4.3) Statistical Language Modelling:

a) The maximum likelihood estimate of the unigram $P_u(w)$ distribution -

$$P_u(w) = \frac{Number\ of\ times\ word\ w\ appears\ in\ corpus}{Total\ number\ of\ words\ in\ corpus}$$

The table of all tokens starting with letter ‘M’, along with their numerical unigram probabilities is shown below:

'MILLION'	0.002073
'MORE'	0.001709
'MR.'	0.001442
'MOST'	0.000788
'MARKET'	0.00078
'MAY'	0.00073
'M.'	0.000703
'MANY'	0.000697
'MADE'	0.00056
'MUCH'	0.000515
'MAKE'	0.000514
'MONTH'	0.000445
'MONEY'	0.000437
'MONTHS'	0.000406
'MY'	0.0004

'MONDAY'	0.000382
'MAJOR'	0.000371
'MILITARY'	0.000352
'MEMBERS'	0.000336
'MIGHT'	0.000274
'MEETING'	0.000266
'MUST'	0.000267
'ME'	0.000264
'MARCH'	0.00026
'MAN'	0.000253
'MS.'	0.000239
'MINISTER'	0.00024
'MAKING'	0.000212
'MOVE'	0.00021
'MILES'	0.000206

b) The maximum likelihood estimate of the bigram distribution $P_b(w'|w)$ -

$$P_b(w'|w) = \frac{Number\ of\ times\ word\ w'\ follows\ word\ w\ in\ corpus}{Total\ number\ times\ word\ w\ appears\ in\ corpus}$$

The table of the most likely words to follow the word ‘THE’ along with their numerical probabilities :

'<UNK>'	0.61502
'U.'	0.013372
'FIRST'	0.01172
'COMPANY'	0.011659
'NEW'	0.009451
'UNITED'	0.008672
'GOVERNMENT'	0.006803
'NINETEEN'	0.006651
'SAME'	0.006287
'TWO'	0.006161

c) Log-likelihood of the sentence “The stock market fell by one hundred points last week” under the unigram and bigram models is :

$$L_u = \log[P_u(The)\ P_u(stock) \ldots \ldots P_u(week)]$$

$$L_u = -64.5094$$

$$L_b = \log[P_b(The \mid <s>)\ P_b(stock \mid The) \ldots \ldots P_u(week \mid last)]$$

$$L_b = -40.9181$$

From the numbers we see that the bigram model yields a higher likelihood.

d) The log-likelihood of the sentence “The sixteen officials sold fire insurance” under the unigram and bigram models is :

$$L_u = \log[P_u(The)\ P_u(sixteen) \ldots \ldots P_u(insurance)]$$

$$L_u = -44.2919$$

$$L_b = \log[P_b(The \mid <s>)\ P_b(sixteen \mid The) \ldots \ldots P_u(insurance \mid fire)]$$

$$L_b = -Inf$$

The pairs “Sixteen officials” and “sold fire” are not observed in the training corpus due to which their bigram probabilities are zero, which force the log-likelihood of the sentence to go to negative infinity. We can avoid this by assigning a very minimal general word-pair occurrence probability to each un-occurring word-pair.

e) The log-likelihood of the sentence from part (d) using a “mixture” model that predicts words from a weighted interpolation of the unigram and bigram models is :

$$\mathrm{P_m}(w'|\mathrm{w}) = \lambda * \mathrm{P_u}(\mathrm{w'}) + (1-\lambda) * P_b(w'|w)$$

where λ lies in [0, 1]

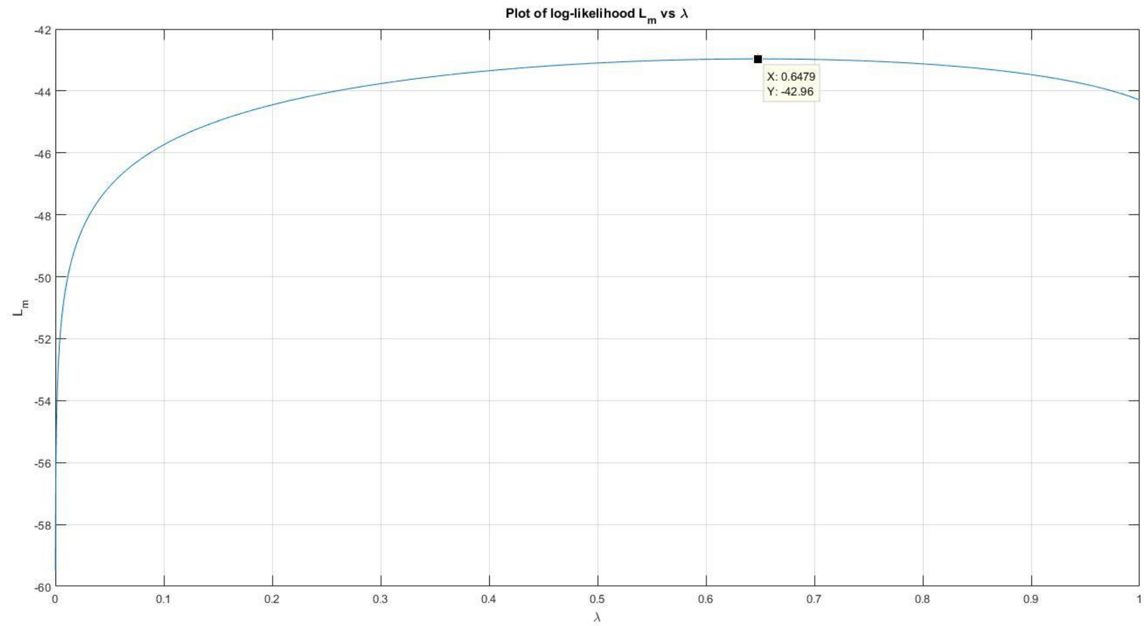
The log – likelihood of the sentence under this model is given by :

$$L_m = \log[\ P_m(the| <s>) \,.\, P_m(sixteen|the) \,.\, P_m(officials|sixteen) \ldots P_m(insurance|fire)]$$

The value of the log-likelihood L_m was plotted as a function of the parameter λ in [0,1], using which the optimal value of lambda was deduced to be :

$$\lambda = 0.6479$$

$$L_m = -42.9641$$



Source code:

```
%% Read data from files

% Make a list of tokens in vocabulary
tokens.words =
textread('Datasets/vocab.txt','%s');

% Counts of each token in corpus of
text
tokens.counts =
textread('Datasets/unigram.txt','%d');

% Counts of pairs of adjacent words
[bigram.ind1, bigram.ind2,
bigram.count] =
textread('Datasets/bigram.txt','%d %d
%d');

% Size of vocabulary
N= length(tokens.words);

%% Part a

% ML estimate of unigram distribution
tokens.priors = tokens.counts /
sum(tokens.counts);

% Table of tokens starting with "M"
with their probabilities
k = 1;
for i = 1:N
    word = tokens.words{i};

    % Check if word starts with "M"
    if(word(1) == 'M')
        Word{k} = word;
        Probability{k} =
tokens.priors(i);
        k = k+1;
    end
end

Word = Word'; Probability =
Probability';

% Make table of the "M" tokens
tokensFromM = table(Word,
Probability);
clear Word Probability word k i

%% Part B

% Probability of word "The" in
vocabulary
IndexThe = find(strcmp(tokens.words,
'THE'));
CountThe = tokens.counts(IndexThe);
tmpIndex = find(bigram.ind1 ==
IndexThe);

% Find index of words following "the"
IndexFollowingThe =
bigram.ind2(tmpIndex);

% Store words following "the"
k = 1;
for i = 1:length(IndexFollowingThe)
    WordsFollowingThe{k} =
tokens.words{IndexFollowingThe(i)};
    k = k+1;
end
WordsFollowingThe =
WordsFollowingThe';

% Store bigram counts of words after
"the"
CountsFollowingThe =
bigram.count(tmpIndex);

% Compute their bigram probabilities
```

```
bigramProbThe = CountsFollowingThe /
CountThe;

% Sort probabilities and get most
likely words
[Probabilities, tmpInd] =
sort(bigramProbThe, 'descend');
Words = WordsFollowingThe(tmpInd);

% Make table of top 10 words
BigramTheTop10 = table(Words(1:10),
Probabilities(1:10));
clear IndexThe CountThe tmpIndex
tmpInd CountsFollowingThe
IndexFollowingThe word Words
Probabilities i k

%% Part C

% Convert given sentence to upper case
sentence = upper('The stock market
fell by one hundred points last
week');

% Log-likelihood under both models
[uLogC, bLogC] =
computeSentenceProbabilities(sentence,
tokens, bigram);

%% Part D

% Repeat steps from Part C
sentence = upper('The sixteen
officials sold fire insurance');
[uLogD, bLogD, pU, pB] =
computeSentenceProbabilities(sentence,
tokens, bigram);

%% Part E

% Vary lambda
lambda = 0.0001 : 0.0001 : 1;
for i = 1:length(lambda)
    % Weighted interpolation of
unigram and bigram models
    pM = lambda(i)*pU + (1-
lambda(i))*pB;
    Lm(i) = log(prod(pM));
end

% Plot of log-likelihood vs lambda
plot(lambda, Lm);
set(gcf, 'color', 'w');
grid on;
xlabel('\lambda');
ylabel('L_{m}');
title('Plot of log-likelihood L_{m} vs
\lambda');
[Peak, PeakIdx] = findpeaks(Lm);
hold on;
% text(lambda(PeakIdx), Peak,
sprintf('Peak = %6.3f', Peak))
plot(lambda(PeakIdx),Peak,'r^','marker
facecolor',[1 0 1])

% Compute probability of words in a
sentence using unigram and bigram
% models

function [uLog, bLog, pU, pB] =
computeSentenceProbabilities(sentence,
tokens, bigram)
    % Make array of words in sentence
    wordsU = strsplit(sentence);
    beginSent = {'<s>'};
    wordsB = [beginSent wordsU];

    % Log-likelihood using unigram
model
    pWords = 1;
    for i = 1:length(wordsU)
        tmpInd =
find(strcmp(tokens.words, wordsU{i}));

        % If word not in vocab, assign
"unknown" token
        if(isempty(tmpInd))
            tmpInd = 1;
        end

        pU(i) = tokens.priors(tmpInd);
        % Compute unigram probability
        pWords = pWords* pU(i);
    end
    uLog = log(pWords);

    % Log-likelihood using bigram
model
    pWords = 1;
    for i = 2:length(wordsB)
        w1 = wordsB{i-1};
        w2 = wordsB{i};

        % Find index of adjacent words
        ind1 =
find(strcmp(tokens.words, w1));

        if(isempty(tmpInd))
            ind1 = 1;
        end
        ind2 =
find(strcmp(tokens.words, w2));
        if(isempty(tmpInd))
            ind2 = 1;
        end

        cnt = 0;

        % Find counts of this pair
        for j = 1:length(bigram.count)
            if((bigram.ind1(j) ==
ind1) && (bigram.ind2(j) == ind2))
                cnt = bigram.count(j);
                break;
            end
        end

        % Compute bigram probability
        pB(i-1) = (cnt /
tokens.counts(ind1));
        pWords = pWords*pB(i-1);
    end
    bLog = log(pWords);
end
```

Q4

Source code - Since python had an simpler package, I used numpy.

```
# coding: utf-8

# In[44]:

import math
import numpy as np


# In[20]:

f = open('nasdaq00.txt','r')
p0 = [float(k) for k in
f.read().split('\n')]
f.close()
f = open('nasdaq01.txt','r')
p1 = [float(k) for k in
f.read().split('\n')]
f.close()


# In[142]:

b = list()
x = list()
for i in range(3,len(p0)):
    b.append(p0[i])
    x.append([p0[i-1],p0[i-
2],p0[i-3]])
b=np.array(b)
x=np.matrix(x)


# In[144]:

pinvx = np.linalg.pinv(x)
coeff = pinvx.dot(b)


# In[148]:

a = np.array(coeff)[0][0]
b = np.array(coeff)[0][1]
c = np.array(coeff)[0][2]
print a,b,c


# In[154]:

err0 = 0.0
for i in range(3,len(p0)):
    err0+=(a*p0[i-1] + b*p0[i-2] +
c*p0[i-3] - p0[i])**2
err0/=(len(p0)-3)*1.0


# In[155]:

err1 = 0.0
for i in range(3,len(p1)):
    err1+=(a*p1[i-1] + b*p1[i-2] +
c*p1[i-3] - p1[i])**2
err1/=(len(p1)-3)*1.0


# In[156]:

err0


# In[157]:

err1
```