DATA SCIENCE DECAL, FALL 2017

Project 2: Sentiment Analysis Bag of Words Model on IMDB reviews

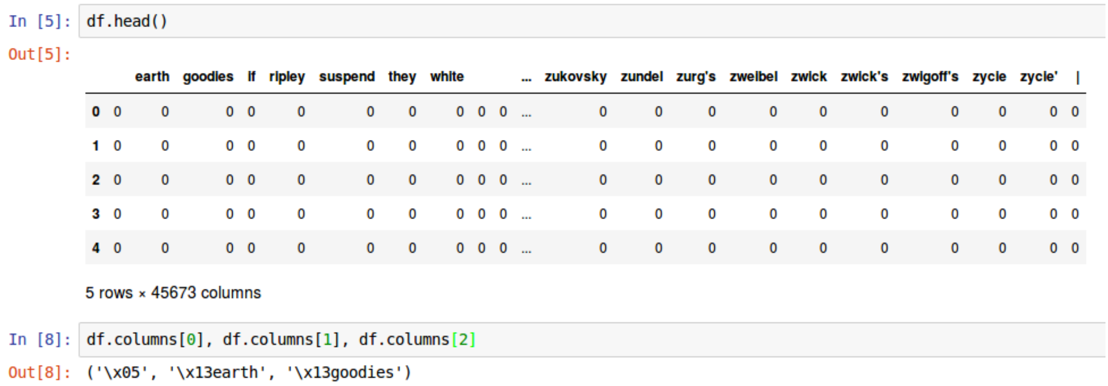
Brandon Hsieh, Dev Ojha, Eva Su

October 25, 2017

CONTENTS

1 Cleaning the data

The first step in our process was to look at our data, when it is unmodified.



```
In [5]: df.head()
Out[5]:
```

		earth	goodies	if	ripley	suspend	they	white			...	zukovsky	zundel	zurg's	zweibel	zwick	zwick's	zwigoff's	zycle	zycle'	|
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 45673 columns

```
In [8]: df.columns[0], df.columns[1], df.columns[2]
Out[8]: ('\x05', '\x13earth', '\x13goodies')
```

Something seems odd here, since the columns should be alphabetically sorted, and they aren't near the beginning. If you print the names of the first few columns, you see that they begin with bytes that aren't in the printable range. For example the ’
x05’ is a control byte corresponding to "ENQUIRY". The inclusion of that byte in the data is likely accident. The ’
x13’ is the control byte for "Device Control 3". This is also likely an accidental inclusion. These bytes are adding columns, which makes our data more sparse. Sparser data makes making conclusions harder. These mistake bytes are also very unlikely to appear in multiple messages, so they are not contributing to sentiment. So to make our data better, we decided to remove these unprintable bytes, and so ’
x05earth’ will count towards the occurences of earth in that message, and not towards ’
x05earth’. This converts our data from 45673 columns to 45643 columns. Thus 30 unhelpful columns are now useful data.

Now in the remaining data, we continued to investigate the columns. We found that words that started with "[", or words that ended in a "." were rendered as different words. Again these aren't usefully contributing to the data, so we added spaces to the punctuation marks before the word were split. Our thought process is that the fact that a word concludes a sentence does not provide any new information regarding its sentiment in the bag of words model, so it should not add new a column, but instead should count as the same word but without the punctuation mark. This made our dataset less sparse, which makes it more suitable for regressions / decision trees. It converted our data from 45643 columns to 45463 columns.

The punctuation marks which spaces are added around are:

["[", "]", "(", ")", "‘", ":", ".", ";", "", ""]

Finally when looking at the data, there are many words which end in an apostrophe, or apostrophe s, to indicate possession. We deemed that the possessive nature of a proper noun does not on its own indicate the sentiment of the word, moreso than the word itself did on its on. (I.e. there is no difference in sentiment between "bob" and "bob's") So as a result, we decided that if a word ends in an apostrophe or an apostrophe s, we removed the apostrophe / apostrophe s. This change converted our data from 45463 columns to 42549 columns.

Later on, while we were working on our later models, we took another look at our columns. We noticed that many of the columns were extremely sparse, and that they had words appearing in only one review. One example of this is "zurg". If we make our models split on words like these, or regress on words like these, we would be fitting our models to noise in the data. There isn't enough data for our models to be able to see how much that particular word indicates the positivity / negativity of a review. Additionally these words are extremely unlikely to appear in any word in the validation set / a potential set which we would like to determine the sentiment of. Thus we decided to remove all words that did not appear in at least 5 different reviews. This means that if a word did not appear in at least .25% of the reviews in our data, then we are going to remove it from our data frame. This is a reasonable thing to do, as it is very unlikely for a new review to contain such a word, and we did not have enough data for logistic regression to assign a meaningful weight to this word. Decision trees would not have taken these words into account to begin with, as they don't help split the data. Removing these words removed 30750 additional columns from our dataset. This leaves 11799 left in our dataset. Note that this step does not invalidate our previous steps in cleaning the data, as those bad columns are now contributing to the correct columns, which potentially saves them from being pruned in this step. Since we performed this step later in our process, we noticed significant speed increases, and we noticed an increase of one percentage point in the basic logistic regression, on average.

In total we have removed about 74% of the columns in the dataset in this cleaning process.

2 Feature Engineering

We thought that adding some additional data about the entries may be useful features in the later logistic regression and decision trees. This is known as feature engineering. Doing this can't hurt, as we are having the models decide which features are the best to use. So if these engineered columns don't help, then they won't be included in the models.

The engineered column we made were:

- File ID - This was not used in regressions, but just for debugging purports.
- Word count in the review - Since we removed some columns, we thought that the regressions may get some useful information from the number of words that are in the review in total. It turned out that these was not an indicative feature in the models.

3 Training/Validation Split

We performed a two-way split of and divided the data into separate partitions. We assumed that the data available is fairly representative of the totality of IMDB movie reviews.

We also use Grid Search with cross validation later on the training set, for Random Forests. We only used it on the training set, instead of the training set and validation set, even though it makes multiple training / validation splits to prevent smart overfitting. The reason we did this, is that the validation data could act as a test set in these scenarios when we are scoring, which gives us more meaningful scores at the end. The drawback is that our models are not as accurate as they could be at their best due to them being provided less data, however as a trade-off we gain more confidence in the truth of their predictions on unseen data, since we did not have a proper test set.

4 Logistic Regression

4.1 Basic Model

After partitioning the dataset into training and validation data, a logistic regression estimator was initialized to the default parameters determined by sklearn:

penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver= 'liblinear', max_iter=100.

With these parameters, we scored the model on the validation set. The accuracy remained between 80% and 85%. We believe that the accuracy varied depending on how the training/validation set was partitioned.

4.2 Tuning Hyperparameters

After getting an idea on the accuracy of the logistic regression model without tuning parameters, we decided to vary the regularization strength, through varying the parameter C– the inverse of regularization strength. Using grid search cross validation, the values for C that were attempted ranged from .001 to 50. And again, the resultant accuracy of our model scored on the validation set varied across separate trials, changing as the partition of training/validation set was changed. At times the optimal C was determined to be less than 1 and other times closer to 10. In the best case, tuning C resulted in a .003 improvement against leaving C at 1.0.

4.3 Backwards Stepwise Selection

An initial attempt was made to perform backwards feature elimination using the recursive feature elimination with cross-validation class through sklearn. However, with the model having more than 35,000+ features, it was determined that it would take too long to process on our own computers.

In an effort to get the backwards stepwise selection to run in a reasonable time, we reduced the number of words (columns) by dropping the columns that had appeared in less than five distinct documents. The reasoning for this is explained in the end of section one. Additionally, we can believe that removing those columns reduced overfitting, as although those rare words appeared in our training set, the likelihood of them appearing further would be slim and therefore not enhance the predictive ability of our model.

Ultimately, through manual feature elimination and backwards step wise selection we were able to reduce the number of features used by our model from 45673 features down to 9447 features without any decrease in the accuracy of the model on the validation set. From this, we believe we greatly reduced overfitting while maintaining the accuracy of our model.

4.4 Further parameter adjustment

Further parameter adjustment could be performed on adjusting the max iteration, the tolerance, the loss penalty, the number of features to keep in backwards stepwise selection, and the step size in backwards stepwise selection. However for our computers performing grid search to optimize all of the hyperparameters was too computationally intensive.

5 Decision Trees

5.1 Overfitting

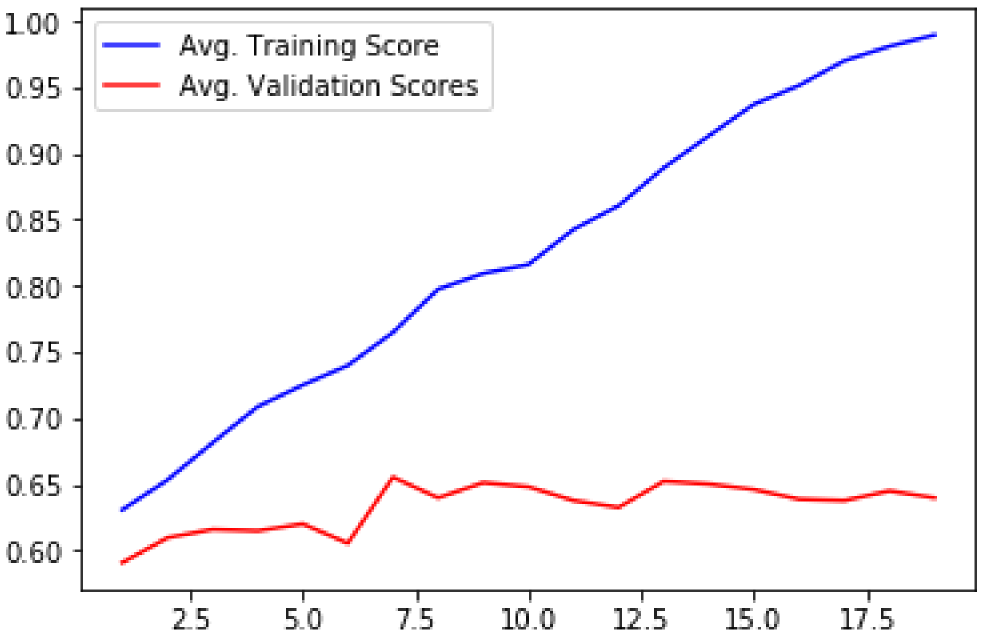
Decision trees are prone to overfitting since they can fit to the training set exactly. As a result they fit to the noise in the sample data, instead of the underlying trends. Often its best to prune the decision trees early to minimize the impact of this overfitting. (Or alternatively let the decision tree overfit, and then post-prune).

5.2 Finding the best parameters

First, we decided to make our criterion "entropy" because we discussed this in lecture, and we understood how this worked. We thought that using a model we understand is preferable to a potentially better model which we don't understand.

We decided to vary the maximum depth, as this directly controls to what degree our decision tree is underfitting / overfitting our data.
To decide the optimal depth, we wrote a quick script to check every depth value in a range 3 times, and average the different scores it got on the validation data. Each of these decision tree classifiers, though given the same training / validation data split, will produce different classifications, since there is inherently some randomness used in sklearn to create decision trees. Different random values are used each time. Thus this is a meaningful average.

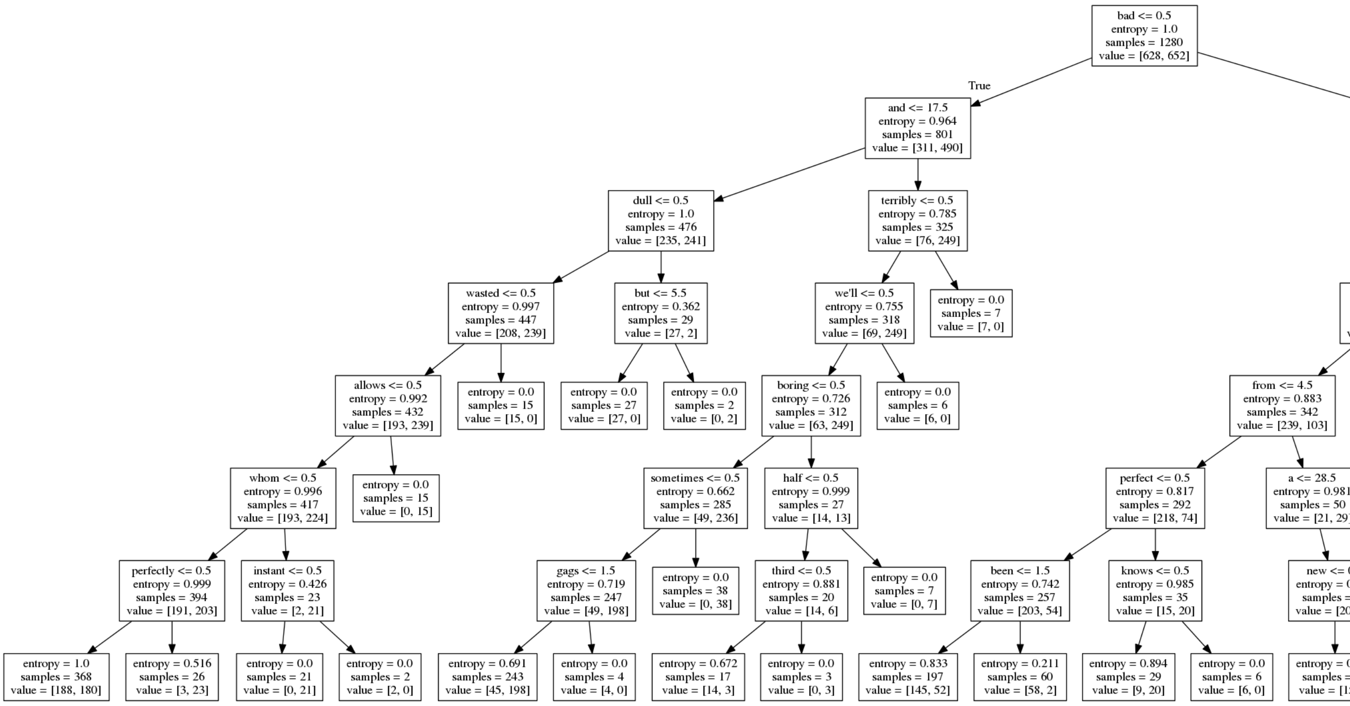
So we first created a graph of all the average training and validation scores for every 4th depth in the range 1 to 100. The graph showed that depths < 21 had higher validation scores. So then we average each maximum depth in the range 1 through 20 over 3 iterations, on a different training / validation data split, and got the following graph:



We ran this several times over different training vs test data splits, and the conclusion was always that the peak was around 6 to 8 for the maximum depth. Thus we used a maximum depth for 7 in our classifier. On average the accuracy of the decision tree on the validation was about 65% at this setting.

5.3 EXTRA ANALYSIS

First we decided to visualize the decision tree, and it came out as:



I have left it truncated to the right so that there is some semblance of readability for it on the page. Its far too wide to properly fit, and to be viewed correctly it has to be zoomed in on, inside of an image viewer. We created this image by exporting the decision tree to graphviz, and then creating an image from graphviz. (Code is in the ipynb)

Then we wanted to look at which words were most indicative of positivity and negativity in the decision tree. The 10 most indicative words, with their importance to their left are: [[0.153, 'bad'], [0.152, 'and'], [0.041, 'dull'], [0.035, 'wasted'], [0.032, 'worst'], [0.031, 'terribly'], [0.029, 'from'], [0.028, 'been'], [0.028, 'superb'], [0.027, "we'll"]]

This makes sense, as we would expect negative words such as bad, wasted, dull, worst and terribly to show that a movie is bad. Likewise superb should indicate that the movie is good. Intriguingly, the frequency of the words "and, from, been, we'll" are indicative of the sentiment of a review. As a humourous aside, when getting the column name from the column index, I had an off by one error, due to the column indices coming from my training dataframe which had "__FileID__" removed, but I was getting the column names from the original dataframe. So the words most indicative of sentimentality were showing as "bacteria" and "ancy".

6 Random Forest Decision Trees

6.1 How they address the overfitting problem

Random forest trees help mitigate the problem of overfitting as they aggregate data into one final result. Random forest classifiers limit overfitting without substantially increasing error due to bias as we are able to set a max depth on the decision trees. Additionally, due to bagging, the features used in each individual tree are different, so when averaged, the result is improved. A way to (non-rigorously) intuit that random forests lower overfitting is that each individual decision tree comprising the random forest is still overfitting to the data. However, due to bagging, each decision tree will overfit in its own distinct way. Recall that bagging changes which entries are given to each decision tree, and which features the decision tree can split on. So each decision tree overfits to the data it was provided, and the ways its splits aren't common to every other tree. So when we average each of these distinct methods of overfitting on different components of our data, we end up with something that is actually representative of our data. This is because the methods that aren't very indicative of the entire data set (overfitting) won't appear in most decision trees, so their contribution will be surprised by the other trees in the forest. However methods that are helpful will strengthen the accuracy of the model.

Thus random forests will not overfit our entire training set like single decision trees do.

6.2 Selecting optimal parameters for the Forest

First, we decided to make our criterion "entropy" because we discussed this in lecture, and we understood how this worked. We believe that using a model we understand is preferable to a potentially better model which we don't understand.

We did this in two steps. First we found what the best results were in the grid search, by incrementing by 5. (I.e. 'n_estimators':[5, 10, 15 ... 45, 50, 55, 60]) This gave us an estimate of values to search in the next iteration of grid searching. In the first grid search, I got {'n_estimators': 60, 'max_depth': 35}, with score .775 as the best set of parameters. By checking the values over a few grid searches, it became clear that the more estimators, the better the model is, but we are limited by computational resources. Thus we chose 100 estimators, just based upon the time it takes to create such a model. After this, we did another grid search (using a different split between validation data and training data to avoid overfitting) to find the optimal depth, in the range of 31- 39 (inclusive). The best depth was 32. Between different validation / test splits, the score of our model with 100 estimators, a max depth of 32, and the entropy as our criterion, the score of the model on the validation data ranged between 77% and 83%. This is significantly better than the single decision tree case, which had a validation accuracy in the 60-70 percent range.

7 Conclusion

We found that Logistic Regressions models on average are the best method for classifying the data, followed by Random Forest Decision Trees, and finally single decision trees.

However the bag of words model is quite limited in how much accuracy it can provide so we thought of a few other ways we may improve the accuracy of the model in a future analysis. We could add some engineered columns, representing the frequency of words which appear in a list which we know to have negative connotations. (e.g. 'bad', 'hate', 'worst', 'terrible) We could derive a similar list for words with positive connotations. This may or may not enhance the model, but it should not worsen the model (as the models would not assign a strong weight to these derived columns). These columns would be linear combinations of other columns, which could make it useless in logistic regression, however it would still be new information to a decision tree. (As this could be a better parameter to split on) We did not include that in this analysis, as it deviated away from the bag of words model which we were told to use.

Another thing that may be useful is to look at pairs of words in addition to just single words. This would increase the data size quadratically, so we would have to remove word pairs that do not appear often. This is known as the 2-grams model, and can generalize to n-grams, n being the number of consecutive words looked at. This allows some more complexity to be realized by the models, as its capturing more information about the movie review.

However, the logistic regressions and random forests we used with the bag of words model did a satisfactory job of predicting the validation data.