Programming assignment 2: Semi-supervised Text Classification

Gitika Meher
Department of Electrical and Computer Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
gkarumur@eng.ucsd.edu

1 Supervised : Performance comparison with baseline

1.1 Baseline Performance

Using the basic Count-Vectorizer, accuracy on train data, dev data and test data was recorded.

Data-set	Train	Dev	Test
Accuracy	0.9821	0.7773	0.77721

1.2 Added Improvements

1.2.1 Use of TF-IDF Vectorizer

Count-Vectorizer is making the assumption that the more a word appears in a text, the more it is representative of its meaning. There are some biases attached with only looking at how many times a word occurs in a text. In particular, the longer the text, the higher it's word counts will be. To fix this issue, I used the Term Frequency (TF) instead of word counts and divided the number of occurrences by the sequence length. I also down-scaled these frequencies so that words that occur all the time have lower values. This down-scaling factor is the Inverse Document Frequency (IDF) and is equal to the logarithm of the inverse word document frequency.

$$\text{TF(word, text)} = \frac{\text{number of times the word occurs in the text}}{\text{number of words in the text}}$$

$$\text{IDF(word)} = \textbf{log}\left[\frac{\text{number of texts}}{\text{number of texts where the word occurs}}\right]$$

$$\text{TF-IDF(word, text)} = \text{TF(word, text)} \times \text{IDF(word)}$$

1.2.2 Removing very common words

Stop Words: A stop word is a commonly used word that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. Treating these type of words as feature words would result in poor performance in text classification. These words can be directly ignored from the corpus in order to obtain a better performance. Stop-Words provided to the vectorizer are:

1.2.3 Use of n-grams instead of a single word as a feature

Features as the combination of words provides better significance rather than considering single words as features. Combination of N words together with N as a hyperparamter was experimented

with. Considering Part of Speech tags combined with with words/n-grams will give an extra set of feature space which increase the performance. By using bigrams, we can discriminate between 'good' and 'not good'.

1.2.4 min-df and max-df

max_df is used for removing terms that appear too frequently, also known as "corpus-specific stop words". min_df is used for removing terms that appear too infrequently. I used min_df of 2 and max_df of 10000 to ignore all words appearing only in 2 or less reviews and ignore all the ones appearing in more than 1000 reviews.

1.2.5 Results

The table below shows the results obtained with the above described methods. Using a TF-IRF vectorizer with bigram and the stop-words being: {'and', 'a', 'the', 'this', 'that', 'an', 'there', 'here', 'those', 'am', 'it','me','with','they','which'} obtained the following results.

Data-set	Train	Dev	Test
Accuracy	0.9611	0.8100	0.79189

Although the accuracy of the training dataset went down, we see an accuracy improvement of about 2% in the development set when compared to the baseline. A similar improvement in the test accuracy is observed.

1.3 Hyper-parameters

1.3.1 n-gram

Bi-gram models worked the best as indicated by the table below. Most of the information that is necessary for this particular application of review classification can be obtained by using bi-grams. Unigrams cannot distinguish between "good" and "not good" and trigrams overfit to the training data.

n-gram	Unigram	Bigram	Trigram
Train Accuracy	0.874	0.898	0.901
Dev Accuracy	0.766	0.7794	0.775

1.3.2 min_df and max_df

The best dev-accuracy is observed with min_df of 2 and max_df 10000. min_df is used for removing terms that appear too infrequently.max_df is used for removing terms that appear too frequently. As we increase the value of min_df, we are ignoring more number of words which could be more relevant. By increasing the max_df, we ignore fewer number of words. So, we observe the best performance when min_df is equal to 2 and max_df, 10000.

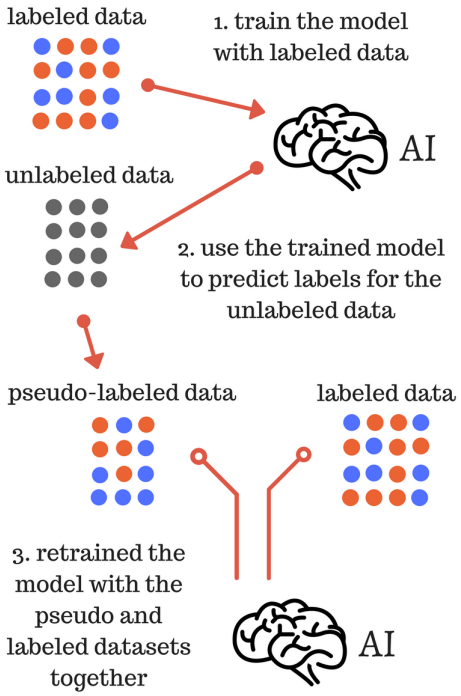
min_df	2	5	20
Train Accuracy	0.928	0.898	0.848
Dev Accuracy	0.783	0.7794	0.766

max_df	100	1000	10000
Train Accuracy	0.95	0.928	0.924
Dev Accuracy	0.772	0.784	0.788

2 Semi-supervised

2.1 Methodology

First, I trained the model on labeled data, then used the trained model to predict labels on the unlabeled data, thus creating pseudo-labels. Later, I combined the labeled data and the newly pseudo-labeled data in a new dataset that is used to train the data. I ran a loop for repeating this process and keeping track of the dev data accuracy. This loop stops whenever the dev data accuracy starts to decrease. The hyper-parameters for this process is the amount of unlabeled data that is added to the labeled data. I only added the unlabeled data to the labeled data that has been classified with a certain confidence level or more.



2.2 Results

Percent of Unlabeled data	25	50	75	100
Dev Accuracy	0.7729	0.7794	0.766	0.757
Test Accuracy	0.7927	0.7912	0.787	0.784

2.3 Hyperparameters

2.3.1 Percentage of Unlabeled data used

Using 25% of the Unlabeled data gave the best results on the dev set. As I increased this percentage, I see that the dev data accuaracy started to fall. However, the test accuracy did not change by much. This is probably because the mistakes in the predictions of unlabeled data are emphasized as I increased the size of this dataset used in training.

Since, the size of unlabeled dataset is much higher than that of training set, the number of bigrams obtained is very high. So, there would be inconsistencies of the bigrams observed with the labels. This is one of the reasons why increasing the unlabeled data for training doesn't improve accuracy. For example, a certain bigram associated with a positive label can also be associated with the negative

one. Or, non-influential bigrams could be associated with the labels which reinforce when trained again.

2.3.2 Confidence level

I ran experiments with using confidence levels as 50% , 75% and 90%. This hyperparameter indicates which pseudo labels of the unlabeled dataset are added to the given training dataset. I observe best results with using a confidence of 50%. That is all predictions are included in the training the model. This is because of the

Confidence level	50	70	80	90
Dev Accuracy	0.7642	0.760	0.7729	0.767
Test Accuracy	0.7927	0.790	0.789	0.787

2.4 Discussion

When I printed out the best 8 words connoting to positive label and the negative label, these are the results. We notice that words like " was not", "do not","you can" have been identified by the

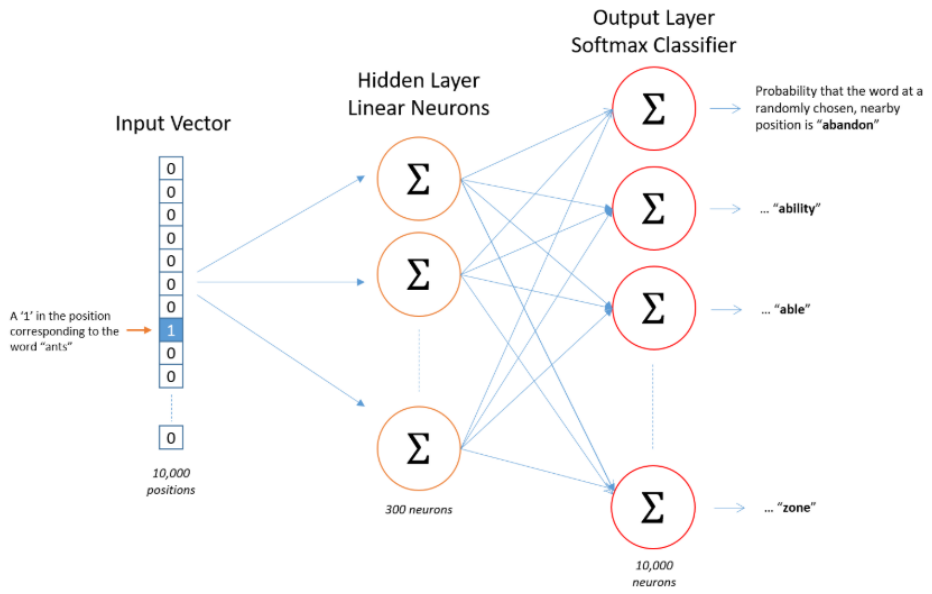
```
---------------------
Top k=8
---------------------
you can
is amazing
love place
was great
excellent
is great
awesome
delicious
---------------------
Bottom k=8
---------------------
worst
horrible
bad
was not
terrible
rude
disappointed
do not
```

semi-supervised model as top features. I notice that these mistakes are being reinforced when I repeat training the model with a parts of unlabeled data. This effect is more noticeable when I increase the percentage of unlabeled data used to train. Hence, I see the best performance with only 25% of the unlabeled data with 50% confidence used for training.

The results obtained after using this method of semi-supervised learning yielded better results than the improved baseline discussed in the previous section. Since, the size of unlabeled dataset is much higher than that of training set, the number of bigrams obtained is very high. So, there would be inconsistencies of the bigrams observed with the labels and non-influential bigrams could be associated with the labels which reinforce when trained again as we saw with the top 8 and bottom 8 words associated.

3 Semi-supervised: Designing better features

I used the word2vec skip-gram representation to create word-embeddings for better features. This model is trained on all the reviews which are part of the train data, dev data and unlabeled data so that the clusterings can be formed by reviewing all the sentences.



3.1 Word2Vec

The underlying assumption of Word2Vec is that two words sharing similar contexts also share a similar meaning and consequently a similar vector representation from the model. For instance: "dog", "puppy" and "pup" are often used in similar situations, with similar surrounding words like "good", "fluffy" or "cute", and according to Word2Vec they will therefore share a similar vector representation. From this assumption, Word2Vec can be used to find out the relations between words in a dataset, compute the similarity between them, or use the vector representation of those words as input for other applications such as text classification or clustering.

3.2 Intuition

The goal of the skip-gram neural network model is to learn the weights of the hidden layer which are the "word vectors" that we're trying to learn. Given a specific word in the middle of a sentence (the input word), the network is going to tell us the probability for every word in our vocabulary of being the "nearby word" that we chose.

The output probabilities are going to relate to how likely it is find each vocabulary word nearby our input word. For example, if we gave the trained network the input word "Soviet", the output probabilities are going to be much higher for words like "Union" and "Russia" than for unrelated words like "watermelon" and "kangaroo".

The neural network is trained to do this by taking word pairs found in our training documents.

3.3 Procedure

I used the Gensim's implementation of word2vec. I first cleaned the input text to strip off of punctuations and fed them as a list of list of words to the word2vec model. I initialized it to automatically detect common phrases (bigrams) from a list of sentences as we saw in the previous sections that bigrams gave the best performance and with a size of 300.
Given a corpus the model loops on the words of each sentence and tries to use the current word to predict its neighbors which acts as its context. This is the Skip-Gram implementation of word2vec.

3.3.1 Training

After training with the word2vec model, each word is represented as a vector of size 300. I took the average of all the vectors of the words in a review and trained these vectors with the labels provided for the training data.

3.4 Performance

Synonymous words for 'good' as identified by the model are 'great', 'decent', 'delicious', 'excellent', 'tasty', 'nice', 'amazing'.
Synonymous words for 'bad' as identified by the model are 'terrible', 'horrible', 'but', 'poor', 'awful'.
Below are the results obtained after training with the embeddings obtained after training the word2vec model which were later used to train a logistic regression unit.

Data-set	Train	Dev	Test
Accuracy	0.8227	0.8013	0.7945

Below is the table which shows a complete comparison of all the models used.

Method	Improv. Baseline	Semi-Sup	Word2vec
Dev Accuracy	0.798	0.8027	0.8100
Test Accuracy	0.7919	0.7927	0.7945