Homework Assignment 4 – Math 118, Winter 2021

Dhruv Chakraborty

March 14, 2021

Problem 1

Consider the following variant of the knapsack problem. Suppose that the Math 118 midterm consists of 5 problems, worth 5, 10, 20, 25 and 35 points respectively. However, the test is only out of 50 points. You may attempt as many of the problems as you like, and your instructor will grade all of them and choose the subset of problems whose totals add up to less than or equal to 50 giving you the best possible score. Suppose you attempt all five questions. In what follows, p_i represents the points achieved on the i-th question, while t_i represents the total points available on the i-th question.

$$p_1 = 4, t_1 = 5 \quad p_2 = 3, t_2 = 10 \quad p_3 = 17, t_3 = 20 \quad p_4 = 20, t_4 = 25 \quad p_5 = 30, t_5 = 35$$

Use dynamic programming to determine the optimal subset of questions, and your maximum achievable score. *(This is essentially the knapsack problem with $B = 50$, $s_i = t_i$ and $v_i = p_i$. Hint: Consider increasing the size of the "knapsack" in increments of* 5*).* Your answer should include the dynamic programming table you construct as part of the algorithm.

Solution:

We first generate the table below by using the fact that for $R \leq 50$ and $f(k,R) = \max_{A \subseteq \{1,\dots,k\}} \left\{ \sum_{i \in A} p_i : \sum_{i \in A} t_i \leq R \right\}$

we have $f(k,R) = \max \begin{cases} f(k-1, R-t_k) + p_k & \text{if } t_k \leq R \\ f(k-1,R) & \text{otherwise} \end{cases}.$

Total points → / Problem number ↓	0	5	10	15	20	25	30	35	40	45	50
0	0	0	0	0	0	0	0	0	0	0	0
1	0	4	4	4	4	4	4	4	4	4	4
2	0	4	4	7	7	7	7	7	7	7	7
3	0	4	4	7	17	21	21	24	24	24	24
4	0	4	4	7	17	21	24	24	27	37	41
5	0	4	4	7	17	21	24	30	34	37	41

Then the maximum achievable score is in the bottom right of the table i.e. it is $f(5,50) = 41$ points. We can find the optimal subset of questions by tracing back how each cell is calculated and including questions corresponding to rows arrived at by a diagonal move. We note in traversing the table from bottom right to top left (shown in red) that these diagonal moves happen in rows 4, 3 and 1. Checking that $p_1 + p_3 + p_4 = 41$ and $t_1 + t_3 + t_4 = 50$, we get that the optimal subset of questions are problems 1, 3 and 4.

The code used to compute the table above is reproduced below.

```
def knapsack(vals, costs, B):
    n = len(vals)+1
    table = [[None for R in range(0, B+1, 5)] for k in range(n)]

    for k in range(n):
        for R in range(0, B+1, 5):
            if k == 0 or R == 0:
                table[k][R//5] = 0
            elif costs[k-1] <= R:
                table[k][R//5] = max(table[k-1][(R-costs[k-1])//5] + vals[k-1], table[k-1][R//5])
            else:
                table[k][R//5] = table[k-1][R//5]

    display(table)
    return table[n-1][B//5]

knapsack([4, 3, 17, 20, 30], [5, 10, 20, 25, 35], 50)
```

Problem 2

Use dynamic programming to find the longest common subsequence between $\mathbf{X} = GGATCGT$ and $\mathbf{Y} = CGAGCTT$. Your answer should include the dynamic programming table you construct as part of the algorithm.

Solution:

We generate the table below by using the fact that $f(\mathbf{X}_i, \mathbf{Y}_j) = \begin{cases} f(\mathbf{X}_{i-1}, \mathbf{Y}_{j-1}) + 1 & \text{if } x_i = y_j \\ \max\{f(\mathbf{X}_{i-1}, \mathbf{Y}_j), f(\mathbf{X}_i, \mathbf{Y}_{j-1})\} & \text{otherwise} \end{cases}$
where $\mathbf{X}_i = x_1 x_2 ... x_i$, $\mathbf{Y}_j = y_1 y_2 ... y_j$ and $f(\mathbf{X}_i, \mathbf{Y}_j)$ is the longest common subsequence of $\mathbf{X}_i$ and $\mathbf{Y}_j$.

	-	C	G	A	G	C	T	T
-	0	0	0	0	0	0	0	0
G	0	←↑ 0	↖ 1	← 1	← 1	← 1	← 1	← 1
G	0	←↑ 0	↑ 1	←↑ 1	↖ 2	← 2	← 2	← 2
A	0	←↑ 0	↑ 1	↖ 2	←↑ 2	←↑ 2	←↑ 2	←↑ 2
T	0	←↑ 0	↑ 1	↑ 2	←↑ 2	←↑ 2	↖ 3	← 3
C	0	↖ 1	←↑ 1	↑ 2	←↑ 2	↖ 3	←↑ 3	←↑ 3
G	0	↑ 1	↖ 2	←↑ 2	↖ 3	←↑ 3	←↑ 3	←↑ 3
T	0	↑ 1	↑ 2	←↑ 2	↑ 3	←↑ 3	↖ 4	↖ 4

Given this table, we can find the longest common subsequences between $\mathbf{X}$ and $\mathbf{Y}$ (of length 4 here) by tracing back a directed path from the bottom right to the top left. The resulting subsequences are:

- *GACT*
- *GGCT*
- *GATT*
- *GGTT*
- *GAGT*

The code used to compute the table above is reproduced below.

```
def lcs(x, y):
    m = len(x)+1
    n = len(y)+1
    table = [[None]*n for i in range(m)]

    for i in range(m):
        for j in range(n):
            if i == 0 or j == 0:
                table[i][j] = 0
            elif x[i-1] == y[j-1]:
                table[i][j] = table[i-1][j-1]+1
            else:
                table[i][j] = max(table[i-1][j], table[i][j-1])

    display(table)
    return table[m-1][n-1]

lcs("GGATCGT", "CGAGCTT")
```

Problem 3

Tom Brady is the greatest quarterback of all time. However even the G.O.A.T. can have off days. In this question you will analyze Brady's performance using a Hidden Markov Model. Recall that as quarterback, Brady's goal is to throw passes that are successfully caught by his receivers. If a pass is successfully caught we say it is "complete" and if it is not caught we say it is "incomplete".

1. Suppose that Brady can be in one of two states: hot or cold. When he is hot, he completes 75% of his passes. When he is cold, he completes 55% of his passes. The probability of transition from hot to cold is 10% the probability of transition from cold to hot is 15%. Sketch a hidden Markov model, as we did in Lecture 21, encoding this information.
2. Suppose that Brady starts a particular game hot. In the first half of the game, we observe the following sequence of passes: $ICCCICCIIII$ where "I" denotes an incomplete pass while "C" denotes a complete pass. Use Viterbi's algorithm to decode the most likely sequence of states that Brady was in during this first half. Include your dynamic programming table as part of your answer.

Solution:

1. We can encode the given data to model Brady's performance as follows.

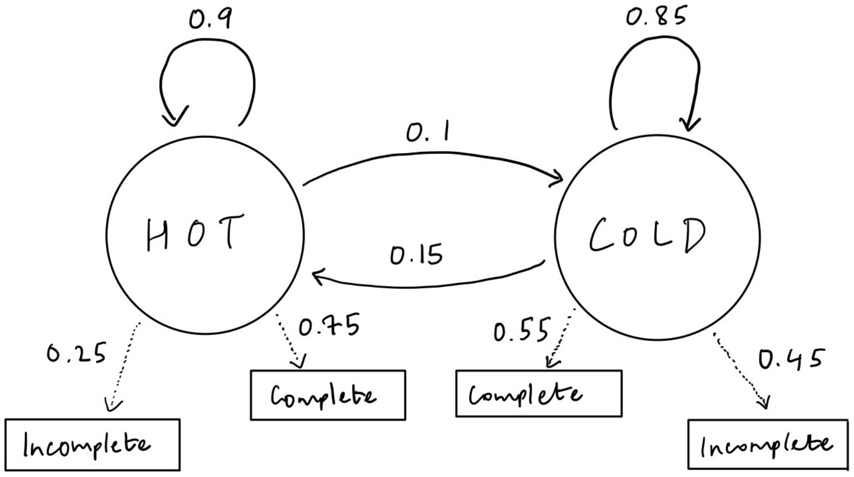

Figure 1: Hidden Markov Model describing Brady's pass completion performance.

2. We use the recursive function $f(i,k) = E_{i,y_k} \max_r A_{ir} f(r, k-1)$ where $E_{ij} = \mathbb{P}(y_t = e_j | x_t = i)$ and $A_{ij} = \mathbb{P}(x_{t+1} = i | x_t = j)$, with initial probabilities $\pi_1 := \mathbb{P}(x_1 = H) = 1$ and $\pi_2 := \mathbb{P}(x_1 = C) = 0$ to generate the table below.

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11
$x_k = H$	0.25	0.16875	0.11391	0.07689	0.0173	0.01168	0.00788	0.00177	0.0004	9e-05	2e-05
$x_k = T$	0.0	0.01375	0.00928	0.00626	0.00346	0.00162	0.00076	0.00035	0.00014	5e-05	2e-05

Using traceback as before, we get that the most likely sequence of states $\mathbf{X} = HHHHHHHHHHH$.

The code used to compute the table above and trace back the optimal sequence of states is reproduced below.

```
def viterbi(y, A, E, pi):
    K = A.shape[0]
    T = len(y)
    T1 = np.empty((K, T), 'd')
    T2 = np.empty((K, T), 'B')

    T1[:, 0] = pi * E[:, y[0]]
    T2[:, 0] = 0

    for i in range(1, T):
        T1[:, i] = np.max(T1[:, i - 1] * A.T * E[np.newaxis, :, y[i]].T, 1)
        T2[:, i] = np.argmax(T1[:, i - 1] * A.T, 1)

    x = np.empty(T, 'B')
    x[-1] = np.argmax(T1[:, T - 1])
    for i in reversed(range(1, T)):
        x[i - 1] = T2[x[i], i]

    return x, T1, T2

y = np.array([1,0,0,0,1,0,0,1,1,1,1])
A = np.array([[0.9, 0.1],[0.15, 0.85]])
E = np.array([[0.75, 0.25], [0.55, 0.45]])
init = np.array([1, 0])

viterbi(y, A, E, init)
```

Problem 4

In this question you will compute certain derivatives used in training logistic regression. The notation is as used in Lecture 26.

1. Show that $\frac{d}{dz}\sigma(z) = \sigma(z)\,(1-\sigma(z))$
2. Show that $\nabla_{\boldsymbol{\theta}}\ell_i(\boldsymbol{\theta}, b) = -\left(y^{(i)} - \sigma\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + b\right)\right)\mathbf{x}^{(i)}$.
3. Show that $\frac{\partial}{\partial b}\ell_i(\boldsymbol{\theta}, b) = -y^{(i)} + \sigma\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + b\right)$

Solution:

1. $\frac{d}{dz}\sigma(z) = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{d}{dz}\left(1+e^{-z}\right)^{-1} = -(1+e^{-z})^{-2}(-e^{-z}) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \times \frac{(1+e^{-z})-1}{1+e^{-z}} = \frac{1}{1+e^{-z}} \times \left(1 - \frac{1}{1+e^{-z}}\right) = \sigma(z)\,(1-\sigma(z))$.

2.

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\ell_i(\boldsymbol{\theta}, b) &= \nabla_{\boldsymbol{\theta}}[-y^{(i)}\log(\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)) - (1-y^{(i)})\log(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))] \\
&= -y^{(i)}\frac{\mathbf{x}^{(i)}[\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))]}{\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)} + (1-y^{(i)})\frac{\mathbf{x}^{(i)}[\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))]}{1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)} \\
&= -y^{(i)}\mathbf{x}^{(i)}(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)) + (1-y^{(i)})\mathbf{x}^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) \\
&= -y^{(i)}\mathbf{x}^{(i)} + y^{(i)}\mathbf{x}^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) + \mathbf{x}^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) - y^{(i)}\mathbf{x}^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) \\
&= -\left(y^{(i)} - \sigma\left(\boldsymbol{\theta}^\top\mathbf{x}^{(i)}+b\right)\right)\mathbf{x}^{(i)}.
\end{aligned}$$

3.

$$\begin{aligned}
\frac{\partial}{\partial b}\ell_i(\boldsymbol{\theta}, b) &= \frac{\partial}{\partial b}\left(-y^{(i)}\log(\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)) - (1-y^{(i)})\log(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))\right) \\
&= -y^{(i)}\frac{\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))}{\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)} + (1-y^{(i)})\frac{\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b))}{1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)} \\
&= -y^{(i)}(1-\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b)) + (1-y^{(i)})\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) \\
&= -y^{(i)} + y^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) + \sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) - y^{(i)}\sigma(\boldsymbol{\theta}^T\mathbf{x}^{(i)}+b) \\
&= -y^{(i)} + \sigma\left(\boldsymbol{\theta}^\top\mathbf{x}^{(i)}+b\right).
\end{aligned}$$

Problem 5

Use the Jupyter notebook provided to implement logistic regression for the Heart Disease data set. You may use the code provided in Lecture 26, and just modify it appropriately. Include code in your answer, as well as the accuracy your model achieves on the test data set.

Solution: The model implemented using the following code achieves 71.25% accuracy on the test set.

```
import numpy as np
import pandas as pd # useful for reading in data
import matplotlib.pyplot as plt

data = pd.read_csv('http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data',
        error_bad_lines=False)

target = 'chd'
features = ['sbp', 'tobacco', 'ldl', 'famhist', 'obesity', 'alcohol', 'age']
data['famhist'] = pd.get_dummies(data['famhist'])['Present']

X, y = data[features].values, data[target].values # 462 total data points

All_indices = range(0,462)
Test_indices = np.random.choice(All_indices,size=80,replace=False)
Train_indices = np.setdiff1d(All_indices,Test_indices)

X_train = X[Train_indices,:]
y_train = y[Train_indices]

X_test = X[Test_indices,:]
y_test = y[Test_indices]

def sigma(z):
    min_output = 0.00001
    max_output = 0.99999
    output = 1.0/(1.0+np.exp(-z))
    output = max(min_output, output)
    output = min(max_output, output)
    return output

def h(theta,b,x):
    return sigma(np.dot(theta,x) + b)

def TrainingLogisticRegression(theta_0,b_0,X,y,alpha,K_max):
    # Function for training logistic regression model
    # Inputs:
        # theta_0,b_0 : (random) initializations for parameters
        # X,y : labeled training data
        # alpha: step size/ learning rate
        # K_max: max number of iterations.
    theta = np.squeeze(theta_0)
    b = b_0
    N = X.shape[0]
    d = X.shape[1]
    loss_function_trajectory = np.zeros([K_max,1])
    for k in range(K_max):
        theta_grad = 0
        b_grad = 0
        loss_function_value = 0
        for i in range(N):
            theta_grad -= (y[i] - h(theta,b,X[i,:]))*X[i,:]
            b_grad -= y[i] - h(theta,b,X[i,:])
            loss_function_value += -y[i]*np.log(h(theta,b,X[i,:]))
                                    -(1-y[i])*np.log(1 - h(theta,b,X[i,:]))
        theta -= alpha*theta_grad
        b -= alpha*b_grad
        loss_function_trajectory[k] = loss_function_value
        if k% 50 == 0:
            print(loss_function_value)
    return theta,b,loss_function_trajectory

# Initializing parameters
theta_0 = np.random.randn(7,1)
b_0 = np.random.randn(1)
alpha = 0.01
K_max = 500

# training
theta,b,loss_function_trajectory = TrainingLogisticRegression(theta_0,b_0,
                                    X_train,y_train,alpha,K_max)

def Classifier(theta,b,x):
    probability = h(theta,b,x) # = sigmoid(theta^{T}x - b)
    # if probability >= 0.5 then predict y = 1. if prob < 0.5 then predict y = 0
    classification = np.round(probability)
    return probability,classification

count = 0

for i in range(80):
    prob, y_pred = Classifier(theta, b, X_test[i,:])
    if y_pred == y_test[i]:
        count += 1

print(count/80)
```