**LAHORE UNIVERSITY OF MANAGEMENT SCIENCES**
**Syed Babar Ali School of Science and Engineering**

**EE212 Mathematical Foundations for Machine Learning and Data Science**
**Summer Semester 2020**

**Laboratory 4 – Bayesian & Statistical Inferences**

Issued: Friday 7 August, 2020

**Total Marks:** 100

**Contribution to Final Assessment:** 2%

**Submission:** 12:15 pm, Friday 7 August, 2020.

# Goal

Machine Learning and Statistics are two very closely related fields. Often, the line between the two fields is blurred since there are many statistical techniques that are invaluable when working on machine learning applications. The goal of this laboratory is to develop an understanding of these statistical techniques to make key inferences in machine learning projects.

# Instructions

If you have any concerns, you can ask us in the live zoom session, or in the chat. Each of you has been allotted TA/RA, so when you are done with the lab, let them know and they will mark it. It is your responsibility to ensure you get your work checked.

Name your files Task1.py, Task2.py and so on. Compress them in a **single** file and name it as LabXX_YourRollNumber. Submit this file on LMS before the deadline. No late submissions will be accepted.

Before starting, import the following libraries from python:

```
import random
import numpy as np
import pandas as pd
from scipy import stats
from matplotlib import pyplot as plt
from numpy.random import seed
from numpy.random import rand
from numpy.random import randn
from scipy.stats import pearsonr
from scipy.stats import ttest_ind
from scipy.stats import mannwhitneyu
```

# Task 1: Bayesian Inferences 1.0 (20)

Bayesian statistics is a particular approach to apply probability related concepts to statistical problems. It provides us with the mathematical tools needed to update our beliefs about random events in light of seeing new data or evidence about those events. In particular, Bayesian inference interprets probability as a measure of confidence that an individual may possess about the occurrence of a particular event. We may have a prior belief about an event but our beliefs are likely to change when new evidence is brought to light. Bayesian statistics gives us solid mathematical means of incorporating our prior beliefs and evidence to produce new posterior beliefs. The mathematical definition of the Bayes theorem is as follows:

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

where probability of event $A$ occurring given that event $B$ has occurred is equal to the probability that they have both occurred, relative to the probability that $B$ has occurred.

1. Consider event $A$ as rainfall and event $B$ as observing clouds. Then, the probability of rain given that we have seen clouds is equal to the probability of rain and clouds occurring together, relative to the probability of seeing clouds at all. Write an expression for the probability of observing clouds given that it is raining i.e. $P(B|A)$
2. Using your answer in 1, derive an expression of $P(A|B)$ in terms of $P(B|A)$

# Task 2: Bayesian Inferences 2.0 (40)

Bayesian statistics provides us with mathematical tools to rationally update our subjective beliefs in light of new data or evidence. For instance, prior to any flips of the coin, an individual may believe that the coin is fair. After a few flip,s the coin continually comes up heads. Thus the prior belief about fairness of the coin is modified to account for the fact that three heads have come up in a row and thus the coin might not be fair. After 500 flips, with 400 heads, the individual believes that the coin is very unlikely to be fair. The posterior belief has been heavily modified and is different from the prior belief of a fair coin. In the example below, consider multiple coin-flips of a coin with unknown fairness. We will use Bayesian inference to update our beliefs on the fairness of the coin as more data (i.e. more coin flips) becomes available. Consider the following code snippet:

```
number_of_trials = [0,2]
data = stats.bernoulli.rvs(0.5, size=number_of_trials[-1])
x = np.linspace(0, 1, 100)
for i, N in enumerate(number_of_trials):
    heads = data[:N].sum()
    ax = plt.subplot(len(number_of_trials) / 2, 2, i + 1)
    ax.set_title("%s trials, %s heads" % (N, heads))
    plt.xlabel("$P(H)$, Probability of Heads")
    plt.ylabel("Density")
    if i == 0:
        plt.ylim([0.0, 2.0])
    plt.setp(ax.get_yticklabels(), visible=False)
```

```
        y = stats.beta.pdf(x, 1 + heads, 1 + N - heads)
        plt.plot(x, y, label="observe %d tosses,\n %d heads" % (N, heads))
        plt.fill_between(x, 0, y, color="#aaaadd", alpha=0.5)
    plt.tight_layout()
    plt.show()
```

The code above allows you to visualize the probability density function for 0 and 2 trials (coin flips). The uniform distribution in the case of 0 coin flips reflects the probability of heads is equally likely to take any value between 0 and 1.

1. Interpret the subplot with 2 trials.
2. Modify the *number_of_trials* array in the code above to append 5,10,20,50,100,5000 trials and interpret the plots.
3. Looking at the results from 10 trials, can you conclude if the coin is a fair coin?
4. What is your inference on the fairness of the coin as you increased the number of trials?

# Task 3: Monty Hall Problem (20)

The Monty Hall problem was first featured on the classic game show "Let's make a Deal". In the final segment of the show, contestants were presented with a choice of three different doors. Behind two of the doors would be a goat, and behind the third would be an extravagant prize such as a car. The contestant begins the game by picking one door. The host, Monty Hall, would then open one of the remaining doors. Monty never opens the door which contained the car. The door Monty opens always contains a goat. At this point, the contestant is asked to make a decision: open the originally selected door or switch to the other unopened door.

1. Using the following definitions, calculate if sticking to the original decision has a higher probability of winning or switching to the remaining door has a higher probability of winning:

   $P(C)$: Probability that car is behind door 1 without knowing what door Monty reveals

   $P(C') : 1 - P(C)$

   $P(G|C)$ : Probability that Monty shows a door with a goat behind it given that there is a car behind door 1

   $P(G|C')$ : Probability that Monty shows a door with a goat behind it given that the car is not behind door 1

2. The code snippet below allows you to simulate the game for many trials. Modify the following code to increase the number of trials to 1000. Verify your answer in part 1.

```
def run_trial(switch_doors, ndoors=3):
    chosen_door = random.randint(1, ndoors)
    if switch_doors:
        revealed_door = 3 if chosen_door==2 else 2
        available_doors = [dnum for dnum in range(1,ndoors+1)
```

```
                                    if dnum not in (chosen_door, revealed_door)]
            chosen_door = random.choice(available_doors)
        return chosen_door == 1

    def run_trials(ntrials, switch_doors, ndoors=3):
        nwins = 0
        for i in range(ntrials):
            if run_trial(switch_doors, ndoors):
                nwins += 1
        return nwins
    ndoors, ntrials = 3, 10
    nwins_without_switch = run_trials(ntrials, False, ndoors)
    nwins_with_switch = run_trials(ntrials, True, ndoors)

    print('Monty Hall Problem with {} doors'.format(ndoors))
    print('Proportion of wins without switching: {:.4f}'
                .format(nwins_without_switch/ntrials))
    print('Proportion of wins with switching: {:.4f}'
                .format(nwins_with_switch/ntrials))
```

# Task 4: Statistical Inferences (20)

In this task we will develop an understanding of inferential statistics. Often, variables in a data-set may be related to one another. It is considered useful in data analysis and modeling to better understand the relationships between variables. The statistical relationship between two variables is referred to as their correlation. A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variable's value decreases. We can quantify the relationship between samples of two variables using a statistical method called Pearson's correlation coefficient, named after the developer of the method, Karl Pearson. The *pearsonr*() function can be used to calculate the Pearson's correlation coefficient for samples of two variables, as shown in the code below:

```
seed(1)
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
corr, p = pearsonr(data1, data2)
print('Pearsons correlation: %.3f' % corr)
```

As the number of variables increase in the dataset, finding correlation between those variables becomes difficult. However, Python makes this process fairly simple with the *corr*() function. The provided dataset for the following code snippet contains 19 columns of features and 1000 rows of observations. Use the code below to visualize the correlation between them:

```
data = pd.read_csv('data_visualization.csv', index_col=0)
corr = data.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
```

```
fig.colorbar(cax)
ticks = np.arange(0,len(data.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(data.columns)
ax.set_yticklabels(data.columns)
plt.show()
```

1. Run the code above and identify which variables are strongly correlated in the dataset.
2. Generate a new dataset of your own, following the same syntax as in the provided csv file. Include at least 10 variables and 20 observations in it. Modify the above code to visualize correlations between the variables in your own dataset.

Data must be interpreted in order to add meaning. We can interpret data by assuming a specific structure to our outcome and use statistical methods to confirm or reject the assumption. The assumption is called a hypothesis and the statistical tests used for this purpose are called statistical hypothesis tests. The assumption of a statistical test is called the null hypothesis, or hypothesis zero (H0). It is often called the default assumption, or the assumption that nothing has changed. A violation of the test's assumption is often called the first hypothesis (H1).

Hypothesis 0 (H0): Assumption of the test holds and is failed to be rejected.
Hypothesis 1 (H1): Assumption of the test does not hold and is rejected at some level of significance.

We can interpret the result of a statistical hypothesis test using a p-value. The p-value is the probability of observing the data, given the null hypothesis is true. A large probability means that the H0 or default assumption is likely. A small value, such as below 5% (0.05) suggests that it is not likely and that we can reject H0 in favor of H1, or that something is likely to be different (e.g. a significant result).

A widely used statistical hypothesis test is the Student's t-test for comparing the mean values from two independent samples. The default assumption is that there is no difference between samples, whereas a rejection of this assumption suggests some significant difference. The tests assumes that both samples were drawn from a Gaussian distribution and have the same variance. The Student's t-test can be implemented in Python via the *ttest_ind*() function.

Below is an example of calculating and interpreting the Student's t-test for two data samples that are known to be different.

```
seed(1)
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 51
stat, p = ttest_ind(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
```

```
    print('Different distributions (reject H0)')
```

A large portion of the field of statistics and statistical methods is dedicated to data where the distribution is known. Data in which the distribution is unknown or cannot be easily identified is called non-parametric. In the case where you are working with non-parametric data, specialized non-parametric statistical methods can be used that discard all information about the distribution. As such, these methods are often referred to as distribution-free methods. A widely used non-parametric statistical hypothesis test for checking for a difference between two independent samples is the Mann-Whitney U test, named after Henry Mann and Donald Whitney. It is the non-parametric equivalent of the Student's t-test but does not assume that the data is drawn from a Gaussian distribution. The test can be implemented in Python via the *mannwhitneyu*() function as shown below:

```
seed(1)
data1 = 50 + (rand(100) * 10)
data2 = 51 + (rand(100) * 10)
stat, p = mannwhitneyu(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Same distribution (fail to reject H0)')
else:
    print('Different distribution (reject H0)')
```