Student: Quoc Tuan Vinh, Ngo
ID: 704526

CS-E4820 – Machine Learning: Advanced Probabilistic Methods
Homework Assignment 7

Problem 1. *"Bayes factors"*

Let $N_1, b_1, N_1 - b_1, \theta_1$ respectively denote the total number of draws, number of black, number of white and the probability of black in the first bag.

Let $N_2, b_2, N_2 - b_2, \theta_2$ respectively denote the total number of draws, number of black, number of white and the probability of black in the second bag.

(1) First, let us calculate $p(x|\theta_1, M), p(x|\theta_2, M)$ for each set of draws:

In the first set of draw, we can obtain the probability of getting b_1 black and $N_1 - b_1$ white in a sequence of N_1 draws as below:

$$p(x|\theta_1, M) = \binom{N_1}{b_1} \theta_1^{b_1} (1 - \theta_1)^{\,b_1}$$

Similarly, we can obtain the same probabilities for the second set of draws:

$$p(x|\theta_2, M) = \binom{N_2}{b_2} \theta_2^{b_2} (1 - \theta_2)^{\,b_2}$$

Therefore:

$$p(x|\theta, M) = p(x|\theta_1, M) p(x|\theta_2, M) = \binom{N_1}{b_1} \theta_1^{b_1} (1 - \theta_1)^{\,b_1} \binom{N_2}{b_2} \theta_2^{b_2} (1 - \theta_2)^{\,b_2} \quad \mathbf{(1)}$$

In model 1, since $\theta_1 = \theta_2 = \theta$, we can transform (1) as:

$$p(x|\theta, M_1) = \binom{N_1}{b_1} \binom{N_2}{b_2} \theta^{b_1 + b_2} (1 - \theta)^{\,b_1 + b_2}$$

In model 1, since $\theta_1 \neq \theta_2$, we will keep (1) as it is:

$$p(x|\theta, M_2) = \binom{N_1}{b_1} \theta_1^{b_1} (1 - \theta_1)^{\,b_1} \binom{N_2}{b_2} \theta_2^{b_2} (1 - \theta_2)^{\,b_2}$$

(2) Second, let us calculate the prior for each model:

Next, since we assume a priori all proportions are equally probable, and a uniform proportion corresponds to the $Beta(1,1)$ distribution, we can calculate the conditional probability of θ against a particular model as:

Model M_1: $\theta_1 = \theta_2 = \theta$

$$p(\theta|M_1) = Beta(\theta|1,1) = B(1,1)^{-1}\theta(1-\theta)^0 = B(1,1)^{-1} = 1$$

Model M_2: $\theta_1 \neq \theta_2$

$$p(\theta|M_2) = p(\theta_1|M_2)\, p(\theta_1|M_2) = Beta(\theta_1|1,1)\, Beta(\theta_2|1,1) = B(1,1)^{-2} = 1$$

(3) Next, we can calculate the model probability as below:

Model 1:

$$p(x|M_1) = \int p(x|\theta, M_1)\, p(\theta|M_1) d\theta = \int \binom{N_1}{b_1}\binom{N_2}{b_2} \theta^{b_1+b_2}(1-\theta)^{\,b_1+b_2} B(1,1)^{-1} d\theta$$
$$= \binom{N_1}{b_1}\binom{N_2}{b_2} B(b_1 + b_2 + 1, N_1 + N_2 - b_1 - b_2 + 1) \quad \mathbf{(2)}$$

Model 2:

$$p(x|M_2) = \int p(x|\theta_1, M) p(\theta_1|M) d\theta_1 \times \int p(x|\theta_2, M) p(\theta_2|M) d\theta_2$$
$$= \int \binom{N_1}{b_1} \theta_1^{b_1}(1-\theta_1)^{\,b_1} B(1,1)^{-1}\, d\theta_1$$
$$\times \int \binom{N_2}{b_2} \theta_2^{b_2}(1-\theta_2)^{\,b_2} B(1,1)^{-1} d\theta_2$$
$$= \binom{N_1}{b_1}\binom{N_2}{b_2} \int \theta_1^{b_1}(1-\theta_1)^{\,b_1} d\theta_1 \int \theta_2^{b_2}(1-\theta_2)^{\,b_2} d\theta_2$$
$$= \binom{N_1}{b_1}\binom{N_2}{b_2} B(b_1+1, N_1 - b_1 + 1) B(b_2 + 1, N_2 - b_2 + 1) \quad \mathbf{(3)}$$

(a) In this case, we have $N_1 = N_2 = 5, b_1 = 3, b_2 = 4$:

From (2), we have:

$$p(x|M_1) = \binom{5}{3}\binom{5}{4} B(8,4) = 50\, B(8,4)$$

From (3), we have:

$$p(x|M_2) = \binom{5}{3}\binom{5}{4} B(4,3)B(5,2) = 50\, B(4,3)B(5,2)$$

Thus the Bayes factor in favor of M_1 is:

$$BF_{M_1,M_2} = \frac{p(x|M_1)}{p(x|M_2)} = \frac{B(8,4)}{B(4,3)B(5,2)} \left(\approx \frac{7.57576E-4}{0.016667 \times 0.03333} \approx 1.3637\right)$$

meaning that with all current assumptions, seeing the data x has increased the probability of model M_1 about 1.3637 time as opposed to model M_2.

(b) In this case, we have $N_1 = N_2 = 500, b_1 = 300, b_2 = 250$:

Similarly to question (a), one can derive

$$p(x|M_1) = \binom{N_1}{b_1}\binom{N_2}{b_2}B(551{,}451\ )$$

and

$$p(x|M_2) = \binom{N_1}{b_1}\binom{N_2}{b_2}B(301{,}201)B(251{,}251)$$

$$BF_{M_1,M_2} = \frac{p(x|M_1)}{p(x|M_2)} = \frac{B(551{,}451\ )}{B(301{,}201)B(251{,}251)} \approx 0.0815$$

meaning that with all current assumptions, seeing the data x has increased the probability of model M_1 about 0.0815 time as opposed to model M_2.

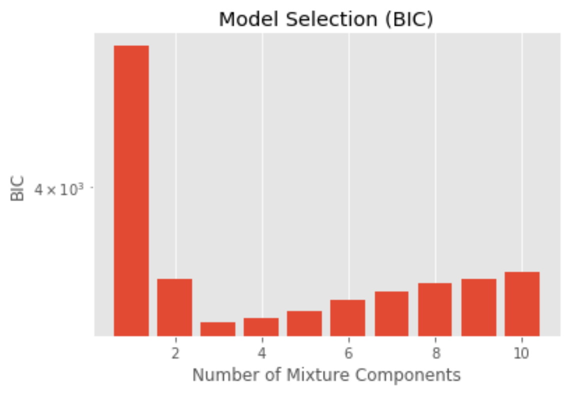
Problem 2. *"Model selection for GMM with BIC and CV"*

For BIC:

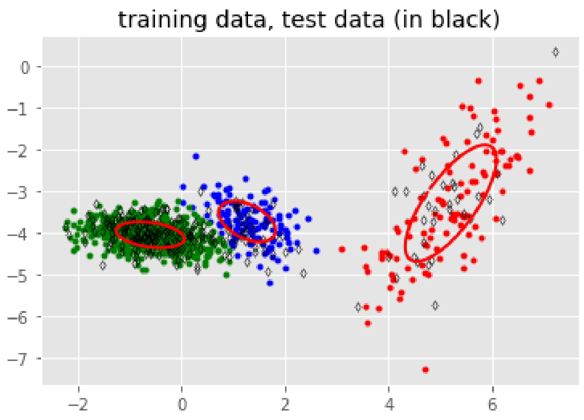
```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
import GMMem
import pickle

np.random.seed(0)

# Fill in the missing parts marked with ?

# ***** DATA
totalComponents = 10  # max number of mixture components

# load data
with open("data.pickle", "rb") as f:
	X, labels = pickle.load(f)

D, N = X.shape   # dimension and number of data points
print(X.shape)

ratio = 0.75
train_ind = np.random.choice(N, int(ratio * N), replace=False)   # training data index
test_ind = np.setdiff1d(np.arange(N), train_ind)                 # test data index

Xtrain = X[:,train_ind]            # training data
Xtrain_labels = labels[train_ind]  # training data labels

Xtest = X[:,test_ind]            # test data
Xtest_labels = labels[test_ind]  # test data labels

# plot training and test data
def plot_data():
	for i in sorted(set(Xtrain_labels)):
		X_comp = Xtrain[:, Xtrain_labels == i]
		plt.plot(X_comp[0], X_comp[1], '.' + 'brgmcyk'[i-1], markersize=6)

	plt.plot(Xtest[0], Xtest[1], 'kd', markersize=4, markeredgewidth=0.5, markerfacecolor="None")

plot_data()
plt.title('training data, test data (in black)')
plt.show()
# ***** Use BIC to select the number of components
# (Only this part differs from the second template, where cross validation is used instead)

BICs = []

for H in range(1, totalComponents+1):    # number of mixture components
        print("H: {}".format(H))

        P, m, S, loglik, phgn = GMMem.GMMem(Xtrain, H, 100)  # fit to data

        numParams = H*(2 + 3)   # number of parameters in the model ?
        BIC = -2*(loglik - 1/2 * numParams * np.log(N*ratio))          # BIC for the model ?
        BICs.append(BIC)

# plot the BIC curve
plt.bar(range(1, totalComponents+1), BICs)
plt.yscale("log", nonposy="clip")
plt.xlabel('Number of Mixture Components')
plt.ylabel('BIC')
plt.title('Model Selection (BIC)')
plt.show()

# select the number of mixture components which minimizes the BIC
h = np.argmin(BICs) + 1

# ***** TRAIN

# Now train full model with selected number of mixture components
P, m, S, loglik, phgn = GMMem.GMMem(Xtrain, h, 100)  # fit to data

# Predict using the full trained model (Use GMMem.GMMloglik)
logl = GMMem.GMMloglik(Xtest, P, m, S) # ?

print('Test Data Likelihood = {0:f}'.format(np.sum(logl))) # ?

# Plot the best GMM model
plot_data()

for i in range(h):
        dV, E = LA.eig(S[i,:,:])

        theta = np.arange(0, 2*np.pi, 0.1)
        p = np.sqrt(dV.reshape(D,1)) * [np.cos(theta), np.sin(theta)]
        x = (E @ p) + np.tile(m[:,i:i+1], (1, len(theta)))

        plt.plot(x[0], x[1], 'r-', linewidth=2)

plt.title('training data, test data (in black)')
plt.show()
```

Results are shown as below:

```
H:  1
H:  2
H:  3
H:  4
H:  5
H:  6
H:  7
H:  8
H:  9
H:  10
```



```
Test Data Likelihood = -529.653042
```

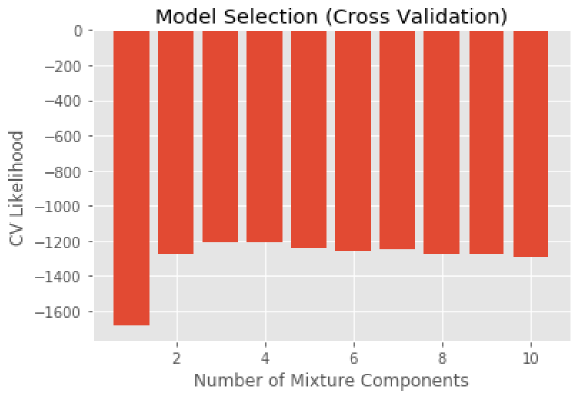


For Cross Validation:

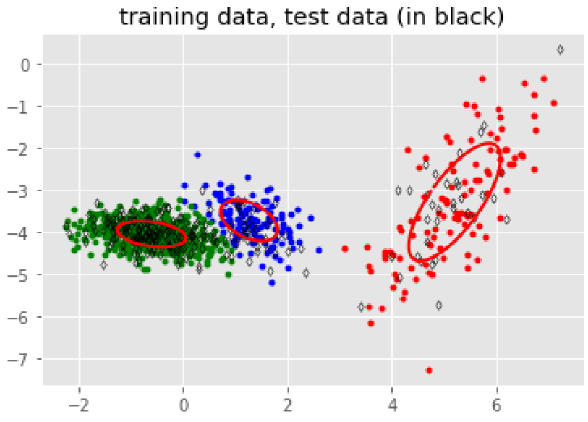
```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
import GMMem
import pickle

np.random.seed(0)

# Fill in the missing parts marked with ?


# ***** DATA
totalComponents = 10  # max number of mixture components

# load data
with open("data.pickle", "rb") as f:
        X, labels = pickle.load(f)

D, N = X.shape   # dimension and number of data points

ratio = 0.75
train_ind = np.random.choice(N, int(ratio * N), replace=False)   # training data index
test_ind = np.setdiff1d(np.arange(N), train_ind)                 # test data index

Xtrain = X[:,train_ind]            # training data
Xtrain_labels = labels[train_ind]  # training data labels

Xtest = X[:,test_ind]            # test data
Xtest_labels = labels[test_ind]  # test data labels

# plot training and test data
def plot_data():
        for i in sorted(set(Xtrain_labels)):
                X_comp = Xtrain[:, Xtrain_labels == i]
                plt.plot(X_comp[0], X_comp[1], '.' + 'brgmcyk'[i-1], markersize=6)

        plt.plot(Xtest[0], Xtest[1], 'kd', markersize=4, markeredgewidth=0.5, markerfacecolor="None")

plot_data()
plt.title('training data, test data (in black)')
plt.show()

# ***** Use cross validation to choose the number of components
# (Only this part differs from the first template, where BIC is used instead)

foldCount = 5    # number of folds

loglikelihoods = np.zeros((totalComponents, foldCount))  # collect log-likelihoods

Nlearning = Xtrain.shape[1]
order = np.random.permutation(Nlearning)  # you can randomize the order of training samples
                                    # when constructing the folds in the loop

for H in range(1, totalComponents+1):     # number of mixture components
        print("H: {}".format(H))

        for fold in range(foldCount):    # K-fold cross validation
                fold_size = int(N/(fold+1))

                val_indices = np.random.choice(N,fold_size*(fold), replace=False)
                training_indices=np.setdiff1d(np.arange(N), val_indices)
                X_train = X[:,training_indices]  # cv training data
                X_val   = X[:,val_indices]       # cv validation data

                # Train model
                P, m, S, loglik, phgn = GMMem.GMMem(X_train, H, 100)   # fit model

                # Predict using the cv trained model for validation data X_val
                logl = GMMem.GMMloglik(X_val, P, m, S)   # use the function GMMem.GMMloglik ?
                loglikelihoods[H-1,fold] = np.sum(logl)  # ?


# plot the accuracy curve
plt.bar(np.arange(1, totalComponents+1), np.mean(loglikelihoods, axis=1))
plt.xlabel('Number of Mixture Components')
plt.ylabel('CV Likelihood')
plt.title('Model Selection (Cross Validation)');
plt.show()

# select the number of mixture components which maximizes the accuracy
h = np.argmax(np.mean(loglikelihoods, axis=1)) + 1


# ***** TRAIN

# Now train full model with selected number of mixture components
P, m, S, loglik, phgn = GMMem.GMMem(Xtrain, h, 100)  # fit to data

# Predict using the full trained model (Use GMMem.GMMloglik)
logl = GMMem.GMMloglik(Xtest, P, m, S) #?

print('Test Data Likelihood = {0:f}'.format(np.sum(logl))) #?


# Plot the best GMM model
plot_data()

for i in range(h):
        dV, E = LA.eig(S[i,:,:])
        theta = np.arange(0, 2*np.pi, 0.1)
        p = np.sqrt(dV.reshape(D,1)) * [np.cos(theta), np.sin(theta)]
        x = (E @ p) + np.tile(m[:,i:i+1], (1, len(theta)))

        plt.plot(x[0], x[1], 'r-', linewidth=2)

plt.title('training data, test data (in black)')
plt.show()
```

Results are shown as below:

```
H:  1
H:  2
H:  3
H:  4
H:  5
H:  6
H:  7
H:  8
H:  9
H:  10
```



```
Test Data Likelihood = -529.653042
```



(a) From the codes and printed results above, we can see that BIC and validation log-likehood results in the same result: 03 components are the most likely.
(b) Test data likelihoods are the same in both cases as well.
(c) Pros and cons of each method:

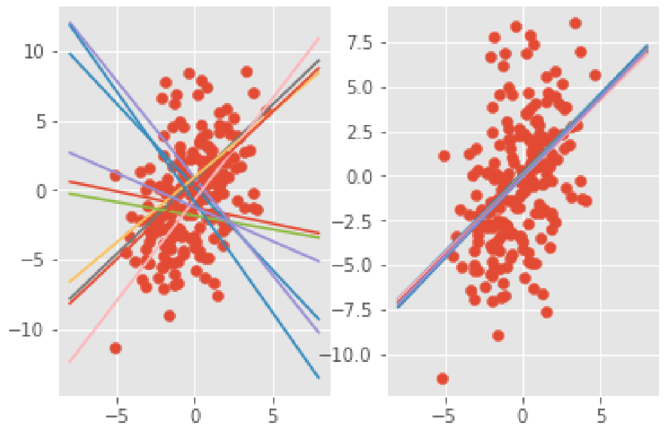
- BIC: Light computation, faster and easy to understand. However, it is a bit tricky to define the correct number of parameters, and needs to be cautious with small N. Plus, Laplace method is just an approximation method so there might be deviation.

- CV: Also easy to understand and more accurate. However, CV requires heavy computation (doing different fold for each mixture of components), especially with big dataset, which costs time subsequently.

Problem 3. "*Bayesian linear regression model using Edward*"

(a) Run the model, one can obtain the solution as below:

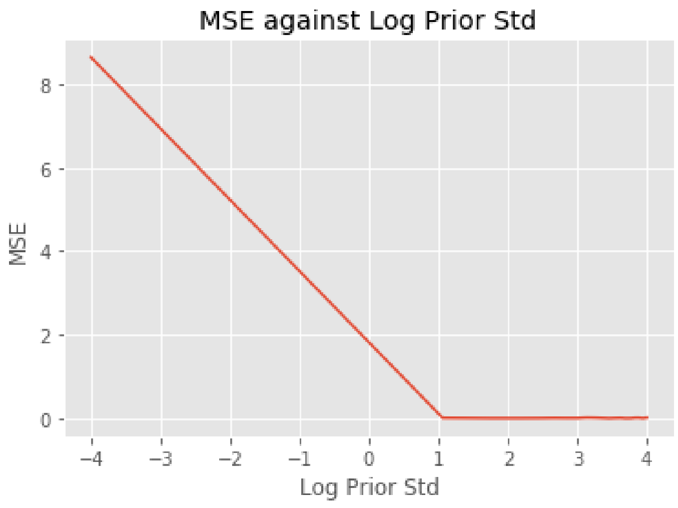
```
Mean squared error:
0.00751544
Mean absolute error:
0.0641012
```



```
Point estimate for STD of weights: 1.0
Correlation between estimated and learned weights:  0.999815475178
```

(b) Grid test over a range of log prior std -4 to 4

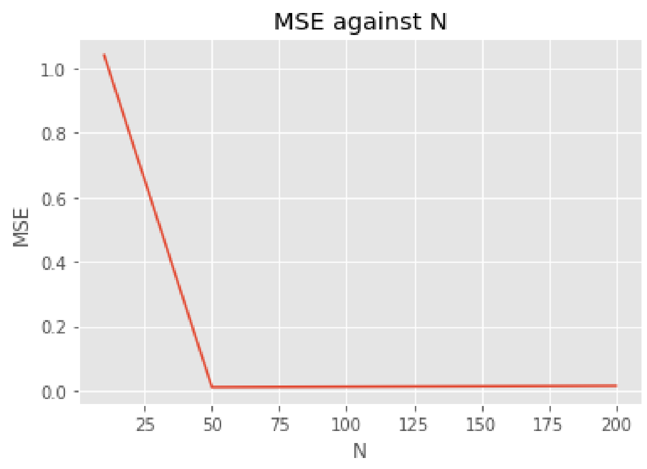
Dividing the range into 20 equal spaces, we got the result as below:



It can be seen that MSE decreases significantly from log prior =-4 till log prior =1 (from above 8. till almost 0), and then stays approximately the same till log prior = 4.

(c) Grid test over N of 10,50,200

```
N [10, 50, 200]
MSE [1.0412712, 0.010112436, 0.014766407]
```



It can be seen that MSE does not vary much between test size of 50 and 200, and both of them are much lower than test size of 10.

Below is the code for part b and c:

#Firstly, let’s define the function to use for both b and c

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import edward as ed
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from edward.models import Normal

plt.style.use('ggplot')

# CREATE DATA
np.random.seed(0)
def build_toy_dataset(N, w):
    D = len(w)
    x = np.random.normal(0.0, 2.0, size=(N,D))
    y = np.dot(x, w) + np.random.normal(0.0, 0.01, size=N)
    return x, y

# MODEL FUNCTION
def model(N,w_prior_std_):
    #print('With %s and %s :' % (N,w_prior_std_num))
    D = 10
    w_true = np.random.randn(D) * 0.5 # so, the true STD of weights is 0.5
    X_train, y_train = build_toy_dataset(N, w_true)
    X_test, y_test = build_toy_dataset(N, w_true)

    # DEFINE MODEL

    X = tf.placeholder(tf.float32, [N, D])
    w_prior_std = tf.constant(w_prior_std_)
    w = Normal(loc=tf.zeros(D), scale=w_prior_std * tf.ones(D))
    b = Normal(loc=tf.zeros(1), scale=tf.ones(1))
    y = Normal(loc=ed.dot(X, w) + b, scale=tf.ones(N))

    # DEFINE VARIATIONAL DISTRIBUTION

    qw = Normal(loc=tf.Variable(tf.random_normal([D])), scale=tf.nn.softplus(tf.Variable(tf.random_normal([D]))))
    # One more variational factor for b:
    qb = Normal(loc=tf.Variable(tf.random_normal([1])), scale=tf.nn.softplus(tf.Variable(tf.random_normal([1]))))

    # INFERENCE
    inference = ed.KLqp({w: qw, b: qb}, data={X: X_train, y: y_train})


    inference.run(n_samples=5, n_iter=250)

    # MODEL CRITICISM
    y_post = ed.copy(y, {w: qw, b: qb})
    return ed.evaluate('mean_squared_error', data={X: X_test, y_post: y_test})

#b
N_grid = 200
w_prior_std_grid = np.array(np.linspace(np.exp(-4),np.exp(4),20), dtype=np.float32)
MSE_list=[]
for j in w_prior_std_grid:
    MSE = model(200,j)
    MSE_list.append(MSE)

print('Prior Std',w_prior_std_grid)
print('MSE',MSE_list)

plt.plot(np.log(w_prior_std_grid), MSE_list)
plt.xlabel('Log Prior Std')
plt.ylabel('MSE')
plt.title('MSE against Log Prior Std')
plt.show()

#c
N_grid = [10,50,200]
MSE_list=[]
for i in N_grid:
    MSE = model(i,1.0)
    MSE_list.append(MSE)

print('N',N_grid)
print('MSE',MSE_list)

plt.plot(N_grid, MSE_list)
plt.xlabel('N')
plt.ylabel('MSE')
plt.title('MSE against N')
plt.show()
```