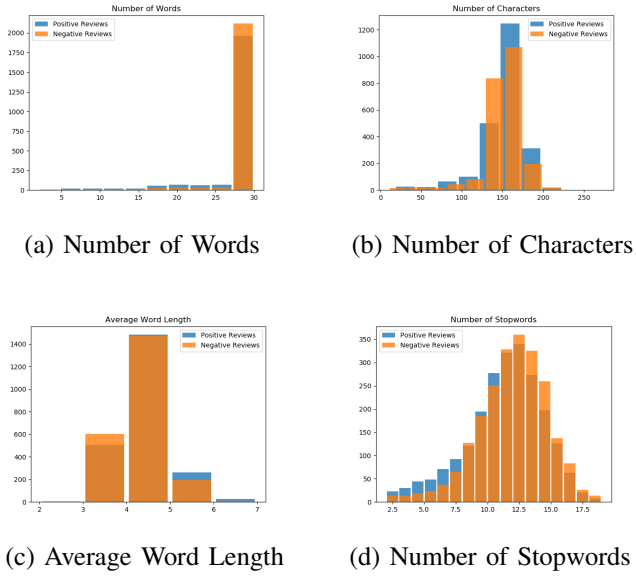
Programming Assignment 2: Semi-supervised Text Classification

Shihan Ran, A53313589
Fall 2019, CSE 256: Statistical NLP

I. Supervised Improve the Basic Classifier



(a) Number of Words (b) Number of Characters

(c) Average Word Length (d) Number of Stopwords

Fig. 1: Some basic statistical information about dataset distribution.

A. *Basis Analysis of Dataset*

After reading the data set, to perform different tasks on it, we should first perform some basic analysis on the dataset distribution. There are lots of features we can consider, like words, characters, stopwords, special characters, numerics, uppercase words and so on. Here due to limited time and resources, we only select a few indicators to illustrate the problem.

1) Number of Words: One of the most basic functions we can extract from text data is the number of words in each sentence. The intuition behind this is that generally speaking, the negative sentiments should contain a lesser amount of words than the positive ones.

From figure 2a, we can clearly see that due to the reason that sentences are cut off when the word length is longer than 30, there aren't many differences between the distribution of positive reviews and negative reviews.

2) Average Word Length: We also present another feature that will calculate the average word length of each sentence. This can also potentially help us in improving our model. The results are shown in figure 1c.

3) Number of Characters: Generally speaking, when solving a Natural language processing problem, the first thing we should do is preprocessing. The preprocessing can include various operations like spelling correction, tokenization and remove stopwords. Before doing preprocessing, calculating the number of stopwords may give us some extra information that we might have been losing before. The results are shown in figure 1d.

B. *Basic Pre-processing*

Before diving into text and feature extraction, we first clean the training data in order to obtain better features. I tried the following preprocessing steps: transform sentences into lower case, removing punctuations, removing stopwords, common words, and rare words. I also tried different tokenizations.

C. *Feature Engineering*

Now we have done all the basic preprocessing steps to clean our data. Now, we move on to extract features.

1) Term frequency: Term frequency (TF) is simply the ratio of the count of a word represented in a sentence to the length of the sentence.

2) Inverse Document frequency: The intuition behind inverse document frequency (IDF) is that a word is not of much use if it's occurred in all the sentences. The IDF of each word is the log of the ratio of the total number of sentences to the number of sentences in which that word is present.

3) Term frequency - Inverse Document frequency: TF-IDF is the multiplication of the TF and IDF which we calculated above. TF-IDF can penalize commonly occurring words. We implement this feature using *TfidfVectorizer* from *sklearn.feature_extraction.text*.

4) Bags of Words: Bag of Words (BoW) refers to the presentation of text which describes the presence of words within the text data. The intuition behind this is that two similar text fields will contain similar kind of words, and will have a similar bag of words. We implement this feature using *CountVectorizer* from *sklearn.feature_extraction.text*.

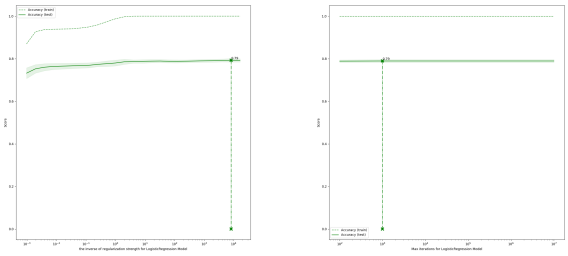
D. *Performances*

The training set contains 4582 sentences while the development set contains 458 sentences.

TABLE I: Supervised Performances

	BoW	**BoW+TFIDF**	**BoW+TFIDF+Tuned *LR***
Train	0.9997	0.9860	**1.0**
Dev	0.7795	0.7773	**0.7948**

We created a model pipeline and performed an exhaustive hyperparameter search using *GridSearchCV* from *sklearn.model_selection*. Some of the performances can be found in figure 2.



(a) Fine-tuned on regularization strength (b) Fine-tuned on max iterations

Fig. 2: Model performances on different hyperparameter settings.

TABLE II: HyperParameters

	Parameters
CountVectorizer	*max/min_df, ngram_range, stop_words*
TfidfVectorizer	*use_idf, smooth_idf, sublinear_tf*
LogisticRegression	*C, solver, max_iter, class_weight*

The best performances were acheived under

- *ngram_range=(1,3)*
- *use_idf=True*, *smooth_idf=False*
- *C=512*, *solver='saga'*, *max_iter=1000*
- DO NOT use any preprocess
- DO NOT remove stopwords

From our experiments, we know that features from count-based vectorization methods like BoW and TF-IDF have some disadvantages:

1) They describe sentences by word occurrences while completely ignoring the relative position information of the words in the sentence (despite using N-grams, which is only a quick fix).
2) TF-IDF word vectors are usually very high dimensional.
3) They are not able to capture semantics.

II. Semi-supervised Exploiting Unlabeled Data

A. *The approach to utilize unlabeled data*

My approach to semi-supervised learning is to iteratively expand the labeled data and retrain the classifier.

Algorithm 1: Semi-Supervised

Input: D_u as unlabeled data, D_l as labeled data, $ratio$, $threshold$

Output: Model parameters θ

$\hat{D}_l \leftarrow D_l, \hat{D}_u \leftarrow D_u$;

while $\hat{D}_u > (1 - ratio) * D_u$ **do**

  $\theta \leftarrow \text{Train}(\hat{D}_l)$;

  $D'_u \leftarrow \text{Predict}(\theta, \hat{D}_u)$;

  $\hat{D}_l \leftarrow \text{Expand}(\hat{D}_l, D'_u, threshold)$;

  $\hat{D}_u \leftarrow \text{Remove}(D'_u, threshold)$;

return θ;

The first thing is to decide which labels to include in $\hat{D}_l$ in every iteration. We choose to include the most confident predictions (i.e., the

probability of these predictions should be bigger than the $threshold$).

The second thing is to decide the stopping criterion. Instead of choosing a fixed number of iterations, we choose to stop when a certain $ratio$ of D_u has been added into the labeled training data $\hat{D}_l$.

In order to make our performances better, We tried different combinations of $ratio$ and $threshold$.

B. *The effect of size of labeled data*

Usually, larger datasets can help us avoid overfitting. If our model has many parameters and it isn't fed enough data, then it will tend to memorize the training set and perform poorly on the dev/test set. To avoid this phenomenon, except for using special training techniques like regularization, we can also use a semi-supervised method. By using semi-supervised learning, it is possible to combine the advantages of working with a smaller labeled dataset to guide the learning process and a larger unlabeled dataset to increase the generalizability of the solution.

In order to make it possible for semi-supervised learning methods to make the most of the labeled and unlabeled data, some assumptions are needed for the underlying structure of data space. The labeled and unlabeled data points should share the same data space. If the distributions are hugely different for those two data, then adding labeled data may and confuse our model and thus decrease our performances.

C. *Features and changes*

There should exist an overlapping between the unlabeled data and labeled data. When adding those unlabeled data into our training data, we will augment those overlapped words. Whether using BoW or TF-IDF, the features of overlapped words will change enormously, which will further lead to the result that their weights may change significantly.

Meanwhile, for those not overlapped words, while their features may remain the same, it can be regarded as their weights decreases.

D. *Performances*

We can refer to figure 3 to see the performances of $Ratio = 0.7$ on different thresholds. Overall speaking, compared with supervised learning, the performances on development set is decreasing as we add more labeled data to our training sets.

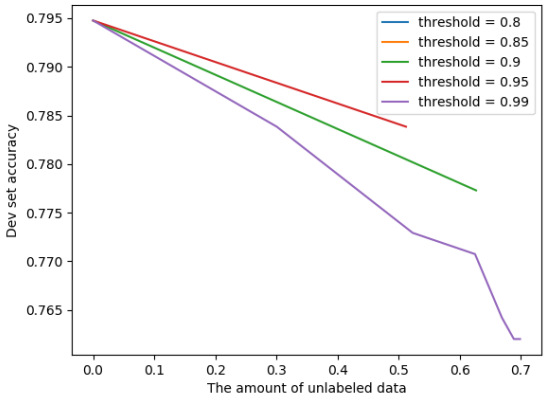


Fig. 3: Performances of Ratio=0.7 on different thresholds

The first thing we should notice is our model can achieve a high accuracy of about 0.8 on the development set and 1.0 on the training set. Thus, our threshold should be bigger than 0.8.

What we should know is that the unlabeled data is so noisy and it even contains some Chinese and Japanese words. If not selected precisely, adding those data as labeled data may confuse your model and decrease the accuracy.

When the threshold is low, each iteration we will have more confident data and thus can add more labeled data into our training data. So after two or three iterations then we will achieve the stopping criterion since the amount of unlabeled data is larger than our ratio. However, a lower threshold means you are less confident about the precision of your prediction. As the threshold increases, each iteration will get a smaller amount of confident data. Thus, we will need more iterations. The higher threshold should mean the prediction is more accurate. There is a tradeoff between the precision and the number of iterations.

E. *Error Analysis on semi-supervised classifier*

I sampled the following sentences and their respective labels generated by supervised learning and semi-supervised learning.

- *"This is a serious matter that needs to be addressed.. I've had my fair share of chipotle, and this one by far is the most stingiest.. Us Fat ass American's"*. Supervised Model labeL: Negative. Semi-Supervised Model Label: Positive. Ground Truth: Negative.
- *'Went there for the first time last night to celebrate a friends birthday. OMG! The porkchops were unbelievable. Met the owner, Eddie. Good friends, great food'* Supervised Model labeL: Negative. Semi-Supervised Model Label: Positive. Ground Truth: Positive.

TABLE III: Statistical Information

	Train	Dev	Unlabeled
Total	4582	458	91524
Positive	2291	229	-
Negative	2291	229	-

TABLE IV: Labels for Unlabeled data

	Supervised Model	Semi-Supervised Model
Total	91524	91524
Positive	44202	43325
Negative	47322	48199

We can see from Table III that the original dataset is pretty balanced. It contains the same number of positive samples and negative samples.

However, according to Table IV, the distribution of our predicted label isn't like the original one. The semi-supervised model tends to predict more negative labels than the supervised model. There are 4331 different labels out of 91524 lines in the prediction from our two models.

III. Semi-supervised Designing better Features

A. *Define better features*

Our goal is to utilize the unlabeled corpus to learn something about the word semantics and use it to identify the words that are likely to indicate the same sentiment.

We choose word2vec. Word2vec is a model that embeds words in a lower-dimensional vector space using a shallow neural network. The result is a set of word-vectors where vectors close together in vector space have similar meanings based on context, and word-vectors distant to each other have differing meanings. There are two versions of this model based on skip-gram and continuous-bag-of-words.

B. *Performances*

We use the word embedding trained on our unlabeled text corpus as our features. Since our unlabeled data isn't big enough, we do a hyperparameter tuning on the dimension of word embedding. The results are shown in Table V. Bigger size values require more training data and training time, but can lead to a better and more accurate model.

TABLE V: Performances on different dimensions of the trained word embedding

	50	100	150	200
Train	0.643	0.655	0.657	0.658
Dev	0.659	**0.670**	0.668	0.666

Overall, the performances of only using word embedding features is worse than our previous supervised model.

IV. Kaggle

User name: shihanran
Display name: Shihan Ran
Email: sran@ucsd.edu

References

[1] Bag-of-words model on Wikipedia
[2] Tokenization on Wikipedia
[3] TF-IDF on Wikipedia
[4] Section 4.2. Feature extraction, scikit-learn User Guide
[5] sckit-learn Feature Extraction API
[6] Working With Text Data, scikit-learn Tutorial
[7] CountVectorizer scikit-learn API
[8] TfidfVectorizer scikit-learn API
[9] HashingVectorizer scikit-learn API