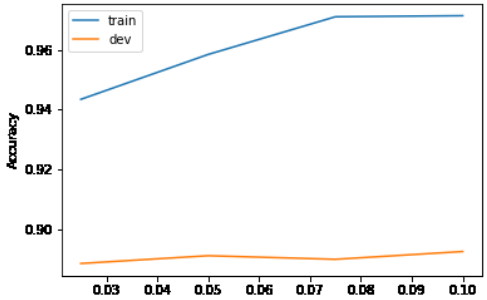
Bryan Sam
Karl Hickel
Marianna Carini
Caesar Phan

NLP Homework 2

Default Features

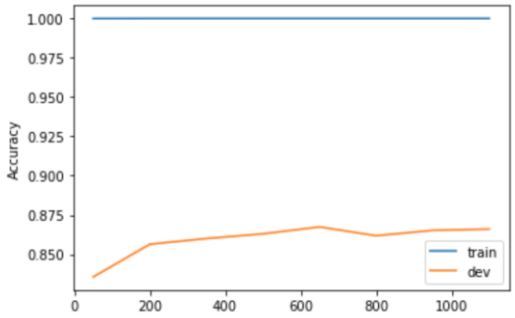
Overall our default features improved considerably in both training and testing accuracies. Our logistic regression did experience some level of overfitting, but not quite as extensive as the overfitting for our random forest. Our random forest model was considerably overfitting with dev values around 0.85 and training accuracies of 1.0. This was true for all of the iterations of our random forest model. Our logistic regression model experienced significantly less overfitting, but there was still some as observed by our chart.

Graph A



As Graph A shown above, we set the range of regularization weight C from 0.025 to 0.1 by increments of 0.025 for the logistic regression. Since C is the inverse of the regularization weight, a large C value leads to small penalty value and chances of the model being overfitting because of the small regularization strength. With the range we set, both training and testing accuracies increase starting from around 0.075 to 0.10. The regularization weight C of 0.10 gives the training and dev accuracy of 97.14% and 89.26% respectively.

Graph B



With Graph B shown above, we set the n_estimators from 50 to 1250 by increments of 50 for the random forest. The n_estimators parameter is the parameter where we decide how many trees will be used in our random forest model. In Graph B, the dev accuracy increases then slightly decreases. Our highest level of accuracy for our test set was 86.6% at 950 n_estimator. This of course was matched with our overfitted training model of 100%. Overall, this was a significant improvement from the previous models range.

Custom Features

For our custom features analysis, our initial approach was to compare the performance between classifiers ‘countvectorizer’ and ‘tfidf’. Since ‘countvectorizer’ yielded significantly higher overall accuracy, we then tried increasing the number of dictionaries to classify positive and negative words. On top of the inqtabs dictionary, we also included the following libraries::‘swn’ and ‘afin’. This increased our overall accuracy by roughly 3% for both our logistic and random forest models. We then attempted to clean the dataset, however, to our surprise, our overall accuracy actually decreased by nearly 4% on average. We added a feature that counts the number of sentences by looking at the punctuation marks at the end of each sentence. Including a feature called num_exclaimations to count the number of exclamation points. The additional features are listed which are affin_pos, affin_neg, neg_count, textblob_polarity, and textblob_subjectivity. These are counting the number of positive and negative words according to the affin library, number of negative words from inqtabs, swn, and affin, and polarity and subjectivity scores from TextBlob.

```
def features(self, review):
    return {
        # --------------------------------------------------------------------
        # 4 example features
        # TODO: Add your own here e.g. word_count, and pos_count
        #neg word count
        #afinn method

        'length': len(review),
        'num_sentences': len(re.findall(r'[\.|\?|!]', review)),
        'num_words': self.word_count(review),
        'pos_count': self.pos_count(review),

        #added features
        'num_exclaimations': review.count('!'),
        'afinn_pos': self.afinn_pos_count(review),
        'afinn_neg': self.afinn_neg_count(review),
        'neg_count': self.neg_count(review),
        'textblob_polarity': TextBlob(review).sentiment[0],
        'textblob_subjectivity': TextBlob(review).sentiment[1]

    }
```

We tuned the ‘countvectorizer’ function by updating the n_gram features to a range between 1 to 3 (i.e. (1,3)) with max features of 1000. On top of the creation of the features we also performed parameters for both logistic regression and random forest. For the random forest algorithm we ran a range for the number of n_estimators from 950 to 1500 with increments of 150. There was also the inclusion of other parameters including the max_features, min_samples_split and leafs. Additionally, we also experimented with min and max df (ranging from 0 to 100, and 0 up to 10000 respectively) and removing English stop words. Ultimately, the best parameters for the logistic regression model utilized the following parameters: CountVectorizer(stop_words={'english'}, min_df=3, max_df=5400, ngram_range=(2,2)). This model returned a ‘dev’ accuracy of 0.8812 in the logistic regression model and 0.84 in the random forest model.

Analysis

For our analysis of the models, Classifier 1 is the default logistic regression, Classifier 2 is the default random forest model, Classifier 3 is the custom logistic regression mentioned above, and Classifier 4 is the custom random forest model using the same parameters as the custom logistic regression model. Below are the evaluation metrics. We can see that the random forest models have the highest accuracy, precision, recall, and F1 scores. The default random forest model has the highest F1 score of the four models.

```
Classifier1 Metrics
Accuracy: 0.95564
Precision: 0.9551666267082235
Recall: 0.95616
F1: 0.9556630552112901

Classifier2 Metrics
Accuracy: 0.973
Precision: 0.969506868895418
Recall: 0.97672
F1: 0.9731000677479775

Classifier3 Metrics
Accuracy: 0.8074
Precision: 0.8115119578435347
Recall: 0.8008
F1: 0.8061203946043889

Classifier4 Metrics
Accuracy: 0.97056
Precision: 0.9702590342181004
Recall: 0.97088
F1: 0.9705694177863083
```

In the default logistic regression model, “excellent” is the positive word with highest weight and “waste” is the negative word with the highest weight. In our custom logistic regression classifier, the textblob polarity feature has the highest positive weight and the number of negative Afinn words has the highest negative weight. In our default random forest classifier, bad, worst, and great are the three words with the highest weight. In our custom random forest classifier, textblob polarity, the number of negative Afinn words, and the number of negative words from inqtabs and swn dictionaries are the three criteria with the highest weight, much like our custom logistic regression classifier.

From the positive reviews where only the custom classifier was correct, we can see that our custom features take into account more context than the default classifier. This proved useful when looking at reviews tackling opposing views. The ‘n_gram’ parameter allowed more combinations of context to be derived from these more substantive reviews. Our custom random forest classifier makes use of bigrams and the use of textblob where the default classifier is only using a bag of words. In the context of the review, the user uses words like “ridiculous” and “frustration” which without context can be classified as negative. Therefore our default classifier has a tougher time with reviews that have added context to words that would otherwise be potentially positive. Overall one of the major disadvantages of our custom features is that the overall runtime is very long. Some text preprocessing steps (e.g. lowercasing words) decreased our overall accuracy, this became more apparent after running our random forest model. In addition to that, the random forest model was heavily over-fitted. Adding regularization parameters considerably reduced this issue; however, it also considerably dropped the dev dataset’s accuracy. An advantage of our custom features is that we can add more libraries with words that have different words with varying sentiment scores. So if one dictionary can’t classify a word as being positive or negative, perhaps the next one will.

Statement of Collaboration

Throughout the duration of the assignment, our group met and communicated on multiple occasions to discuss methodologies to improve the accuracy of our models. Our communication was strictly between our group members.