram Bank Size 30000	81.61

Table 3: Results of Newsgroup Classification

Batch Size	Time (seconds)	Deviation
1	149.49	0.48
2	77.03	0.35
4	41.54	0.29
8	22.85	0.18
16	13.13	0.27
32	8.20	0.20
64	5.84	0.10
128	4.55	0.06
256	3.99	0.03
512	3.91	0.02

Table 4: Batching Benchmark

The training speed benefits from batching training samples, especially when the batch size is relatively small (less than 64).

5 Conclusion (4pt)

Based on the experiments we did for this project. We found that the bag of words feature works well for the Newsgroup classifier while the char n gram feature works well for the Proper Name classier,due to different formats and volumes of the two training datasets. We should also be careful when choosing appropriate parameters like number of iterations as it’s subtle to make trade-offs between overfitting and model performance.