Machine Learning, Fall 2017
Homework 1

Sample Solutions

1. LINEAR ALGEBRA AND PROBABILITY REVIEW

(a) Suppose x_0 is a point on the hyperplane $\theta \cdot x + \theta_0 = 0$, we can get a vector from point y to point x_0:

$$v = y - x_0$$

As θ is the norm vector of the hyperplane, thus, the distance d from point y to the hyperplane can be computed by:

$$d = \frac{v \cdot \theta}{|v||\theta|}|v| = \frac{y \cdot \theta - x_0 \cdot \theta}{|\theta|} = \frac{y \cdot \theta + \theta_0}{|\theta|}$$

(b) As X and Y are independent, $P(X = x) = \frac{\alpha^x e^{-\alpha}}{x!}$, $P(Y = y) = \frac{\beta^y e^{-\beta}}{y!}$, Z = X + Y,

$$\begin{aligned}
P(Z = z) &= \sum_{i=0}^{z} P(X = i)P(Y = z - i) \\
&= \sum_{i=0}^{z} \frac{\alpha^i e^{-\alpha}}{i!} \frac{\beta^{z-i} e^{-\beta}}{(z-i)!} \\
&= \sum_{i=0}^{z} \frac{\alpha^i e^{-\alpha} \beta^{z-i} e^{-\beta}}{i!(z-i)!} \\
&= e^{-\alpha-\beta} \sum_{i=0}^{z} \frac{\alpha^i \beta^{z-i}}{i!(z-i)!} \\
&= e^{-\alpha-\beta} \sum_{i=0}^{z} \frac{z!}{i!(z-i)!} \frac{\alpha^i \beta^{z-i}}{z!} \\
&= e^{-\alpha-\beta} \sum_{i=0}^{z} C_z^i \frac{z!}{\alpha^i \beta^{z-i}} \\
&= e^{-\alpha-\beta} \frac{(\alpha+\beta)^z}{z!}
\end{aligned}$$

According to the definition of Poisson distribution, we can get that Z is also a Poisson random variable and its rate $\gamma = \alpha + \beta$

3. LINEAR REGRESSION

```
import sys
print (sys.version)

2.7.12 (default, Nov  1 2016, 10:50:56)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
csv = 'https://www.dropbox.com/s/oqoyy9p849ewzt2/linear.csv?dl=1'
data = np.genfromtxt(csv,delimiter=',')
X = data[:,1:]
Y = data[:,0]

import theano
import theano.tensor as T
d = X.shape[1] # dimension of feature vectors
n = X.shape[0] # number of training samples
learn_rate = 0.5 # learning rate for gradient descent

x = T.matrix(name='x') # feature matrix
y = T.vector(name='y') # response vector
w = theano.shared(np.zeros((d,1)),name='w') # model parameters

risk = T.sum((T.dot(x,w).T - y)**2)/2/n # empirical risk
grad_risk = T.grad(risk, wrt=w) # gradient of the risk

train_model = theano.function(inputs=[],
                    outputs=risk,
                    updates=[(w, w-learn_rate*grad_risk)],
                    givens={x:X, y:Y})

n_steps = 50
for i in range(n_steps):
    train_model()
print(w.get_value())

[[-0.57392068]
 [ 1.35757059]
 [ 0.01527565]
 [-1.88288076]]

(2) Exact Solution:

np.dot(np.linalg.inv(np.dot(X.T,X)), np.dot(X.T, Y))
```

```
array([-0.57392068,  1.35757059,  0.01527565, -1.88288076])
```

```
The answer is the same as (1)
```

```
(3) Using sklearn:
```

```
from sklearn import linear_model
regr = linear_model.LinearRegression(fit_intercept=False)
regr.fit(X, Y)
print('Coefficients:', regr.coef_)
```

```
('Coefficients:', array([-0.57392068,  1.35757059,
0.01527565, -1.88288076]))
```

```
The answer is the same as (1) and (2)
```

```
(4) Stochastic Gradient Descent:
```

```
from theano import shared
from numpy.random import shuffle

mini_batch_size = 5
n_batches = n/mini_batch_size
index = T.lscalar()
train_x = shared(np.array(X))
train_y = shared(np.array(Y))
train_model = theano.function(inputs=[index],
                    outputs=risk,
                    updates=[(w, w-learn_rate*grad_risk)],
                    givens={x:train_x[index * mini_batch_size: (index + 1) * min
                            y:train_y[index * mini_batch_size: (index + 1) * min
n_steps = 50
for i in range(n_steps):
    learn_rate = learn_rate / (i + 1)
    sample_index = np.arange(n)
    shuffle(sample_index)
    X = [X[i] for i in sample_index]
    Y = [Y[i] for i in sample_index]
    train_x = shared(np.array(X))
    train_y = shared(np.array(Y))
    for j in range(n_batches):
        train_model(j)
print (w.get_value())
```

```
[[-0.57205036]
 [ 1.35861364]
 [ 0.01744707]
 [-1.88281916]]
```

The answer **is** nearly the same as (1), (2), (3)