Introduction to Machine Learning (CSCI-UA.473): Homework 1

Alvin Lee

06 September 2022

Solutions

Probability and Calculus

Question 1

We begin this question by considering all the possible outcomes where Player 1 wins. Player 1 wins if and only if Player 1 is the first to succeed. Lets denote a success for Player 1 as P1 and a miss as P1', and the same for Player 2. The winning combinations for Player 1, assuming Player 1 shoots first are:

$$P1, P1'P2'P1, P1'P2'P1'P2'P1, P1'P2'P1'P2'P1'P2'P1, \ldots$$

And so the probability that Player 1 wins is the sum of the probabilities of the winning combinations, which is as follows:

$$\mathbb{P}(Win) = \mathbb{P}(P1) + \mathbb{P}(P1')\mathbb{P}(P2')\mathbb{P}(P1) + \mathbb{P}(P1')\mathbb{P}(P2')\mathbb{P}(P1')\mathbb{P}(P2')\mathbb{P}(P1) + \ldots \tag{1}$$

We know that $\mathbb{P}(P1) = \frac{1}{5}$, $\mathbb{P}(P2) = \frac{1}{4}$ and since a player either succeeds or misses, this implies that $\mathbb{P}(P1') = 1 - \mathbb{P}(P1) = \frac{4}{5}$, and for the same reason $\mathbb{P}(P2') = \frac{3}{4}$ We can see from equation (1) that this becomes the summation:

$$\mathbb{P}(Win) = \frac{1}{5} + \sum_{n=0}^{\infty} \left(\left(\frac{3}{4} \right)^n \left(\frac{4}{5} \right)^n \left(\frac{1}{5} \right) \right) \tag{2}$$

We can simplify this summation with the following steps:

$$\mathbb{P}(Win) = \frac{1}{5} + \sum_{n=0}^{\infty} \left(\left(\frac{3}{5} \right)^n \left(\frac{1}{5} \right) \right)$$

$$\mathbb{P}(Win) = \frac{1}{5} + \left(\frac{1}{5} \right) \sum_{n=0}^{\infty} \left(\frac{3}{5} \right)^n$$

By using the formula

$$\sum_{a=0}^{\infty} a^i = \frac{1}{1-a} \ , \ |a| < 1 \tag{3}$$

We finish with

$$P(Win) = \frac{1}{5} + \frac{1}{5} \cdot \frac{1}{1-\frac{3}{5}} = \frac{1}{5} + \frac{1}{5} \cdot \frac{5}{2} = \frac{1}{5} + \frac{1}{2} = \frac{7}{10}$$

So the probability that Player 1 wins given Player 1 goes first is

$$\mathbb{P}(Win) = \frac{7}{10} \tag{4}$$

Question 2

The initial thought when starting this problem is to use Bayes Theorem which states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{5}$$

In the case of this question, we can state $P(A)$ as the probability that you have COVID, and $P(B)$ as the probability that you test positive. We are given that $P(A) = 0.01$, $P(B|A) = 0.9$, and $P(B|A') = 0.1$ where A' is the situation that you do **not** have COVID. Before using Bayes Theorem, we must find $P(B)$. We can use the following formula:
given $P(Y) + P(Y') = 1$,

$$P(X) = P(X|Y)P(Y) + P(X|Y')P(Y') \tag{6}$$

Which we derive from the following steps:

$$P(X) = P(X \cap Y) + P(X \cap Y')$$

$$P(X) = \frac{P(X \cap Y)P(Y)}{P(Y)} + \frac{P(X \cap Y')P(Y')}{P(Y')}$$

$$P(X) = P(X|Y)P(Y) + P(X|Y')P(Y')$$

Using this equation:

$$P(B) = P(B|A)P(A) + P(B|A')P(A')$$

$$P(B) = 0.9 \cdot 0.01 + 0.1 \cdot 0.99$$

$$P(B) = 0.009 + 0.099 = 0.108$$

Finally, we now use Bayes Theorem to find the solution:

$$P(A|B) = \frac{0.9 \cdot 0.01}{0.108} \approx 0.0833$$

The probability you have COVID given you tested positive is 8.33%

Question 3

We will consider if the following function is a Probability Density Function (PDF)

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ \frac{1}{(1+x)} & otherwise \end{cases} \tag{7}$$

A Probability Density Funciton must satisfy two conditions

(i) $f(x)$ must be nonnegative for each value of the random variable

(ii) The integral over all values of the random variable must equal 1

We know that $f(x)$ is nonnegative for each value of the random variable. Now we solve the integral over all values of the random variable. We know $f(x) = 0,\ \forall\ x < 0$, so to find the integral we solve

$$\int_0^\infty \frac{1}{(1+x)} dx \tag{8}$$

Which we can do by the following

$$\int_0^\infty \frac{1}{(1+x)} dx = ln(1+x) \Big|_0^\infty = \infty$$

We can conclude that the integral over all values of the random variable is greater than 1, so (ii) is not satisfied, meaning $f(x)$ is not a PDF

Question 4

Given two independent variables X and Y with the same density function:

$$f(x) = \begin{cases} 2x & if\ 0 \leq x \leq 1 \\ 0 & otherwise \end{cases} \tag{9}$$

Since X and Y are independent random variables, the joint probability density function of X and Y is:

$$f_{x,y}(x,y) = f_x(x) f_y(y) \tag{10}$$

We calculate $P(X + Y \leq 1)$

$$\begin{aligned} & P(X+Y \leq 1) \\ &= \int\int_{\{x+y \leq 1\}} f(x)f(y)dxdy \\ &= \int\int_{\{x \geq 0, y \geq 0, x+y \leq 1\}} f(x)f(y)dxdy \\ &= \int_0^1 \int_0^{1-y} f(x)f(y)dxdy \end{aligned}$$

$$= \int_0^1 2y(x^2 \Big|_0^{1-y})dy$$
$$= 2\int_0^1 y(1-2y+y^2)dy$$
$$= \frac{y^4}{4} - \frac{2y^3}{3} + \frac{y^2}{2} \Big|_0^1$$
$$= 2(\frac{1}{4} - \frac{2}{3} + \frac{1}{2})$$
$$= \frac{1}{6}$$

Question 5

We compute expected value by using the following equation:

$$\mathbb{E}(Y) = \int_{-\infty}^{\infty} yf(y)dy \tag{11}$$

We know $Y = g(X) = e^X$ and $X \sim Unif(0,1)$ which implies $f_X(x) = 1$. We can find

$$\mathbb{E}(Y) = \mathbb{E}(e^X) = \int_0^1 e^x f(x)dx = \int_0^1 e^x dx = e^x\Big|_0^1 = e - 1$$

Question 6

We know $X_i \sim Poisson(\lambda)$. The Central Limit Theorem states, for $X_1, \ldots, X_n$ i.i.d with mean μ and variance σ^2 Let $\overline{X_n} = n^{-1}\sum_{i=1}^n$. Then

$$Z_n \equiv \frac{\sqrt{n}(\overline{X_n} - \mu)}{\sigma} \xrightarrow{d} Z \tag{12}$$

This states that with large enough n, the average of random variables has a distribution that is approximately normal.
We know $\mu = meanofX_i = \lambda = 5$ and also for a poisson distribution $\lambda = var(X_i) = \sigma^2 = 5$ Using the Central Limit Theorem, we know:

$$Z_n = \frac{\sqrt{n}(\overline{X_n} - \mu)}{\sigma} = \frac{\sqrt{125}(\overline{X_n} - 5)}{\sqrt{5}} = 5(\overline{X_n} - 5) \sim Z \tag{13}$$

We can compute

$$P(\overline{X}_n \leq 5.5) = P(5(\overline{X}_n - 5) \leq 5(5.5 - 5)) \approx P(Z \leq 2.5) \tag{14}$$

We can use a z-score table to find that

$$P(Z \leq 2.5) = 0.9938$$

Question 7

Let $X_n = f(W_n, X_{n-1})$ for $n = 1, \ldots, P$, for some function $f()$.
we can show the following:

$$\frac{\partial X_n}{\partial X_0} = \partial_2 f(W_n, X_{n-1}) \frac{\partial X_{n-1}}{\partial X_0},\ n = 1, \ldots, P \tag{15}$$

By the chain rule, we can find for

$$E = \|C - X_p\|^2 \tag{16}$$

$$\frac{\partial E}{\partial X_0} = -2\|C - X_P\| \cdot \frac{\partial X_P}{\partial X_0} \tag{17}$$

$$= -2\|C - X_P\| \cdot \partial_2 f(W_P, X_{P-1}) \cdot \ldots \cdot \partial_2 f(W_1, X_0) \frac{\partial X_0}{\partial 0}$$
$$= -2\|C - X_P\| \cdot \partial_2 f(W_P, X_{P-1}) \cdot \ldots \cdot \partial_2 f(W_1, X_0) \cdot 1$$

Linear Algebra

Question 8

Let A be the matrix $\begin{bmatrix} 2 & 6 & 7 \\ 3 & 1 & 2 \\ 5 & 3 & 4 \end{bmatrix}$ and let x be the column vector $\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$. We can write A^T as $\begin{bmatrix} 2 & 3 & 5 \\ 6 & 1 & 3 \\ 7 & 2 & 4 \end{bmatrix}$ and x^T as $\begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$ We can compute the matricies by using the following formula. For $B = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$ where r is a row of B and $C = \begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix}$ where c is a column of C

$$BC = \begin{bmatrix} r_1 c_1 & r_1 c_2 & r_1 c_3 \\ r_2 c_1 & r_2 c_2 & r_2 c_3 \\ r_3 c_1 & r_3 c_2 & r_3 c_3 \end{bmatrix} \tag{18}$$

Using this equation we can find

$$Ax = \begin{bmatrix} 2*2 + 6*3 + 7*4 \\ 3*2 + 1*3 + 2*4 \\ 5*2 + 3*3 + 4*4 \end{bmatrix} = \begin{bmatrix} 50 \\ 17 \\ 35 \end{bmatrix} \tag{19}$$

$$A^T = \begin{bmatrix} 2 & 3 & 5 \\ 6 & 1 & 3 \\ 7 & 2 & 4 \end{bmatrix} \tag{20}$$

$$x^T A = \tag{21}$$

$$\begin{bmatrix} (2*2 + 3*3 + 4*5) & (2*6 + 3*1 + 4*3) & (2*7 + 3*2 + 4*4) \end{bmatrix}$$
$$= \begin{bmatrix} 33 & 27 & 36 \end{bmatrix}$$

Question 9

We know the following statement is true: If A is a square matrix and $det(A) \neq 0$ then A is invertible.

(a) Let $A = \begin{bmatrix} 6 & 2 & 3 \\ 3 & 1 & 1 \\ 10 & 3 & 4 \end{bmatrix}$

$$det(A) = 6(det(\begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix})) - 2(det(\begin{bmatrix} 3 & 1 \\ 10 & 4 \end{bmatrix})) + 3(det(\begin{bmatrix} 3 & 1 \\ 10 & 3 \end{bmatrix})) \tag{22}$$

$$= 6(1) - 2(2) + 3(-1) = -1 \neq 0$$

So A is invertible. Now we find the inverse of A. We know by definition of inverse matricies, that $AA^{-1} = I$. We can therefore find A^{-1} by finding elementary matricies $E_1, E_2, \ldots$ such that $(E_1 E_2 \ldots)A = I$ and let $(E_1 E_2 \ldots) = A^{-1}$. After doing these steps, we find that:

$$A^{-1} = \begin{bmatrix} -1 & -1 & 1 \\ 2 & 6 & -3 \\ 1 & -2 & 0 \end{bmatrix}$$

(b) Let $B = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 2 \\ 1 & 4 & 5 \end{bmatrix}$

$$det(B) = 0 + 2(det(\begin{bmatrix} 1 & 3 \\ 1 & 5 \end{bmatrix})) - 2(det(\begin{bmatrix} 1 & 2 \\ 1 & 4 \end{bmatrix})) \tag{23}$$

$$= 2(2) - 2(2) = 0$$

So B is not invertible.

Question 10

An eigenvalue, often denoted as $\lambda \in \mathbb{R}$ is the scalar factor such that when a linear transformation is applied to a vector, the result differs from the vector by a factor of λ. The eigenvector is the corresponding vector that changes by the eigenvalue when the linear transformation is applied. In other words, the eigenvalue is such that, for a matrix A and a vector v

$$Av = \lambda v \tag{24}$$

To find the eigenvector and eigenvalues, we can solve:

$$Av - \lambda v = 0$$

$$(A - \lambda I)v = 0 \tag{25}$$

for non-zero vector v, this will only have a solution if

$$\det(A - \lambda I) = 0$$

We can use the above to find the eigenvalues. Once we have the eigenvalues we can just them plug into the equation (25) and find the associated eigenvectors. Using the above method to compute the eigen values, we can find:

$$\det\left(\begin{bmatrix} 1-\lambda & 0 & -1 \\ 1 & 0-\lambda & 0 \\ -2 & 2 & 1-\lambda \end{bmatrix}\right) = 0 \tag{26}$$

$$(1-\lambda)\det\left(\begin{bmatrix} -\lambda & 0 \\ 2 & 1-\lambda \end{bmatrix}\right) - 1(\det\left(\begin{bmatrix} 1 & -\lambda \\ -2 & 2 \end{bmatrix}\right)) = 0 \tag{27}$$

$$-\lambda^3 + 2\lambda^2 + \lambda - 2 = 0 \tag{28}$$

$$(-\lambda^2 + 1)(\lambda - 2) = 0 \tag{29}$$

$$\lambda = 1, -1, 2 \tag{30}$$

al6178_hw1

September 20, 2022

1 Homework 1: Math Foundations for ML

The is the coding potion of Homework 1. The homework is aimed at testing the ability to code up mathematical operations using Python and the `numpy` library.

For each problem, we provide hints or example test cases to check your answers (see the `assert` statements below). Your full submission will be autograded on a larger batch of randomly generated test cases.

1.1 Note on the autograding process

For this assignment, we are using `nbgrader` for autograding. We recommend that you use JupyterLab or Jupyter notebook to complete this assignment for compatibility.

The cells containing example test cases also serve as placeholders for the autograder to know where to inject additional random tests. Notice that they are always after your solution; moving/deleting them will cause the tests to fail, and we'd have to manually regrade it. They are marked with `DO NOT MOVE/DELETE` and set to read-only just in case.

The autograder tests will call the functions named `solve_system`, `split_into_train_and_test`, `closest_interval`. You may not change the function signature (function name and argument list), but otherwise feel free to add helper functions in your solution. You can also make a copy of the notebook and use that as a scratchpad.

To double check your submission format, restart your kernel (Menu bar -> Kernel -> Restart Kernel); execute all cells from top to bottom, and see if you can pass the example test cases.

[2]:
```
import numpy as np
from numpy.linalg import inv
```

1.2 Part 1: Systems of linear equations

Given n equations with n unknown variables ($n \leq 4$), write a function `solve_system` that can solve this system of equations and produce an output of value for each variable such that the system of equations is satisfied.

The system of equations will be provided as a list of strings as seen in `test_eq`.

You may assume that the variables are always in $\{a, b, c, d\}$, the system has a unique solution, and all coefficients are integers.

[3]:
```
def solve_system(equations):
    """
    Takes in a list of strings for each equation.
    Returns a numpy array with a row for each equation value
    """
    n = len(equations)
    variables = np.array(["a","b","c","d"])
    matrix = np.zeros((n,n))
    v = np.zeros((n,1))
    for row in range(0,len(equations)):
        equation_split = equations[row].split()
        sign = 1
        for j in range(0,len(equation_split)):
            #change sign for coefficient, default pos (for the first entry␣
 ↪without sign)
            if equation_split[j] == "+":
                sign = 1
            if equation_split[j] == "-":
                sign = -1
            # if the equation has the variable we can check the coef and add it␣
 ↪to our matrix
            if equation_split[j] in variables:
                i = np.where(variables == equation_split[j])[0]
                # if any of the variables have no coefs default to 1 or -1␣
 ↪depending on sign
                try:
                    int(equation_split[j-1])
                    is_dig = True
                except ValueError:
                    is_dig = False
                if j == 0 or is_dig == False:
                    matrix[row,i] = sign
                # anything else set the matrix as the coef
                else:
                    matrix[row,i] = int(equation_split[j-1]) * sign
            # if theres an equal sign we add it to vector v (solution vector)
            if equation_split[j] == "=":
                v[row,0] = int(equation_split[j+1])
    #to solve we just multiply (A^-1)v
    return np.linalg.solve(matrix,v)

    raise NotImplementedError()
```

[4]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

def test_eq(sys_eq):
```

```
    results = solve_system(sys_eq)
    expected = np.array([
    [14.75521822],
    [-7.33776091],
])
    assert np.allclose(expected, results)

test_eq([
    '30 a + 73 b = -93',
    '-29 a - 53 b = -39',
])
```

1.3 Part 2: Split a dataset into test and train

(For this question, using an existing implementation (e.g. `sklearn.model_selection.train_test_split`) will give 0 points.)

In supervised learning, the dataset is usually split into a train set (on which the model is trained) and a test set (to evaluate the trained model). This part of the homework requires writing a function `split_into_train_and_test` that takes a dataset and the train-test split ratio as input and provides the data split as an output. The function takes a `random_state` variable as input which when kept the same outputs the same split for multiple runs of the function.

Note: if `frac_test` does not result in an integer test set size, round down to the nearest integer.

Hints: - The input array `x_all_LF` should not be altered after the function call. - Running the function with the same seed multiple times should yield the same results. - Every element in the input array should appear either in the train or test set, but not in both.

[33]:
```
def split_into_train_and_test(x_all_LF, frac_test=0.5, seed=None):
    ''' Divide provided array into train and test sets along first dimension
    User can provide random number generator object to ensure reproducibility.
    Args
    ----
    x_all_LF : 2D np.array, shape = (n_total_examples, n_features) (L, F)
        Each row is a feature vector
    frac_test : float, fraction between 0 and 1
        Indicates fraction of all L examples to allocate to the "test" set
        Returned test set will round UP if frac_test * L is not an integer.
        e.g. if L = 10 and frac_test = 0.31, then test set has N=4 examples
    seed : integer or None
        If int, will create RandomState instance with provided value as seed
        If None, defaults to current numpy random number generator np.random.
    Returns
    -------
    x_train_MF : 2D np.array, shape = (n_train_examples, n_features) (M, F)
        Each row is a feature vector
        Should be a separately allocated array, NOT a view of any input array
    x_test_NF : 2D np.array, shape = (n_test_examples, n_features) (N, F)
```

```
    Each row is a feature vector
    Should be a separately allocated array, NOT a view of any input array
Post Condition
--------------
This function should be side-effect free. Provided input array x_all_LF
should not change at all (not be shuffled, etc.)
Examples
--------
>>> x_LF = np.eye(10)
>>> xcopy_LF = x_LF.copy() # preserve what input was before the call
>>> train_MF, test_NF = split_into_train_and_test(
...     x_LF, frac_test=0.201, random_state=np.random.RandomState(0))
>>> train_MF.shape
(7, 10)
>>> test_NF.shape
(3, 10)
>>> print(train_MF)
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
>>> print(test_NF)
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
## Verify that input array did not change due to function call
>>> np.allclose(x_LF, xcopy_LF)
True
References
----------
For more about RandomState, see:
https://stackoverflow.com/questions/28064634/
↪random-state-pseudo-random-numberin-scikit-learn
'''
if seed is None:
    rng = np.random.RandomState()
else:
    rng = np.random.RandomState(seed)

num_rows, num_cols = x_all_LF.shape
num_split = int(num_rows*frac_test)
randomInts = rng.choice(num_rows,num_rows,replace=False)
i = 0
x_train_MF = []
```

```
    x_test_NF = []

    for row in x_all_LF:
        if randomInts[i] >= num_split:
            x_train_MF.append(row)
            i+=1
        else:
            x_test_NF.append(row)
            i+=1
    x_train_MF = np.array(x_train_MF)
    x_test_NF = np.array(x_test_NF)
    return x_train_MF, x_test_NF
    # YOUR CODE HERE
    raise NotImplementedError()
```

[34]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

N = 10
x_LF = np.eye(N)
xcopy_LF = x_LF.copy() # preserve what input was before the call
train_MF, test_NF = split_into_train_and_test(x_LF, frac_test=0.2, seed=0)
```

1.3.1 Part 3: Solving a Search Problem

Given a list of N intervals, for each interval $[a, b]$, we want to find the closest non-overlapping interval $[c, d]$ greater than $[a, b]$.

An interval $[c, d]$ is greater than an non-overlapping interval $[a, b]$ if $a < b < c < d$.

The function `closest_interval` takes in the list of intervals, and returns a list of indices corresponding to the index of the closest non-overlapping interval for each interval in the list. If a particular interval does not have a closest non-overlapping interval in the given list, return `-1` corresponding to that element in the list.

[7]:
```
def closest_interval(intervals):
    x = []
    y = []
    closest = []
    for arr in intervals:
        x.append(arr[0])
        y.append(arr[1])
    x = np.array(x)
    y = np.array(y)
    for i in range(0,len(intervals)):
        if len(np.where(x == min((j for j in x if (j > y[i])),␣
↪default=None))[0])==0:
            a = -1
```

```
        else:
            a = np.where(x == min((j for j in x if (j > y[i])),␣
 ↪default=None))[0][0]
        closest.append(a)
    return closest
    # YOUR CODE HERE
    raise NotImplementedError()
```

[8]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

intervals = np.array([[3, 6], [1, 6], [4, 6], [3, 7], [6, 9], [2, 6], [5, 8],␣
 ↪[0, 3]])

expected_closest_intervals = closest_interval(intervals)

# Evaluate
results = np.array([-1, -1, -1, -1, -1, -1, -1, 2])
assert np.allclose(expected_closest_intervals, results)
```