**Information Retrieval - CS 7800**

**Assignment-2**

Sai Vijay Kumar Surineela Manoj Kumar Reddy Avula Prabhu Charan Murari

U01096171 U01067535 U01099304

surineela.2@wright.edu avula.25@wright.edu murari.4@wright.edu

1. There are two major classifiers of Naive Bayes multinomial and Bernoulli in this assignment we are going with Bernoulli.
2. In Naive Bayes Multivariate Bernoulli classifier, the term frequency doesn't matter all that matters is whether the term is existing or not. it is similar to binary vector space model.
3. We developed Manual Naive Bayes Multivariate Bernoulli classifier according to the Pseudo code provided and Scikit-learn Naive Bayes Multivariate Bernoulli classifier.
4. Load text and Transpose from numpy library were also used in the development.

Code used:-

import numpy as np

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.naive\_bayes import BernoulliNB

trainData = "trainMatrixModified.txt"

testData = "testMatrixModified.txt"

trainClassData = "trainClasses.txt"

testClassData = "testClasses.txt"

trainNPArray = np.loadtxt(trainData)

testNPArray = np.loadtxt(testData)

trainClasses = np.loadtxt(trainClassData)[:, 1]

testClasses = np.loadtxt(testClassData)[:, 1]

testClasses = testClasses.astype(int)

trainClasses = trainClasses.astype(int)

def trainBernoulliNB(training, classes):

N = len(classes)

N1 = len(np.nonzero(classes)[0])

N0 = N - N1

priorProb = [N0/N, N1/N]

condProb = []

classes = list(classes)

for term in training:

d0t = 0

d1t = 0

for index, tf in enumerate(term):

if tf != 0:

if ((classes[index]) == 0):

d0t += 1

else:

d1t += 1

prob = [(d0t + 1)/(N0 + 2), (d1t + 1)/(N1 + 2)]

condProb.append(prob)

return priorProb, condProb

def applyBernoulliNB(prior, condProb, testclass):

score = np.log10(prior)

score0 = score[0]

score1 = score[1]

transposeTestClass = np.transpose(testclass)

res = []

for doc in transposeTestClass:

for index, term in enumerate(doc):

if term != 0:

score0 += np.log10(condProb[index][0])

score1 += np.log10(condProb[index][1])

else:

score0 += np.log10(1 - condProb[index][0])

score1 += np.log10(1 - condProb[index][1])

finalscore = [score0, score1]

c = np.argmax(finalscore)

res.append(c)

return np.array(res)

priorPob, condProb = trainBernoulliNB(trainNPArray, trainClasses)

prediction = applyBernoulliNB(priorPob, condProb, testNPArray)

def evaluationMat(act, pred, fName):

tn = np.sum((act == 1) & (pred == 1))

fn = np.sum((act == 0) & (pred == 1))

tp = np.sum((act == 0) & (pred == 0))

fp = np.sum((act == 1) & (pred == 0))

tottal = tp+tn+fn+fp

prec = tp/(tp+fp)

rec = tp/(tp+fn)

f1 = (2\*prec\*rec)/(prec + rec)

acc = (tp + tn)/tottal

cm = np.array([[tp, fp], [fn, tn]])

name = f"{fName}.txt"

with open(name, 'w') as fileContent:

fileContent.write(f"Accuracy: {acc:.2f}\n")

fileContent.write(f"Precision: {prec:.2f}\n")

fileContent.write(f"Recall: {rec:.2f}\n")

fileContent.write(f"F-1 score: {f1:.2f}\n")

fileContent.write(f"Confusion matrix:\n")

fileContent.write(f"{cm[0][0]} {cm[0][1]}\n{cm[1][0]} {cm[1][1]}")

fileContent.close()

matrixDisplay = metrics.ConfusionMatrixDisplay(confusion\_matrix = cm, display\_labels = [0, 1])

matrixDisplay.plot()

plt.title(f'Confusion Matrix for {fName}')

plt.savefig(f'{fName}ConfusionMatrix.png')

evaluation = [prec, rec, f1, acc]

evaluation\_names = ['Precision', 'Recall', 'F1 Score', 'Accuracy']

plt.figure(figsize=(8, 4))

plt.plot(evaluation\_names, evaluation, 'o-', color='blue')

plt.title(f'Evaluation Metrics for {fName}')

plt.xlabel('Metric')

plt.ylabel('Value')

plt.ylim([0, 1])

plt.savefig(f'{fName}Evaluation.png')

evaluationMat(testClasses, prediction, "BNB\_manual")

bnb = BernoulliNB(alpha=1.0)

bnb.fit(trainNPArray.T, trainClasses)

skpred = bnb.predict(testNPArray.T)

evaluationMat(testClasses, skpred, "BNB\_scikit")