```
from sklearn.datasets import load_iris
iris = load_iris()


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
     Mounted at /content/gdrive
```

```
cd "/content/gdrive/MyDrive/1- Academics/CS 383/Assignment 2"
```

```
     /content/gdrive/MyDrive/1- Academics/CS 383/Assignment 2
```

```
spamdata = pd.read_csv('spambase.data', header=None)
train = spamdata.sample(frac=(2/3), random_state=0)
test = spamdata.drop(train.index)


trainY = train[57]
trainX = train.drop(columns=[57])

mean = trainX.mean()
std = trainX.std()
trainX = (trainX-mean)/std


testY = test[57]
testX = test.drop(columns=[57])
testX = (testX-mean)/std
#testX.insert(0, 'Bias', 1)
```

Logistic Regression

```
def sigmoid(x, theta):

  val = -(x @ theta)
  return 1/(1+np.exp(val))

def cost(y, x, thetas):

  activations = y*-np.log(sigmoid(x, thetas)) + (1-y)*-np.log(1 - sigmoid(x, thetas))
  cost = sum(activations)
```

```
  return cost


X = iris.data[:, :2]
y = (iris.target != 0) * 1
X[:,0] = (X[:,0] - X[:, 0].mean())/X[:, 0].std()
X[:,1] = (X[:,1] - X[:, 1].mean())/X[:, 1].std()
thetas = np.random.uniform(low=-1, high=1, size=2)
lr = 0.01


bias = np.ones(len(X)).reshape(150, 1)
X_bias = np.concatenate((bias, X), axis=1)


thetas = np.random.uniform(low=-1, high=1, size=3)


iter = 10000
counter = 0
stoppingthresh = 2E-23
old_loss = 1E9
lr = 0.01

while counter < iter:

  counter = counter+1
  loss = cost(y, X_bias, thetas)
  if abs(loss-old_loss) < stoppingthresh:
    break

  thetas = thetas + lr/2* X_bias.T @ (y - sigmoid(X_bias, thetas))
  old_loss = loss
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: RuntimeWarning: divide b

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: RuntimeWarning: invalid
```

```
c = -thetas[0]/thetas[2]
m = -thetas[1]/thetas[2]
xsample = np.asarray(list(range(-2, 2)))
ysample = m*xsample + c


thetas
```

```
array([ 8.34583606, 13.06817275, -6.55300008])
```

```
from sklearn.linear_model import LogisticRegression
```

```
lgr = LogisticRegression(penalty='none', solver='lbfgs', max_iter=10000)
lgr.fit(X, y)
```

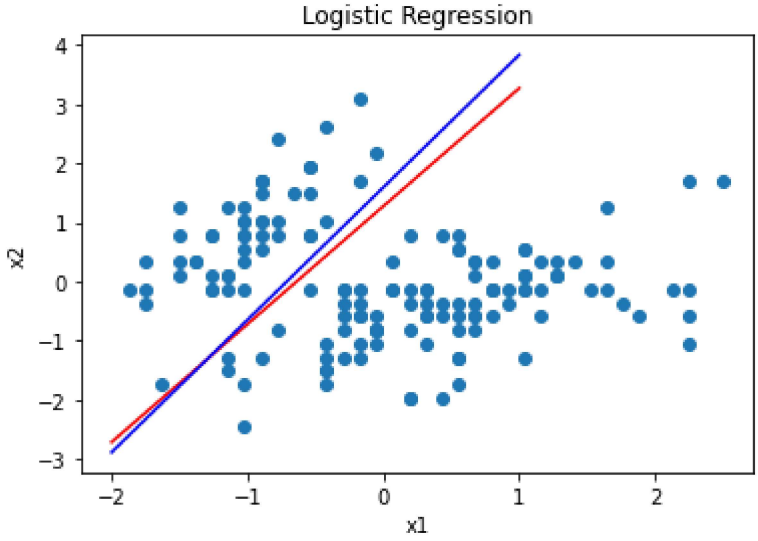
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
b = lgr.intercept_
coef = lgr.coef_[0]
c = -b/coef[1]
m = -coef[0]/coef[1]
```

```
xsampleval = np.asarray(list(range(-2, 2)))
ysampleval = m*xsampleval + c[0]
```

```
plt.scatter(X[:,0], X[:,1])
plt.plot(xsample, ysample, 'r', label='scratch')
plt.plot(xsampleval, ysampleval, 'b', label='sklearn')
plt.title('Logistic Regression')
plt.xlabel('x1')
plt.ylabel('x2')
```

```
Text(0, 0.5, 'x2')
```



Logistic Regression Classifier

```
X = trainX.to_numpy()
Y = trainY.to_numpy()
X_test = testX.to_numpy()
Y_test = testY.to_numpy()
```

```
bias = np.ones(len(X)).reshape(len(X), 1)
X_bias = np.concatenate((bias, X), axis=1)


thetas = np.random.uniform(low=-1, high=1, size=58)
lr = 0.01


iter = 1500
counter = 0
stoppingthresh = 2E-23
old_loss = 1E9
lr = 0.01

while counter < iter:

  counter = counter+1
  loss = cost(Y, X_bias, thetas)
  if abs(loss-old_loss) < stoppingthresh:
    break

  thetas = thetas + lr/57* X_bias.T @ (Y - sigmoid(X_bias, thetas))
  old_loss = loss
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: RuntimeWarning: divide b

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: RuntimeWarning: invalid
```

```
y_score = sigmoid(X_bias, thetas)
y_class = np.zeros(len(y_score))
pos_mask = np.where(y_score > 0.5)
y_class[pos_mask] = 1


def accuracy(y, y_pred):

  total = 0
  for i in range(len(y)):
    if y[i] == y_pred[i]:
      total = total + 1

  return total/len(y)

def precision(y, y_pred):
  tp = 0
  fp = 0

  for i in range(len(y)):
    if y_pred[i] == 1:
```

```
        if y[i] == 1:
          tp = tp+1

        else:
          fp = fp+1

    return tp/(tp+fp)

  def recall(y, y_pred):
    tp = 0
    fn = 0

    for i in range(len(y)):
      if y_pred[i] == 1:

        if y[i] == 1:
          tp = tp+1

      else:
        if y[i] == 1:
          fn = fn+1

    return tp/(tp+fn)

  def f1(precision, recall):
    return (2*precision*recall)/(precision+recall)


  acc = accuracy(Y, y_class)
  p = precision(Y, y_class)
  r = recall(Y, y_class)
  f = f1(p, r)

  print(acc, p, r, f)
```

```
0.9272905119008803 0.9267676767676768 0.8900565885206144 0.9080412371134021
```

Naive Bayes Classifier

```
def gauss(x, mu, sig):
  exponent = np.exp((-1/2)* ((x-mu)/sig)**2)
  prob = 1/(sig*np.sqrt(2*np.pi)) * exponent
  return prob


X = trainX.to_numpy()
Y = trainY.to_numpy()
X_test = testX.to_numpy()
Y_test = testY.to_numpy()
```

```
oneidx = np.where(Y == 1)[0]
zeroidx = np.where(Y == 0)[0]
p1 = len(oneidx)/len(Y)
p0 = 1 - p1


X_class_0 = X[zeroidx].copy()
X_class_1 = X[oneidx].copy()


zero_mean = X_class_0.mean(axis=0).reshape(57, 1)
zero_std = X_class_0.std(axis=0).reshape(57, 1)


one_mean = X_class_1.mean(axis=0).reshape(57, 1)
one_std = X_class_1.std(axis=0).reshape(57, 1)


test_prob_0 = gauss(X_test, zero_mean.T, zero_std.T)
test_prob_1 = gauss(X_test, one_mean.T, one_std.T)

f_test_prob_0 = test_prob_0.prod(axis=1)*p0
f_test_prob_1 = test_prob_1.prod(axis=1)*p1


y_assign = []

for i in range(len(f_test_prob_0)):
  if f_test_prob_0[i] > f_test_prob_1[i]:
    y_assign.append(0)

  else:
    y_assign.append(1)


y_pred = np.asarray(y_assign)


acc = accuracy(y, y_pred)
p = precision(y, y_pred)
r = recall(y, y_pred)
f = f1(p, r)

print(acc, p, r, f)
```

```
0.6733333333333333 0.6711409395973155 1.0 0.8032128514056226
```

Decision Tree Classifier

```
X = trainX.to_numpy()
Y = trainY.to_numpy()
X_test = testX.to_numpy()
Y_test = testY.to_numpy()


oneidx = np.where(Y == 1)[0]
zeroidx = np.where(Y == 0)[0]
p1 = len(oneidx)/len(Y)
p0 = 1 - p1


X_class_0 = X[zeroidx].copy()
X_class_1 = X[oneidx].copy()


entropy = -p0*np.log2(p0) + -p1*np.log2(p1)


zero_mean = X_class_0.mean(axis=0).reshape(57, 1)
zero_std = X_class_0.std(axis=0).reshape(57, 1)


one_mean = X_class_1.mean(axis=0).reshape(57, 1)
one_std = X_class_1.std(axis=0).reshape(57, 1)


prob_0 = gauss(X, zero_mean.T, zero_std.T)
prob_1 = gauss(X, one_mean.T, one_std.T)


prob_0.dtype
```

```
dtype('float64')
```

```
eps = np.finfo(np.float64).eps


mask_0 = np.where(prob_0 < eps)
mask_1 = np.where(prob_1 < eps)


prob_0[mask_0] = eps
prob_1[mask_1] = eps


entropy_all = -prob_0*np.log2(prob_0) + -prob_1*np.log2(prob_1)
IG = entropy - entropy_all.sum(axis=0)/3067


IG.argmax()
```

```
31
```

```
entropy_all[31]
```

```
array([ 1.03449145e+00, -1.57751768e-01,  1.04839683e+00, -1.28442278e+02,
        1.04592185e+00,  9.78475274e-01, -1.79948539e-01,  7.93281993e-01,
        1.01015399e+00,  1.04995073e+00,  1.02432887e+00,  1.05241208e+00,
        5.43182519e-01,  1.05758611e+00,  2.76159054e-02,  7.61350278e-01,
        5.55642315e-01,  1.01614573e+00,  1.01955496e+00, -1.67526942e+00,
        9.55150342e-01,  8.91650511e-01, -1.63694540e+00,  1.00309670e+00,
       -5.78153968e+00, -5.47796547e+00, -1.73560866e+02,  7.22970956e-01,
       -7.33115955e+01, -3.33829761e-01, -7.39275668e+00, -4.21074724e+15,
       -1.58420476e+00, -5.53584142e+00, -5.37430553e+00,  2.81692684e-01,
        9.85725939e-01, -1.56571448e+00, -1.78002374e+00,  6.95118609e-01,
       -3.03120279e+02, -4.25139228e+01, -8.23035430e-01, -8.15201331e+00,
        6.17665654e-01, -2.27897877e+00, -6.08839969e-01, -4.94752541e+00,
        7.13723461e-01,  1.06049599e+00,  6.01907748e-01,  1.01301778e+00,
       -4.64657586e-01,  7.30579698e-01, -1.82214392e+00, -1.69372924e+00,
        9.50397290e-01])
```

```
prob_0_test = gauss(X_test, zero_mean.T, zero_std.T)
prob_1_test = gauss(X_test, one_mean.T, one_std.T)
```

```
prob_0_test[0][31] < prob_1_test[1][31]
```

```
True
```

```
Y_test[31]
```

```
1
```

```
for i in range(len(prob_0_test)):
  print((prob_0_test[i][31] < prob_1_test[i][31]),Y_test[i])
```

✓ 0s completed at 10:10 PM