Yuechen Zhao
1126125
zzyc1993@uw.edu
CSE 446 HW2

1.

(1)

P(X-ray Shadow | Bronchitis) = 2/3
P(Dyspnea | Bronchitis) = 2/3
P(Lung inflammation | Bronchitis) = 3/3 = 1

P(X-ray Shadow | Tuberculosis) = 2/3
P(Dyspnea | Tuberculosis) = 1/3
P(Lung inflammation | Tuberculosis) = 1/3

(2)

P(Bronchitis) = 1/2 * 2/3 * 1/3 * 1 = 1/9
P(Tuberculosis) = 1/2 * 2/3 * 2/3 * 1/3 = 2/27
P(Bronchitis) > P(Tuberculosis)
Thus Based on this learned Naive Bayes model, this patient is more likely to have Bronchitis.

(3)

One sign of overfitting in Naïve Bays model is having zero or one conditional possibilities. My model has this problem, where P(Lung inflammation | Bronchitis) = 1. If a typical Bronchitis patient does not have lung inflammation, the zero term would dominate and give the wrong diagnosis.

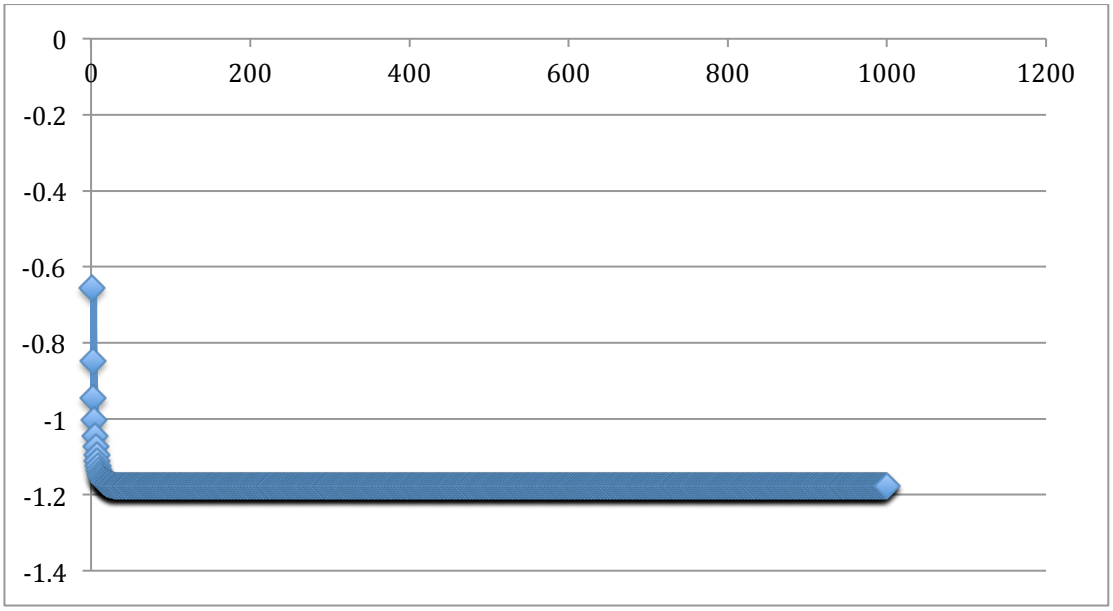
2.
2.4

(1)

$$w_i^{t+1} = w_i^t + \eta \times (\sum_j x_i^j (y^j - \frac{\exp(w0 + \Sigma wixi)}{1 + \exp(w0 + \Sigma wixi)}) - \lambda w_i^t)$$

(2)

PART A



PART B

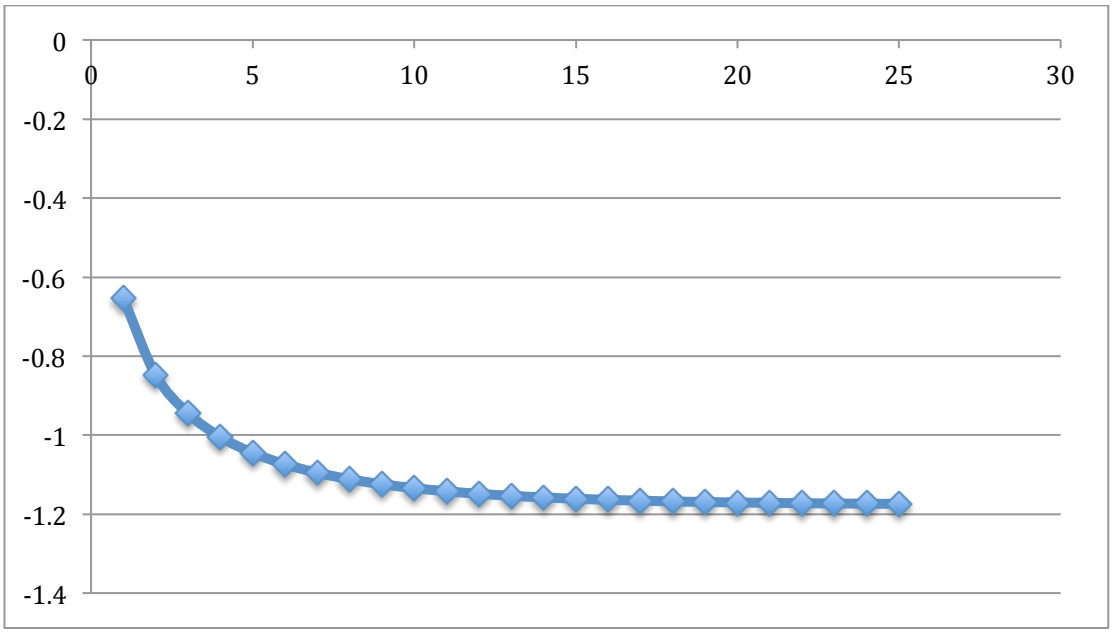
After 1000 iterations, the weights predict the test data to have 0 clicks, and 1000 no-clicks. The number of correct predictions is 946. Thus the prediction is 94.6% accurate.

(3)

PART A:

The number of iteration until threshold achieved is 24.

PART B:



PART C:

After 1000 iterations, the weights predict the test data to have 0 clicks, and 1000 no-clicks, which is the same as the above value. The number of correct predictions is 946. Thus the prediction is 94.6% accurate.

(4)

For λ = 0, the l^2 norm = $\sqrt{w1^2 + w2^2 + w3^2 + \cdots}$ = 1.01133732478
For λ = 0.3, the l^2 norm = $\sqrt{w1^2 + w2^2 + w3^2 + \cdots}$ = 0.5195872482

I would argue that λ = 0.3 is more preferred. Because each time an update occurs, a non-zero λ pushes back the weights to its original value a bit, which would take into account prior beliefs. Also, it adds a non-zero term to the log loss function, which would take into account large weights values. Also when λ = 0, the weight vector tends to have relatively large updates in every step, so λ = 0.3 is generally more preferred.

2.5:

(1)

For class 1:
Precision = truePositive / #elements labeled to positive = 0 / 0,
which is undefined.
Recall = truePositive / #elements actually belong = 0 / 54 = 0.

For class 0:
Precision = trueNegative / #elements labeled to negative = 946/1000
= 0.946
Recall = trueNegative / #elements actually belong = 946/946 = 1

(2)

After 2 hours of 5000 iterations, the result is the same with the previous one.

For class 1:
Precision = truePositive / #elements labeled to positive = 0 / 0,
which is undefined.
Recall = truePositive / #elements actually belong = 0 / 54 = 0.

For class 0:
Precision = trueNegative / #elements labeled to negative = 946/1000
= 0.946
Recall = trueNegative / #elements actually belong = 946/946 = 1

3.

(1)

For the perceptron that is trained on the oversampled data

For class 1:

Precision = 0.0552825552826

Recall = 0.833333333333

For class 0:

Precision = 0.951612903226

Recall = 0.18710359408

Unlike the logistic regression model, which predicts non-clicks dominantly, the perceptron model tends to predict more evenly. However, the weight vector tends to have relatively bigger update in each step, and it does not converge very quickly. Also, the perceptron model is much easier to implement, and it runs much faster than logistic model.

(2)

First of all, I set a hyperparameter T = 1…1000, and I used a development set. I trained the data using part of the training set, and recorded the maximum sum of recalls for class 1 and 0 (of the test set). Then, I used the development set to verify the max sum of recalls. Then I re-ran the training set, and "break"-ed the outermost loop (before it hits T = 1000) once the sum of recalls reached the max value. Therefore, I can record the weights that maximize the sum of recalls for class 1 and 0 of test set.

(3)

My perceptron does not seem to converge when run on the oversampled training data. This might be because the oversampled training data is not general enough, and thus the points are not separable. Therefore the separating plane keeps changing constantly, and cannot converge.

(4)

I will prefer perceptron first, because perceptron is much easier to implement, and perceptron runs much faster than logistic regression. Though perceptron might not be a good choice if the data points are not separable, it is worth the try.