Problem3

(a)

```
[('MILLION', 0.002072759168154815),
 ('MORE', 0.0017088989966186725),
 ('MR.', 0.0014416083492816956),
 ('MOST', 0.0007879173033190295),
 ('MARKET', 0.0007803712804681068),
 ('MAY', 0.0007298973156289532),
 ('M.', 0.0007034067394618568),
 ('MANY', 0.0006967290595970209),
 ('MADE', 0.0005598610827336895),
 ('MUCH', 0.0005145971758110562),
 ('MAKE', 0.0005144626437991272),
 ('MONTH', 0.00044490959363187093),
 ('MONEY', 0.00043710673693999306),
 ('MONTHS', 0.0004057607781605526),
 ('MY', 0.0004003183467688823),
 ('MONDAY', 0.00038198530259784006),
 ('MAJOR', 0.00037089252670515475),
 ('MILITARY', 0.00035204581485220204),
 ('MEMBERS', 0.00033606096579846475),
 ('MIGHT', 0.00027358919153183117),
 ('MEETING', 0.00026573741410834 27),
 ('MUST', 0.0002665079156312084),
 ('ME', 0.00026357267173457725),
 ('MARCH', 0.0002597935452176646),
 ('MAN', 0.0002528834918776787),
 ('MS.', 0.0002389900041002911),
 ('MINISTER', 0.00023977273580605944),
 ('MAKING', 0.00021170446604452378),
 ('MOVE', 0.0002099555498894477),
 ('MILES', 0.00020596851026319035)]
```

(b)

```
<UNK> 0.030112266401635904
U. 0.0006547370650823069
FIRST 0.0005738410470713228
COMPANY 0.0005708312580181669
NEW 0.00046275824180569056
UNITED 0.0004246088480686425
GOVERNMENT 0.0003331087209463285
NINETEEN 0.00032562869667498317
SAME 0.0003078239529934448
TWO 0.0003016392809728248
```

(c)

```
log-likelihood for unigram = -64.50944034364878

log-likelihood for bigram = -40.91813213378977
```

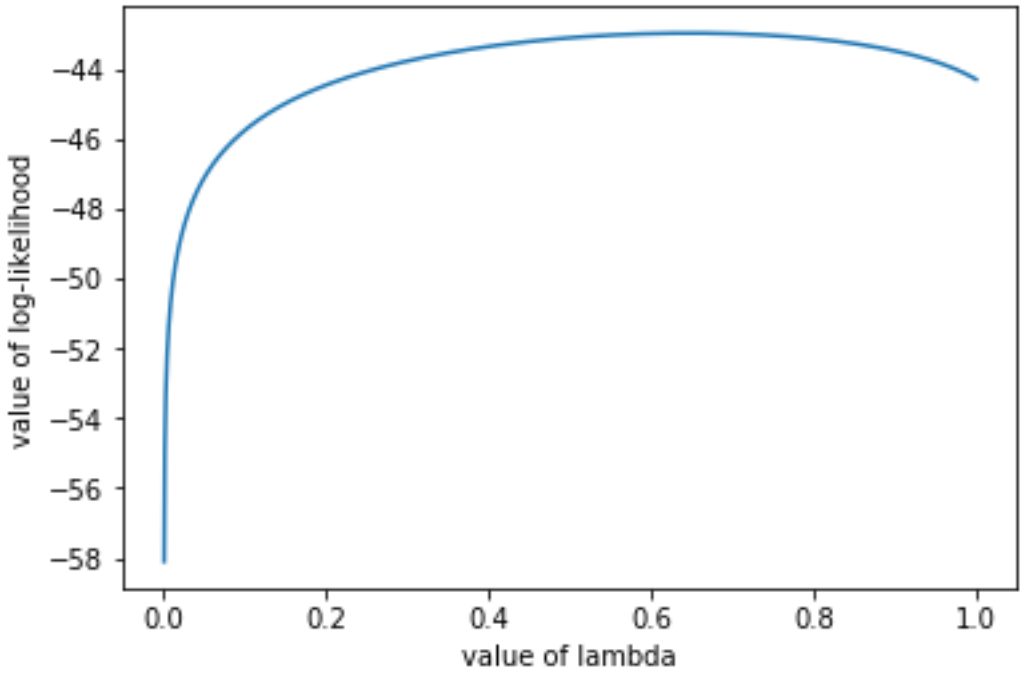
Bigram model yields the highest log-likelihood.

(d)
log-likelihood for unigram = -44.291934473132606

P_b(officals|sixteen) and **P_b(fire|sold)** are not observed in the training corpus.

The fact that 2 pairs of adjacent words in this sentence are not observed in the training corpus makes bigram models have no generalization to unseen information. Without the probability of these 2 pairs of adjacent words, we can't get the log-likelihood from the bigram model(if we regard the probability of these 2 pairs of adjacent words as 0, then we can't compute the log-likelihood since there is not meaningful to calculate log(0)).

(e)
lambda = 0.65



problem4

(1) Use the data of year 2000, a1=0.95067337,a2=0.01560133, a3=0.03189569
(2) I choose dealing with data in 2001 from the 4th day in that year.
MSE for 2000 = 13902.40107637
MSE for 2001 = 2985.09792411

I won't recommend this model because basically, this model mainly use the data one day(since a1 is 0.95, just ignore a2 and a3) before to predict the current data. Since in stock market, the f luctuation of data doesn't have a linear relationship, it is not accurate to predict the current N asdaq use the data one, two, three days before.

Source code:
Problem3

```
#import the data
with open('hw4_vocab.txt') as f:
    vocab = f.read().splitlines()
with open('hw4_unigram.txt') as f1:
    unigram = f1.read().splitlines()
bigram = []
with open('hw4_bigram.txt') as f2:
    for line in f2:
        bigram.append(line.strip().split('\t'))
```

```
#(a)
for i in range(len(unigram)):
    unigram[i] = int(unigram[i])
total_num_word = sum(unigram)
unigram_prob = []
for i in range(len(unigram)):
    unigram_prob.append(unigram[i]/total_num_word)
word_with_prob = list(zip(vocab,unigram_prob)) #vocabulary number, prob of that vocabulary

token_begin_M = []
for j in range(len(word_with_prob)):
    if word_with_prob[j][0][0] == 'M':
        token_begin_M.append(word_with_prob[j])
print(token_begin_M)
#end of (a)
```

```
#the string in bigram become integer
for i in range(len(bigram)):
    bigram[i][0] = int(bigram[i][0])
    bigram[i][1] = int(bigram[i][1])
    bigram[i][2] = int(bigram[i][2])
#form dictionary of number and actual word, number is the key
number_and_word=dict(zip(range(1,501),vocab))
word_and_number=dict(zip(vocab,range(1,501)))#now word is the key
#number_and_word[16]=='THAT'
#word_and_number['THAT']==16
```

```
#calculate the prob of every row in bigram
def bi_prob(a):
    bigram_1 =[d for d in bigram if d[0]==a]
    sum_1=0
    for d in bigram_1:
        sum_1+=d[2]
    bigram_1_prob = []
    for d in bigram_1:
        bigram_1_prob.append(d[2]/sum_1)
    return bigram_1_prob
#bigram_prob is the total prob of bigram
bigram_prob=[]
for i in range(1,501):
    bigram_prob.append(bi_prob(i))
```

```
#(b) for word "the"
the_prob=bigram_prob[word_and_number['THE']-1]
the_bigram=[d[1] for d in bigram if d[0]==4] #get the order of word which goes after word "the"
the_word_prob =list(zip(the_bigram,the_prob))
sorted_by_second=sorted(the_word_prob,key=lambda tup:tup[1],reverse=True)#sort from high to low
for i in range(10):
    print(number_and_word[sorted_by_second[i][0]],sorted_by_second[i][1])
#end of (b)
```

```
#(c) calculate L_unigram
import math
sentence_u=['THE','STOCK','MARKET','FELL','BY','ONE','HUNDRED','POINTS','LAST','WEEK']
log_unigram=0
for i in sentence_u:
    row_number=word_and_number[i]-1
    log_unigram=log_unigram+math.log(word_with_prob[row_number][1])
log_unigram
#end of calculate of L_unigram
```

```
#(c) calculate L_bigram
def log_prob_bigram(value,value_next):  #calculate for logPb(|)
    bigram_value =[d for d in bigram if d[0]==value]
    sum_value=0
    for d in bigram_value:
        sum_value+=d[2]
    for i in bigram_value:
        if i[1]==value_next:
            time = i[2]
    node_prob=time/sum_value
    return math.log(node_prob)
sentence_b=['<s>','THE','STOCK','MARKET','FELL','BY','ONE','HUNDRED','POINTS','LAST','WEEK']
number_sentence_b=[word_and_number[i] for i in sentence_b]
log_bigram=0
for i in range(len(number_sentence_b)-1):
    value=number_sentence_b[i]
    value_next=number_sentence_b[i+1]
    log_bigram+=log_prob_bigram(value,value_next)
log_bigram
#end of calculate of L_bigram
```

```
#(d)calculate L_unigram
sentence_in_d=['THE','SIXTEEN','OFFICIALS','SOLD','FIRE','INSURANCE']
log_unigram_d=0
for i in sentence_in_d:
    row_number=word_and_number[i]-1
    log_unigram_d=log_unigram_d+math.log(word_with_prob[row_number][1])
print('log-likelihood for unigram = ',log_unigram_d)
#end of calculate L_unigram
```

```
#(d)calculate L_bigram
sentence_in_d=['<s>','THE','SIXTEEN','OFFICIALS','SOLD','FIRE','INSURANCE']
number_sentence_in_d=[word_and_number[i] for i in sentence_in_d]
for i in range(0,2):
    value=number_sentence_in_d[i]
    value_next=number_sentence_in_d[i+1]
    prob=log_prob_bigram(value,value_next)
    print('likelihood of P(',number_and_word[value_next],'|',number_and_word[value],')',' = ',prob)
for i in range(3,4):
    value=number_sentence_in_d[i]
    value_next=number_sentence_in_d[i+1]
    prob=log_prob_bigram(value,value_next)
    print('likelihood of P(',number_and_word[value_next],'|',number_and_word[value],')',' = ',prob)
for i in range(5,6):
    value=number_sentence_in_d[i]
    value_next=number_sentence_in_d[i+1]
    prob=log_prob_bigram(value,value_next)
    print('likelihood of P(',number_and_word[value_next],'|',number_and_word[value],')',' = ',prob)
```

```
#(e)
import matplotlib.pyplot as plt
def prob_mgram(value,value_next,lam):  #calculate for Pb(|)
    bigram_value =[d for d in bigram if d[0]==value]
    sum_value=0
    for d in bigram_value:
        sum_value+=d[2]
    for i in bigram_value:
        if i[1]==value_next:
            time = i[2]
            break
        else:
            time = 0
    p_b=time/sum_value #P_b(w'|w)
    p_m=(1.0-lam)*p_b+lam*word_with_prob[value_next-1][1]
    log_pm = math.log(p_m)
    return log_pm


sentence_e = ['<s>','THE','SIXTEEN','OFFICIALS','SOLD','FIRE','INSURANCE']
number_sentence_in_d=[word_and_number[i] for i in sentence_in_d]
```

```
value_lam=[x/5000 for x in range(1,5000)]
value_Lm=[]
for j in range(len(value_lam)):
    lam=value_lam[j]
    L=0
    for i in range(6):
        value=number_sentence_in_d[i]
        value_next=number_sentence_in_d[i+1]
        L+=prob_mgram(value,value_next,lam)
    value_Lm.append(L)
plt.plot(value_lam,value_Lm)
plt.ylabel('value of log-likelihood')
plt.xlabel('value of lambda')
plt.show()
```

problem4

```
#import the data
with open('hw4_nasdaq00.txt') as f:
    nasdaq00 = f.read().splitlines()
with open('hw4_nasdaq01.txt') as f1:
    nasdaq01 = f1.read().splitlines()
for i in range(len(nasdaq00)):
    nasdaq00[i] = float(nasdaq00[i])
for i in range(len(nasdaq01)):
    nasdaq01[i] = float(nasdaq01[i])
```

```
#(a)
import numpy as np
train_y=nasdaq00[3:]
train_x = [nasdaq00[i:i-3:-1] for i in range(3,len(nasdaq00)-1)]
train_x_1 = nasdaq00[2::-1]
train_x.insert(0,train_x_1)
matrix_A=np.zeros((3,3))
for i in train_x:
    row = np.matrix(i)
    matrix_A+=row.T*row
matrix_b=np.zeros((3,1))
for i in range(len(train_y)):
    column =  np.matrix(train_x[i]).T
    matrix_b += train_y[i]*column
Ainv= np.linalg.inv(matrix_A)
w= np.matmul(Ainv,matrix_b)
w
```

```
#(b)
MSE_t=[0]
for i in range(len(train_y)):
    predict = np.matmul(w.T,np.matrix(train_x[i]).T)
    square_diff=(predict - train_y[i])**2
    square_diff=np.array(square_diff.flatten().tolist()[0])
    MSE_t =  MSE_t+square_diff
MSE = MSE_t/len(train_y)
```

```
test_y=nasdaq01[3:]
test_x = [nasdaq01[i:i-3:-1] for i in range(3,len(nasdaq01)-1)]
test_x_1 = nasdaq01[2::-1]
test_x.insert(0,test_x_1)
MSE_t=[0]
for i in range(len(test_y)):
    predict = np.matmul(w.T,np.matrix(test_x[i]).T)
    square_diff=(predict - test_y[i])**2
    square_diff=np.array(square_diff.flatten().tolist()[0])
    MSE_t =  MSE_t+square_diff
MSE = MSE_t/len(test_y)
```

```
#calculate all the prediction value for 2000
predict_train=[]
for i in range(len(train_y)):
    predict = np.matmul(w.T,np.matrix(train_x[i]).T)
    predict = np.array(predict.flatten().tolist()[0])
    predict_train.append(predict)
len(predict_train)
```