4.1 Maximum likelihood estimation of a multinomial distribution

A 2D-sided die is tossed many times, and the result of each toss are recorded as data.

We suppose that in the course of the experiment, the d^{th} side of the die is observed C_d times.

We assume that the tosses are i.i.d according to the probabilities of the die.

a) Log-likelihood

We let $X \in \{1,2,3,\dots,2D\}$ denote the outcome of a toss, and let $p_d = P(X=d)$ denote the probabilities of the die.

Express the log-likelihood $\mathcal{L} = \log P(\text{data})$ of the observed results in terms of the probabilities p_d and the counts C_d.

$$\begin{aligned}
\mathcal{L} &= \log P(\text{data}) \\
&= \log\left[\prod_{c=1}^{C_d} P(X_c)\right] \\
&= \sum_{d=1}^{2D}\sum_{c=1}^{C_d} \log \underbrace{P(X_c = d)}_{p_d} \\
&= \underline{\underline{\sum_{d=1}^{2D} C_d \log p_d}}
\end{aligned}$$

b) Maximum likelihood estimate

Derive the maximum likelihood estimates of the die's probabilities p_d. Specifically, we are to maximize the expression for $\mathcal{L}$ from a) s.t. the constraints $\sum_{d=1}^{2D} p_d = 1$ & $p_d \geq 0$.

We apply a Lagrange multiplier for the equality constraint to get

$$\mathcal{L} = C_d \log p_d - \lambda\left[\sum_{d=1}^{2D} p_d - 1\right] \qquad (1)$$

We take the partial derivative of $\mathcal{L}$ with respect to p_d to get

$$\frac{\partial \mathcal{L}}{\partial p_d} = \frac{C_d}{p_d} - \lambda = 0 \qquad \text{to find the optimum}$$

$$\Rightarrow \quad p_d = \frac{C_d}{\lambda} \qquad (2)$$

From the constraint in the problem, we have that

$$\sum_{d=1}^{2D} p_d = 1 = \frac{\sum_{d=1}^{2D} C_d}{\lambda} \quad \Rightarrow \quad \lambda = \sum_{d=1}^{2D} C_d \qquad (3)$$

From equation (2) we now have

$$p_d = \frac{C_d}{\lambda} = \frac{C_d}{\sum_{d=1}^{2D} C_d}$$

As the count C_d is nonnegative, the estimate of p_d is nonnegative as well. ■

c) **Even vs odd**

Compute the probability $P(X \in \{2,4,6,\dots,2D\})$ that a roll is even and $P(X \in \{1,3,5,\dots,2D-1\})$ that a roll is odd. Show that they're equal when $\sum_{d=1}^{2D}(-1)^d p_d = 0$.

$$P(X \in \{2,4,6,\dots,2D\}) = \sum_{d=1}^{2D} \frac{(1+(-1)^d)}{2} p_d$$

$$P(X \in \{1,3,5,\dots,2D-1\}) = \sum_{d=1}^{2D} \frac{(1-(-1)^d)}{2} p_d$$

$$\sum_{d=1}^{2D}(-1)^d p_d = 0 \Rightarrow -\sum_{d=1}^{2D} \frac{(1-(-1)^d)}{2} p_d + \sum_{d=1}^{2D} \frac{(1+(-1)^d)}{2} p_d = 0$$

$$\Rightarrow \sum_{d=1}^{2D} \frac{(1-(-1)^d)}{2} p_d = \sum_{d=1}^{2D} \frac{(1+(-1)^d)}{2} p_d \quad \blacksquare$$

d) **Maximum likelihood estimate**

We suppose it is known a priori that p_{even} is equal to p_{odd}.

Derive the maximum likelihood estimates of the die's probs p_d subject to the added constraint.

Specifically, maximize ℓ from a) subject to the constraints $\sum_{d=1}^{2D} p_d = 1$, $\sum_{d=1}^{2D}(-1)^d p_d = 0$, $p_d \geq 0$.

We can now rewrite $\ell = \sum_{d=1}^{2D} C_d \log p_d = \sum_{d=1}^{D} C_{2d} \log p_{2d} + \sum_{d=1}^{D} C_{2d-1} \log p_{2d-1}$

Since the two are independently constrained, we can subdivide the problem into an odd part and an even part.

Odd:

We introduce a Lagrange multiplier for the equality constraint

$$\ell = C_{2d} \log p_{2d} - \lambda_1 \left[\sum_{d=1}^{D} p_{2d} - \frac{1}{2} \right] \tag{4}$$

We take the partial derivative of ℓ with respect to p_d to get

$$\frac{\partial \ell}{\partial p_{2d}} = \frac{C_{2d}}{p_{2d}} - \lambda_1 = 0 \quad \text{to find the optimum}$$

$$\Rightarrow p_{2d} = \frac{C_{2d}}{\lambda_1} \tag{5}$$

From the constraint in the problem, we have that

$$\sum_{d=1}^{D} p_{2d} = \frac{1}{2} = \frac{\sum_{d=1}^{D} C_{2d}}{\lambda_1} \Rightarrow \lambda_1 = 2\sum_{d=1}^{D} C_{2d} \tag{6}$$

From equation (5) we now have

$$p_{2d} = \frac{C_{2d}}{\lambda_1} = \frac{C_{2d}}{2\sum_{d=1}^{D} C_{2d}}$$

As the count C_{2d} is nonnegative, the estimate of p_{2d} is nonnegative as well.

Even:

We introduce a Lagrange multiplier for the equality constraint

$$\mathcal{L} = C_{2d-1}\log p_{2d-1} - \lambda_2\left[\sum_{d=1}^{D} p_{2d-1} - \frac{1}{2}\right] \tag{7}$$

We take the partial derivative of $\mathcal{L}$ with respect to p_d to get

$$\frac{\partial \mathcal{L}}{\partial p_{2d-1}} = \frac{C_{2d-1}}{p_{2d-1}} - \lambda_2 = 0 \qquad \text{to find the optimum}$$

$$\Rightarrow \quad p_{2d-1} = \frac{C_{2d-1}}{\lambda_2} \tag{8}$$

From the constraint in the problem, we have that

$$\sum_{d=1}^{D} p_{2d-1} = \frac{1}{2} = \frac{\sum_{d=1}^{D} C_{2d-1}}{\lambda_2} \quad \Rightarrow \quad \lambda_2 = 2\sum_{d=1}^{D} C_{2d-1} \tag{9}$$

From equation (8) we now have

$$p_{2d-1} = \frac{C_{2d-1}}{\lambda_2} = \frac{C_{2d-1}}{2\sum_{d=1}^{D} C_{2d-1}}$$

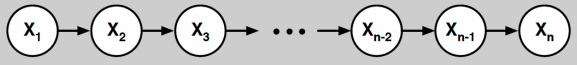
As the count C_{2d-1} is nonnegative, the estimate of p_{2d-1} is nonnegative as well.

Thus, as all counts C_d are nonnegative, so the estimate of p_d is nonnegative as well. ■

4.2 Maximum likelihood estimation in belief networks

We consider the two DAGs below, G_1 and G_2, over the same nodes $\{X_1, X_2, \ldots, X_n\}$, that differ only in edge direction.

G_1



G_2

We suppose that we have a fully observed data set $\{x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}\}_{t=1}^{T}$ in which each example provides a complete instantiation of the notes in these DAGs. We let $\text{COUNT}_i(x)$ denote the number of examples in which $X_i = x$ and we let $\text{COUNT}_i(x, x')$ denote the number of examples in which $X_i = x$ and $X_{i+1} = x'$.

a) Express the maximum likelihood estimates for the CPTs in G_1 in terms of these counts.

For the root node, we have

$$P_{1_{ML}}(X_1 = x_1) = \frac{\text{COUNT}(X_1 = x_1)}{T}$$

For the nodes with parents, we have

$$P_{i+1_{ML}}(X_{i+1} = x_{i+1} \mid X_i = x_i) = \frac{\text{COUNT}(X_i = x_i, X_{i+1} = x_{i+1})}{\text{COUNT}(X_i = x_i)}$$

b) Express the maximum likelihood estimates for the CPTs in G_2 in terms of these counts.

For the last node, we have

$$P_{n_{ML}}(X_n = x_n) = \frac{\text{COUNT}(X_n = x_n)}{T}$$

For the other nodes, we have

$$P_{i_{ML}}(X_i = x_i \mid X_{i+1} = x_{i+1}) = \frac{\text{COUNT}(X_{i+1} = x_{i+1}, X_i = x_i)}{\text{COUNT}(X_{i+1} = x_{i+1})}$$

c) Using the answers from a) and b), show that the ML CPTs for G_1 and G_2 from this data set give rise to the same joint distribution over the nodes $\{X_1, \dots, X_n\}$

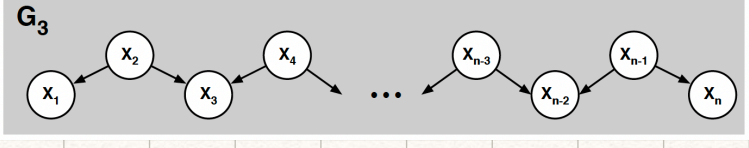
$$P(G_1) = P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

$$= P(X_1 = x_1) \prod_{i=1}^{n-1} P(X_{i+1} = x_{i+1} \mid X_i = x_i)$$

$$= \frac{\text{COUNT}(X_1 = x_1)}{T} \prod_{i=1}^{n-1} \frac{\text{COUNT}(X_i = x_i, X_{i+1} = x_{i+1})}{\text{COUNT}(X_i = x_i)}$$

$$= \frac{\prod_{i=1}^{n-1} \text{COUNT}(X_i = x_i, X_{i+1} = x_{i+1})}{T \prod_{i=2}^{n-1} \text{COUNT}(X_i = x_i)}$$

$$= \frac{\text{COUNT}(X_n = x_n)}{T} \prod_{i=1}^{n-1} \frac{\text{COUNT}(X_{i+1} = x_{i+1}, X_i = x_i)}{\text{COUNT}(X_{i+1} = x_{i+1})} = P(X_n = x_n) \prod_{i=1}^{n-1} (X_i = x_i \mid X_{i+1} = x_{i+1}) = P(G_2) \quad \blacksquare$$



Suppose that some but not all edges in these DAGs were reversed, as in G_3 shown above.

d) Would the maximum likelihood CPTs for G_3 also give rise to the same joint distribution?

In G_1 and G_2, we have all CPTs given by for example $P(X_3 \mid X_2)$ or $P(X_2 \mid X_3)$.

In G_3 we get $P(X_3 \mid X_2, X_4)$. Therefore, as we dont have the same T for G_3 as we have in G_1 and G_2.

As the individual probabilities are not equal $\forall i$, the joint probability will not be the same.

4.3 Statistical language modeling

a) Compute the maximum likelihood estimate of the unigram distribution $P_u(w)$ over words w. Print out a table of all the tokens (i.e., words) that start with the letter "M", along with their numerical unigram *probabilities* (not counts). (You do not need to print out the unigram probabilities for all 500 tokens.)

	Words	Unigram Probs
53	MILLION	0.002073
68	MORE	0.001709
76	MR.	0.001442
120	MOST	0.000788
121	MARKET	0.000780
125	MAY	0.000730
129	M.	0.000703
130	MANY	0.000697
158	MADE	0.000560
177	MUCH	0.000515
179	MAKE	0.000514
202	MONTH	0.000445
208	MONEY	0.000437
226	MONTHS	0.000406
229	MY	0.000400
246	MONDAY	0.000382
255	MAJOR	0.000371
274	MILITARY	0.000352
286	MEMBERS	0.000336
355	MIGHT	0.000274
365	MEETING	0.000266
369	MUST	0.000267
373	ME	0.000264
374	MARCH	0.000260
384	MAN	0.000253
402	MS.	0.000239
403	MINISTER	0.000240
459	MAKING	0.000212
472	MOVE	0.000210
478	MILES	0.000206

b) Compute the maximum likelihood estimate of the bigram distribution $P_b(w'|w)$. Print out a table of the ten most likely words to follow the word "THE", along with their numerical bigram probabilities.

```
      w1   w2  count(w1,w2)  Bigram Probs
993    4    1       2371132      0.615020
1058   4   70         51556      0.013372
1064   4   79         45186      0.011720
1060   4   73         44949      0.011659
1050   4   61         36439      0.009451
1165   4  184         33435      0.008672
1086   4  103         26230      0.006803
1029   4   39         25641      0.006651
1282   4  308         24239      0.006287
1014   4   23         23752      0.006161

 We access the words from the vocab dataframe to print them out nicely
Word: <UNK>              Probability: 0.61502
Word: U.                 Probability: 0.013372
Word: FIRST              Probability: 0.01172
Word: COMPANY            Probability: 0.011659
Word: NEW                Probability: 0.009451
Word: UNITED             Probability: 0.008672
Word: GOVERNMENT         Probability: 0.006803
Word: NINETEEN           Probability: 0.006651
Word: SAME               Probability: 0.006287
Word: TWO                Probability: 0.006161
```

c) Consider the sentence **"The stock market fell by one hundred points last week."** Ignoring punctuation, compute and compare the log-likelihoods of this sentence under the unigram and bigram models:

$$\mathcal{L}_u = \log\Big[P_u(\textbf{the})\,P_u(\textbf{stock})\,P_u(\textbf{market})\,\ldots\,P_u(\textbf{points})\,P_u(\textbf{last})P_u(\textbf{week})\Big]$$

$$\mathcal{L}_b = \log\Big[P_b(\textbf{the}|\langle\textbf{s}\rangle)\,P_b(\textbf{stock}|\textbf{the})\,P_b(\textbf{market}|\textbf{stock})\,\ldots\,P_b(\textbf{last}|\textbf{points})\,P_b(\textbf{week}|\textbf{last})\Big]$$

In the equation for the bigram log-likelihood, the token $\langle s\rangle$ is used to mark the beginning of a sentence. Which model yields the highest log-likelihood?

```
Unigram log-likelihood: -64.50944
Bigram log-likelihood: -40.91813
```

We observe that the bigram model yields a better log-likelihood.

d) Consider the sentence **"The sixteen officials sold fire insurance."** Ignoring punctuation, compute and compare the log-likelihoods of this sentence under the unigram and bigram models:

$$\mathcal{L}_u = \log\Big[P_u(\textbf{the})\,P_u(\textbf{sixteen})\,P_u(\textbf{officials})\,\ldots\,P_u(\textbf{sold})\,P_u(\textbf{fire})\,P_u(\textbf{insurance})\Big]$$

$$\mathcal{L}_b = \log\Big[P_b(\textbf{the}|\langle\textbf{s}\rangle)\,P_b(\textbf{sixteen}|\textbf{the})\,P_b(\textbf{officials}|\textbf{sixteen})\,\ldots\,P_b(\textbf{fire}|\textbf{sold})\,P_b(\textbf{insurance}|\textbf{fire})\Big]$$

Which pairs of adjacent words in this sentence are not observed in the training corpus? What effect does this have on the log-likelihood from the bigram model?

```
Unigram log-likelihood: -44.29193

The combination of SIXTEEN and OFFICIALS is not present in the bigram data, resulting in log(0) which is -inf
The combination of SOLD and FIRE is not present in the bigram data, resulting in log(0) which is -inf
Bigram log-likelihood: -inf
```

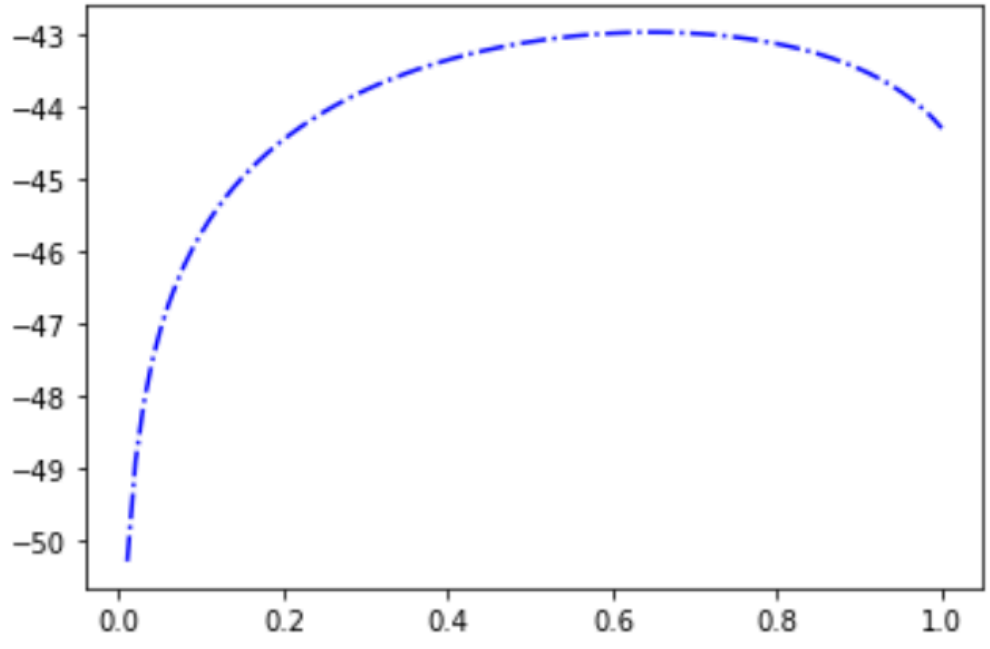
e) Consider the so-called *mixture* model that predicts words from a weighted interpolation of the unigram and bigram models:

$$P_m(w'|w) = \lambda P_u(w') + (1-\lambda)P_b(w'|w),$$

where $\lambda \in [0,1]$ determines how much weight is attached to each prediction. Under this mixture model, the log-likelihood of the sentence from part (d) is given by:

$$\mathcal{L}_m = \log\Big[P_m(\textbf{the}|\langle\textbf{s}\rangle)\,P_m(\textbf{sixteen}|\textbf{the})\,P_m(\textbf{officials}|\textbf{sixteen})\,\ldots\,P_m(\textbf{fire}|\textbf{sold})\,P_m(\textbf{insurance}|\textbf{fire})\Big]$$

Compute and plot the value of this log-likelihood $\mathcal{L}_m$ as a function of the parameter $\lambda \in [0,1]$. From your results, deduce the optimal value of λ to two significant digits.




```
Highest lambda value is 0.65
```

f) Submit a printout of your source code for the previous parts of this problem.

a)

Compute the maximum likelihood estimate of the unigram distribution $P_u(w)$ over words w. Print out a table of all the tokens (i.e., words) that start with the letter "M", along with their numerical unigram probabilities (not counts).

[1]:
```
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
pd.options.mode.chained_assignment = None

# Import vocabulary and unigrams as pandas dataframes
vocab_data = pd.read_csv('hw4_vocab.txt',sep="\t",quoting=csv.
 ↪QUOTE_NONE,header=None,names=["Words"])
unigram_data = pd.read_csv('hw4_unigram.txt',sep="\t",␣
 ↪header=None,names=["Count"])

# Add a column for the unigram probabilities in the vocab dataframe
vocab_data["Unigram Probs"] = unigram_data["Count"]/unigram_data["Count"].sum()

# Print the words starting with M and their probabilities
vocab_data[vocab_data["Words"].str.startswith("M")]
```

[1]:

	Words	Unigram Probs
53	MILLION	0.002073
68	MORE	0.001709
76	MR.	0.001442
120	MOST	0.000788
121	MARKET	0.000780
125	MAY	0.000730
129	M.	0.000703
130	MANY	0.000697
158	MADE	0.000560
177	MUCH	0.000515
179	MAKE	0.000514
202	MONTH	0.000445
208	MONEY	0.000437
226	MONTHS	0.000406
229	MY	0.000400
246	MONDAY	0.000382
255	MAJOR	0.000371
274	MILITARY	0.000352
286	MEMBERS	0.000336
355	MIGHT	0.000274
365	MEETING	0.000266
369	MUST	0.000267
373	ME	0.000264
374	MARCH	0.000260
384	MAN	0.000253
402	MS.	0.000239
403	MINISTER	0.000240
459	MAKING	0.000212
472	MOVE	0.000210
478	MILES	0.000206

b)

Compute the maximum likelihood estimate of the bigram distribution $P_b(w'|w)$. Print out a table of the ten most likely words to follow the word "THE", along with their numerical bigram probabilities.

[2]:
```
# Read the bigram file as a dataframe
bigram_data = pd.read_csv('hw4_bigram.
 ↪txt',sep="\t",header=None,names=["w1","w2","count(w1,w2)"])
# Retrieve the index of the word "THE" in the previous dataframe
index = vocab_data.index
condition = vocab_data["Words"] == "THE"
the_index_list = index[condition]
the_index = the_index_list[0]+1

# Make a new dataframe with only the "THE" word as "w1"
the_bigram = bigram_data[bigram_data["w1"]==the_index]

# Save the bigram probabilities as a column in the bigram dataframe
word_sum = the_bigram["count(w1,w2)"].sum()
the_bigram["Bigram Probs"] = the_bigram["count(w1,w2)"]/word_sum
the_bigram = the_bigram.sort_values("Bigram Probs", ascending=False)
ten_words_after_the = the_bigram.head(10)
print(ten_words_after_the)

print("\n We access the words from the vocab dataframe to print them out nicely")
```

```
# Save words and probabilities as lists to print out nicely
bigram_prob_list = ten_words_after_the["Bigram Probs"].tolist()
second_word = []
# Add all second words to list to easily print out
second_word.extend((vocab_data.loc[0,"Words"],vocab_data.
 ↪loc[69,"Words"],vocab_data.loc[78,"Words"],vocab_data.
 ↪loc[72,"Words"],vocab_data.loc[60,"Words"],vocab_data.
 ↪loc[183,"Words"],vocab_data.loc[102,"Words"],vocab_data.
 ↪loc[38,"Words"],vocab_data.loc[307,"Words"],vocab_data.loc[22,"Words"]))
for i in range(len(second_word)):
    if (second_word[i]=="GOVERNMENT"):
        print("Word: {} \t Probability: {}".format(second_word[i],␣
 ↪round(bigram_prob_list[i],6)))
    else:
        print("Word: {} \t \t Probability: {}".format(second_word[i],␣
 ↪round(bigram_prob_list[i],6)))
```

```
      w1   w2  count(w1,w2)  Bigram Probs
993    4    1       2371132      0.615020
1058   4   70         51556      0.013372
1064   4   79         45186      0.011720
1060   4   73         44949      0.011659
1050   4   61         36439      0.009451
1165   4  184         33435      0.008672
1086   4  103         26230      0.006803
1029   4   39         25641      0.006651
1282   4  308         24239      0.006287
1014   4   23         23752      0.006161
```

```
 We access the words from the vocab dataframe to print them out nicely
Word: <UNK>              Probability: 0.61502
Word: U.                 Probability: 0.013372
Word: FIRST              Probability: 0.01172
Word: COMPANY            Probability: 0.011659
Word: NEW                Probability: 0.009451
Word: UNITED             Probability: 0.008672
Word: GOVERNMENT         Probability: 0.006803
Word: NINETEEN           Probability: 0.006651
Word: SAME               Probability: 0.006287
Word: TWO                Probability: 0.006161
```

c)

Consider the sentence **"The stock market fell by one hundred points last week."** Ignoring punctuation, compute and compare the log-likelihoods of this sentence under the unigram and bigram models:

$$\mathcal{L}_u = log[P_u((\textbf{The}))\mathbf{P_u}((\textbf{stock}))\mathbf{P_u}((\textbf{market}))...\mathbf{P_u}((\textbf{last}))\mathbf{P_u}((\textbf{week}))]$$

$$\mathcal{L}_b = log[P_b((\textbf{The}|<\mathbf{s}>))\mathbf{P_b}((\textbf{stock}|\textbf{the}))\mathbf{P_b}((\textbf{market}|\textbf{stock}))...\mathbf{P_b}((\textbf{last}|\textbf{points}))\mathbf{P_b}((\textbf{week}|\textbf{last}))]$$

In the equation for the bigram log-likelihood, the token $< s >$ is used to mark the beginning of a sentence. Which model yields the highest log-likelihood?

[3]:
```
# I see now that I should have saved bigram probabilities for all words, so I␣
 ↪will do that
bigram_probs = []
for index,row in bigram_data.iterrows():
    bigram_probs.append(row["count(w1,w2)"]/unigram_data.
 ↪iloc[row["w1"]-1]["Count"])
bigram_data["Bigram Probs"] = bigram_probs
```

[4]:
```
sentence1 = "THE STOCK MARKET FELL BY ONE HUNDRED POINTS LAST WEEK"
sentence1 = sentence1.split()
indeces1 = []
for word in sentence1:
    index = vocab_data.index
    condition = vocab_data["Words"] == word
    word_index = index[condition]
    indeces1.append(word_index[0])
```

[5]:
```
# Unigram log-likelihood
Lu = 0
for index in indeces1:
    Lu += np.log(vocab_data.iloc[index]["Unigram Probs"])
print(f"Unigram log-likelihood: {np.round(Lu,5)}")
```

```
Unigram log-likelihood: -64.50944
```

[6]:
```
sentence2 = "<s> THE STOCK MARKET FELL BY ONE HUNDRED POINTS LAST WEEK"
sentence2 = sentence2.split()
indeces2 = []
for word in sentence2:
    index = vocab_data.index
    condition = vocab_data["Words"] == word
    word_index = index[condition]+1
    indeces2.append(word_index[0])
```

[13]:
```
# Bigram log-likelihood
Lb = 0
for i in range(len(indeces2)-1):
    # Last word on the line is .values[0]
    Lb+=np.
 ↪log(bigram_data[(bigram_data["w1"]==indeces2[i])&(bigram_data["w2"]==indeces2[i+1])]["Bigram␣
 ↪Probs"].values[0])
```

```
print(f"Bigram log-likelihood: {np.round(Lb,5)}")
```

```
Bigram log-likelihood: -40.91813
```

We observe that the bigram model yields a better log-likelihood.

d)

Consider the sentence **"The sixteen officials sold fire insurance."** Ignoring punctuation, compute and compare the log-likelihoods of this sentence under the unigram and bigram models:

$$\mathcal{L}_u = log[P_u(\mathbf{The})\mathbf{P_u}(\mathbf{sixteen})\mathbf{P_u}(\mathbf{officials})...\mathbf{P_u}(\mathbf{fire})\mathbf{P_u}(\mathbf{insurance})]$$

$$\mathcal{L}_b = log[P_b(\mathbf{The}|<\mathbf{s}>)\mathbf{P_b}(\mathbf{sixteen}|\mathbf{the})\mathbf{P_b}(\mathbf{officials}|\mathbf{sixteen})...\mathbf{P_b}(\mathbf{fire}|\mathbf{sold})\mathbf{P_b}(\mathbf{insurance}|\mathbf{fire})]$$

Which pairs of adjacent words in this sentence are not observed in the training corpus? What effect does this have on the log-likelihood from the bigram model?

[8]:
```
sentence3 = "THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
sentence3 = sentence3.split()
indeces3 = []
for word in sentence3:
    index = vocab_data.index
    condition = vocab_data["Words"] == word
    word_index = index[condition]
    indeces3.append(word_index[0])
```

[9]:
```
# Unigram log-likelihood
Lu = 0
for index in indeces3:
    Lu += np.log(vocab_data.iloc[index]["Unigram Probs"])
print(f"Unigram log-likelihood: {np.round(Lu,5)}")
```

```
Unigram log-likelihood: -44.29193
```

[10]:
```
sentence4 = "<s> THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
sentence4 = sentence4.split()
indeces4 = []
for word in sentence4:
    index = vocab_data.index
    condition = vocab_data["Words"] == word
    word_index = index[condition]+1
    indeces4.append(word_index[0])
```

[15]:
```
# Bigram log-likelihood
Lb = 0
for i in range(len(indeces4)-1):
    try:
        # Last word on the line is .values[0]
```

```
        prob =␣
 ↪bigram_data[(bigram_data["w1"]==indeces4[i])&(bigram_data["w2"]==indeces4[i+1])]["Bigram␣
 ↪Probs"].values[0]
    except IndexError:
        print(f"The combination of {sentence4[i]} and {sentence4[i+1]} is not␣
 ↪present in the bigram data, resulting in log(0) which is -inf")
        Lb = float('-inf')
    else:
        Lb+=np.log(prob)
print(f"Bigram log-likelihood: {np.round(Lb,5)}")
#print(f"The combination of {sentence4[i]} and {sentence4[i+1]} is not present␣
 ↪in the bigram data")
```

```
The combination of SIXTEEN and OFFICIALS is not present in the bigram data,
resulting in log(0) which is -inf
The combination of SOLD and FIRE is not present in the bigram data, resulting in
log(0) which is -inf
Bigram log-likelihood: -inf
```

e)

Consider the so-called mixture model that predicts words from a weighted interpolation of the unigram and bigram models:

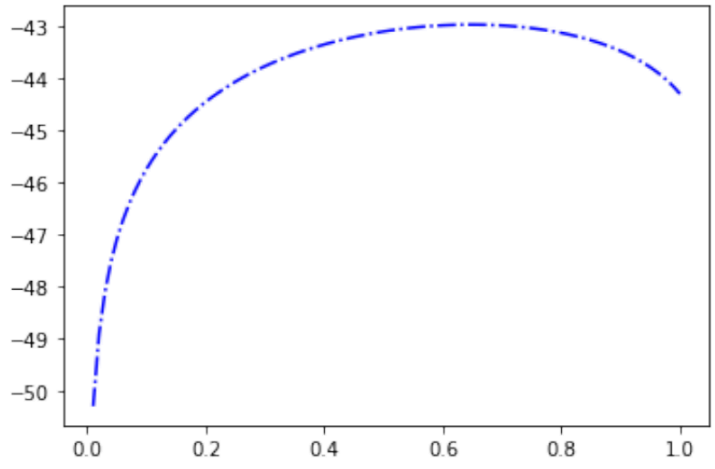
$$P_m(w'|w) = \lambda P_u(w') + (1-\lambda)P_b(w'|w)$$

where $\lambda \in [0,1]$ determines how much weight is attached to each prediction. Under this mixture model, the log-likelihood of the sentence from part (d) is given by:

$$\mathcal{L}_m = log[P_m(\mathbf{The}|<\mathbf{s}>)\mathbf{P_m}(\mathbf{sixteen}|\mathbf{the})\mathbf{P_m}(\mathbf{officials}|\mathbf{sixteen})...\mathbf{P_m}(\mathbf{fire}|\mathbf{sold})\mathbf{P_m}(\mathbf{insurance}|\mathbf{fire})].$$

Compute and plot the value of this log-likelihood $\mathcal{L}_m$ as a function of the parameter $\lambda \in [0,1]$. From your results, deduce the optimal value of λ to two significant digits.

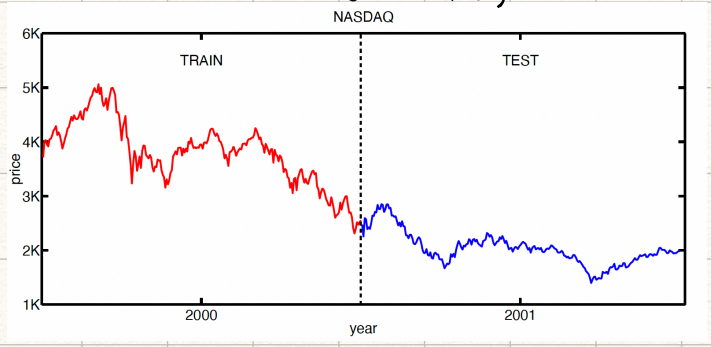
[12]:
```
sentence5 = "<s> THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
sentence5 = sentence5.split()
indeces5 = []
for word in sentence5:
    index = vocab_data.index
    condition = vocab_data["Words"] == word
    word_index = index[condition]+1
    indeces5.append(word_index[0])

def mixture_prob(lam, p_uni, p_bi):
    return (np.log(lam*p_uni+(1-lam)*p_bi))

def mixture_mle(sentence5, lam):
    p = 0
    for i in range(len(sentence5) - 1):
        p_uni = vocab_data[vocab_data["Words"]==sentence5[i+1]]["Unigram Probs"].
↪values[0]
        try:
            # Last word on the line is .values[0]
           ␣
↪p_bi=bigram_data[(bigram_data["w1"]==indeces5[i])&(bigram_data["w2"]==indeces5[i+1])]["Bigram
↪Probs"].values[0]
        except IndexError:
            p_bi = 0
        p_ = mixture_prob(lam,p_uni,p_bi)
        p += p_
    return p

x = []
y = []
for i in np.linspace(0.01, 1, 100):
    x.append(i)
    y.append(mixture_mle(sentence5, i))
plt.plot(x, y, 'b-.')
plt.show()

np.array(x)
np.array(y)
print(f"Highest lambda value is {round(x[np.argmax(y)],2)}")
```




```
Highest lambda value is 0.65
```

4.4 Stock market prediction

In this problem, we will apply a simple linear model to predicting the stock market. We will download the files containing the NASDAQ indices at the close of business days in 2000 and 2001.



a) Linear coefficients

How accurately can the index on one day be predicted by a linear combination of the three preceding indices? Using only data from the year 2000, compute the linear coefficients (a_1,a_2,a_3) that maximize the log-likelihood $\mathcal{L} = \sum_t \log P(x_t|x_{t-1}, x_{t-2}, x_{t-3})$, where:

$$P(x_t|x_{t-1}, x_{t-2}, x_{t-3}) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\Big(x_t - a_1 x_{t-1} - a_2 x_{t-2} - a_3 x_{t-3}\Big)^2\right],$$

and the sum is over business days in the year 2000 (starting from the fourth day).

```
A values are (a1,a2,a3) = (0.950673,0.015601,0.031896).
```

b) Root mean squared prediction error

For the coefficients estimated in part (a), compare the model's performance (in terms of root mean squared error) on the data from the years 2000 and 2001. (A rhetorical question: does a lower prediction error in 2001 indicate that the model worked better that year?)

```
RMSE Year 2000: 117.90844361778286
RMSE Year 2001: 54.486240219686444
```

We observe that the RMSE is lower in 2001 than in 2000, so the prediction is better on paper.

However, the stock market is very hard to predict, so this might just have been a coincidence.

c) Source code

Turn in a print-out of your source code.

a)

How accurately can the index on one day be predicted by a linear combination of the three preceding indices? Using only data from the year 2000 , compute the linear coefficients (a_1, a_2, a_3) that maximize the log-likelihood $\mathcal{L} = \sum_t \log P(x_t|x_{t-1}, x_{t-2}, x_{t-3})$, where:

$$P(x_t|x_{t-1}, x_{t-2}, x_{t-3}) = \frac{1}{\sqrt{2\pi}} \exp[-\frac{1}{2}(x_t - a_1 x_{t-1} - a_2 x_{t-2} - a_3 x_{t-3})^2]$$

and the sum is over business days in the year 2000 (starting from the fourth day).

[26]:
```
import numpy as np
from numpy.linalg import inv
from numpy import transpose as T
```

[32]:
```
# Initializating the matrices for X and Y
X_2000 = []
Y_2000 = []

# Opening file and iterating through lines to predict
with open('nasdaq00.txt','r') as f:
    t_3 = -1
    t_2 = -1
    t_1 = -1
    for line in f.readlines():
        line = line.rstrip()
        if t_1 != -1:
            X_2000.append([t_3, t_2, t_1])
            Y_2000.append(float(line))
        t_1 = t_2
        t_2 = t_3
        t_3 = float(line)

# Computing a values
X_2000 = np.matrix(X_2000)
Y_2000 = np.matrix(Y_2000).reshape(-1,1)
```

```
a_2000 = inv(T(X_2000) * X_2000) * T(X_2000) * Y_2000
a_1,a_2,a_3 = round(a_2000.item(0),6), round(a_2000.item(1),6), round(a_2000.
↪item(2),6)
print(f"A values are (a1,a2,a3) = ({a_1},{a_2},{a_3}).")
```

```
A values are (a1,a2,a3) = (0.950673,0.015601,0.031896).
```

[33]:
```
# Initializating the matrices for X and Y
X_2001 = []
Y_2001 = []

# Opening file and iterating through lines to predict
with open('nasdaq01.txt','r') as f:
    t_3 = -1
    t_2 = -1
    t_1 = -1
    for line in f.readlines():
        line = line.rstrip()
        if t_1 != -1:
            X_2001.append([t_3, t_2, t_1])
            Y_2001.append(float(line))
        t_1 = t_2
        t_2 = t_3
        t_3 = float(line)

# Computing matrices for MSE computing
X_2001 = np.matrix(X_2001)
Y_2001 = np.matrix(Y_2001).reshape(-1,1)
a_2001 = inv(T(X_2001) * X_2001) * T(X_2001) * Y_2001
```

[36]:
```
rmse_2000 = np.sqrt(np.square(Y_2000 - np.dot(X_2000,a_2000)).mean())
rmse_2001 = np.sqrt(np.square(Y_2001 - np.dot(X_2001,a_2001)).mean())
print(f"RMSE Year 2000: {rmse_2000}")
print(f"RMSE Year 2001: {rmse_2001}")
```

```
RMSE Year 2000: 117.90844361778286
RMSE Year 2001: 54.486240219686444
```

We observe that the RMSE is lower in 2001 than in 2000, so the prediction is better on paper.

However, the stock market is very hard to predict, so this might just have been a coincidence.