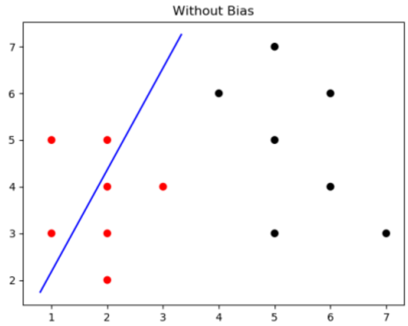
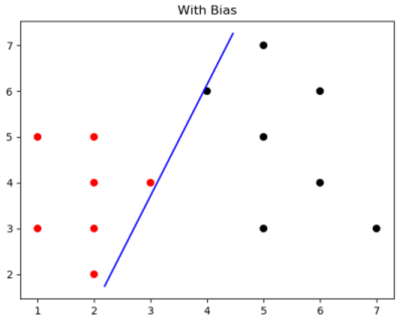
Haoran Wang
00274-00605

1. Perceptron (40 Points)
1.1 Perceptron Details (20 Points)

1. The equation with bias: f(x)=$\begin{cases} 1\ if\ bias + \sum w_j x_j \geq 0 \\ 0\ if\ bias + \sum w_j x_j < 0 \end{cases}$

   Bias shifts the decision boundary away from the origin, in the direction of w.

2. For graph (a) and (b), it won't matter, because those data sets cannot be linearly separated. So, in order to separate those two, we need a multi-layer perceptron. For graph (c), a perceptron without bias will get high accuracy because based on the plot, it is evenly distributed along 0. For graph (d), a perceptron with bias will get high accuracy because based on the plot, the decision boundary is approximately 3.5. Therefore, if we add a bias of positive 3.5, it will classify more accurately than no bias.

3. We can engineer a new feature by caluculating the correlation between (city, state) and priceRange. This new feature can provide us information about the realtion between geometric information and local economy which is reflected by priceRange.

4. Weights denote a vector of weights for each feature in the data set. The weight for each feature shows how it is going to affect activation. Zero weight means the activation is the same regardless of the value of this feature. Positive weight will cause the activation to increase along the direction of the vector. Negative weight will cause the activation to decrease along the direction of the vector. Bias denote the adjustment on activation and cause the plane to shift.
   We need to update weights and bias in a perceptron so that in the end, the perceptron will converge if it is linearly seperatable and correctly classify every training example.
   In line 8 and 9, we update the weights and bias accordingly. After some iterations, the error will become 0 due to updated weights and bias and the iteration will stop. Therefore, the perceptron is converged if it is linearly seperatable.

5. a)

Pseudocode for Average Perceptron:

$function\ TRAIN(D, MaxIter)$

// initialize weights and bias

$W_i \leftarrow 0, for\ all\ i = 1, \dots, n$

$b \leftarrow 0$

// initialize cached weights and bias

$U_i \leftarrow 0, for\ all\ i = 1, \dots, n$

$c \leftarrow 0$

// initialize counter

$counter \leftarrow 1$

$for\ iter = 1, \dots, MaxIter\ do$

$for\ all\ (x, y) \epsilon D\ do$

$error \leftarrow y - f(x)$

$if\ error\ then$

// update weights and bias

$b \leftarrow b + error$

$w_i \leftarrow w_i + (error \times x_i), for\ all\ i = 1, \dots, n$

// update cached weights and bias

$c \leftarrow c + error \times counter$

$U_i \leftarrow U_i + (error \times x_i) \times counter, for\ all\ i = 1, \dots, n$

end if

$counter \leftarrow counter + 1$

end for

end for

return $W_i - \frac{1}{counter} U_i\ for\ all\ i = 1, \dots, n$, $b - \frac{1}{count} c$

b)

The advantage of average perceptron is that we can simply maintain a running sum of the averaged weight vector and averaged bias. It will generalize better to test data because the weight vectors can survive a long time to gain more accuracy than weight vectors that are overthrown quickly.

2. Naïve Bayes (40 Points)

2.1 Naïve Bayes Details (20 Points)

1. P(Y|X) = MAP(Y) = argmax(P(X|Y)*P(Y))

2. Let C_i denote the class label

   Let $W_0, W_1, W_2, \dots, W_n$ denote different features of a given row W

   P($C_i|W$) = $\frac{P(W|C_i)P(C_i)}{P(W)}$ = $\frac{P(W_0, W_1, \dots, W_n|C_i)P(C_i)}{P(W)}$

   Because we assume conditional independence,

   $P(W_0, W_1, \dots, W_n|C_i) = P(W_0|C_i)P(W_1|C_i) \dots P(W_n|C_i)$

   P($C_i|W$) = $\frac{P(W_0|C_i)P(W_1|C_i) \dots P(W_n|C_i) * P(C_i)}{P(W)}$

   Also, use LaPlace smoothing and log probability

   where P(W) = 1/number of rows in the table

   P(Class=0) = 1- P(Class=1)

   Pick the value of C_i for which P(W)*P(C_i|W) is maximum

3. We assume conditional independence, meaning each attribute is independent of each other. Therefore, $P(W_0, W_1, \ldots, W_n|C_i) = P(W_0|C_i)P(W_1|C_i) \ldots P(W_n|C_i)$. This assumption is not true because the attributes interact with each other. Such as stars and noiseLevel affect each other. However, this assumption is necessary for us to implement naïve bayes classifier

4. P(C_i) is the prior probability of that class.
   Smoothing is to handle the situation where the value of a certain attribute has zero probability because it doesn't appear in training data. It will give more accurate final probabilities.

5. We need to calculate P($X_i|GoodForGroup$) (14 parameters)and P($X_i|\neg GoodForGroup$) (14 parameters)as well as P($GoodForGroup$) (1 parameter) and P($\neg GoodForGroup$)) (1 parameter), X_i being the attributes.
   NBC parameters = CPDs + prior
   Therefore, there are 29 parameters that needs to be calculated.

6. $P(alcohol \mid goodForGroups) = \prod_{j=1}^{k} goodForGroups_j^{I(alchol=j)}$, where I(alcohol = j) is an indicator function
   $\frac{P(alcohol \mid goodForGroups)+1}{P(goodForGroups)+k}$,where k is the number of possible values for alcohol

7. a) stars
   with smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_attr.py train-set.csv stars
CPD: stars = 0.4372
```

   without smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_wo_sm_attr.py train-set.csv stars
CPD: stars = 0.4377
```

   b) waiterService
   with smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_attr.py train-set.csv waiterService
CPD: waiterService = 0.5997
```

   without smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_wo_sm_attr.py train-set.csv waiterService
CPD: waiterService = 0.5999
```

c) caters
with smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_attr.py train-set.csv caters
CPD: caters = 0.9356
```

without smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_wo_sm_attr.py train-set.csv caters
CPD: caters = 0.9359
```

d) attire
with smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_attr.py train-set.csv attire
CPD: attire = 0.4971
```

without smoothing:

```
wang2226 at data in ~/cs373/wang2226-hw4
$ python nbc_wo_sm_attr.py train-set.csv attire
CPD: attire = 0.4973
```

Smoothing does not affect CPD significantly in those attributes. Most of the values of each attribute already appear in the training set. Therefore, smoothing doesn't have a huge effect over those four attributes. Caters shows the most association with the class.

3. Analysis (20 Points)

Q1

a)

trainin set size	mean zero-one loss(NBC)	mean zero-one loss(Avg)
1%	0.22727	0.23351
10%	0.22333	0.20431
50%	0.22445	0.19455
70%	0.22879	0.19425

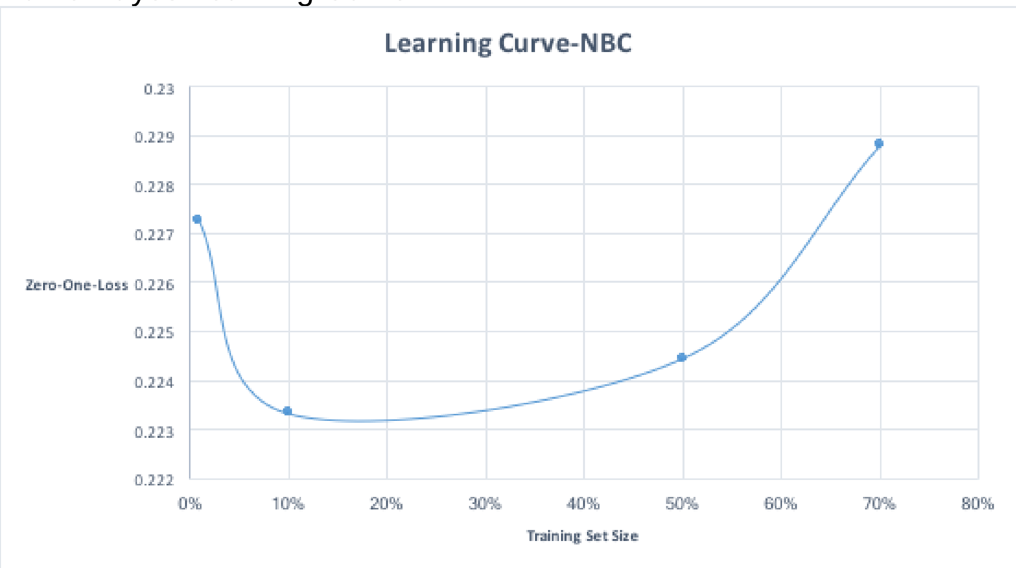
b)

train set size	mean square loss(NBC)
1%	0.68848
10%	0.69596
50%	0.69661
70%	0.69675

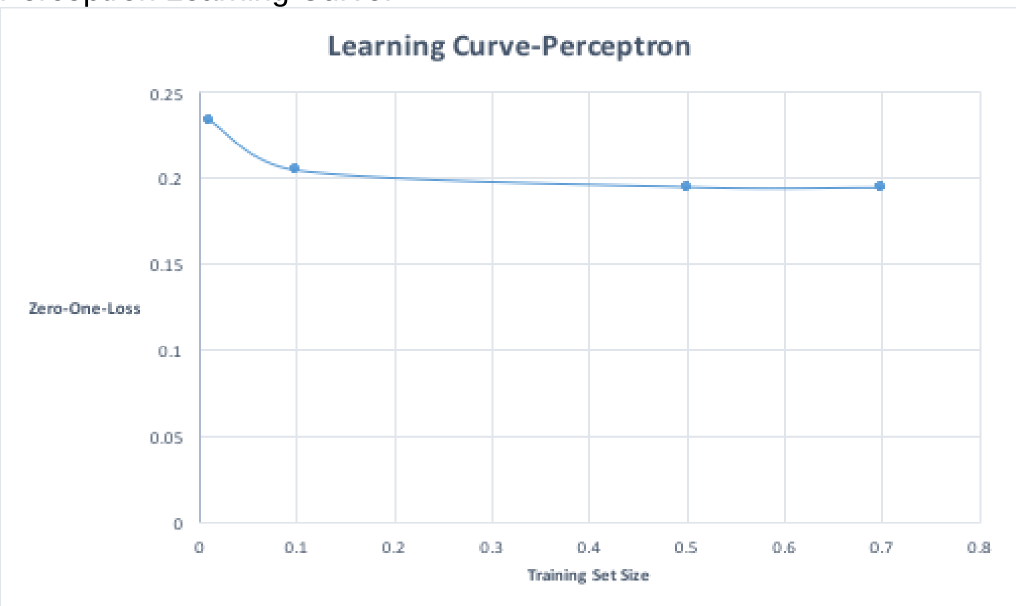
Q2

a)

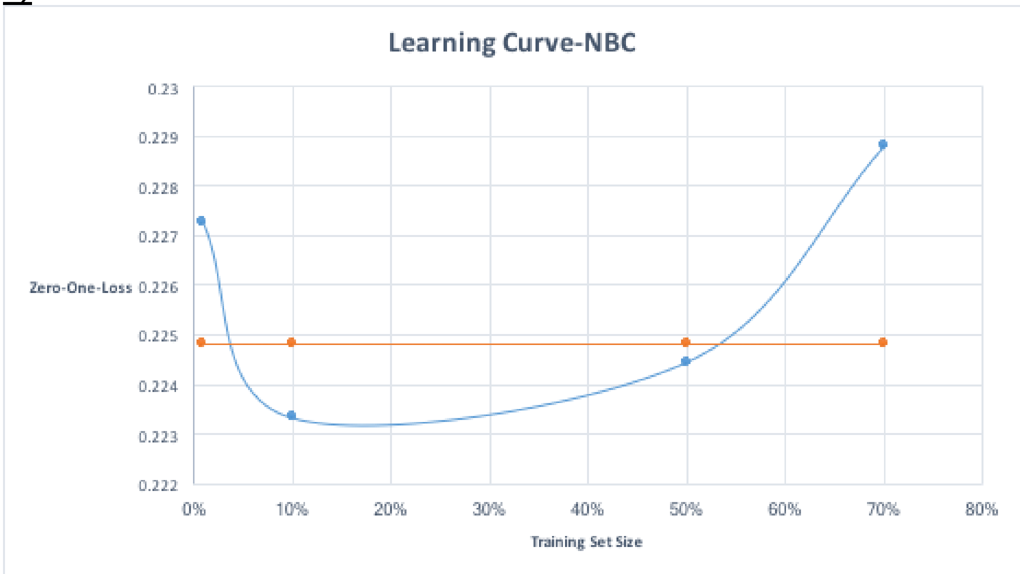
Naïve Bayes Learning Curve:

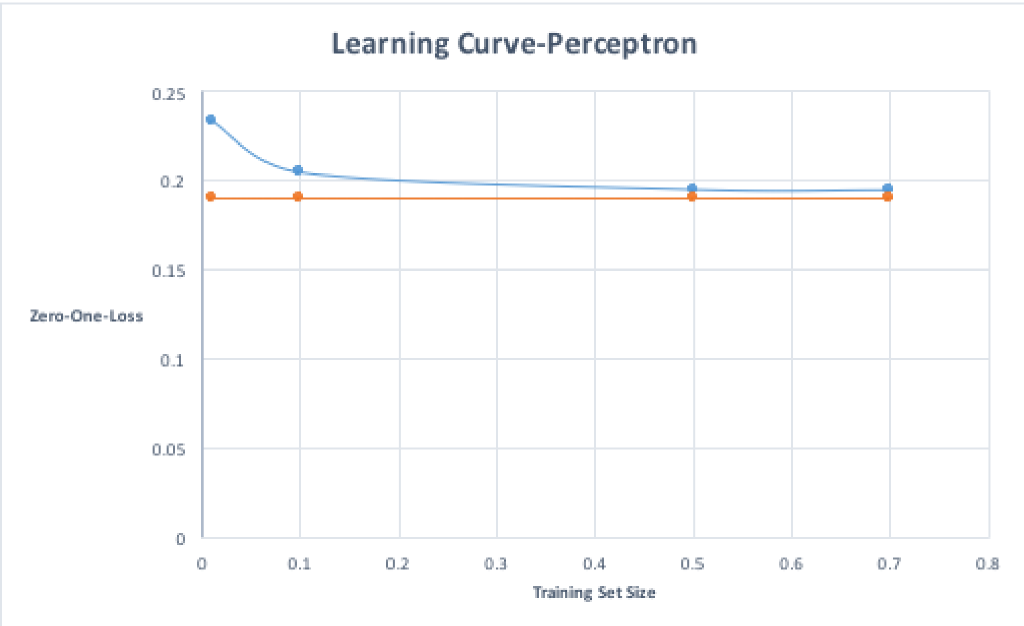


Perceptron Learning Curve:



b)

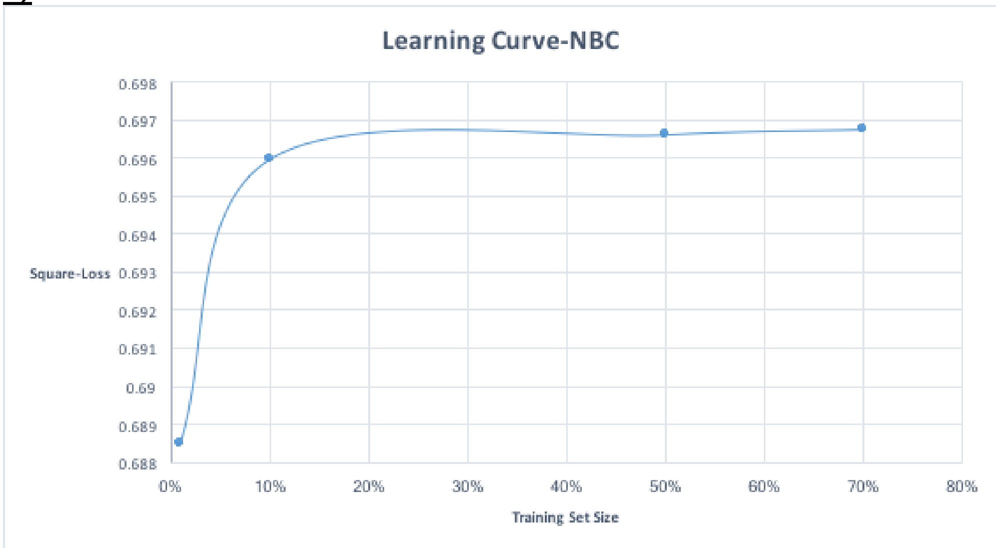




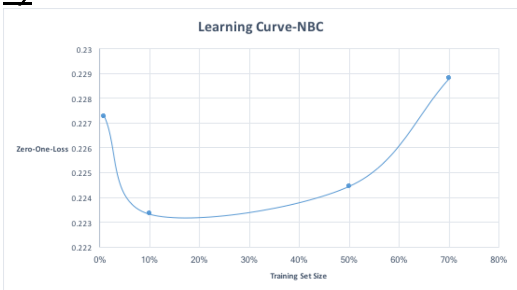
c)
Naïve Bayes has two intercept points with baseline and overall it is not close to baseline. Perceptron is close to baseline overall and getting closer with training set size increase.
Naïve Bayes has lowest zero-one loss at 10% training set size and has a overall shape of a 'Nike' function. Percetron has decreasing zero-one loss with training set size increasing.

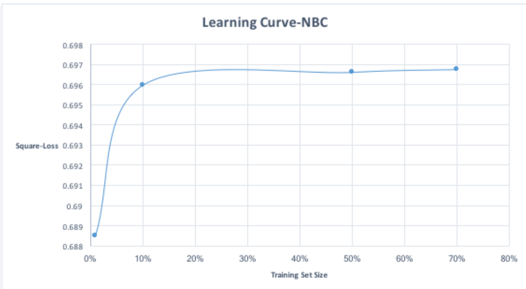
Q3

a)



b)





(zero-one loss) *(squared loss)*

Based on the graph above, zero-one loss has its lowest at 10%, squared loss has its lowest at 1%. Squared loss performs better because it almost converges after roughly 20%. On the other hand, zero-one loss decrease first and than increase without showing any consistency. I think squared loss is a better metric because it shows how the classifier perform accurately.