CS 613 - Machine Learning

Assignment 2 - Classification

Name
CS 613 Machine Learning
Fall 2021
Dr. Edward Kim
Drexel University

Introduction

In this assignment you will perform classification using Logistic Regression, Naive Bayes and Decision Tree classifiers. You will run your implementations on a binary class dataset and report your results.

You may **NOT** use any functions from a ML library in your code unless explicitly told otherwise.

Submission

For your submission, upload to Blackboard a single zip file containing:

1. PDF Writeup and PDF of Jupyter Notebook (can be the same PDF)
2. Python notebook Code

The PDF document should contain the following at the top:

1. Answers to Theory Questions

1.1

- a. What is the sample entropy, $H(Y)$ from this training data (using log base 2) (2pts)?
- b. What are the information gains for branching on variables x_1 and x_2 (2pts)?
- c. Draw the decision tree that would be learned by the ID3 algorithm without pruning from this training data (3pts)?

1.2

- a. What are the class priors, $P(A = Yes), P(A = No)$? (2pt)
- b. Find the parameters of the Gaussians necessary to do Gaussian Naive Bayes classification on this decision to give an A or not. Standardize the features first over all the data together so that there is no unfair bias towards the features of different scales (2pts).
- c. Using your response from the prior question, determine if an essay with 242 characters and an average word length of 4.56 should get an A or not (3pts).

1.3

- a. How could you use a validation set to determine the user-defined parameter k?

2. Requested Logistic Regression thetas and plots

3. Requested Classification Statistics

> precision:
> recall:
> f1_score:
> accuracy:

4. Requested Classification Statistics

> precision:
> recall:
> f1_score:
> accuracy:

5. Requested Classification Statistics

> precision:
> recall:
> f1_score:
> accuracy:

Datasets

Iris Dataset (sklearn.datasets.load_iris)

The Iris flower data set or Fishers Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

The iris data set is widely used as a beginner's dataset for machine learning purposes. The dataset is included in the machine learning package Scikit-learn, so that users can access it without having to find a source for it. The following python code illustrates usage.

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

Spambase Dataset (spambase.data)

This dataset consists of 4601 instances of data, each with 57 features and a class label designating if the sample is spam or not. The features are *real valued* and are described in much detail here:

> https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names (https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names)

Data obtained from: https://archive.ics.uci.edu/ml/datasets/Spambase (https://archive.ics.uci.edu/ml/datasets/Spambase)

Imports

In [7]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib as mpl

from pprint import pprint
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from scipy.special import logsumexp
```

1 Theory

1 Consider the following set of training examples for an unknown target function: $(x_1, x_2) \rightarrow y$

In [3]:
```
theory_data1 = pd.DataFrame({
    "Y" : ['+', '+', '+', '+', '-', '-', '-', '-'],
     "x1" : ['T', 'T', 'F', 'F', 'T', 'T', 'F', 'F'],
     "x2" : ['T', 'F', 'T', 'F', 'T', 'F', 'T', 'F'],
     "Count" : [3, 4, 4, 1, 0, 1, 3, 5]
})
theory_data1['Y'] = abs(theory_data1['Y'].astype('category').cat.codes - 1)
theory_data1
```

Out[3]:

	Y	x1	x2	Count
0	1	T	T	3
1	1	T	F	4
2	1	F	T	4
3	1	F	F	1
4	0	T	T	0
5	0	T	F	1
6	0	F	T	3
7	0	F	F	5

a. What is the sample entropy, $H(Y)$ from this training data (using log base 2) (2pts)?

In [844]:
```
entropy=-((9/21)*math.log2(9/21)+(12/21)*math.log2(12/21))
entropy
```

Out[844]: 0.9852281360342515

p(0)=9/21 p(1)=12/21 from the above data. Entropy = sum of (-pi*log2(pi)) = -((9/21)*log2(9/21)+(12/21)*log2(12/21)) H(Y) = 0.9852281360342515

b. item What are the information gains for branching on variables x_1 and x_2 (2pts)?

In [855]:
```
8/21*((-7/8)*math.log2(7/8)+(-1/8)*math.log2(1/8))
```

Out[855]: 0.20707216883794147

In [852]:
```
13/21*((-5/13)*math.log2(5/13)+(-8/13)*math.log2(8/13))
```

Out[852]: 0.5950512314951136

In [853]:
```
ig1=entropy-(0.20707216883794147+0.5950512314951136)
```

In [854]:
```
ig1
```

Out[854]: 0.18310473570119645

Information gain= H(Y)-H(Y/X1) = .985229 - (-8/21*((-7/8)*math.log2(7/8)+(-1/8)*math.log2(1/8)))-13/21*((-5/13)*math.log2(5/13)+(-8/13)*math.log2(8/13)) =0.18310473570119645

In [856]:
```
10/21*((-7/10)*math.log2(7/10)+(-3/10)*math.log2(3/10))
```
Out[856]: 0.4196623329669965

In [857]:
```
11/21*((-5/11)*math.log2(5/11)+(-6/11)*math.log2(6/11))
```
Out[857]: 0.5206824917260249

In [858]:
```
ig2=entropy-(0.4196623329669965+0.5206824917260249)
```

In [859]:
```
 ig2
```
Out[859]: 0.04488331134123014

Information gain= H(Y)-H(Y/X2) = .985229 - (-10/21*((-7/10)math.log2(7/10)+ (-3/10)math.log2(3/10)))-13/21*((11/21*((-5/11)*math.log2(5/11)+(-6/11)*math.log2(6/11)) =0.04488331134123014

c. item Draw the deicion tree that would be learned by the ID3 algorithm without pruning from this training data (3pts)?

In [ ]:
```

```

2

We decided that maybe we can use the number of characters and the average word length an essay to determine if the student should get an A in a class or not. Below are five samples of this data:

In [900]:
```
theory_data2 = pd.DataFrame({
    "# of Chars" : [216, 69, 302, 60, 393],
    "Average Word Length" : [5.68, 4.78, 2.31, 3.16, 4.2],
    "Give an A" : ['Yes', 'Yes', 'No', 'Yes', 'No']})
theory_data2['bool'] = theory_data2['Give an A'].astype('category').cat.cod
theory_data2
```
Out[900]:

	# of Chars	Average Word Length	Give an A	bool
0	216	5.68	Yes	1
1	69	4.78	Yes	1
2	302	2.31	No	0
3	60	3.16	Yes	1
4	393	4.20	No	0

a. What are the class priors, $P(A = Yes)$, $P(A = No)$? (2pt)

P(A=Yes)=3/5 P(A=no)=2/5

b. Find the parameters of the Gaussians necessary to do Gaussian Naive Bayes classification on this decision to give an A or not. Standardize the features first over all the data together so that there is no unfair bias towards the features of different scales (2pts).

In [901]:
```
def Standardise(string1):
    m1=np.mean(string1)
    std1=np.std(string1)
    string2=(string1-m1)/std1
    return string2
```

In [902]:
```
Y_train=np.array(theory_data2["bool"])
theory_data2=theory_data2.drop(["Give an A"],axis=1)
theory_data2=theory_data2.drop(["bool"],axis=1)
X_train=theory_data2.apply(Standardise)
```

In [903]:
```
Y_train=Y_train.reshape(-1,1)
Y_train
```

Out[903]:
```
array([[1],
       [1],
       [0],
       [1],
       [0]], dtype=int8)
```

In [904]:
```
df_zero= X_train[Y_train == 0]
df_one = X_train[Y_train==1]
```

In [905]:
```
df_zero
```

Out[905]:

	# of Chars	Average Word Length
2	0.723720	-1.447277
4	1.424342	0.146752

In [906]:
```
df_one
```

Out[906]:

	# of Chars	Average Word Length
0	0.061593	1.394987
1	-1.070181	0.635925
3	-1.139473	-0.730386

In [907]:
```
def method(string3):
    string4=np.mean(string3)
    return string4
def method1(string3):
    string4=np.var(string3)
    return string4
```

In [908]:
```
df_zero1=df_zero.apply(method)
df_zero2=df_zero.apply(method1)
df_one1=df_one.apply(method)
df_one2=df_one.apply(method1)
```

In [909]:
```
import math
def norm1(X,mean,var):
    x2=2*var
    x3=(X-mean)**2
    x4=2*math.pi*var
    Ck=(1/np.sqrt(x4))*(np.exp(-x3/x2))
    return Ck
```

In [910]:
```
X_one=norm1(X_train,df_one1,df_one2)
X_zero=norm1(X_train,df_zero1,df_zero2)
priority0 = len(df_zero) / len(X_train)
priority1 = len(df_one) / len(X_train)
print(priority0)
print(priority1)
```

```
0.4
0.6
```

In [915]:
```
a_prob=X_one*priority1
nota_prob=X_zero*priority0
print(a_prob)
print(nota_prob)
```

```
   # of Chars  Average Word Length
0    0.160358             0.149728
1    0.353501             0.265074
2    0.014237             0.027646
3    0.323439             0.113372
4    0.000227             0.258097
      # of Chars  Average Word Length
0  6.994277e-03             0.007440
1  3.334723e-09             0.054451
2  2.762924e-01             0.121439
3  9.744360e-10             0.199209
4  2.762924e-01             0.121439
```

prior = 1/n_class

final_probabilities = [] for i in k: class_prob = np.prod(i) * prior final_probabilities.append(class_prob)

In [914]:
```
print(np.prod(a_prob))
print(np.prod(nota_prob))
```

```
# of Chars              5.933885e-08
Average Word Length    3.210620e-05
dtype: float64
# of Chars              1.734977e-21
Average Word Length    1.190158e-06
dtype: float64
```

c. Using your response from the prior question, determine if an essay with 242 characters and an average word length of 4.56 should get an A or not (3pts).

In [ ]:

3

Consider the following questions pertaining to a k-Nearest Neighbors algorithm (1pt):

a. How could you use a *validation set* to determine the user-defined parameter k?

---

2 Logistic Regression

Let's train and test a *Logistic Regression Classifier* to classify flowers from the Iris Dataset.

First download import the data from sklearn.datasets. As mentioned in the Datasets area, The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. We will map this into a binary classification problem between Iris setosa versus Iris virgincia and versicolor. We will use just the first 2 features, width and length of the sepals.

For this part, we will be practicing gradient descent with logistic regression.

Use the following code to load the data, and binarize the target values.

```
iris = skdata.load_iris()
X = iris.data[:, :2]
y = (iris.target != 0) * 1
```

Write a script that:

1. Reads in the data with the script above.
2. Standardizes the data using the mean and standard deviation
3. Initialize the parameters of θ using random values in the range [-1, 1]
4. Do **batch** gradient descent
5. Terminate when absolute value change in the loss on the data is less than 2^{-23}, or after $10,000$ iterations have passed (whichever occurs first).
6. Use a learning rate $\eta = 0.01$.
7. While the termination criteria (mentioned above in the implementation details) hasn't been met
   a. Compute the loss of the data using the logistic regression cost
   b. Update each parameter using **batch** gradient descent

Plot the data and the decision boundary using matplotlib. Verify your solution with the LogisticRegression sklearn method.

```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression(penalty='none',solver='lbfgs',max_iter=100
00)
lgr.fit(X,y)
```

In your writeup, present the thetas from gradient descent that minimize the loss function as well as plots of your method versus the built in LogisticRegression method.

In [146]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```

In [147]:
```
#iris
```

In [148]:
```
'''sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (
#iris['target']
```

Out[148]:
```
"sepal length (cm)','sepal width (cm)','petal length (cm)','petal width
(cm)'],"
```

In [149]:
```
X = iris.data[:, :2]
y = (iris.target != 0) * 1
y=y.reshape(-1,1)
#print(X)
#print(y)
```

In [150]:
```
y.shape
```

Out[150]:
```
(150, 1)
```

In [151]:
```
def Standardise(string1):
    m1=np.mean(string1)
    std1=np.std(string1)
    string2=(string1-m1)/std1
    return string2
```

In [152]:
```
X1=Standardise(X[:,0])
X2=Standardise(X[:,1])
ones = np.ones(X.shape[0]).reshape(X.shape[0], 1)
X1=X1.reshape(-1,1)
X2=X2.reshape(-1,1)
X3=np.c_[ones,X1,X2]
```

In [153]:
```
#X3
```

In [154]:
```
from numpy.random import seed
from numpy.random import rand
seed(42)
size=X3.shape[1]
size
values = rand(size)
theta=np.array(values).reshape(-1,1)
theta1=np.array(values).reshape(-1,1)
theta2=theta
```

calculate gradient theta = theta -gradientTheta while(True): calculate gradient newTheta = theta - gradient if gradient is very close to zero and abs(newTheta-Theta) is very close to zero: break from loop # (The algorithm has converged) theta = newTheta

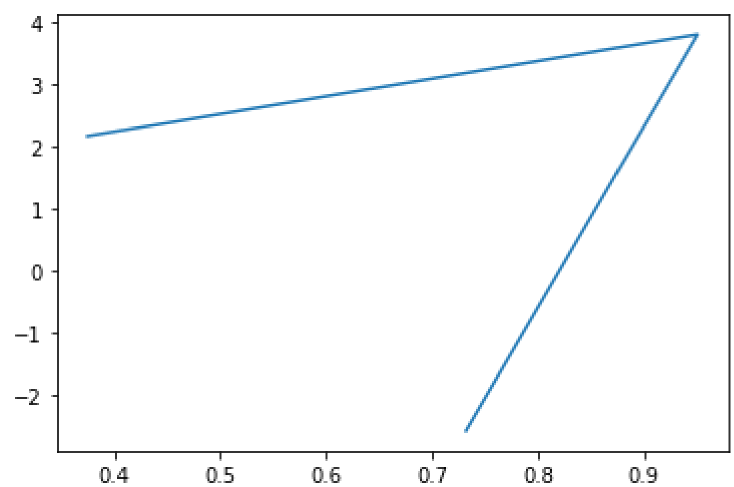
In [155]:
```
def loss1(gx, y):
    cost=0
    for i, j in zip(gx,y):
        if ( i!= 1 and i != 0 ):
            cost = cost + j*np.log(i)+(1-j)*np.log(1-i)

    return np.mean(cost)
```

In [157]:
```
loss=0
count=0
eta=0.01
iter = 10000
train_error = [1]
while count<iter:
    gx1=1/(1+np.exp(-X3@theta))
    grad=X3.transpose() @ (gx1 - y)
    theta=theta-((eta/X3.shape[0])*grad)
    #print(theta)
    cost2=loss1(gx1,y)
    train_error.append(cost2)
    if abs(train_error[-2]-train_error[-1])<=2**-23:
        break
    count=count+1
theta
```

Out[157]:
```
array([[ 2.17513889],
       [ 3.81031692],
       [-2.56300942]])
```

In [159]:
```
from matplotlib import pyplot as plt
plt.plot(theta2, theta)
```

Out[159]:
```
[<matplotlib.lines.Line2D at 0x7f82c8fb5190>]
```



In [948]:
```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression(penalty='none',solver='lbfgs',max_iter=10000)
lgr.fit(X3,y)
```

```
/Users/brindakulkarni/opt/anaconda3/lib/python3.8/site-packages/sklearn/u
tils/validation.py:63: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  return f(*args, **kwargs)
```

Out[948]:
```
LogisticRegression(max_iter=10000, penalty='none')
```

In [949]:
```
lgr.coef_
```

Out[949]:
```
array([[ 19.97028527,  60.36395919, -28.8049862 ]])
```

3 Logistic Regression Spam Classification

Let's train and test a *Logistic Regression Classifier* to classifiy Spam or Not from the Spambase Dataset.

First download the dataset *spambase.data* from Blackboard. As mentioned in the Datasets area, this dataset contains 4601 rows of data, each with 57 continuous valued features followed by a binary class label (0=not-spam, 1=spam). There is no header information in this file and the data is comma separated.

Write a script that:

1. Reads in the data.
2. Randomizes the data.
3. Selects the first 2/3 (round up) of the data for training and the remaining for testing (you may use **sklearn train_test_split** for this part)
4. Standardizes the data (except for the last column of course) using the training data
5. Initialize the parameters of θ using random values in the range [-1, 1]
6. Do *batch gradient descent*
7. Terminate when absolute value change in the loss on the data is less than 2^{-23}, or after $1,500$ iterations have passed (whichever occurs first, this will likely be a slow process).
8. Use a learning rate $\eta = 0.01$.
9. Classify each testing sample using the model and choosing the class label based on which class probability is higher.
10. Computes the following statistics using the testing data results:
    [lecture 3 b]
    a. Precision
    b. Recall
    c. F-measure
    d. Accuracy

Implementation Details

1. Seed the random number generate with zero prior to randomizing the data
2. There are a lot of θs and this will likely be a slow process

In your report you will need

1. The statistics requested for your Logistic classifier run.

In [5]:
```
import pandas as pd
import numpy as np
df1=pd.read_csv("spambase.data",header=None)
```

In [6]:
```
df1 = df1.sample(frac = 1)
```

In [7]:
```
Y=df1[57]
X=df1.drop([57],axis=1)
Y=np.array(Y)
Y=Y.reshape(-1,1)
Y
```

Out[7]:
```
array([[1],
       [1],
       [1],
       ...,
       [0],
       [0],
       [1]])
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.34)
```

In [9]:
```
Y_train
```

Out[9]:
```
array([[1],
       [1],
       [1],
       ...,
       [0],
       [0],
       [1]])
```

In [10]:
```
print(len(X_train)," ",len(X_test))
```

```
3036   1565
```

In [11]:
```
def Standardise(string1):
    m1=np.mean(string1)
    std1=np.std(string1)
    string2=(string1-m1)/std1
    return string2
```

In [12]:
```
X_train=X_train.apply(Standardise)
#X_train
```

In [13]:
```
ones = np.ones(X_train.shape[0]).reshape(X_train.shape[0], 1)
X_train=np.concatenate((ones,X_train),1)
X_train.shape
```

Out[13]:
```
(3036, 58)
```

In [14]:
```
from numpy.random import seed
from numpy.random import rand
seed(42)
size=X_train.shape[1]
values = rand(size)
theta=np.array(values).reshape(-1,1)
theta1=np.array(values).reshape(-1,1)
theta.shape
```

Out[14]:
```
(58, 1)
```

In [15]:
```
gx=1/(1+np.exp(-X_train@theta))
gx
```

Out[15]:
```
array([[0.9996303 ],
       [0.00693386],
       [0.9999065 ],
       ...,
       [0.03476937],
       [0.45202125],
       [0.0055623 ]])
```

In [16]:
```
gx1=np.log(gx)
gx2=np.log(1-gx)
gx1
```

```
<ipython-input-16-d299603e6eba>:2: RuntimeWarning: divide by zero encount
ered in log
  gx2=np.log(1-gx)
```

Out[16]:
```
array([[-3.69768364e-04],
       [-4.97133903e+00],
       [-9.35074504e-05],
       ...,
       [-3.35901834e+00],
       [-7.94026092e-01],
       [-5.19174370e+00]])
```

In [17]:
```
def loss1(gx, y):
    cost=0
    for i, j in zip(gx,y):
        if ( i!= 1 and i != 0 ):
            cost += j*np.log(i)+(1-j)*np.log(1-i)

    return np.mean(cost)
```

In [18]:
```
loss=0
count=0
eta=0.01
iter = 10000
train_error = [1]
while count<iter:
    gx=1/(1+np.exp(-X_train@theta))
    grad=X_train.transpose() @ (gx - Y_train)
    theta=theta-((eta/X_train.shape[0])*grad)
    #print(theta)
    cost2=loss1(gx,Y_train)
    train_error.append(cost2)
    if abs(train_error[-2]-train_error[-1])<=2**-23:
        break
    count=count+1
theta
```

Out[18]:
```
array([[-0.49792168],
       [-0.12033906],
       [-0.12664462],
       [ 0.13296111],
       [ 0.37495036],
       [ 0.35708362],
       [ 0.17670686],
       [ 1.08024375],
       [ 0.30829383],
       [ 0.32568932],
       [ 0.04978401],
       [ 0.09051493],
       [-0.14363676],
       [-0.04696918],
       [ 0.11327776],
       [ 0.3538238 ],
       [ 0.39877652],
       [ 0.38564674],
       [ 0.21212101],
       [ 0.17347728],
       [ 0.53586228],
       [ 0.24005825],
       [ 0.43469973],
       [ 0.97050643],
       [ 0.47923177],
       [-0.93761023],
       [-0.75891843],
       [-0.68895864],
       [ 0.14234934],
       [-0.35847343],
       [-0.27614312],
       [-0.24338315],
       [-0.43435494],
       [-0.35589355],
       [ 0.40896515],
       [-0.23828744],
       [ 0.15636356],
       [-0.13030599],
       [-0.07754049],
       [-0.23110903],
```

```
       [-0.22135012],
       [-0.28164779],
       [-0.64299131],
       [-0.14763768],
       [-0.31326566],
       [-0.62455775],
       [-0.45438662],
       [-0.18363917],
       [-0.26554588],
       [-0.31694926],
       [-0.00395129],
       [-0.06864494],
       [ 0.41116475],
       [ 1.30262568],
       [ 0.49397032],
       [ 1.18194084],
       [ 0.55580615],
       [ 0.31243488]])
```

In [ ]:
```
Accuracy = (TP + TN)/(TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
```

4 Naive Bayes Classifier

Let's train and test a *Naive Bayes Classifier* to classifiy Spam or Not from the Spambase Dataset.

First download the dataset *spambase.data* from Blackboard. As mentioned in the Datasets area, this dataset contains 4601 rows of data, each with 57 continuous valued features followed by a binary class label (0=not-spam, 1=spam). There is no header information in this file and the data is comma separated. As always, your code should work on any dataset that lacks header information and has several comma-separated continuous-valued features followed by a class id $\in 0, 1$.

Write a script that:

1. Reads in the data.
2. Randomizes the data.
3. Selects the first 2/3 (round up) of the data for training and the remaining for testing
4. Standardizes the data (except for the last column of course) using the training data
5. Divides the training data into two groups: Spam samples, Non-Spam samples.
6. Creates Gaussian models for each feature for each class.
7. Classify each testing sample using these models and choosing the class label based on which class probability is higher.
8. Computes the following statistics using the testing data results:
   a. Precision
   b. Recall

c. F-measure
d. Accuracy

Implementation Details

1. Seed the random number generate with zero prior to randomizing the data
2. If you decide to work in log space, realize that python interprets 0log0 as inf. You should identify this situation and either add an EPS (very small positive number) or add a very largenegative number to the log sum.

In your report you will need:

1. The statistics requested for your Naive Bayes classifier run.

In [102]:
```
import pandas as pd
df1=pd.read_csv("spambase.data",header=None)
```

In [103]:
```
df1 = df1.sample(frac = 1)
```

In [104]:
```
train = df1.sample(frac=0.66, random_state=42)
test = df1.drop(train.index)
print(f"No. of training set: {train.shape[0]}")
print(f"No. of testing set: {test.shape[0]}")
```

```
No. of training set: 3037
No. of testing set: 1564
```

In [105]:
```
def Standardise(string1):
    m1=np.mean(string1)
    std1=np.std(string1)
    string2=(string1-m1)/std1
    return string2
```

In [106]:
```
Y_train=np.array(train[57])
Y_train=Y_train.reshape(-1,1)
train=train.drop([57],axis=1)
train=train.apply(Standardise)
Y_test=np.array(test[57])
Y_test=Y_test.reshape(-1,1)
test=test.drop([57],axis=1)
test=test.apply(Standardise)
#train[57]=Y_train
```

In [107]:
```
df_zero= train[Y_train == 0]
index1 = df_zero.index
index1.name = "Non Spam Samples"
df_one = train[Y_train==1]
index = df_one.index
index.name = "Spam Samples"
```

In [108]:
```
def method(string3):
    string4=np.mean(string3)
    return string4
def method1(string3):
    string4=np.var(string3)
    return string4
```

In [109]:
```
df_zero1=df_zero.apply(method)
df_zero2=df_zero.apply(method1)
df_one1=df_one.apply(method)
df_one2=df_one.apply(method1)
```

In [110]:
```
#test_zero2
```

In [111]:
```
import math
def norm1(X,mean,var):
    x2=2*var
    x3=(X-mean)**2
    x4=2*math.pi*var
    Ck=(1/np.sqrt(x4))*(np.exp(-x3/x2))
    return Ck
```

In [112]:
```
X_one=norm1(train,df_one1,df_one2)
X_zero=norm1(train,df_zero1,df_zero2)
priority0 = len(df_zero) / len(train)
priority1 = len(df_one) / len(train)
print(priority0)
print(priority1)
```

```
0.6042146855449456
0.3957853144550543
```

In [113]:
```
X_test1=norm1(test,df_one1,df_one2)
X_test1
```

Out[113]:

	0	1	2	3	4	5	6	7	8	
3507	0.423867	1.381900	0.296082	0.250618	0.293189	0.325384	0.240290	0.298267	0.288782	0.
933	0.378700	1.381900	0.008076	0.250618	0.117588	0.293578	0.011559	0.277289	0.288782	0.
192	0.378700	0.279717	0.296082	0.250618	0.354427	0.293578	0.240290	0.277289	0.288782	0.
3421	0.378700	1.381900	0.296082	0.250618	0.293189	0.293578	0.240290	0.277289	0.288782	0.
4414	0.378700	1.381900	0.138409	0.250618	0.293189	0.018062	0.240290	0.277289	0.288782	0.
...	...	...	...	...	...	...	...	...	...	
1614	0.378700	1.563959	0.296082	0.250618	0.374249	0.301893	0.240290	0.277289	0.288782	0.
1132	0.266917	1.381900	0.296082	0.250618	0.338133	0.044573	0.240290	0.277289	0.288782	0.
97	0.378700	1.165277	0.417745	0.250618	0.383402	0.293578	0.240290	0.277289	0.288782	0.
2072	0.378700	1.381900	0.296082	0.250618	0.293189	0.293578	0.240290	0.277289	0.288782	0.
3509	0.376008	1.452351	0.296082	0.250618	0.383186	0.316621	0.240290	0.256981	0.288782	0.

1564 rows × 57 columns

In [114]:
```
X_test0=norm1(test,df_zero1,df_zero2)
X_test0
```

Out[114]:

	0	1	2	3	4	5	6	7	٤
3507	0.383826	0.310702	0.370577	2.608648	0.413478	0.485956	1.846038e+00	0.433179	0.51184(
933	0.372270	0.310702	0.003611	2.608648	0.033407	0.488592	3.203748e-92	0.634675	0.51184(
192	0.372270	0.298641	0.370577	2.608648	0.260602	0.488592	1.846038e+00	0.634675	0.51184(
3421	0.372270	0.310702	0.370577	2.608648	0.413478	0.488592	1.846038e+00	0.634675	0.51184(
4414	0.372270	0.310702	0.075568	2.608648	0.413478	0.000087	1.846038e+00	0.634675	0.51184(
...	...	...	...	...	...	...	...	...	..
1614	0.372270	0.313919	0.370577	2.608648	0.416547	0.218483	1.846038e+00	0.634675	0.51184(
1132	0.214007	0.310702	0.370577	2.608648	0.231212	0.000884	1.846038e+00	0.634675	0.51184(
97	0.372270	0.312734	0.340949	2.608648	0.396859	0.488592	1.846038e+00	0.634675	0.51184(
2072	0.372270	0.310702	0.370577	2.608648	0.413478	0.488592	1.846038e+00	0.634675	0.51184(
3509	0.304911	0.314252	0.370577	2.608648	0.341926	0.268307	1.846038e+00	0.097763	0.51184(

1564 rows × 57 columns

In [115]:
```
Spam_probability=X_one*priority1
NonSpam_probability=X_zero*priority0
```

In [116]:
```
np.prod(Spam_probability)
```

Out[116]:
```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
5      0.0
6      0.0
7      0.0
8      0.0
9      0.0
10     0.0
11     0.0
12     0.0
13     0.0
14     0.0
15     0.0
16     0.0
17     0.0
18     0.0
19     0.0
20     0.0
21     0.0
22     0.0
23     0.0
24     0.0
25     0.0
26     0.0
27     0.0
28     0.0
29     0.0
30     0.0
31     0.0
32     0.0
33     0.0
34     0.0
35     0.0
36     0.0
37     0.0
38     0.0
39     0.0
40     0.0
41     0.0
42     0.0
43     0.0
44     0.0
45     0.0
46     0.0
47     0.0
48     0.0
49     0.0
50     0.0
51     0.0
52     0.0
53     0.0
54     0.0
```

```
55    0.0
56    0.0
dtype: float64
```

In [117]:
```
np.prod(NonSpam_probability)
```

```
/Users/brindakulkarni/opt/anaconda3/lib/python3.8/site-packages/numpy/cor
e/_methods.py:51: RuntimeWarning: overflow encountered in reduce
  return umr_prod(a, axis, dtype, out, keepdims, initial, where)
```

Out[117]:
```
0      0.0
1      0.0
2      0.0
3      NaN
4      0.0
5      0.0
6      0.0
7      0.0
8      0.0
9      0.0
10     0.0
11     0.0
12     0.0
13     0.0
14     0.0
15     0.0
16     0.0
17     0.0
18     0.0
19     0.0
20     0.0
21     0.0
22     0.0
23     0.0
24     0.0
25     0.0
26     0.0
27     0.0
28     0.0
29     0.0
30     0.0
31     0.0
32     0.0
33     0.0
34     0.0
35     0.0
36     0.0
37     0.0
38     0.0
39     0.0
40     0.0
41     0.0
42     0.0
43     0.0
44     0.0
45     0.0
46     0.0
47     0.0
48     0.0
49     0.0
50     0.0
```

```
51    0.0
52    0.0
53    0.0
54    0.0
55    0.0
56    0.0
dtype: float64
```

In [122]:
```
Y_test.shape
```

Out[122]:
```
(1564, 1)
```

In [123]:
```
matrix = {'TP':0, 'FP':0, 'TN': 0, 'FN':0}
count=0
for sp,nsp, gt in zip(X_test1,X_test0, Y_test):
    count=count+1
    sp_mul = np.prod(sp)
    nsp_mul = np.prod(nsp)
    #print(nsp_mul)
    pred = 1 if sp_mul > nsp_mul else 0
    if pred == 0 and gt == 0:
        matrix['TN'] +=1
    elif pred == 1 and gt == 1:
        matrix['TP'] +=1
    elif pred == 0 and gt == 1:
        matrix['FN'] +=1
    elif pred == 1 and gt == 0:
        matrix['FP'] +=1
matrix
```

Out[123]:
```
{'TP': 0, 'FP': 0, 'TN': 34, 'FN': 23}
```

In [120]:
```
precision = matrix['TP']/(matrix['TP']+matrix['FP'])
print('Precision: ',precision)
# b. Recall
recall = matrix['TP']/(matrix['TP']+matrix['FN'])
print('Recall: ',recall)
# c. F-measure
F1 = (2*precision*recall)/(precision+recall)
print('F1: ',F1)
# d. Accuracy
acc = (matrix['TP']+matrix['TN'])/(matrix['TP']+matrix['TN']+matrix['FP']+m
print("Accuracy: ",acc)
```

```
---------------------------------------------------------------------------
--
ZeroDivisionError                         Traceback (most recent call las
t)
<ipython-input-120-3e01da6eddda> in <module>
----> 1 precision = matrix['TP']/(matrix['TP']+matrix['FP'])
      2 print('Precision: ',precision)
      3 # b. Recall
      4 recall = matrix['TP']/(matrix['TP']+matrix['FN'])
      5 print('Recall: ',recall)

ZeroDivisionError: division by zero
```

In [431]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
# fit the model
model.fit(train, Y_train)
model.score(test,Y_test)
```

Out[431]:
```
0.6227621483375959
```

5 Decision Trees

Let's train and test a Decision Tree to classify Spam or Not from the Spambase Dataset.

Write a script that:

1. Reads in the data.
2. Randomizes the data.
3. Selects the first 2/3 (round up) of the data for training and the remaining for testing
4. Standardizes the data (except for the last column of course) using the training data
5. Divides the training data into two groups: Spam samples, Non-Spam samples.
6. Trains a decision tree using the ID3 algorithm without any pruning.
7. Classify each testing sample using your trained decision tree.
8. Computes the following statistics using the testing data results:
   a. Precision

b. Recall
c. F-measure
d. Accuracy

Implementation Details

1. Seed the random number generate with zero prior to randomizing the data
2. Depending on your perspective, the features are either continuous or finite discretize. The lattercan be considered tru since the real-values are just the number of times a feature is observedin an email, normalized by some other count. That being said, for a decision tree we normallyuse categorical or discretized features. So for the purpose of this dataset, look at therange of each feature and turn them into binary features by choosing a threshold. I suggest using the median or mean.

In your report you will need:

1. The statistics requested for your Decision Tree classifier run.

Notes

Notes hand typed from Wikipedia; nearly verbatim

Algorithm

The ID3 - iterative dichotomiser 3 - is used to generate a decision tree from a dataset. ID3 is the precusor to C4.5 and is typically used in machine learning and natural language processing domains.

The ID3 algorithm begins witht the origianl set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy H(Y) or the information gain IG(S) of that attribute.

It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split or partitioned be the selected attribute to produce subsets of the data. For example, a node can be split into child nodes based on the subsets of the population whose ages are less than 50, between 50 and 100, and greater than 100). The algorithm continues to recurse on each subset, considering only attributes never slected before.

Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class; in which case the node is turned into a leaf node and labelled with the class of the examples.
- there are no more attributes to be selected, but the examples still do not belong to the same class. In this case, the node is made a leaf node and lablled with the most common class of the examples in the subset.
- ther are no examples in the subset, which happens when no example in the parent set was found to match a specific value of the selected attribute. An example could be the absence of a person among the population with age over 100 years. The leaf node is created and labelled with the most common class of the examples in the parent nodes set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node (internal node) representing the selected attribute on which the data was split, and terminal nodes (leaf nodes) representing the class label of the final subset of this branch.

Summary

1. Calculate the entropy of every attribute α of S.
2. Partition (split) the set S into subsets using the attribute for which the resulting entropy after splitting is minimized; or equivalenty, information gain is maximum.
3. Make a decision tree node containing that attribute.
4. Recurse on subsets using the remaining attributes.

Pseudocode

```
ID3 (Examples, Target_Attribute, Attributes)

    Create a root node for the tree

    If all examples are positive: Return the single-node tree Root
 with label = +
    If all examples are negative: Return the single-node tree Root
 with label = -
    If number of predicting attributes is empty: Return the single-
node tree Root

    If Root not None: label = most common value of the target attri
bute in the example

    Else If Root is None:

            A = the attribute the best classifies the examples
            Decision Tree attribute for Root = A

            For value $v_i$ in A:
                Add a new tree branch below Root, correspongin to t
he test A = $v_i$
                Let Examples($v_i$) be the subset of examples that
 have the value $v_i$ for A
                If Examples($v_i$) is empty:
                    Then below this new branch add a leaf node with
label = most common target value in the examples
                Else below this new branch add the subtree ID3(Exam
ples($v_i$), Target_attribute, Attributes - {A})

    End
    Return Root
```

Properties ID3 does not guarantee an optimal solution. It can converge on local optima. It uses a greedy strategy by selecting the locally best attribute to split the dataset on each iteration. The algorithms optimality can be improved by using backtracking during the search for the optimal decision tree at the cost of possibly taking longer.

ID3 can overfit the training data. To avoid overfitting, smaller decision trees should be preferred over larger ones. This algorithm usually produces small trees but it does not always produce the smallest tree possible.

ID3 is harder to use on continuous data than on factored data (factored data has a discrete number of possible values, thus reducing the possible barnch points). If the values of any given attribute are continuous, then there are many more places to split the data on this attribute, and searching for the best value to split on can be time consuming.

Usage ID3 is used by training on data set S to produce a decision tree which is stored in memory. At runtime, this decision tree is used to classify new test cases (feature vectors) by traversing the decision tree using the features of the datum to arrive at a leaf node. The class of this terminal node is the class the test case is classified as.

In [165]:
```
import pandas as pd
df1=pd.read_csv("spambase.data",header=None)
```

In [166]:
```
df1 = df1.sample(frac = 1)
```

In [167]:
```
train = df1.sample(frac=0.66, random_state=42)
test = df1.drop(train.index)
print(f"No. of training set: {train.shape[0]}")
print(f"No. of testing set: {test.shape[0]}")
```

```
No. of training set: 3037
No. of testing set: 1564
```

In [168]:
```
def Standardise(string1):
    m1=np.mean(string1)
    std1=np.std(string1)
    string2=(string1-m1)/std1
    return string2
```

In [169]:
```
Y_train=train[57]
train=train.drop([57],axis=1)
train=train.apply(Standardise)
train[57]=Y_train
```

In [173]:
```
def Change_value(string1):
    m1=np.mean(string1)
    i=0
    while i<string1.shape[0]:
        if string[i1<m1:
           string2=0
        else:
            string2=1
        i=i+1
    return string2
```

```
  File "<ipython-input-173-3fda9f158b7a>", line 6
    string2=0
           ^
SyntaxError: invalid syntax
```

In [174]:
```
train=train.apply(Change_value)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call las
t)
<ipython-input-174-8249aaf952f0> in <module>
----> 1 train=train.apply(Change_value)

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py in apply
(self, func, axis, raw, result_type, args, **kwds)
   7766                 kwds=kwds,
   7767             )
-> 7768         return op.get_result()
   7769 
   7770     def applymap(self, func, na_action: Optional[str] = None) ->
 DataFrame:

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/apply.py in get_r
esult(self)
    183             return self.apply_raw()
    184 
--> 185         return self.apply_standard()
    186 
    187     def apply_empty_result(self):

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/apply.py in apply
_standard(self)
    274 
    275     def apply_standard(self):
--> 276         results, res_index = self.apply_series_generator()
    277 
    278         # wrap results

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/apply.py in apply
_series_generator(self)
    288             for i, v in enumerate(series_gen):
    289                 # ignore SettingWithCopy here in case the user mu
tates
--> 290                 results[i] = self.f(v)
    291                 if isinstance(results[i], ABCSeries):
    292                     # If we have a view on v, we need to make a c
opy because

<ipython-input-36-835f1a61a045> in Change_value(string1)
      1 def Change_value(string1):
      2     m1=np.mean(string1)
----> 3     if string1<m1:
      4         string2=0
      5     else:

~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/generic.py in __n
onzero__(self)
   1440     @final
   1441     def __nonzero__(self):
-> 1442         raise ValueError(
   1443             f"The truth value of a {type(self).__name__} is ambig
```

```
uous. "
   1444              "Use a.empty, a.bool(), a.item(), a.any() or a.all
()."

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool
(), a.item(), a.any() or a.all().
```

In [120]:
```
df_zero = df1[df1[57] == 0]
index1 = df_zero.index
index1.name = "Non Spam Samples"
df_one = df1[df1[57]==1]
index = df_one.index
index.name = "Non Spam Samples"
```

In [121]:
```
df_zero
```

Out[121]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52
Non Spam Samples																
1813	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.022	0.022	0.019	0.022	0.022
1814	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.85	...	0.299	0.000	0.000	0.149	0.000
1815	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.000	0.000	0.000	0.000	0.000
1816	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.000	0.131	0.000	0.262	0.000
1817	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.000	0.104	0.324	0.000	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4596	0.31	0.0	0.62	0.0	0.00	0.31	0.0	0.0	0.0	0.00	...	0.000	0.232	0.000	0.000	0.000
4597	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.000	0.000	0.000	0.353	0.000
4598	0.30	0.0	0.30	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.102	0.718	0.000	0.000	0.000
4599	0.96	0.0	0.00	0.0	0.32	0.00	0.0	0.0	0.0	0.00	...	0.000	0.057	0.000	0.000	0.000
4600	0.00	0.0	0.65	0.0	0.00	0.00	0.0	0.0	0.0	0.00	...	0.000	0.000	0.000	0.125	0.000

2788 rows × 58 columns

In [122]:
```
df_one
```

Out[122]:

Non Spam Samples	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.000	0.778	0
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.000	0.132	0.000	0.372	0
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.010	0.143	0.000	0.276	0
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.137	0.000	0.137	0
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.135	0.000	0.135	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1808	0.00	0.00	0.00	0.0	0.00	0.23	0.00	0.00	0.00	0.00	...	0.077	0.038	0.000	0.000	0
1809	0.39	0.00	0.00	0.0	0.00	0.39	0.79	0.00	0.00	0.39	...	0.000	0.064	0.000	0.640	0
1810	0.00	0.00	0.77	0.0	0.38	0.38	0.38	0.00	0.00	0.77	...	0.063	0.127	0.255	0.510	0
1811	0.00	0.00	0.00	0.0	0.53	0.00	0.53	0.00	0.53	0.00	...	0.000	0.000	0.000	0.082	0
1812	0.00	0.31	0.42	0.0	0.00	0.10	0.00	0.52	0.21	0.52	...	0.000	0.016	0.000	0.887	0

1813 rows × 58 columns

In [ ]: