CSC411 Project III

Duan, Mengli
1001233033

Khatta, Shamama
1001

March 2017

1 Part 1

The Cornell Movie Review dataset consists of 2000 movie reviews that are split into two categories: positive or negative reviews. Each review consists of a variable number of words. Some reviews contain multiple words that are the same. Additionally, the reviews consist of characters that do not form words, such as punctuation. In the positive reviews, it is clear to see that there are more words with positive connotations such as “darling”, “extraordinary”, or “fantastic”. Alternatively, the negative reviews often contain words with negative connotations associated with them, such as “unimpressive”, “stupid”, or “annoying”. As there are some distinct difference in the proportion of the types of words used in the different categories of the reviews, it should be possible to predict whether the review is positive or negative given that the appropriate data pre-processing steps are taken. That is provided that all punctuation and whitespace are removed, and words are split into their own features. We expect that common adjective words that appear in everyday language will be the strongest predictors of the type of a review.

Positive keywords that may be useful

amazing - 118
great - 691
good - 1396

Negative keywords that may be useful

awful - 79
bad - 818
gross - 38

2 Part 2

To use naive bayes' algorithm to predict whether a review is positive or negative, we must find the conditional probability of being positive or negative given a review. That is, we aim to find:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{1}$$

Where:
Y = 1 if the review is positive; 0 if the review is negative
X is a single review that we want to classify

To classify a single review, we must compute the conditional probability that the review is positive as well as the conditional probability that the review is negative. We classify the review as positive or negative depending on which conditional probability is greater.

More explicitly, for every review, we compute:

$$P(Y = 1|X) = \frac{P(X|Y = 1)P(Y = 1)}{P(X)} and P(Y = 0|X) = \frac{P(X|Y = 0)P(Y = 1)}{P(X)} \tag{2}$$

We choose to classify the review X based on:

$$P(Y|X) = \begin{cases} 1 & P(Y = 1|X) \geq P(Y = 0|X) \\ 0 & otherwise \end{cases}$$

```
```

Listing 1: Example with line numbers enabled

Since X is a single review, and a single review can consist of multiple words, then to compute the probability that a review is positive or negative, we actually need to break the review into its features and consider the probability of its individual features. Since naive bayes' algorithm assumes that the features are independent, we can compute the conditional probability of a review being positive as:

$$P(Y = 1|X) = P(x_0|Y = 1)P(x_1|Y = 1)...P(x_n|Y = 1) \tag{3}$$

We can ignore P(Y) from the original equation in this calculation since it is constant. Additionally, since P(X) has no semantic interpretation, we can ignore it as well. Hence, we only need to obtain the conditional probability of each word in the review in order to find our desired outcome.

To compute the probability of a word given the class of the review, we use the following equation:

$$P(x_i|Y) = \frac{count(x_i \text{in y class samples})}{count(\text{y class samples})} \tag{4}$$

Note, that since the probabilities are being multiplied, having a word appear in a review that has never been seen can result in the probability of the entire review being driven to zero. For this, reason we add the parameters m and k to the equation above. m and k are tunable parameters that give some form of interpretation to unseen words and prevents our probabilities from being driven to zero needlessly. Hence, we actually compute:

$$P(x_i|Y) = \frac{count(x_i \text{in y class samples}) + mk}{count(\text{y class samples}) + k} \tag{5}$$

Lastly, since equation 3 requires that we compute the probabilities of many small number (that is the probabilities of $P(x_i|Y)$ where i is a word in the review is always on a scale from 0..1), this can lead to underflow. To prevent this, when we are trying to find $P(Y|X)$, we use that:

$$a_1a_2...a_n = exp(loga_1 + loga_2 + ... + loga_n)$$

to instead just approximate $P(Y|X)$ as:

$$P(Y=1|X) = log(P(x_0|Y=1)) + log(P(x_1|Y=1)) + ... + log(P(x_n|Y=1)) \tag{6}$$

Since *exp* is a constant, it is not necessary that we add it to our computation of $P(Y=1|X)$. Because the log of a small number is a larger number, this solves our underflow issue.We tuned the parameters m and k by trial and error. We found that the value of m and k that were the best for us were m = 1.1 and k = 0.1.

3 Part 3

10 words that most strongly predict that the review is positive:
silverado', 'wilma', 'constraints', 'unequivocal', 'clueless', 'evens', 'efficiency', 'helmet', 'mayo', 'laundry'

'

10 words that most strongly predict that the review is negative:
thermopolis', 'batteries', 'borrr', 'problematic', 'horse', 'jenna', 'seamlessly', 'slash', 'intelligent', 'married'

To obtain our predictors:
To obtain the best predictors, we chose the top 10 highest conditional probabilities from our negative set of words and the top ten conditional probabilities from the positive class of words.

4 Part 4

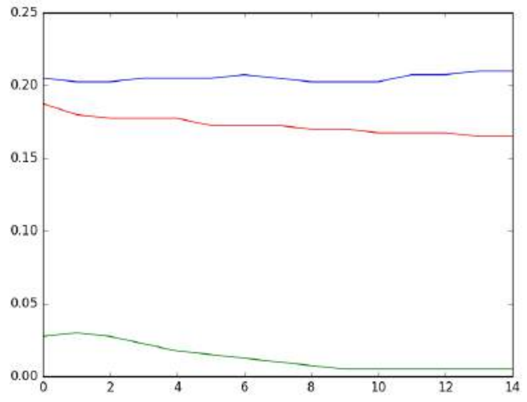
Train a Logistic Regression model on the same dataset. For a single movie review, For a single review r the input to the Logistic Regression model will be a k -dimensional vector v , where v[k]=1 if the k -th keyword appears in the review r

The way the input features are constructed are as the following: For a single review r, the input features is a vector v, [1xk] where k is the number of unique words in the training set. That is, all words that appear in positive and negative training class. If the i-th word appear in the review r, then v[k] = 1 and otherwise v[k] = 0. For m reviews in our training set, we have our input features with a dimension of [mxk]. To apply logistic regression, we initialize our weights, θ, as [2xm] where 2 refers the polarity of the review, i.e. positive or negative review. We used tensorflow to construct our logistic model.

The learning curves is shown below. From the graph, we can see a clear The regularization parameter seems to be trivial in our models. Ideally, increasing the value of should improved our performance from over-fitting. Given the size of training set is only 200, our model doesn't exhibits over fitting, over 5000 iterations. The performance on the training set converges to 100%, and 78.6% on the test set.

The experiments to get above results are as followed. We trained our models with the following sizes, 400, 1400 respectively. We observed that with = 0 and = 0.05 the performance on test set has significant improvements.

Our learning curve is show below. It plots error rate for every 100 iterations.



5 Part 5

The formulation of both Logistic Regression and Naive Bayes can generalized to what is shown below:

$$\sum_{i=0}^{k} \theta_i I_i(x) = \theta_0 + \theta_1 I_1 + \theta_2 I_2 + ... + \theta_k I_k > 0.5$$

The interpretation of the parameters in the formula is justified in the table below. Note k here refers to the number of all words that appear in both positive and negative training class. i means the i-th word in the training class.

Paras	Naive Bayes	Logistic Regression
θ_i	log(P(w_i|+)/P(w_i|−))	weights
$I_i(x)$	1 if w_i in r 0 otherwise	binary values
thr	0.5	0.5

I_i here is a binary variable for each positive review r_i, which represents the status whether the i-th word in the review r_i. For further details, please see in discussion in part 4.

6 Part 6

The top 100 s that you obtained using Logistic Regression:
For positive reviews: 'impacted', 'frenzy', 'eruption', 'cellist', 'vow', 'geologic', 'zdenek', 'kathryn', 'leeper', 'characterizations', 'jan', 'allen', 'burst', "dante's", "afi's", 'atheist', 'paramount', 'rigors', 'stallone', 'snoozing', 'geologist', 'tropical', 'dragons', 'yells', 'dumping', 'railway', 'volcanos', 'chums', 'heat', 'bigelow', 'considered', 'multiplex', 'procedure', 'downer', "ricci's", 'understood', 'slice', 'twister', 'volcano', 'theorists', 'enrages', 'gallons', "town's", 'visceral', 'kolya', 'straps', 'bachelor', 'sesame', 'brill', 'hoped', 'debating', 'abode', 'ingred', 'connections', 'partaking', 'lori', 'unborn', 'abruptly', 'reporter', 'bundle', 'darryl', 'ghreat', 'someday', 'volcanic', 'proposed', 'collapses', 'jobs', 'digitally', 'emo', 'busey', 'phones', 'humphry', 'hampered', 'napkin', 'mace', 'beth', 'confession', 'mst', 'pets', 'daring', 'solaris', 'animus', 'richest', "'volcano'", 'jay', 'quake', 'defensive', 'kkk', 'tenants', 'business', 'labute', 'kenny', "'cheezy'", 'secret', 'table', 'okay', "'twister'", 'quandary', 'expecting', 'replacement'.

For negative reviews: 'horrendous', 'words', 'extraordinarily', 'titular', 'patterson', 'staros', 'wordy', 'spade', 'divorced', 'sw', "lemmon's", 'caller', 'wooden', 'dinky', 'civil', 'robbery', 'driller', 'transformed', 'weighted', 'generating', 'lion', "nolte's", 'aiding', 'winslow', 'recover', 'cheeze', 'fights', 'toronto', 'amazed', 'toys', 'col', 'ribisi', 'turbulent', 'immense', 'merits', 'havilland', 'performers', 'version', 'gaping', 'raids', 'holidays', 'pontificating', 'littered', 'packed', 'y', 'invest', 'renound', 'grad', 'kitty', 'georges', 'courteney', 'affordable', 'fireballs', 'frasier', 'undermining', 'adaptation', 'subjects', 'intersect', 'ineffectual', 'emphasis', 'pleaser', 'cusack', 'leaves', 'gorgeous', 'routes', 'waltz', 'craving', 'tinted', 'olin', 'wayne', 'triumphant', 'hippie', 'compass', 'copland', 'patrolman', 'basing', 'imaginable', 'giovanni', 'incestuous', 'bolt', 'notebook', 'obscure', 'masters', 'collector', 'phenomena', "lawrence's", 'refusing', 'skarsg', 'forensic', 'realms', 'randle', "helen's", 'st', "wallace's", 'superficial', 'animal', 'innumerable', 'deceptively', 'founded', 'primate'.

The top 100 s that you obtained using Naive Bayes:

['silverado', 'wilma', 'constraints', 'unequivocal', 'clueless', 'evens', 'efficiency', 'helmet', 'mayo', 'laundry', 'hurley', 'does', "it'd", 'mansions', 'dating', 'marital', 'drivel', 'wincer', 'unreasonable', 'aiding', 'slotted', 'lied', 'yankee', 'flawed', 'drafted', 'macbeth', 'poison', 'dalton', 'creepiest', "henson's", 'wanna', 'sporadic', 'vietnam', 'pound', 'defines', 'kitschy', 'italian', 'wanted', 'aforementioned', 'hootworthy', 'inadvertent', 'rattlesnakes', 'slowed', 'launch', 'sanitarium', 'ceo', 'suspected', 'falsely', 'ably', 'nora', 'greenpeace', 'prodigious', 'complains', 'thinks', 'preparing', 'carrier', 'maniacal', 'humiliating', "africa's", 'overlooking', 'fifth', "someone's", 'master', 'melbourne', 'knock', 'yakuza', 'barb', 'requisite', 'waaaaay', 'propelling', 'paranoia', 'slang', 'saps', 'presences', 'croon', 'cringes', 'rating', 'story', 'round', 'maker', 'cds', 'dzunza', 'unprofessional', 'wright', 'bombarding', 'danner', 'north', 'polar', 'assured', 'submarines', 'facemask', 'messed', 'toy', 'sacrificial', 'tenth', 'romania', 'wary', 'returning', 'motivations', 'priestley']

['abortive', 'prototype', 'trump', "sorvino's", 'rehashing', 'innocently', 'dunces', 'pun', 'tattooed', 'leering', 'void', 'stomped', 'passport', 'stud', 'transparently', 'op', 'studying', 'customer', 'scarborough', 'ugliness', 'richness', 'films', 'jax', 'selena', 'title', 'aging', 'talkies', 'sublimated', 'mosaic', 'bluntly', 'impersonation', 'discovers', 'frenchman', 'laborious', 'oblivion', 'dont', 'bowl', 'parachute', 'leader', 'outta', 'reccemond', 'ray', 'cannon', 'shirts', 'bothersome', 'floor', 'attar', 'prevents', 'ponder', 'puns', 'smitten', 'throwback', 'tossed', 'lead', 'pacify', 'alyson', 'differnt', 'got', 'adolf', 'basketball', 'care', 'yo', 'nikita', 'creep', "arthur's", 'phil', 'blooded', 'trucking', 'salt', 'tickets', 'energetic', 'occur', 'overthrow', 'grace', 'exploding', 'hardworking', 'disaster', 'carries', 'appear', 'lingerie', 'money', 'growing', 'modern', "nun's", 'papers', 'passionately', 'reporter', 'friendship', 'nest', 'killings', 'acts', 'graft', 'thirty', 'scully', 'prinze', 'cast', 'conti', 'dorn', 'proudly', 'touts']

7 Part 7

The performance of our logistic regression is shown below:

Training Set: 0.667022
Validation Set: 0.614249

The objective of this part is to find out whether the target word appear together with the word using logistic regression. That is, given (word, target), our model would output 1 if word and target are next to each other and 0 otherwise.

In order to accomplish this, we generate features are as follows:
- The features are tuples of words, i.e. (words, target).
- As each word is represented by the word2vector embeddings, a [1x128] vector, our input features for one training example , therefore, have the dimension of [1x256].

Inputs to logistic regression:
In order to train our logistic classifier, we generate tuples of words from our reviews and split them into two categories: adjacent or distant where adjacent tuples are tuples of words that were next to each other in the reviews and distant tuple are tuples that had a distance greater than 1 across all the reviews. We use data from about 65 reviews for our input training matrix. The total number of adjacent tuples that we have to work with is about 50 in every batch and the amount of distant tuples we had to work with was about 50 in every batch. We also generated a corresponding training Y matrix that basically had a mapping such that Y at a particular index was 1 if the training input matrix, X, at the corresponding index was an adjacent tuple and 0 otherwise. When we generate our inputs, we decide to include an equal number of positive class (i.e., adjacent tuples) and negative class (i.e., distant tuples). We found that not doing so often skewed our classifier to classify outputs as negative more often and this negatively impacted our performance.

To train our model:
We train our model using logistic regression. We chose to use the sigmoid activation function with 1 unit to be able to classify our inputs. Results less than 0.5 are chosen as the negative class (i.e., not adjacent) and results >= 0.5 are chosen to be the positive class (i.e., adjacent). We define our weights to be a vector with a dimension of [1x256]. Our logistic regression comes up with a hypothesis that is [n*1] where n is the number of training examples (number of tuples) that we send in.

In order to train our classifier, we decided to use batches. When we weren't using batches, we found that we had a much lower accuracy and additionally it took us much longer to converge. Since different batches had somewhat different composition of tuples, using batches enabled us to cycle through our training data more quickly and also allowed us to prevent overfitting. It also allowed us to cycle through more data in general which helped improve our accuracy .Through trial and error we found that the best learning rate for us was 0.00005. Additionally, the regularization penalty, through trial and error, did not do much to help us obtain a better performance. This is likely because regularization helps ensure that some features do not overpower other in their importance and this is not very applicable to this scenario. Using smaller batches (about 100) resulted in having a larger variance in the range of values that we would obtain for our performance. For example, with training size of about 100, we got results between 0.65 - 0.75. Using larger batches resulted in more stable range of values for performance, however, were much slower to converge and, as such, at the time of termination were lower than what we obtained from smaller batches.

8 Part 8

To finding words that are the most similar to target words, we choose to check for cosine similarity. The final result there are "the most similar words to story are how, film, mon, magma, rightly, ingenious, bane, walsh, manipulative, inclusion" and "the most similar words to good are could, hes, whether, awash, understand, think, meanings, fascinated, hirschfeld, delpy". The two interesting examples that demonstrates that word2vec works are as followed.

word2vec(gloriously) = word2vec(joy) + word2vec(happy)
word2vec(great) = word2vec(better) + word2vec(good)

To do this, we first compute the sum of the embeddings of these two words. We treat the sum as our new target word to feed into our similar words finder function, which is part8(), and what we get are the word that has the most cosine similarity with the sum embeddings. The word we find for 'joy' and 'happy' is 'gloriously'; for 'better' and 'good' is 'great', which makes sense.

9 Part 9

To generate the dataset on which Logistic Regression works a significantly better than Naive Bayes, we want to increase the randomness of our dataset. Similar idea of what we did in Part 5 in Project 2, logstic regression generally is expected to have a better performance than naive bayes when given fairly noisy input. Thus, we swapped the first 20% of training set for pos and neg. The results shows us that logistic regression perform steadily than naive bayes.
The code snipped is as followed: