CMPS 242 Third Homework, Fall 2018

Weiting Zhan (wzhan83@ucsc.edu)
Constant Maximilien Kuakou Bossou (cbossou@ucsc.edu)
Fahim Hasan Khan (fkhan4@ucsc.edu)

Wednesday November 7 2018

Problem 1

Hack the Titanic is a problem in the Kaggle website for predicting who survived the Titanic disaster using machine learning. If we take a look at the Leaderboard, there are several teams who achieve 100 percent accuracy in this problem.

There are ways to obtain a perfect score without using any sophisticated machine learning algorithms at all. One of the methods to achieve a perfect leaderboard score without learning from the data would be using the real data or fact. In this case, the Passenger Id and the survival information can be generated as a one-to-one mapped dataset based on the real-life information. The Titanic database is very public knowledge, and the full dataset is available from many sources on the internet.

We have tested a simple learning model without any optimization implemented in Python using SKlearn Support Vector Machine (SVM) based on the Age, Passenger class, Sex and Fare and the accuracy was 0.79 percent which is close to the average accuracy (78-85%) other models in the leaderboard. The scores with accuracy 100 percent are over-fitting examples. If a model is "overfit" to a dataset, then it is not going to work effectively outside of the dataset it is trained on. This means that the model would have low accuracy on another sample of data taken from a similar dataset. So, that would imply the machine learning model is not successful. Kaggle only uses 50 percent of the test data for the public leaderboard to avoid showing the actual accuracy of the overfitted models until the contest is over.

Problem 2

Perceptron algorithm with noise experiment. We Implemented the Perceptron algorithm presented in class using Python based on the simple concept, $\mathbf{w}_{new} := \mathbf{w}_{old} + t\mathbf{x}$ when the algorithm makes a mistake for examples with 15 features. We have create a set of 300 instances for part (a),(b) and (c) and another set of 500 instances where the instances are selected uniformly from the boolean

hyper-cube $\{\pm 1\}^{15}$. The test sets and training sets are generated using Python and saved as CSV dataset to be used by the Perceptron.

Part (a)

We have labeled our 300 instances with the first component of the instances, so $t_i = x_{i,1}$ to create a labeled sample. For this experiment, only 1 epoch occurred with 10 prediction errors before producing a hypothesis that correctly labels the training set.

Part (b)

We have relabeled the 300 instances training set to $t_i = +1$ or -1 using the given condition. In this experiment, 3 epochs occurred Before producing a hypothesis that correctly labels the training set. We encountered a total 73 prediction errors, 54, 14 and 5 in epochs 1,2 and 3 respectively.

It was expected that this experiment would take more epochs as the labels are mapped to all of the features of each instance, making it more complicated to predict. The previous one was simple one-to-one mapping with only the first feature of each instances.

Part (c)

We have created a third training set of with 300 instances with label noise based on the given formula and ran three epochs through this noisy training set by shuffling the order of the training set between epochs and storing the 900$\mathbf{w}$ vectors produced by the algorithm. Then we created the 5 hypothesis as instructed.

Then, we have created a test set of 500 new instances with random label noise and tested the 5 hypothesis for evaluating their accuracy. The result of the experiment is given in the table below.

Name of Hypothesis	Number of incorrect predictions	Percentage of Error
Last Hypothesis	148	29.6
Average hypothesis	119	23.8
Voted hypothesis	126	25.2
Last epoch average	109	21.8
Last epoch vote	118	23.6

We can observe that the *last epoch average* has the lowest error rate and thus, the highest accuracy. The *last hypothesis* has the highest error rate and thus, the lowest accuracy, which is expected.

Problem 3

We have generated a 100-example training set using given parameters for linear regression.

Part (a)

We perform MLE estimation of $\mathbf{w}$ and β. We have used the optimize module from *scipy* for this task.

After we made some tests, we realized that in the case of 1000 training points, the parameters estimate of $w1$, $w2$ and $w3$ is pretty close to its original values 2, 1, 3. But the parameters $w0$ and β does not converge to its original parameters 0.2 and 1. Instead, they stayed close to their initially randomly chosen values. When we used 10000 training points, we realized that all the estimated values get closer to their original values, but the overall gain is small. Whereas $w1$, $w2$ and $w3$ get much more closer to their original values 2, 1, 3, $w0$ and β does not converge to their original values still but moved a bit towards of those values.

Part (b)

We repeated the first part considering additional parameters in the model.

In this case, only $w2$ and $w3$ converge to their original values. The rest of the parameters ($w1$, $w4$, $w5$ and β) don't. $w4$ and $w5$ were not originally used in the data generation, therefore it is expected their estimated values do not make much sense for us. However, just like in part a and for 1000 training points, $w0$ and β stayed close to their initially randomly chosen values. When we used 10000 training points, we realized that $w1$, $w2$ and $w3$ get much more closer to their original values 2, 1, 3, $w0$ and β does not converge to their original values still but moved a bit towards of those values. Also the values of $w4$ and $w5$ remain meaningless for the experiment.

Part (c)

We estimated $w = (w0; w1; w2; w3)$ using Bayesian linear regression.

Using Bayesian linear regression, all of the w vector components are very close to their original values 0.2, 2, 1, 3, 1. Hence, the generated curve based on estimated w and the "true" curve (with the original values) are very similar. In particular, in the case of 100 training points, the curves align less. In the case of 1000 training points, the curves seem to perfectly align. And in the case of 10000 training points, we no longer notice any difference between the two curves. The curves of the fifth degree model do not follow the same pattern as the curves of the third degree model. This is due to the fact that $w4$ and $w5$ were not part of the original parameters.

Problem 4

We are trying to find the probability that email is spam given the text in the email. Bayes Theorem notation:

$$p(Hypothesis|Evidence) = \frac{p(Evidence|Hypothesis)p(Hypothesis)}{p(Evidence)} \tag{1}$$

In this email classification problem,

$$p(Spam|\omega_1, ..., \omega_n) = \frac{p(\omega_1, ..., \omega_n|Spam)p(Spam)}{p(\omega_1, ..., \omega_n)} \tag{2}$$

$$p(Ham|\omega_1, ..., \omega_n) = \frac{p(\omega_1, ..., \omega_n|Ham)p(Ham)}{p(\omega_1, ..., \omega_n)} \tag{3}$$

In order to simplify the math, we make Naive Bayes assumption that each word is independent of all other words.

By making the Naive Bayes, we can break down the numerator into the following.

$$p(Spam|\omega_1, ..., \omega_n) = \frac{p(\omega_1|Spam)p(\omega_2|Spam)...p(\omega_n|Spam)p(Spam)}{p(\omega_1, ..., \omega_n)} \tag{4}$$

$$p(Ham|\omega_1, ..., \omega_n) = \frac{p(\omega_1|Ham)p(\omega_2|Ham)...p(\omega_n|Ham)p(Ham)}{p(\omega_1, ..., \omega_n)} \tag{5}$$

Our decision rule is :

$$p(Spam|\omega_1, ..., \omega_n) \begin{cases} > p(Ham|\omega_1, ..., \omega_n) & \text{Spam,} \\ = p(Ham|\omega_1, ..., \omega_n) & \text{Unable to predict} \\ < p(Ham|\omega_1, ..., \omega_n) & \text{Ham} \end{cases} \tag{6}$$

We can get rid of the denominator since its only purpose is to scale the numerator.

So we get:

$$p(Spam|\omega_1, ..., \omega_n) \propto p(\omega_1|Spam)p(\omega_2|Spam)...p(\omega_n|Spam)p(Spam) \tag{7}$$

$$p(Ham|\omega_1, ..., \omega_n) \propto p(\omega_1|Ham)p(\omega_2|Ham)...p(\omega_n|Ham)p(Ham) \tag{8}$$

Since the value of probability is in between [0,1], multiply probability 100 times, for example, the calculation of $p(Spam|\omega_1, ..., \omega_n)$ and $p(Ham|\omega_1, ..., \omega_n)$,have the risk of to end up to zero. To prevent this, we use log probability. We have the property of logarithm:

$$\log(xy) = \log(x) + \log(y) \tag{9}$$

Apply (9) to equation (7) and (8) by taking the log of each side of the equation:

$$\log p(Spam|\omega_1, ..., \omega_n) \propto \log p(Spam) + \sum_{i=1}^{n} \log p(\omega_i|Spam) \tag{10}$$

$$\log p(Ham|\omega_1, ..., \omega_n) \propto \log p(Ham) + \sum_{i=1}^{n} \log p(\omega_i|Ham) \tag{11}$$

For the prior probability of spam $P(Spam)$ and ham $P(Ham)$ are calculated by counting how many messages are spam/ham, dividing by the total number of messages.

$$p(Spam) = \frac{N_{spam}}{N_{Spam} + N_{Ham}} \tag{12}$$

$$p(Ham) = \frac{N_{Ham}}{N_{Spam} + N_{Ham}} \tag{13}$$

N_{Spam}: the total number of Spam in the data set.
N_{Ham}: the total number of Spam in the data set.
Define the condition probability, $P(\omega_i|Spam)$ and $p(\omega_i|ham)$:

$$p(\omega_i|Spam) = \frac{T_{\omega_i+1}}{\sum_{\omega \in spam} T_{\omega_i+1}} \tag{14}$$

$$p(\omega_i|Ham) = \frac{T_{\omega_i+1}}{\sum_{\omega \in Ham} T_{\omega_i+1}} \tag{15}$$

T_{ω_i} : the number of occurrences of word i ω_i in the training data from class Spam.

Part (a) Decisions

Algorithm:

1. Compute Spam and Ham priors by using equation 12 and 13.

2. For each (email , label) pair, get rid of the punctuation, and stop words ,tokenize the document into words.

3. From the training data, for each word, either add it to the vocabulary for spam/ham dictionary , and add the word to the global dictionary.And count the frequency of the word. After iterate the training data, we get a Spam dictionary, Ham dictionary and global dictionary.

4.For the test data, apply Naive Bayes as described above. For example, given a document, we need to iterate each of the words and compute $\log p(\omega_i|Spam)$ and sum $\log p(\omega_i|Spam)$ all up. We also compute $\log p(\omega_i|Ham)$

and sum $\log p(\omega_i|Ham)$ all up. Then we add the log Spam priors and log Ham priors to get the score.

5 . use the decision rule to predict the label is Spam or Ham:

Our decision rule is :

$$p(Spam|\omega_1,...,\omega_n)\begin{cases} > p(Ham|\omega_1,...,\omega_n) & \text{Spam,} \\ = p(Ham|\omega_1,...,\omega_n) & \text{Unable to predict} \\ < p(Ham|\omega_1,...,\omega_n) & \text{Ham} \end{cases} \tag{16}$$

Part (b) Accuracy on the test set

$$Accuracy = \frac{TP + TN}{TP + FP + FN + FN} \tag{17}$$

TP: True positive,prediction is Spam and it's spam
TN: True negative, prediction is Spam and it's ham
FP: false positive,prediction is Ham and it's ham
FN: False Negative, prediction is Ham and it's Spam
Corpus size = 15525 emails;
Collected 15525 feature sets;
Training set size = 12420 emails;
Test set size = 3105 emails;
Accuracy on the training set = 0.7981481481481482;
Accuracy of the test set = 0.7716586151368761;

Part (c) Prior probabilities for Spam and Ham

For the prior probability of spam $P(Spam)$ and ham $P(Ham)$ are calculated by counting how many messages are spam/ham, dividing by the total number of messages.

$$p(Spam) = \frac{N_{spam}}{N_{Spam} + N_{Ham}} \tag{18}$$

$$p(Ham) = \frac{N_{Ham}}{N_{Spam} + N_{Ham}} \tag{19}$$

N_{Spam}: the total number of Spam in the data set.
N_{Ham}: the total number of Spam in the data set.

Part (d) Without and with Laplace smoothing

Without Laplace Smoothing: Since log of 0 is undefined. The code will have errors when running the algorithm. For example, if the world "Happy" never appeared in the spam of training data, then the

$$p(\omega_i|Ham) = 0 \tag{20}$$

which leads to:

$$\log p(\omega_i|Ham) = \log 0 \tag{21}$$

There will be error by running this way. ValueError: math domain error

With Laplace Smoothing: Define the condition probability, $P(\omega_i|Spam)$ and $p(\omega_i|ham)$: known as Laplace Smoothing, there will not be situation like $\log 0$:

$$p(\omega_i|Spam) = \frac{T_{\omega_i+1}}{\sum_{\omega \in spam} T_{\omega_i+1}} \tag{22}$$

$$p(\omega_i|Ham) = \frac{T_{\omega_i+1}}{\sum_{\omega \in Ham} T_{\omega_i+1}} \tag{23}$$

Part (e) the most discriminate words based on the learned probabilities

The discriminative words are those that have very different probabilities in the two classes:

```
Most Informative Features
                kaminski = True              ham : spam   =    425.7 : 1.0
                 shirley = True              ham : spam   =    163.0 : 1.0
                     hpl = True              ham : spam   =    144.3 : 1.0
                   vince = True              ham : spam   =    139.2 : 1.0
                  valium = True             spam : ham    =     73.2 : 1.0
                 melissa = True              ham : spam   =     63.6 : 1.0
```

So our discriminative words are:kaminski, shirley, hpl, vince, valium, melissa.