# Semi-Supervised Text Classification

*Statistical Natural Language Processing - PA2*

**Saideep Reddy Pakkeer**

University of California San Diego

29.04.2019

# INTRODUCTION

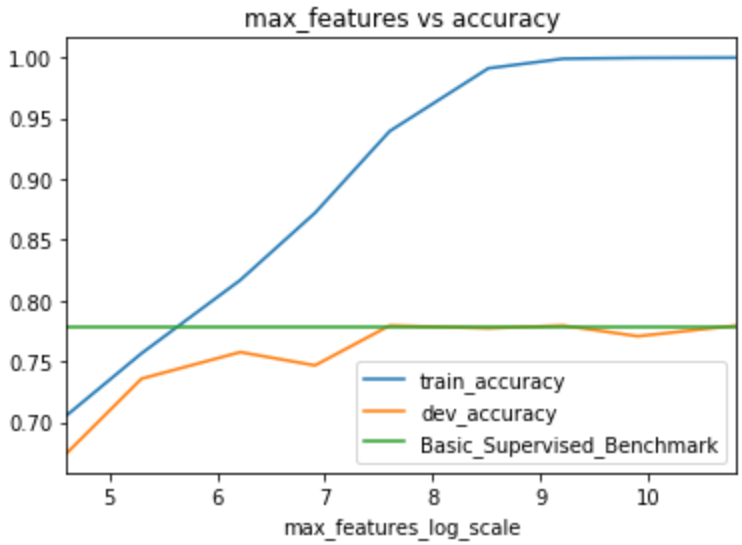
We are going to build a text classification model to identify the sentiment in the sentences. We have 3 sets of data: Train data of 4,582 sentences with a label attached (POSITIVE/NEGATIVE), development data (dev data) with 458 sentences (with labels) and 91,524 sentences of unlabeled data.

## SUPERVISED: Improving the Basic Classifier

Our task here is to improve the classifier by trying different things like hyper parameter tuning, building extra features using only the available data

## COUNTVECTORIZER (Strategy 1)

To start with, I experimented with the given starter code and see how the number of features affected the performance on dev set. Below is the graph illustrating the how the accuracy is affected with the number of features. Only other hyper-parameters I changed here is used n_gram (1,2 & 3)




## TF-IDF (Strategy 2)

To start with, I used the TF-IDF from sklearn to create the features for the text classification. Most of the analysis below is based on tf-idf features which is:

"Term frequency": $tf(t,d)$ = number of times the term *t* appears in the document *d*

e.g. tf("Taylor Swift", that news article) = 3

"Inverse document frequency": $idf(t,D) = log\frac{N}{|\{d \in D : t \in d\}|}$

term (e.g. "Taylor Swift") — set of documents

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

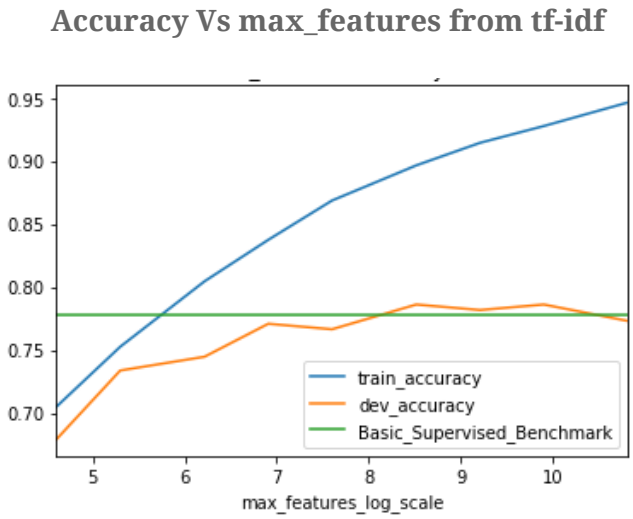
There are several hyper-parameters to vary in this setting to evaluate the performance on dev set. For this exercise, I have experimented with max_features (which is the maximum features from the tf-idf vector created ignoring the ones with least term-frequency), the regularization of logit model, n-gram range (unigram/bigram/trigram), stopwords (in English). After few experiments, I have decided to use all unigram,

bigram & trigrams without using stopwords as they seemed to give best performance. As we have very small amount of data, I haven't removed any words (including stopwords).

## Hyperparameter tuning

Below is a graph showing the performance of the model on train and dev set. The greenline is for benchmark provided in the starter code.

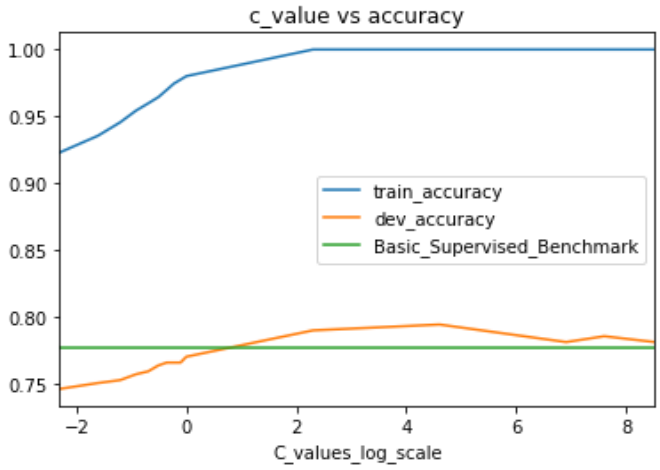
### Tuning max_features of TF-IDF:




As you can see the training accuracy keeps increasing with more feature being added but after a point the dev set performance doesn't increase. It is going to over fit but we can keep adding the features and tune the regularization parameter to compensate which is shown below

### Tuning regularization:

Next we tune the regularization parameter with all the features from tf-idf. Below is a plot of the accuracy vs the C value (regularization parameter in logistic regression). Lower the C means stronger the regularization. The accuracy on train obviously increases (and reaches 100%) when we don't regularize at all. But the dev accuracy doesn't increase. We can select that hyper-parameter for our final model



## FINAL MODEL FOR SUPERVISED

For our final model, we scale the term-frequency in log scale (sublinear_tf = T) which converts tf to 1+ log(tf), word_tokenizer instead of default tokenization in tf-idf and adding more features using doc2vec implementation and then finally using the best hyperparameters for regularization we get the following accuracy on dev set

### TF-IDF final

Accuracy on dev set**: 0.82751**
Confusion Matrix: **[[190 39] [ 40 189]]** with almost equal false positives and false negatives.
After submitting this model for Kaggle the accuracy on Kaggle: **0.79906**

The reason for this performance is I let all the features to be used in the model and let the regularization determine the best features for the model. This is obviously not a great model because the low amount of train data (4500 points) used to test data of ~91000 points and also it has a very high train accuracy (almost 100%) hence quite overfitted.

### *Top Features of our model:*

As we are using tf-idf we can see what are the unigrams/bigrams that are most predictive for our logit model. Below are features with highest coefficients for logistic regression

Features with the most positive coefficients: **great, excellent, amazing, delicious, awesome, love, friendly, spot, city, this is, but my, best, which was, the best, fantastic, good, yum, true, loved, perfect**

Features with the most negative coefficients: **disappointing, would, ok, to be, disappointed, was, but, called, asked, to, star, no, went, the worst, rude, terrible, bad, worst, horrible, not**

## SEMI-SUPERVISED: Exploiting Unlabeled Data

In the previous section, we haven't utilized the unlabeled data to build the classifier. Now, we devise a semi-supervised approach to design the classifier. We take different amounts of unlabeled data score using the existing classifier to make accurate predictions.

### Scheme used:

For semi-supervised learning, we take the most confident scores from the top as well as bottom and add them to our existing data and train the model again. For example, the way we add 100 points from unlabeled dataset is take 50 points with highest confidence for positive sentiment and 50 points with negative sentiment and add them to training data.

As a stopping criteria, we decide to stop the algorithm when the accuracy on dev set doesn't increase significantly (or keeps increasing). So, when the dev accuracy stagnates we stop adding new points from unlabeled dataset to train data.

### Accuracy (for 20000 features)

Here is the table how the accuracy varied when we fixed the number of features (20k) and used the above scheme to train the model. Later we don't limit the number of features and let regularization take care of it

| | Accuracy (min-df=5, sublinear_tf=False) | | Accuracy (min-df=4, sublinear_tf=True) | |
|---|---|---|---|---|
| Data points from Unlabeled data set | Train | Dev | Train | Dev |
| 100 | 0.9130 | 0.7947 | 1 | 0.7838 |
| 200 | 0.9150 | 0.7947 | 1 | 0.7816 |
| 400 | 0.9185 | 0.7969 | 1 | 0.7860 |
| 1000 | 0.9267 | 0.7947 | 1 | 0.7816 |
| 2000 | 0.9392 | 0.7925 | 1 | 0.7816 |
| 4000 | 0.9524 | 0.7947 | 1 | 0.7794 |
| 10000 | 0.9714 | 0.7903 | 1 | 0.7991 |

I decided to terminate after 10,000 unlabeled data points to the training data in case of using fixed number features. This is the stopping criteria as the accuracy drops after that point (3rd column). As you see, the accuracy on the dev set (last column in the above table) increases slightly as we use more and more unlabeled data for training but it stagnates after that.

### If we don't restrict the number of features and use all vocabulary for tf-idf:

As a next experiment, I tried no restriction to number of features and two schemes:

1. Each iteration, I add the most confident predictions from unlabeled data as before
2. Each iteration, I add a fixed number of unlabeled data and score the unlabeled data and then score the unlabeled data again to get the confident predictions.

In the second scheme, if your model is not performing well, it keeps getting worse as each time you are taking new prediction probabilities from that iterations model. Below (in the last column) you see exactly that happening.

| | Accuracy (confident predictions based on train data) | | Accuracy (confident predictions based on train+unlabeled data trained iteratively) | |
|---|---|---|---|---|
| Data points from Unlabeled data set | Train | Dev | Train | Dev |
| 100 | 1 | 0.8296 | 1 | 0.8296 |
| 1000 | 1 | 0.8275 | 1 | 0.8275 |
| 2000 | 1 | 0.8296 | 1 | 0.8253 |
| 4000 | 1 | 0.8231 | 1 | 0.8165 |
| 6000 | 1 | 0.8209 | 1 | 0.8122 |
| 10000 | 1 | 0.8165 | 1 | 0.8078 |
| 20000 | 1 | 0.8209 | 1 | 0.7925 |

## Comparison of supervised and semi-supervised classifiers:

Weights of features before(supervised) and after(semi-supervised):

| Feature (unigram/bigram/trigram) | Weight before(sup) | Weight After(semi-sup) |
|---|---|---|
| perfect | 6.99 | 11.71 |
| love | 10.05 | 20.48 |
| great | 17.23 | 33.48 |
| not | -8.74 | -26.41 |
| horrible | -14.89 | -26.41 |
| bad | -8.42 | -15.29 |

The above words in semi-supervised model had significantly higher weights compared to supervised classifier

The final predictions submitted had almost been the same as supervised with marginal difference.

## Error Analysis:

This is the confusion matrix of our final model on the dev set:

[[190  39]  (189 true positives & 190 true negatives)
 [ 40 189]]

We'll analyze the false positives and negative below. Here are a few examples from our dev set where the model failed to predict the right sentiment. The sentiment shown below is the actual sentiment given in the data

1. **NEGATIVE** : I'll be straight up on here.  I am definitely a club rat, I love music, dancing, and hanging out w/ friends, to have a good time, etc.  The  (*This one seem tough to say that it's going to be negative*)
2. **POSITIVE** : Since 2007 we have sent our dogs to daycare. We have used Noda Bark and Board, Dogs All Day, City Dog, and MetroDog. The Barker Lounge is the only daycare
3. **NEGATIVE** : After being gone from the Las Vegas scene for a few years I was amazed at how many new celebrity owned restaurants had popped up. My boyfriend and I both
4. **NEGATIVE** : Vor einigen Wochen habe ich in diesem Restaurant in Ermangelung einer Qype Empfehlung in der Umgebung (die geöffnet hatte) hier zu Abend gegessen. Ich muss ehrlich sagen: Ich war maßlos  (*A different altogether in the corpus which will not have any words matching)*
5. **NEGATIVE** : Round Two! ding-doh?!I recently was at an event that featured the Rusty Pickle Truck. They were giving out samples of their pickles and a Pulled Pork BBQ Slider. The Pickles
6. **POSITIVE** : I came to The Olive Branch Bistro in Bruntsfield on Saturday at about 6pm with my mum. From the menu outside it looked pretty expensive with with some mains

The above examples I hand picked suggest that the model is not very bad as they seem tough to identify the sentiment in them. The last two definitely seem difficult to classify without the complete sentence

## SEMI-SUPERVISED: Designing Better Features

One approach that we can take is to take a low-dimensional embedding of words and cluster similar words together. In a new sentence we again take a low dimensional representation and find the similarity with any of the words in a cluster (say of positive sentiment) and make an indicator as a new feature. But for this we have to take embeddings of each word in a sentences and somehow aggregate them together into one feature of fixed length. It is a tough problem to combine these representations of individual words to a representation for a sentence.

Hence, here we take a simpler approach to design new features where we use the doc2vec and append them to the already engineered semi-supervised features. The main difference between doc2vec from word2vec implementation is it used paragraph-id (or sentence id here) in training the model [2].

Just compare with our initial model, I used only top 10,000 features from our logit model(based on the absolute value of coefficients) as base line and kept adding the doc2vec features to see if the performance improves.

| Number of features Baseline (without doc2vec) | | Doc2Vec included (5000 features from Doc2Vec) | |
|---|---|---|---|
| Num of features | accuracy | Num of features | accuracy |
| 5000 | 75.76% | 5000+5000 | 75.98% |
| 10000 | 79.47% | 10000+5000 | 79.65% |
| 20000 | 80.34% | 20000+5000 | 80.34% |

There is a slight improvement in the dev accuracy if we are using lesser number of features (which should be done at the end of the day because we don't use these many features when we don't have many data points)

## Kaggle

Display name: **duckface** ; Username: **saideep91**; email: **120070045@iitb.ac.in**

## REFERENCES


1. http://cseweb.ucsd.edu/classes/fa18/cse258-a/slides/lecture10.pdf
2. https://cs.stanford.edu/~quocle/paragraph_vector.pdf
3. https://radimrehurek.com/gensim/models/doc2vec.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer.transform