CSC411: Project 3

Due on Tuesday, March 21, 2017

Katie Datsenko, Loora Zhuoran Li

March 22, 2017

Part 1

The dataset consists of 1000 positive reviews and 1000 negative reviews for movies taken from the IMDB database. While there are many words that are dependent on the movie itself, are ambiguous, or are part of everyday language, some of the word inevitably express particular emotions towards the movie in question. For instance, a sample phrase from a positive review that expresses a positive sentiment is 'this is the most fun film of the summer', where the notable keyword is 'fun'. A sample phrase from a negative review is 'didn't they read the bad dialogue, the cheezy lines', where the notable keyword is 'bad' and perhaps 'cheezy'. Note that 'cheezy' has incorrect spelling, which represents an obstacle for semantic analysis. Other weaknesses of semantic keyword analysis are its inability to handle language structures such as sentence negation, sarcasm, and the terseness or tone of a phrase. However, we believe there still is lot of information available for determining the polarity of a review based on simple key word detection.

We selected the example keywords below based on the **difference in frequency** in reviews of the class we are interested in, and reviews of the opposite class. Those with greater polarity in frequency as selecting as keywords that may have be 'useful' for determining whether a review is of that class.

Examples for specific keywords and their statistics:

```
================== RUNNING PART 1 ==================
The number of positive reviews is 1000
The number of negative reviews is 1000
3 keywords useful for Positive reviews:
Word: both Positive count: 377.0 Negative count: 243.0
Word: also Positive count: 604.0 Negative count: 466.0
Word: life Positive count: 471.0 Negative count: 307.0
3 keywords useful for Negative reviews:
Word: bad Positive count: 255.0 Negative count: 507.0
Word: worst Positive count: 43.0 Negative count: 194.0
Word: plot Positive count: 367.0 Negative count: 509.0
```

Part 2

There were two parameters to tune, m and k. To tune either of them, we did a grid search. For each parameter we made a list of values to try. We set the limit value of k to be higher than m since the value in the final result should typically appear as a probability value (between 0 and 1) as it represents the conditional probability of a particular key word appearing for a review conditioned as one of the classes (positive or negative), and a large value of m could produce values greater than 1. We experimented with different step values, from a coarse grained larger step size to a finer grained step value (at the cost of greater runtime to search), and ran a nested loop to compare the performance with each pair of values on the validation set. The value pair that scores the highest on the validation set is taken as the final tuned values. The relevant code snippets are shown below.

```
# CHANGE THE GRID VALUES.....!!!!
m_grid = arange(0.2, 2, 0.2)
k_grid = arange(0.5, 10, 0.5)

def train_bayes(m_grid, k_grid, \
posDict, negDict, prob_p, count_p, prob_n, count_n, valid_set, valid_l):
    best_accuracy = -1
    for m in m_grid: # Smoothing parameter tuning loops
        print("m: " + str(m))
        for k in k_grid:
            print("k: " + str(k))
            accuracy = classify_bayes(valid_set, valid_l, posDict, negDict, \
            prob_p, count_p, prob_n, count_n, m, k)
            print(accuracy)
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_params = (m, k)
    return best_params
```

Our results are as follows:

```
Final tuned parameters m=1.0 k=2.0
Performance on training set: 97.75
Performance on validation set: 86.0
Performance on test set: 75.5
```

To avoid underflow in the computations, we took the log of the expression of $P(a_1, a_2, ...a_n|class) * P(class)$ representing the MAP probability of a class assignment (for example, the positive class) conditioned on a sample review (consisting of the keywords a_i). We ignored the denominator expression $P(a_1, a_2, ...a_n)$ in our computation since it will be equal for either class and thus not useful for classification. Thus our new expression becomes

$$\sum_i \log(P(a_i|class)) + \log(P(class)) = \sum_i [\log(P(a_i = 1|class)) - \log(count(class))] + \log(P(class))$$

We assign a class based on which probability is greater for the sample review, the MAP computation for the positive class or negative class. To this end, we also skipped the step of taking the exp of the expression since both log and exp are increasing functions and taking exp does not change the relative values. The relevant code snippets are included below:

```
def make_class_prediction(sample, class_worddict, class_prob, count_class, m, k):
    prediction = 0
    # remove duplicates
    sample_words = set(sample)
    #compute P(a1, ...an | class)
    for word in sample_words:
        if word in class_worddict:
            p_ai = float(class_worddict[word] + m*k)
        else:
            p_ai = float(m*k)
        prediction += log(float(p_ai)) - log(float(count_class + k))
    #prediction = exp(prediction)
    return prediction + log(class_prob)

def classify_bayes(x, t, posDict, negDict, prob_p, p_count, prob_n, n_count, m, k=1):
    hits = 0
    for i in range(len(x)):
        sample = x[i]
        # Compute the negative and positive probabilities
        positive_prediction = make_class_prediction(sample, posDict, prob_p, p_count, m, k)
        negative_prediction = make_class_prediction(sample, negDict, prob_n, n_count, m, k)

        prediction = 0
        # We assign a classification based on which probability is greater
        if positive_prediction > negative_prediction:
            prediction = 1
        if prediction == t[i]:
            hits += 1
    return float(hits) / len(t)
```

Part 3

List the 10 words that most strongly predict that the review is positive, and the 10 words that most strongly predict that the review is negative. State how you obtained those in terms of the the conditional probabilities used in the Naive Bayes algorithm.
For each word a_i in the training set of reviews we store the log of the conditional probability of a word with respect to the class

$$\log(P(a_i|class)) + \log(P(class))$$

in a dictionary named *wordDict* stored under the respective class (i.e. either $wordDict[word][0]$ for positive or $wordDict[word][1]$ for negative). This allows us to easily compute the bayes classification probability for a sample review consisting of multiple words by taking the sum of the log of conditional probabilities, or the sum of the corresponding entries for each word a_i in the *wordDict*.
To obtain the words that strongly predict one of the classes, we used the log odds ratio of the conditional probabilities of a_i between the positive and negative classes. For example, the ratio of the positive to the negative class may be expressed as:

$$\log(\frac{P(class=1|a_i)}{P(class=0|a_i)})$$

$$= \log\Big(\frac{(\frac{P(a_i=1|class=1)P(class=1)}{P(a_i)})}{(\frac{P(a_i=1|class=0)P(class=0)}{P(a_i)})}\Big)$$

$$= \log(P(a_i|class=1)) - \log(P(a_i|class=0)) + log(\frac{P(class=1)}{P(class=0)})$$

(The $log(\frac{P(class=1)}{P(class=0)})$ term may be excluded since it is the same for every word a_i):

$$\approx \log(P(a_i|class=1)) - \log(P(a_i|class=0))$$

The reasoning behind the log odds ratio is that it will be greater than one for features (keywords) that cause belief in the Positive Class to be greater relative to the Negative Class. The features that have the greatest impact at classification time are those with both a high probability (because they appear often in the data) and a high odds ratio (because they strongly bias one label versus another). Computing the log odds requires a simple subtraction between the *wordDict* entries, as the log function is already applied on them.

```
================== RUNNING PART 3 ==================
10 most strongly Positive word predictions:
['bold', 'spielbergs', 'ideals', 'lovingly', 'gattaca', 'wonderfully',
'dread', 'fashioned', 'astounding', 'outstanding']
10 most strongly Negative word predictions:
['turkey', 'ludicrous', 'lifeless', 'feeble', 'unimaginative', 'stupidity',
'sucks', 'nonsense', 'insipid', 'insulting']
```

The relevant code snippet is shown below:

```
def part3(wordDict):
    logOdds = []

    for word in wordDict:
        logOdds.append((wordDict[word][0] - wordDict[word][1], word))
    logOdds.sort()
```

```
    vocab_size = len(logOdds)
    print("10 most strongly Positive word predictions:")
    print([word for val, word in logOdds[(vocab_size-10):]])
    print("10 most strongly Negative word predictions:")
    print([word for val, word in logOdds[:10]])
```

Part 4

For the Logistic Regression model we used the same model structure as for multinomial logistic regression in Project 2, except that since we have only two classes, thus our one-hot encoding assigns $[1, 0]$ for the positive class or $[0, 1]$ for the negative class. We train a single fully connected layer with a softmax for the output units. The loss function is still negative log loss. We show that the use of the multinomial logistic regression model with 2 classes reduces to simple binary logistic regression with cross-entropy loss.
The output of the 2 softmax units can be summarized as the following vector (we assume the bias b of each unit is integrated into each θ with a dummy multiplier of 1 in x):

$$h_\theta(x) = \frac{1}{\exp(\theta^{(1)\top}x) + \exp(\theta^{(2)\top}x)} \begin{bmatrix} \exp(\theta^{(1)\top}x) \\ \exp(\theta^{(2)\top}x) \end{bmatrix}$$

Dividing both the numerators and denominators of the two components of the vector by $\exp(\theta^{(1)\top}x)$ we have

$$h_\theta(x) = \frac{1}{\frac{\exp(\theta^{(1)\top}x)}{\exp(\theta^{(1)\top}x)} + \frac{\exp(\theta^{(2)\top}x)}{\exp(\theta^{(1)\top}x)}} \begin{bmatrix} \frac{\exp(\theta^{(1)\top}x)}{\exp(\theta^{(1)\top}x)} \\ \frac{\exp(\theta^{(2)\top}x)}{\exp(\theta^{(1)\top}x)} \end{bmatrix}$$

$$= \frac{1}{\exp(\theta^{(1)\top}x - \theta^{(1)\top}x) + \exp(\theta^{(2)\top}x - \theta^{(1)\top}x)} \begin{bmatrix} \exp(\theta^{(1)\top}x - \theta^{(1)\top}x) \\ \exp(\theta^{(2)\top}x - \theta^{(1)\top}x) \end{bmatrix}$$

$$= \frac{1}{\exp(\vec{0}^\top x) + \exp((\theta^{(2)} - \theta^{(1)})^\top x)} \begin{bmatrix} \exp(\vec{0}^\top x) \\ \exp(\theta^{(2)} - \theta^{(1)})^\top x) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \\ \frac{\exp((\theta^{(2)}-\theta^{(1)})^\top x)}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \\ 1 - \frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \end{bmatrix}$$

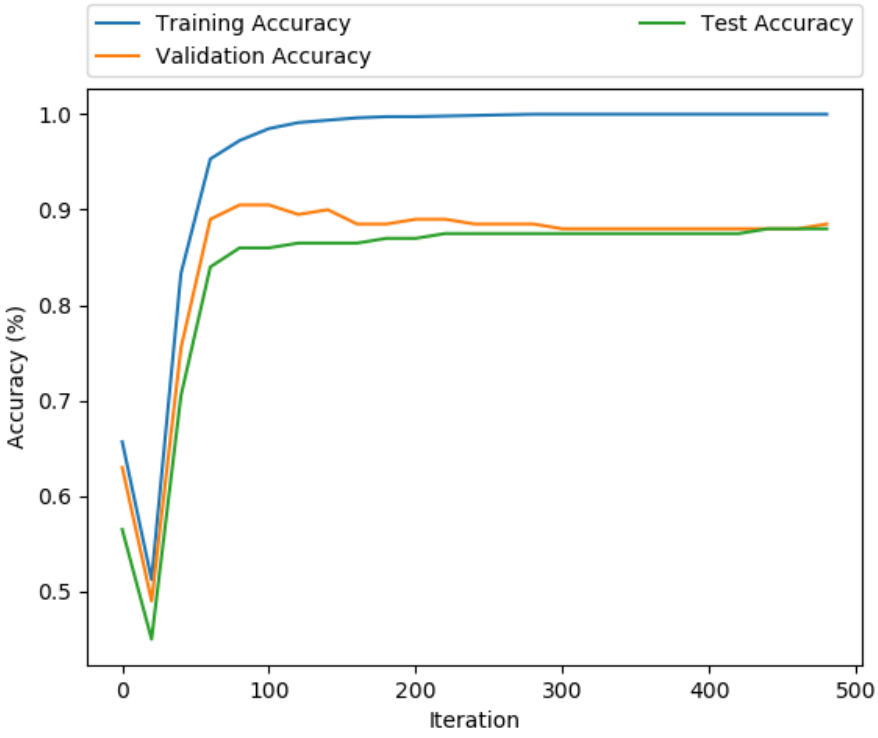
If we replace $\theta^{(1)} - \theta^{(2)}$ in the expression with a single parameter vector θ', we find that the softmax regression predicts the probability of one of the classes as $\frac{1}{1+\exp(-(\theta')^\top x)}$, and the probability of the other class as $1 - \frac{1}{1+\exp(-(\theta')^\top x)}$. Thus the model is equivalent to binary logistic regression, the only difference is that θ' is overparameterized as $\theta^{(1)} - \theta^{(2)}$, however good performance of the overparameterized model can be achieved with enough iterations of the training algorithm.

In our loss function, we take the dot product of the target y_- with the log of the softmax output y:

$$\begin{bmatrix} y^{(i)} & 1 - y^{(i)} \end{bmatrix} \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x)} \\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x)} \end{bmatrix}$$

This exactly corresponds to the cross-entropy loss function in binary logistic regression.

The regularization parameter is selected in a similar way as m, k in Part 2. We do a grid search, trying multiple values of the regularization coefficient of the L2 penalty of the $\theta^{(2)}$, $\theta^{(1)}$ parameters (or the weights in our single layer fully connected softmax network) and we note where the descent algorithm performance on the validation set begins to decay, indicating we should cap our grid search to the last used value which demonstrated improvement for the performance of our validation set. We selected 0.0085 as the regularization coefficient.
The learning curves of the Logistic Regression model are shown below:


Training Accuracy
Validation Accuracy
Test Accuracy
Accuracy (%)
1.0
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400
500
Iteration

Part 5

For Logistic Regression, in Part 4 we showed that the vectorized softmax output for either class is:

$$= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \\ 1 - \frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} \end{bmatrix}$$

Just as in binary logistic regression, we choose the class with the *argmax* probability. This is the same as choosing $y = 1$ if

$$\frac{1}{1+\exp((\theta^{(2)}-\theta^{(1)})^\top x)} > 0.5 \iff (\theta^{(2)}-\theta^{(1)})x > 0$$

.

Previously, we assumed that the bias term was included in the θ's. Now we can bring it out:

$$(\theta^{(2)}-\theta^{(1)})x + (b^{(2)}-b^{(1)}) > 0$$

$$= (b^{(2)}-b^{(1)}) + (\theta_1^{(2)}-\theta_1^{(1)})x_1 + (\theta_2^{(2)}-\theta_2^{(1)})x_2 + \ldots + (\theta_k^{(2)}-\theta_k^{(1)})x_k > 0$$

Thus for the expression $\theta_0 + \theta_1 I_1(x) + \theta_2 I_2(x) + \ldots + \theta_k I_k(x) > thr$, each $I_i(x)$ represents the presence of the word a_i from the vocabulary in the sample review x (value is 0 or 1). k is the size of the vocabulary (the total number of unique words in the training set from both the positive and negative classes). θ_0 represents the difference in the bias of the units $(b^{(2)} - b^{(1)})$. The coefficient θ_i of each $I_i(x)$ corresponds to $(\theta_i^{(2)} - \theta_i^{(1)})$ in the expression above, which is the weight connecting the output unit for the negative class (1) to the input unit x_i subtracted from the weight connecting the output unit for the positive class (2) to the input unit x_i. Thus θ_i represents the polarity of the i^{th} keyword. If the i^{th} keyword is strongly positive, θ_i is strongly positive, and vice-versa.

For Naive Bayes, when we classify a sample review, we choose the *argmax* of the conditional probabilities of each class ($1 = positive$ or $0 = negative$) conditioned on the sample review. This is the same as choosing $y = 1$ if

$$\frac{P(class = 1|a_1, \ldots, a_n)}{P(class = 0|a_1, \ldots, a_n)} > 1$$

If we apply a log to either side of the inequality we have

$$\log(P(class = 1|a_1, \ldots, a_n)) - \log(P(class = 0|a_1, \ldots, a_n)) > 0$$

Expanding out we have

$\sum_i \Big(\log(P(a_i|class = 1)) \Big) + \log(P(class = 1)) - \log(P(a_i, \ldots, a_n))$

$- \sum_i \Big(\log(\frac{(P(a_i=1|class=0))}{count(class=0)}) \Big) - \log(P(class = 0)) - (-\log(P(a_i, \ldots, a_n))) > 0$

$$= \log\Big(\frac{P(class = 1)}{P(class = 0)}\Big) + \sum_i \Big(\log(P(a_i|class = 1)) - \log(P(a_i|class = 0)) \Big) > 0$$

Thus if the expression $\theta_0 + \theta_1 I_1(x) + \theta_2 I_2(x) + \ldots + \theta_k I_k(x) > thr$ represents the classification of a single sample review, the bias term θ_0 is the log of the ratio of prior probability of the positive class to the negative class. k is the size of the vocabulary (the total number of unique words in the training set from both the positive and negative classes). Each I_i is a binary 0 or 1 value representing the presence of the i^{th} keyword from the vocabulary in the sample review. The corresponding coefficient θ_i is the log odds value for the word a_i we obtained in Part 3. θ_i. θ_i should represent the polarity value of the i^{th} word. As explained in Part 3, the log odds value for the word represents the bias of one label versus another.

Part 6

Each θ_i corresponds to the polarity value of the i^{th} word in the training set vocabulary, as explained in Part 6. We are choosing the 100 greatest (most positive) θs, which means we are selecting the θs of the words with the strongest polarity for the positive class of reviews.

From the results we can see that Naive bayes and Logistic regression select some of the same words in their top 100. Examples include 'outstanding', 'terrific', and 'wonderfully', which qualitatively are very positive. Among both there exist noisy words such as 'also' with a ranking of 3 for logistic regression, and 'gattaca' with a ranking of 5 for Naive Bayes. Some of these may be explained by the bias in the training data since positive reviews are longer than negative reviews, thus there is a chance that a word such as 'also' may appear in more positive reviews.

It appears that Naive Bayes more consistently selects keywords with a more immediately strong positive emotion in its top 100 than Logistic regression, for example 'uplifting', 'melancholy', 'lovingly', and 'masterfully' are selected by Naive Bayes but are not present in the list for Logistic Regression. There are more adjectives and adverbs for Naive Bayes. In Logistic Regression in the top of the list we can immediately see words that are not as descriptive; some are nouns, some can be used as emphasis or complimenting words for a positive description. For example, 'life', 'most', 'especially', 'true', 'both', 'together', 'many' are words that are unique to the top 100 for Logistic Regression. These patterns can perhaps be explained by the differences in the models.

In Naive Bayes, each feature's weight is set independently, based on how much it correlates with the label. The result is that that if some of the features are dependent on each other and they often occur together (especially if the feature space is large), the prediction might be overestimating their influence in the whole sample space because we are multiplying probability values of two or more highly correlated features that really represent a single feature, and assume that these features could encapsulate a large union of positive training samples, whereas in reality they cover a smaller intersection. Thus, we see many similar adjectives and adverbs selected and pairings such as ('masterfully' and 'weaves'), which may occur together in the text, for the many of the same texts, but are not necessarily the most representative of the entire sample space. For instance, it may not always be the case that the review would contain 'masterfully' or 'weaves'. However, in logistic regression, all the weights are set together such that the linear decision function tends to be high for positive classes and low for negative classes. Thus Logistic Regression may relax the Naive Bayes assumption and be able to compensate for correlated features by weighting them lower. In fact, we see words that are more peculiar as a positive description, but may be even more generalized than Naive Bayes. For example, 'life' and 'fun'.

Finally, the theta values for logistic regression are smaller than for Naive Bayes. This is could be due again to the fact that NB fits feature weights independently, while LR accounts for correlations amongst features and thus together with regularization sets weights to be smaller.

```
================== RUNNING PART 6 ==================
Printing top 100 thetas for logistic regression:

Theta 1 0.370288282633 Pos Word: hilarious
Theta 2 0.341819703579 Pos Word: others
Theta 3 0.32956302166 Pos Word: also
Theta 4 0.310441970825 Pos Word: fun
Theta 5 0.29780715704 Pos Word: life
Theta 6 0.295863032341 Pos Word: perfectly
```

```
Theta 7 0.293822616339 Pos Word: memorable
Theta 8 0.292977154255 Pos Word: most
Theta 9 0.286952733994 Pos Word: terrific
Theta 10 0.281670749187 Pos Word: overall
Theta 11 0.281082808971 Pos Word: performances
Theta 12 0.276417016983 Pos Word: sometimes
Theta 13 0.264105260372 Pos Word: american
Theta 14 0.263832807541 Pos Word: perfect
Theta 15 0.259931981564 Pos Word: especially
Theta 16 0.256685435772 Pos Word: well
Theta 17 0.254457831383 Pos Word: true
Theta 18 0.254110217094 Pos Word: job
Theta 19 0.249710828066 Pos Word: both
Theta 20 0.249392807484 Pos Word: together
Theta 21 0.247464090586 Pos Word: yet
Theta 22 0.244372099638 Pos Word: quite
Theta 23 0.243289008737 Pos Word: wonderfully
Theta 24 0.238423168659 Pos Word: many
Theta 25 0.238133817911 Pos Word: very
Theta 26 0.237949088216 Pos Word: excellent
Theta 27 0.236174911261 Pos Word: performance
Theta 28 0.235593289137 Pos Word: enjoyable
Theta 29 0.229307442904 Pos Word: seen
Theta 30 0.228570848703 Pos Word: best
Theta 31 0.228331148624 Pos Word: great
Theta 32 0.223626852036 Pos Word: makes
Theta 33 0.220908969641 Pos Word: details
Theta 34 0.219353586435 Pos Word: different
Theta 35 0.217106878757 Pos Word: flaws
Theta 36 0.210115492344 Pos Word: ending
Theta 37 0.207596465945 Pos Word: people
Theta 38 0.206161439419 Pos Word: family
Theta 39 0.205819502473 Pos Word: light
Theta 40 0.205211788416 Pos Word: deserves
Theta 41 0.203720211983 Pos Word: your
Theta 42 0.20241111517 Pos Word: world
Theta 43 0.200672328472 Pos Word: though
Theta 44 0.200606048107 Pos Word: pace
Theta 45 0.200246855617 Pos Word: set
Theta 46 0.19957524538 Pos Word: simple
Theta 47 0.198278710246 Pos Word: ive
Theta 48 0.197140872478 Pos Word: while
Theta 49 0.197102129459 Pos Word: brilliant
Theta 50 0.196320056915 Pos Word: works
Theta 51 0.195221245289 Pos Word: easily
Theta 52 0.194099128246 Pos Word: him
Theta 53 0.193452358246 Pos Word: town
Theta 54 0.193373888731 Pos Word: chemistry
Theta 55 0.192798674107 Pos Word: throughout
```

```
Theta 56 0.192309722304 Pos Word: extremely
Theta 57 0.190947294235 Pos Word: gives
Theta 58 0.18970644474 Pos Word: definitely
Theta 59 0.189220011234 Pos Word: fight
Theta 60 0.187702104449 Pos Word: although
Theta 61 0.18651214242 Pos Word: four
Theta 62 0.18610650301 Pos Word: may
Theta 63 0.184525519609 Pos Word: soundtrack
Theta 64 0.184228926897 Pos Word: right
Theta 65 0.182777166367 Pos Word: see
Theta 66 0.18255341053 Pos Word: allows
Theta 67 0.181220784783 Pos Word: twists
Theta 68 0.179337233305 Pos Word: portrayed
Theta 69 0.178353592753 Pos Word: outstanding
Theta 70 0.178338199854 Pos Word: enjoyed
Theta 71 0.173395931721 Pos Word: surprised
Theta 72 0.171155959368 Pos Word: wonderful
Theta 73 0.171056509018 Pos Word: naturally
Theta 74 0.169605761766 Pos Word: entertaining
Theta 75 0.169468715787 Pos Word: change
Theta 76 0.16915653646 Pos Word: class
Theta 77 0.167777106166 Pos Word: today
Theta 78 0.167124301195 Pos Word: back
Theta 79 0.167083710432 Pos Word: era
Theta 80 0.167033970356 Pos Word: leave
Theta 81 0.165266931057 Pos Word: oscar
Theta 82 0.165243148804 Pos Word: realistic
Theta 83 0.164840936661 Pos Word: intelligent
Theta 84 0.164191395044 Pos Word: above
Theta 85 0.162372589111 Pos Word: pulp
Theta 86 0.162127256393 Pos Word: subtle
Theta 87 0.162071466446 Pos Word: bill
Theta 88 0.161928340793 Pos Word: couple
Theta 89 0.161036044359 Pos Word: takes
Theta 90 0.160862445831 Pos Word: eventually
Theta 91 0.159853026271 Pos Word: classic
Theta 92 0.159403383732 Pos Word: thankfully
Theta 93 0.159205660224 Pos Word: other
Theta 94 0.158646672964 Pos Word: breathtaking
Theta 95 0.157781720161 Pos Word: follows
Theta 96 0.157499045134 Pos Word: plenty
Theta 97 0.156933516264 Pos Word: break
Theta 98 0.156833425164 Pos Word: shows
Theta 99 0.156503647566 Pos Word: still
Theta 100 0.15550801158 Pos Word: mind


Printing top 100 thetas for naive bayes:
```

```
Theta 1 2.13585539483 Pos Word: outstanding
Theta 2 2.03676449218 Pos Word: astounding
Theta 3 1.90323309956 Pos Word: fashioned
Theta 4 1.90323309956 Pos Word: dread
Theta 5 1.88521459406 Pos Word: wonderfully
Theta 6 1.8291251274 Pos Word: gattaca
Theta 7 1.74908241973 Pos Word: lovingly
Theta 8 1.74908241973 Pos Word: ideals
Theta 9 1.69192400589 Pos Word: spielbergs
Theta 10 1.69192400589 Pos Word: bold
Theta 11 1.68008954824 Pos Word: flawless
Theta 12 1.68008954824 Pos Word: damon
Theta 13 1.66207104274 Pos Word: weaknesses
Theta 14 1.66207104274 Pos Word: weakness
Theta 15 1.66207104274 Pos Word: offbeat
Theta 16 1.66207104274 Pos Word: notoriety
Theta 17 1.66207104274 Pos Word: niccol
Theta 18 1.66207104274 Pos Word: masterfully
Theta 19 1.66207104274 Pos Word: en
Theta 20 1.66207104274 Pos Word: downside
Theta 21 1.66207104274 Pos Word: continuing
Theta 22 1.66207104274 Pos Word: coens
Theta 23 1.66207104274 Pos Word: burbank
Theta 24 1.63129938407 Pos Word: hatred
Theta 25 1.63129938407 Pos Word: avoids
Theta 26 1.62233071409 Pos Word: finest
Theta 27 1.61555102711 Pos Word: era
Theta 28 1.59955068576 Pos Word: anger
Theta 29 1.56676086294 Pos Word: mixing
Theta 30 1.56676086294 Pos Word: uplifting
Theta 31 1.56676086294 Pos Word: uncut
Theta 32 1.56676086294 Pos Word: trumans
Theta 33 1.56676086294 Pos Word: melancholy
Theta 34 1.56676086294 Pos Word: marvelous
Theta 35 1.56676086294 Pos Word: jude
Theta 36 1.56676086294 Pos Word: introspective
Theta 37 1.56676086294 Pos Word: freed
Theta 38 1.56676086294 Pos Word: crimson
Theta 39 1.56676086294 Pos Word: brisk
Theta 40 1.52593886842 Pos Word: religion
Theta 41 1.52593886842 Pos Word: poignant
Theta 42 1.52593886842 Pos Word: magnificent
Theta 43 1.51546756855 Pos Word: chilling
Theta 44 1.49776799145 Pos Word: thematic
Theta 45 1.49776799145 Pos Word: strengths
Theta 46 1.47714870425 Pos Word: beautifully
Theta 47 1.46140034728 Pos Word: weaves
Theta 48 1.46140034728 Pos Word: unassuming
Theta 49 1.46140034728 Pos Word: topping
```

```
Theta 50 1.46140034728 Pos Word: tool
Theta 51 1.46140034728 Pos Word: sparked
Theta 52 1.46140034728 Pos Word: soviet
Theta 53 1.46140034728 Pos Word: skarsgard
Theta 54 1.46140034728 Pos Word: separately
Theta 55 1.46140034728 Pos Word: rebels
Theta 56 1.46140034728 Pos Word: outlook
Theta 57 1.46140034728 Pos Word: noah
Theta 58 1.46140034728 Pos Word: mustsee
Theta 59 1.46140034728 Pos Word: meryl
Theta 60 1.46140034728 Pos Word: linney
Theta 61 1.46140034728 Pos Word: layered
Theta 62 1.46140034728 Pos Word: jordans
Theta 63 1.46140034728 Pos Word: jo
Theta 64 1.46140034728 Pos Word: jabba
Theta 65 1.46140034728 Pos Word: hypocrisy
Theta 66 1.46140034728 Pos Word: hesitation
Theta 67 1.46140034728 Pos Word: hawthorne
Theta 68 1.46140034728 Pos Word: gunplay
Theta 69 1.46140034728 Pos Word: gretchen
Theta 70 1.46140034728 Pos Word: flynt
Theta 71 1.46140034728 Pos Word: existed
Theta 72 1.46140034728 Pos Word: envy
Theta 73 1.46140034728 Pos Word: drawback
Theta 74 1.46140034728 Pos Word: divine
Theta 75 1.46140034728 Pos Word: devilish
Theta 76 1.46140034728 Pos Word: carver
Theta 77 1.46140034728 Pos Word: aladdin
Theta 78 1.46140034728 Pos Word: performed
Theta 79 1.44540000593 Pos Word: terrific
Theta 80 1.4236600193 Pos Word: spike
Theta 81 1.4236600193 Pos Word: seamless
Theta 82 1.4236600193 Pos Word: reminder
Theta 83 1.4236600193 Pos Word: recalls
Theta 84 1.4236600193 Pos Word: mulan
Theta 85 1.4236600193 Pos Word: missteps
Theta 86 1.4236600193 Pos Word: keen
Theta 87 1.4236600193 Pos Word: fascination
Theta 88 1.4236600193 Pos Word: elliot
Theta 89 1.4236600193 Pos Word: detract
Theta 90 1.4236600193 Pos Word: deft
Theta 91 1.4236600193 Pos Word: cheerful
Theta 92 1.4236600193 Pos Word: captures
Theta 93 1.4236600193 Pos Word: addresses
Theta 94 1.40424193344 Pos Word: regard
Theta 95 1.40424193344 Pos Word: gripping
Theta 96 1.40424193344 Pos Word: depicted
Theta 97 1.40424193344 Pos Word: innocence
Theta 98 1.37438897029 Pos Word: refreshing
```

```
Theta 99 1.37438897029 Pos Word: breathtaking
Theta 100 1.34361731162 Pos Word: workings
```

Part 7

Our goal is to show the given word2vec embeddings work for figuring out whether a word t appears together with another word w or not using Logistic Regression.

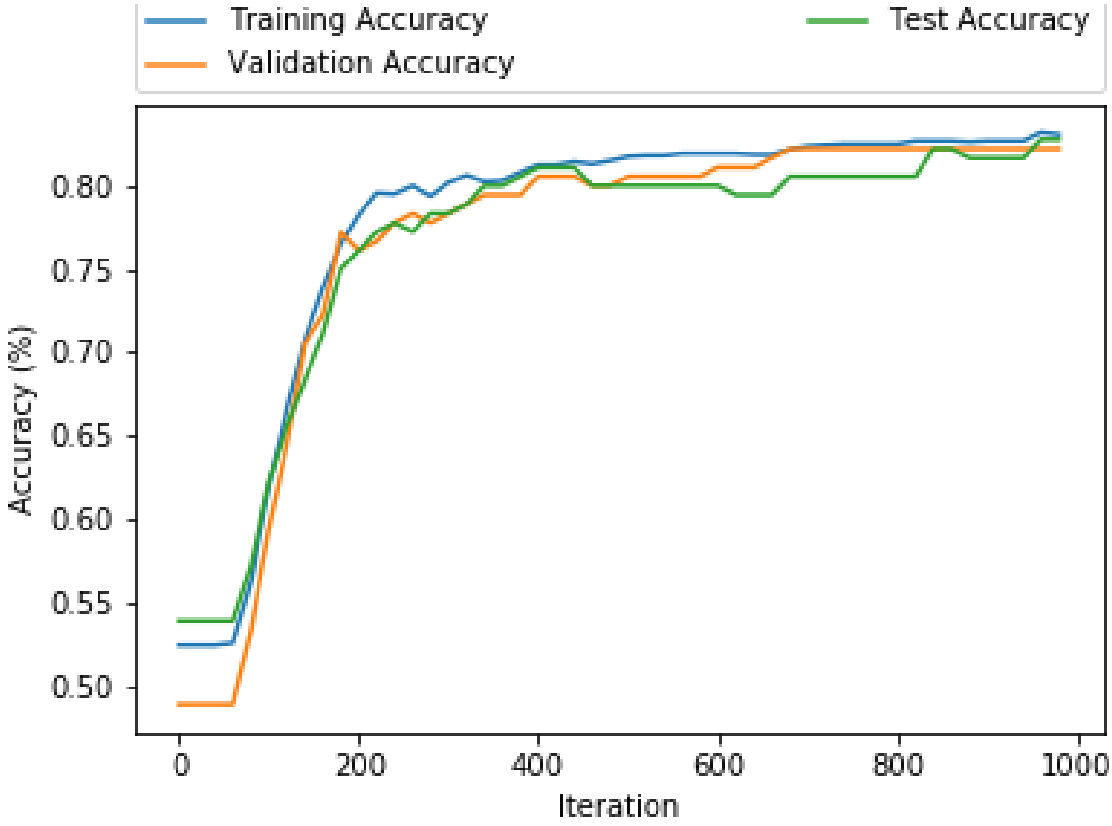
Experiment:

Data preparation:
Considering the running time, we restricted our dataset to a reasonable size, while maintaining its demonstration ability. Since we are evaluating each word's context, there is no difference between selecting words from positive or negative reviews for our experiment. Without loss of generality, we randomly sampled 500 positive reviews, and took a small random set of words appeared in these sampled reviews as test vocab, all words in this set should be within the given embeddings. (To make things easier, we chose words with higher than average frequencies, so that there will be enough ([context], target) pairs to investigate) Then we walked through each pair of adjacent words ([context], target) in the sampled reviews, if target exists in test vocab, we add the context word(within the given embeddings) to a list corresponding with this target. To create samples of adjacent words, for each target word in test vocab, we paired it up with each of its context words in the list, and label 1 for each pair. To create samples of unadjacent words, for each word in test vocab, we randomly sampled 500 words in the given embeddings. If the sampled word not adjacent with target, then we create a pair, and add label 0. Randomly shuffle the dataset. We took a training set with 1600 pairs of words, a validation set with 180 pairs and a test set with 180 pairs.

Logistic Model:
The logistic model used is very similar with the one from Part4.
The input layer is of size 128*128, and is fully connected to the output layer. We are still using one-hot encoding for the outputs: [0, 1], [1, 0] represent adjacent and unadjacent pairs respectively.
With experiments, we noticed the performance rate is a little higher with L1 regularization, so we chose to use Lasso regularization for our model. Tuning the parameters with grid search, We finalized λ= 0.0075 for the decay penalty and $\alpha = 0.005$ for gradient descent.

Performance:
After 1000 iterations,
The final performance on the training set is: 0.830625
The final validation set accuracy is: 0.822222
The final performance on the test set is: 0.827778
The learning curves of logistic regression model are shown in the following page.

Thus, we conclude that the given embeddings are useful for predictions for context of words.


Training Accuracy
Validation Accuracy
Test Accuracy
Accuracy (%)
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
200
400
600
800
1000
Iteration

Part 8

We took the embedding for each word with its embedding given, and calculate the cosine distance from this vector to the vector of embedding for "story" and "good" respectively. The smaller distance with the target embedding, the more similar in context are considered.

Results shown below: (ordered with similarity in decrease)
10 words that appear in similar contexts with story:
('plot', 0.38237626762673049)
('film', 0.48049664497375488)
('benito', 0.52907824516296387)
('simmer', 0.54510831832885742)
('sitter', 0.54638951463824337)
('lift', 0.56182195999360562)
('domineering', 0.56201087689270057)
('ricci', 0.56806393646431719)
('interviews', 0.56924068927764893)
('acclaim', 0.56942308362859984)
==========
10 words that appear in similar contexts with good:
('bad', 0.47773188352584839)
('great', 0.50537475943565369)
('wonderful', 0.52286860346794128)
('reinforcing', 0.53241243958473206)
('decent', 0.53433024883270264)
('funny', 0.53938731785937066)
('manipulate', 0.54369816444505248)
('underused', 0.54780560731887817)
('admiral', 0.5600935554560158)
('perplexing', 0.570656418800354)

Two more examples:
Using the same technique of looking for minimum distance among embedding vectors, we found the following interesting relationships:

1. word2vec(men) - word2vec(women) = word2vec(man) - word2vec(woman)
2. word2vec(he) - word2vec(him) = word2vec(she) - word2vec(her)

Code and experiment results shown below

```
# Want to find a, b, c, d,  s.t. a+b = c+d
def find_examples(a, b, c):
    a_ind = list(indices.keys())[list(indices.values()).index(a)]
    b_ind = list(indices.keys())[list(indices.values()).index(b)]
    c_ind = list(indices.keys())[list(indices.values()).index(c)]
    combined = embeddings[a_ind, :] + embeddings[b_ind, :] - embeddings[c_ind, :]
    likely = {}
    for i in range(embeddings.shape[0]):
        s_em = embeddings[i, :]
        dist = scipy.spatial.distance.cosine(s_em, combined)
```

```
        likely[indices[i]] = dist
    most_likely = sorted(likely.items(), key=operator.itemgetter(1))[1:4]
    print (most_likely)

find_examples("men", "woman", "women")
find_examples("she", "him", "her")
```

```
[('men', 0.40587792116513122), ('gesture', 0.56893305567265173), ('man', 0.58096081473026873)]
[('he', 0.40665561029182806), ('him', 0.4891522414810473), ('it', 0.49161990854717386)]
```