Q1 / Maximum Likelihood Estimation

(a) Consider i.i.d random variables, $x_1, \dots, x_n$ ~~from~~ Gamma distribution with pdf

$$f(x;\alpha,\lambda) = \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x^{\alpha-1} \exp(-\lambda x), \; x \geq 0$$

Suppose the parameter $\alpha > 0$ is known, find the MLE of λ.

We start with the function $f(x;\alpha,\lambda) = \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x^{\alpha-1} \exp(-\lambda x)$

Then we calculate the likelihood function:

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x_i^{\alpha-1} \exp(-\lambda x_i)$$

Then we calculate the log likelihood as:

$$l(\theta) = \sum_{i=1}^{n} \log\left(\frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x_i^{\alpha-1} \exp(-\lambda x_i)\right) \Rightarrow$$

$$\Rightarrow \sum_{i=1}^{n} \log\left(\frac{1}{\Gamma(\alpha)} x_i^{\alpha-1}\right) + \sum_{i=1}^{n} \log(\lambda^{\alpha} \exp(-\lambda x_i)) \Rightarrow$$

$$\Rightarrow \sum_{i=1}^{n} \log\left(\frac{1}{\Gamma(\alpha)} x_i^{\alpha-1}\right) + \sum_{i=1}^{n} \log(\lambda^{\alpha}) + \sum_{i=1}^{n} \log(\exp(-\lambda x_i)) \Rightarrow$$

$$\Rightarrow \sum_{i=1}^{n} \log\left(\frac{1}{\Gamma(\alpha)} x_i^{\alpha-1}\right) + n\alpha \log(\lambda) - \lambda \sum_{i=1}^{n} x_i$$

Then we take the first derivative of $l(\theta)$ with respect to λ:

$$\frac{\partial l(\theta)}{\partial \lambda} = \frac{n\alpha}{\lambda} - \sum_{i=1}^{n} x_i$$

Then we take the second derivative to get:

$$\frac{\partial^2 l(\theta)}{\partial \lambda^2} = \frac{-n\alpha}{\lambda^2} < 0.$$

Once the second derivative is always negative, the log likelihood function is concave, therefore the maximum is where $\frac{\partial l(\theta)}{\partial \lambda} = 0$.

Q1) Maximum Likelihood Estimation continued

(a) cont.

Therefore we solve the equation $\frac{n\alpha}{\lambda} - \sum_{i=1}^{n} x_i = 0 = \frac{\partial \ell(\theta)}{\partial \lambda}$

to get $\lambda = \frac{n\alpha}{\sum_{i=1}^{n} x_i}$ recall $\frac{\sum_{i=1}^{n} x_i}{n} = \bar{x}$ therefore

$$\boxed{\hat{\lambda}_{MLE} = \frac{\alpha}{\bar{x}}}$$

(b) Let $x_1, \dots, x_n$ be i.i.d d-dimensional Gaussian random variables distributed according to $\mathcal{N}(\mu, \Sigma)$. That is,

$$f(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})\right)$$

find the MLE for vector $\vec{\mu}$.

We start with the function $\frac{1}{\sqrt{2\pi^d |\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})\right)$

we then get the likelihood function to be:

$L(\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi^d |\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}_i-\vec{\mu}_i)^T \Sigma^{-1} (\vec{x}_i-\vec{\mu}_i)\right)$ Then, we calculate log likelihood:

$\ell(\theta) = \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi^d |\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}_i-\vec{\mu}_i)^T \Sigma^{-1} (\vec{x}_i-\vec{\mu}_i)\right)\right)$ then we use log rules to get:

$$\ell(\theta) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T \Sigma^{-1}(\vec{x}_i-\vec{\mu})$$

Then we take the first derivative of $\ell(\theta)$ with respect to μ:

$$\frac{\partial}{\partial \mu}\ell(\theta) = \frac{\partial}{\partial \mu}\left(-n\frac{d}{2}\log(2\pi)\right) - \frac{\partial}{\partial \mu}\left(n\frac{1}{2}\log|\Sigma|\right) - \frac{\partial}{\partial \mu}\left(\frac{1}{2}\sum_{i=1}^{n}(\bar{x}_i-\mu)^T \Sigma^{-1}(\vec{x}_i-\mu)\right)$$

Recall that $\frac{\partial}{\partial a}a^T b = \frac{\partial}{\partial a}b^T a = b$, therefore $\frac{\partial}{\partial \mu}\ell(\theta)$ reduces to...

$$\frac{\partial}{\partial \mu}\ell(\theta) = \sum_{i=1}^{n}\Sigma^{-1}(x_i-\mu).$$

we then take the second derivative to get

$$\frac{\partial^2}{\partial \mu^2}\ell(\theta) \frac{\partial}{\partial \mu}\sum_{i=1}^{n}\Sigma^{-1}x_i - \frac{\partial}{\partial \mu}\Sigma^{-1}n\mu \Rightarrow \frac{\partial^2}{\partial \mu^2} = -n\Sigma^{-1} < 0$$

Therefore, once the second derivative is always negative, the maximum is where $\frac{\partial \ell(\theta)}{\partial \mu} = 0$

Q1) Maximum Likelihood Estimation Contnued

(b) cont.

$$\sum_{i=1}^{n} \Sigma^{-1}(x_i - \mu) = 0 \Rightarrow \sum_{i=1}^{n} \Sigma^{-1} x_i - \Sigma^{-1} n \mu = 0 \Rightarrow$$

$$\Rightarrow \sum_{i=1}^{n} \cancel{\Sigma^{-1}} x_i = \cancel{\Sigma^{-1}} n \mu \Rightarrow \frac{1}{n} \sum_{i=1}^{n} x_i = \mu$$ recall $\frac{1}{n} \sum_{i=1}^{n} x_i = \bar{x}$

There fore, $\boxed{\hat{\mu}_{MLE} = \bar{x}}$

ML HW2 Question 2

February 26, 2021

1 Bayesian Spam Filtering

In this problem you will apply the naive Bayes classifer to the problem of spam detection, using a benchmark database assembled by researchers at Hewlett-Packard. Download the file spambase.data from Brightspace under HW2, and issue the following commands to load the data. In Python:

[425]:

```
import numpy as np
z = np.genfromtxt('spambase.data', dtype=float, delimiter=',')
np.random.seed(0) #Seed the random number generator
rp = np.random.permutation(z.shape[0]) #random permutation of indices
z = z[rp,:] #shuffle the rows of z
x = z[:,:-1]
y = z[:,-1]
```

Here X is n x d, where n = 4601 and d = 57. The different features correspond to different properties of an email, such as the frequency with which certain characters appear. y is a vector of labels indicating spam or not spam. For a detailed description of the dataset, visit the UCI Machine Learning Repository, or Google 'spambase'.

To evaluate the method, treat the first 2000 examples as training data, and the rest as test data. Fit the naive Bayes model using the training data (i.e., estimate the class-conditional marginals), and compute the misclassification rate (i.e., the test error) on the test data. The code above randomly permutes the data, so that the proportion of each class is about the same in both training and test data.

Note: On the spam detection problem, please note that you will get a different test error depending on how you quantize values that are equal to the median. It makes a difference whether you quantize values equal to the median to 1 or 2. You should quantize all medians the same way - I'm not suggesting that you try all 2d combinations. So just make sure you try both options, and report the one that works better.

1.1 (a)

Quantize each variable to one of two values, say 1 and 2, so that values below the median map to 1, and those above map to 2.

To get the median in Python, use np.median(a,axis=0).

Report the test error. As a sanity check, what would be the test error if you always predicted the same class, namely, the majority class from the training data?

Note: In class you may learn the Laplace Smoothing technique but in this problem you don't need to implement this technique.

[426]:

```
print(x.shape, y.shape)
```

```
(4601, 57) (4601,)
```

[427]:

```
med = np.median(x, axis=0)
for i in range(len(x)):
    for j in range(len(x[i])):
        if x[i][j] <= med[j]:
            x[i][j] = 0
        else:
            x[i][j] = 1
```

[428]:

```
#split into train and test data
train_size = 2000
x_train = x[:train_size, :]
x_test = x[train_size:, :]
y_train = y[:train_size]
y_test = y[train_size:]

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(2000, 57) (2601, 57) (2000,) (2601,)
```

[429]:

```
num_examples, num_features = x_train.shape
num_classes = len(np.unique(y_train))

print(num_examples, num_features, num_classes)
```

```
2000 57 2
```

[430]:

```
#num of documents in a class divided by the total number of documents is priors
priors = [0,0]
for i in y_train:
    if i==0:
        priors[0] += 1
    else:
        priors[1] += 1

print(priors)
```

```
[1193, 807]
```

[431]:

```
#calculate n_k for all classes
n_kt = []
for cls in range (num_classes):
    sum = 0
    for i in range (len(y_train)):
        if y_train[i] == cls:
            sum += 1
    n_kt.append(sum)
```

[432]:
```
#calculate feature amounts for n_kl
features = np.zeros((num_examples,num_features,num_classes))
for cls in range (num_classes):
    for feat in range(num_features):
        for i in {0,1}:
            sum = 0
            for r in range(len(y_train)):
                if y_train[r] == cls and i == x_train[r][feat]:
                    sum += 1
            features[cls][feat][i] = sum
```

[433]:
```
def predict(r):
    answer = []
    for cls in range (num_classes):
        p = 1

        for feat in range(num_features):

            #calculate n_kl
            n_kl = features[cls][feat][int(x_test[r][feat])]

            #input n_k from n_kt
            n_k = n_kt[cls]

            p *= n_kl/float(n_k)

        answer.append(priors[cls]*p)
    return np.argmax(answer)
```

[434]:
```
def accuracy_percent(test_data, prediction):
    sum = 0
    for i in range(len(test_data)):
        if test_data[i] == prediction[i]:
            sum += 1
    return ((sum/len(y_test))*100)
```

[435]:
```
predictions = []
for i in range (len(y_test)):
    predictions.append(predict(i))

print(str(accuracy_percent(y_test, predictions))+"%")
```

```
89.38869665513263%
```

I have found that if I allow quantized values that are equal to the median go into class 1 (aka 2) then I get a percent accuracy of 75.74%.

If I allow quantized values that are equal to the median to be go into class 0 (aka 1) then I get a percent accuracy of 89.39%.

Therefore, experimentally it seems that the better accuracy favors the <= case into class 0.