Section 2. Logistic Regression Spam Classification

> Import Necessary Libraries

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split as tts
from tqdm import tqdm
from IPython.display import Markdown as md
```

> Reading in the Data
>
> > Randomizing the Data
> >
> > Splitting the data into X and Y vectors

In [ ]:
```
# Read in the data
data = np.genfromtxt('spambase.data', delimiter=',')
dataMat = np.array(data)
# Set RNG with seed = 0
np.random.seed(0)
np.random.shuffle(dataMat)
# Splitting the data into X and Y vectors
X = dataMat[:, :-1]
Y =  np.reshape(dataMat[:, -1], (-1, 1))
```

> Train-Test Split on the data

In [ ]:
```
# Split the training and testing sets in a 2:1 ratio
trainX, testX, trainY, testY = tts(X, Y, test_size=0.33)
```

> Standardizing the Data using the training data
>
> > Take the mean and the standard deviation

In [ ]:
```
mean = trainX.mean(axis=0)
std = trainX.std(axis=0, ddof=1)
####################################################################
trainX_std = (trainX - mean) / std
bias = np.ones((trainX_std.shape[0], 1))
TRAIN_X = np.append(bias, trainX_std , axis=1)

####################################################################
testX_std = (testX - mean) / std
bias = np.ones((testX_std.shape[0], 1))
TEST_X = np.append(bias, testX_std , axis=1)
```

> Perform Batch Gradient Descent Using the Sigmoid Function

In [ ]:
```
HYPERPARAMETERS = {
  "eta" : 0.01,
  "term" : 2 ** (-23),
  "EPSILON" : 10**(-7),
  "n_iterations" : 1500,
}
def sigmoid(x, thetas):
      return 1 / (1 + np.exp(-x @ thetas))
def dLdtheta(x, y, g):
      return x.T @ (g - y)
def L(x, y, g):
      return -1 / TRAIN_X.shape[0] * y.T @ np.log2(g + HYPERPARAMETERS["EPSILON"]) + \
        (1 - y.T) @ np.log2(1-g + HYPERPARAMETERS["EPSILON"])

thetas = np.random.uniform(-1, 1, (TRAIN_X.shape[1], 1))
prev_cost = 0
for i in tqdm(range(HYPERPARAMETERS["n_iterations"]), ascii=True, desc="Training Logistic Regression Spam Classification"):
  g = sigmoid(TRAIN_X, thetas)
  cost = L(TRAIN_X, trainY, g)
  gradient = dLdtheta(TRAIN_X, trainY, g)

  # update thetas by batch gradient descent
  thetas -= HYPERPARAMETERS["eta"] * gradient
  if np.abs(prev_cost - cost) < HYPERPARAMETERS["term"]:
    i = HYPERPARAMETERS["n_iterations"]
  prev_cost = cost



res = "$$y ="
for idx, theta in enumerate(thetas):
  # print(f'theta_{idx}: {theta[0]:0.4f}')
  if idx != 0:
    res += f' {theta[0]:=+0.4f}x_' + '{' + str(idx) + '}'
    if idx % 10 == 0:
          res += '\\\\'
  else:
      res += f'{theta[0]:= 0.4f}\\\\'
res += "$$"

md(res)
```

```
Training Logistic Regression Spam Classification: 100%|##########| 1500/1500 [00:02<00:00, 528.55it/s]
```

Out[ ]:

$$y = -9.2500 \\ -0.1349x_{1} -0.1705x_{2} +0.2577x_{3} +5.8790x_{4} +1.0009x_{5} +0.4762x_{6} +1.7518x_{7} +0.3820x_{8} +0.4454x_{9} +0.1312x_{10}\\ -0.2819x_{11} -0.1539x_{12} -0.0678x_{13} -0.1441x_{14} +0.2585x_{15} +2.3166x_{16} +0.9160x_{17} +0.3563x_{18} +0.7995x_{19} +1.2420x_{20}\\ +1.0888x_{21} +0.1253x_{22} +1.5966x_{23} +0.5895x_{24} -7.0444x_{25} -0.7566x_{26} -28.3051x_{27} +0.2577x_{28} -1.9638x_{29} -0.2072x_{30}\\ -3.1573x_{31} -0.6525x_{32} -0.5522x_{33} +0.4426x_{34} -6.2377x_{35} +0.6847x_{36} -0.4328x_{37} -0.5104x_{38} -0.7728x_{39} +0.5821x_{40}\\ -9.9408x_{41} -2.4063x_{42} -0.4328x_{43} -2.3827x_{44} -2.0925x_{45} -2.3488x_{46} -0.8797x_{47} -2.4694x_{48} -0.4723x_{49} -0.5083x_{50}\\ -0.6144x_{51} +0.8888x_{52} +1.4709x_{53} +2.7268x_{54} -1.1341x_{55} +1.9189x_{56} +0.5019x_{57}$$

> Classification

In [ ]:
```python
spam_threshold = 0.50

yhat = sigmoid(TEST_X , thetas)
predictions = np.where(yhat >= spam_threshold, 1, 0)

TP = FP = TN = FN = 0
for prediction , truth in zip(predictions, testY):
    if prediction == truth:
        if truth == 1:
            TP += 1
        else:
            TN += 1
    else:
        if prediction == 1:
            FP += 1
        else:
            FN += 1
print(TP, FP , TN , FN)
Precision =  TP / (TP + FP)
Recall = TP / (TP + FN)
F_1 = (2 * Precision * Recall) / (Precision + Recall)
Accuracy = (TP + TN) / yhat.shape[0]
```

```
561 123 788 47
```

In [ ]:
```python
md(f"$$Precision = {Precision*100:0.4f}\%," + "\\hspace{5pt}" +\
     f"Recall = {Recall*100:0.4f}\%,"+ "\\hspace{5pt}" +\
         f"F_1  = {F_1*100:0.4f}\%," + "\\hspace{5pt}" +\
             f"Accuracy = {Accuracy*100:0.4f}\%$$")
```

Out[ ]:

$$Precision = 82.0175\%,\hspace{5pt}Recall = 92.2697\%,\hspace{5pt}F_1 = 86.8421\%,\hspace{5pt}Accuracy = 88.8084\%$$

Section 3: Naive Bayes Classifier

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split as tts
from scipy.stats import norm
from IPython.display import Markdown as md
```

In [ ]:
```python
# Read in the data
data = np.genfromtxt('spambase.data', delimiter=',')
dataMat = np.array(data)
# Set RNG with seed = 0
np.random.seed(0)
np.random.shuffle(dataMat)
# Splitting the data into X and Y vectors
X = dataMat[:, :-1]
Y =  np.reshape(dataMat[:, -1], (-1, 1))
```

In [ ]:
```python
# Split the training and testing sets in a 2:1 ratio
trainX, testX, trainY, testY = tts(X, Y, test_size=0.33, random_state=1, shuffle=False)
```

In [ ]:
```python
mean = trainX.mean(axis=0)
std = trainX.std(axis=0, ddof=1)
####################################################################
trainX_std = (trainX - mean) / std
bias = np.ones((trainX_std.shape[0], 1))
TRAIN_X = np.append(bias, trainX_std , axis=1)

####################################################################
testX_std = (testX - mean) / std
bias = np.ones((testX_std.shape[0], 1))
TEST_X = np.append(bias, testX_std , axis=1)
```

In [ ]:
```python
spam_mask = np.asarray(np.where(trainY == 1, True, False)).reshape(-1)
non_spam_mask = np.invert(spam_mask)

spam_train = np.compress(spam_mask, trainX, axis=0)
non_spam_train = np.compress(non_spam_mask, trainX, axis=0)

spam_train_mean = np.mean(spam_train, axis=0)
non_spam_train_mean = np.mean(non_spam_train, axis=0)

spam_train_std = np.std(spam_train, axis=0, ddof=1)
non_spam_train_std = np.std(non_spam_train, axis=0, ddof=1)

spam_prior = spam_mask.shape[0] / trainY.shape[0]
non_spam_prior = non_spam_mask.shape[0] / trainY.shape[0]
```

In [ ]:
```python
TP = FP = TN = FN = 0
# adding the epsilon because there will be divide by zero errors
spam_norm = norm.pdf(testX, spam_train_mean, spam_train_std + np.finfo(float).eps)
non_spam_norm = norm.pdf(testX, non_spam_train_mean, non_spam_train_std + np.finfo(float).eps)


p_spam = np.nan_to_num(np.prod(spam_norm, axis=1) * spam_prior)
p_non_spam = np.nan_to_num(np.prod(non_spam_norm, axis=1) * non_spam_prior)

predictions = np.asarray(np.where(p_spam >= p_non_spam, 1, 0)).reshape(-1)
for prediction , truth in zip(predictions, testY):
    if prediction == truth:
        if truth == 1:
            TP += 1
        else:
            TN += 1
    else:
        if prediction == 1:
            FP += 1
        else:
            FN += 1

print(TP, FP , TN , FN)
Precision =  TP / (TP + FP)
Recall = TP / (TP + FN)
F_1 = (2 * Precision * Recall) / (Precision + Recall)
Accuracy = (TP + TN) / yhat.shape[0]
```

```
561 337 609 12
```

In [ ]:
```python
md(f"$$Precision = {Precision*100:0.4f}\%," + "\\hspace{5pt}" +\
     f"Recall = {Recall*100:0.4f}\%,"+ "\\hspace{5pt}" +\
         f"F_1  = {F_1*100:0.4f}\%," + "\\hspace{5pt}" +\
             f"Accuracy = {Accuracy*100:0.4f}\%$$")
```

Out[ ]:

$$Precision = 62.4722\%,\hspace{5pt} Recall = 97.9058\%,\hspace{5pt} F_1 = 76.2746\%,\hspace{5pt} Accuracy = 77.0244\%$$

Section 4: Decision Trees

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split as tts
from scipy import stats
from IPython.display import Markdown as md
```

In [ ]:
```
# Read in the data
data = np.genfromtxt('spambase.data', delimiter=',')
dataMat = np.array(data)
# Set RNG with seed = 0
np.random.seed(0)
np.random.shuffle(dataMat)
# Splitting the data into X and Y vectors
X = dataMat[:, :-1]
Y =  np.reshape(dataMat[:, -1], (-1, 1))
```

In [ ]:
```
# Split the training and testing sets in a 2:1 ratio
trainX, testX, trainY, testY = tts(X, Y, test_size=0.33)
```

In [ ]:
```
mean = trainX.mean(axis=0)
std = trainX.std(axis=0, ddof=1)
####################################################################
trainX_std = (trainX - mean) / std
bias = np.ones((trainX_std.shape[0], 1))
TRAIN_X = np.append(bias, trainX_std , axis=1)

####################################################################
testX_std = (testX - mean) / std
bias = np.ones((testX_std.shape[0], 1))
TEST_X = np.append(bias, testX_std , axis=1)
```

In [ ]:
```
spam_mask = np.asarray(np.where(trainY == 1, True, False)).reshape(-1)
non_spam_mask = np.invert(spam_mask)

spam_train = np.compress(spam_mask, trainX, axis=0)
non_spam_train = np.compress(non_spam_mask, trainX, axis=0)
```

In [ ]:
```
binary_train_x = (TRAIN_X > TRAIN_X.mean(axis=0)).astype(int)
binary_test_x = (TEST_X > TEST_X.mean(axis=0)).astype(int)
```

In [ ]:

```
class Node:
    def __init__(self, value=None):
        self.value = value
        self.right = None
        self.left = None

class DecisionTree():
    def ID3(self, examples, attributes, default):
        X = examples[:, :-1]
        Y = examples[:, -1]
        if len(X) == 0:
            return Node(default)
        elif ((Y[0] == Y).all()):
            return Node(Y[0])
        elif (len(attributes) == 1):
            return Node(stats.mode(Y).mode[0])
        else:
            best = self.choose_attribute(attributes, examples)
            start = Node(attributes[best])
            newAttrs = np.delete(attributes, best)
            newExamples = np.delete(examples, best, axis=1)
            non_spam = newExamples[(examples[:, best] == 0)]
            start.left = self.ID3(non_spam, newAttrs, stats.mode(Y).mode[0])
            spam = newExamples[(examples[:, best] == 1)]
            start.right = self.ID3(spam, newAttrs, stats.mode(Y).mode[0])
        return start

    def choose_attribute(self, attributes, examples):
        attributes_entropies = []
        eps = np.finfo(float).eps
        for i in range(len(attributes)-1):
            r0 = examples[(examples[:, i] == 0)]
            r1 = examples[(examples[:, i] == 1)]
            n0 = r0[(r0[:, -1] == 0)]
            p0 = r0[(r0[:, -1] == 1)]
            n1 = r1[(r1[:, -1] == 0)]
            p1 = r1[(r1[:, -1] == 1)]
            h0 = self.H(p0.shape[0], n0.shape[0], r0.shape[0])
            h1 = self.H(p1.shape[0], n1.shape[0], r1.shape[0])
            entropy = (r0.shape[0]/len(examples) * h0) + (r1.shape[0]/len(examples) * h1)
            attributes_entropies.append(entropy)
        return np.argmin(attributes_entropies)

    def H(self, p, n, num_rows):
        pos_p = p/(num_rows+np.finfo(float).eps)
        neg_p = n/(num_rows+np.finfo(float).eps)
        return - neg_p*np.log2(neg_p + np.finfo(float).eps) \
               - pos_p*np.log2(pos_p + np.finfo(float).eps)

    def predict(self, root, testX, testY):
        predictions = []
        for i in range(testX.shape[0]):
            node = root
            while True:
                if testX[i][node.value] == 0:
                    node = node.left
                else:
                    node = node.right
                if node.left is None and node.right is None:
                    predictions.append(node.value)
                    break
        return predictions
```

In [ ]:

```python
dt = DecisionTree()
root = dt.ID3(np.hstack([binary_train_x, trainY]), np.arange(0, 57), stats.mode(trainY).mode[0])
predictions = dt.predict(root, binary_test_x, testY)
###################################################################################################
TP = FP = TN = FN = 0
for prediction , truth in zip(predictions, testY):
    if prediction == truth:
        if truth == 1:
            TP += 1
        else:
            TN += 1
    else:
        if prediction == 1:
            FP += 1
        else:
            FN += 1

print(TP, FP , TN , FN)
Precision =  TP / (TP + FP)
Recall = TP / (TP + FN)
F_1 = (2 * Precision * Recall) / (Precision + Recall)
Accuracy = (TP + TN) / yhat.shape[0]
```

```
532 59 852 76
```

In [ ]:

```python
md(f"$$Precision = {Precision*100:0.4f}\%," + "\\hspace{5pt}" +\
     f"Recall = {Recall*100:0.4f}\%,"+ "\\hspace{5pt}" +\
         f"F_1  = {F_1*100:0.4f}\%," + "\\hspace{5pt}" +\
             f"Accuracy = {Accuracy*100:0.4f}\%$$")
```

Out[ ]:

$$Precision = 90.0169\%,\hspace{5pt}Recall = 87.5000\%,\hspace{5pt}F_1 = 88.7406\%,\hspace{5pt}Accuracy = 91.1126\%$$