Performance of Text Classifiers on Complexed Word Identification

Diep (Emma) Vu

I. Introduction

In this report I will provide the results of the baselines I used for the Text Classifiers on the task of Complexed Word Identification. I also tested the performance of various models and found out Support Vector Machine (SVM) is the best Classifier on this task. I also run my model on the unlabeled test set to see the result and perform error analysis on the models.

II. Baselines

1. Simple Baseline

The first simple baseline for the classifier is to label all words as complex. Here is the results of the accuracy, precision, recall and f-score on the training and development data:

Simple baseline	**Accuracy**	**Precision**	**Recall**	**F-score**
Training	42.8250%	42.8250%	100%	59.9685%
Development	44%	44%	100%	61.1111%

As we can see here, with a simple baseline, the results are not promising as F-score is only around 60% and accuracy and precision is lower than 45%. However, the result of recall on both training and development is 100%, because we label all words as complex.

2. Word-Length Baseline

I also measured other baselines and this time I used the length of each word to predict its complexity. The longer the word, the more complex it is. I tried every threshold value between [1,10] and I found out that the **best threshold** is **7.** Here is the results of the accuracy, precision, recall and f-score of threshold 7 on the training and development data:

Word-length baseline	**Accuracy**	**Precision**	**Recall**	**F-score**
Training	68.9%	59.7021%	84.2382%	69.8789%
Development	70.5%	61.8367%	86.0795%	71.9715%

Using this baseline, the results are better in terms of f-score and accuracy (around 70%) compared to the simple baseline's results.

3. Word-Frequency Baseline

The final baseline I tried is on word frequency from *Google NGram* frequencies. The less frequent the word appears, the more complex it is. In this training dataset, the **minimum** frequency is **40** and the **maximum** frequency is **47376829651.** Since the range is too huge, I tried the range [1000000, 100000000] and used the granularity of 100000. I found out that the **best frequency threshold** is at **19900000** and the time it took to run all possible thresholds is 2.5433731079101562 seconds**.** Here are the results:

Word-frequency baseline	**Accuracy**	**Precision**	**Recall**	**F-score**
Training	64.4250%	55.7769%	81.7280%	66.3036%
Development	69.25%	60.7724%	84.9432%	70.8531%

In comparison to Word-Frequency thresholding, it seems that the Word-length thresholding model works better since all accuracy, precision, recall, f-score on both training and development data of Word-Length threshold are higher than that of Word-Frequency threshold.

It is also interesting to notice that in the *load_ngram_counts* function that is already given, only the first character of the token is lowercase to update the count. However, if there are duplicate case-sensitive words such as “Girls” and “girls”, then the count frequency might not be updated correctly. I suspect this small issue will somewhat affect the performance of the Word-Frequency threshold. If there’s not many duplicate words like that, it will not make a huge difference I suppose.

III. Classifiers

1. Naive Bayes and Logistic Regression

I evaluated several models on *word length* and *word frequency* as features. For all the features before training and testing, I have to normalize them by finding the mean and standard deviation to make sure that they are all in the common range to avoid dominance in magnitude. Here is the results for Naive Bayes and Logistic Regression on training and development data:

Naive Bayes model	**Accuracy**	**Precision**	**Recall**	**F-score**
Training	55.25%	48.8766%	97.7817%	65.1751%
Development	55.25%	49.5702%	98.2955%	65.9048%

Logistic Regression model	**Accuracy**	**Precision**	**Recall**	**F-score**
Training	74.0250%	71.7419%	64.9154%	68.1581%
Development	77%	76.0870%	69.6023%	72.7003%

Looking at the results, we can see that Logistic Regression generally performs better on the task, with higher accuracy, precision, f-score (over 70%) whereas Naive Bayes only have nearly perfect recall. One of the reasons perhaps because in Naive Bayes, the occurrence of a rare word can greatly affect the prediction, while Logistic Regression can handle them in a more controlled manner by giving them a small weight.

2. My Own Model

Besides Naive Bayes (NB) and Logistic Regression (LR), I trained other models on the task which are Decision Tree (DT), Random Forest (RF) and Support Vector Machine (SVM). Thus, I added some additional features to train on such as *number of syllables, number of WordNet synonyms (synsets)*. Here are the results on the training and development data:

Training data	**NB**	**LR**	**RF**	**DT**	**SVM**
Accuracy	62.1500%	74.1250%	98.9750%	98.9750%	74.4750%
Precision	53.2421%	71.8147%	99.0610%	99.4675%	69.7941%
Recall	95.3882%	65.1489%	98.5406%	98.1319%	71.2201%
F-score	68.3396%	68.3196%	98.8001%	98.7952%	**70.4999%**

Development data	**NB**	**LR**	**RF**	**DT**	**SVM**
Accuracy	62.15%	77.75%	75%	69.8750%	76.8750%
Precision	53.2421%	77.7070%	72.2222%	66.4688%	73.7892%
Recall	95.3882%	69.3182%	70.1705%	63.6364%	73.5795%
F-score	68.3396%	73.2733%	71.1816%	65.0218%	**73.6842%**

Of all the models, I found out that **SVM performed best** with an F-score greater than 70% compared to the rest. It's also interesting to see that Random Forest and Decision

Tree were **overfitting** in the training data. Perhaps the reason why is because the data is not big enough to train on. It is also because they build complex trees that fit the training data well but may not generalize well to new, unseen data. Potential steps to avoid overfitting would be fine tuning those models, which is beyond the scope of this project.

3. Error Analysis

I then performed error analysis on the best model which is Support Vector Machine with development data in order to find categories of words and patterns on which the model is working best on or making errors. I sampled 20 words and here are the results:

	Value	**Examples**
True Positive	259	['publicly', 'economics', 'coauthor', 'explosion', 'clattered', 'microbial', 'fatally', 'wetlands', 'courtesy', 'asteroid-linked', 'whistling', 'parachute', 'genetically', 'anti-terrorist', 'geologist', 'timetable', 'endangering', 'professors', 'electives', 'embody']
True Negative	356	['linked', 'cattle', 'troop', 'stands', 'uses', 'backed', 'noise', 'records', 'meat', 'war', 'attend', 'video', 'learns', 'child', 'square', 'corals', 'language', 'housing', 'entered', 'muddy']
False Positive	92	['library', 'searched', 'disconsolate', 'everything', 'consumption', 'democracy', 'unlikely', 'spontaneous', 'elections', 'scientists', 'citizen', 'existing', 'veterans', 'hillsides', 'five-time', 'bathroom', 'monotonously', 'offstage', 'implications', 'featured']
False Negative	93	['pawned', 'clash', 'defying', 'microbe', 'attended', 'grooms', 'rash', 'emptier', 'stripped', 'assist', 'gathered', 'beef', 'ousted', 'protest', 'attire', 'pylons', 'combine', 'trailing', 'assess', 'stiffen']

I suspect words with inflectional endings with “s”, “ed”, “ly” made the model perform poorly such as “implications”, “searched”, “unlikely”. It is also interesting to see that all the simple words that the model predicts correctly mostly have less than 2 syllables such as “cattle”, “troop”, “noise”.