31/05/2021 **Mohd. Aftab Alam**
02013302717

EXPERIMENT 5

Aim :

Develop a Machine Learning Method to Classify your Incoming Mails.

Introduction :

Email filtering is an extremely useful and practical problem. Email has become a mainstream form of communication. We obtain useful information each day via email but there is problem of getting unwanted emails.

Many users receive numerous unwanted emails each day which is why a spam filter has been created. The spam filter needs to sort incoming mail into wanted and unwanted. This can be tricky because the filter could allow too much spam into the inbox or could label some legitimate emails as spam. Machine learning can help solve this problem. The email client can be trained to learn where to put each email.

A machine learning algorithm for email filtering will take in a set of labeled messages as the input and will output correct labels for the testing data.

Preprocessing :

Not all of the information in an email is necessary or even useful information. There is a lot of unrelated information that serves no purpose in classifying the email. This information can be eliminated from the email and not processed by the machine learning algorithm. The email header can be discarded. The user rarely sees or pays attention to this information anyhow so it can be ruled out.

Before attempting to eliminate things like stop words or meaningless words, certain sets of data need to be preserved. Email addresses are useful and shouldn't be split apart. The data needs to be tokenized.

Words that appear with a high frequency can be ignored since they will appear in almost all samples. Common words, most punctuation, pronouns, simple verbs and adverbs can be thrown out since they will appear in almost all of the data so they can be ignored. The word "the" will appear multiple times in each sample; however, it adds no value. The words thrown out make up a stop list which there are many predefined lists of stop words that can be used.

After stop words are discarded, the remaining words are stemmed. Stemming reduces words to their root word. This allows for better comparisons between words. If the words "running" and "runner" both appeared, they would be treated as two different features. By stemming, they are both reduced to "run" which allows them to be grouped together. Stemming helps the program to correctly classify an email. The algorithm must depend on multiple words to create a classification for the email.

Techniques :

There are many machine learning techniques available to filter emails. Many spam filtering techniques use text categorization methods to label the data. Machine learning can easily be applied to this problem.

The techniques that can be used are: **Naïve Bayes Classifier, *k*-Nearest Neighbors (KNN), Artificial Neural Networks (ANN),** and **Support Vector Machines (SVM).**

Implementing Naïve Bayes Classifier :

```
import csv

import random

import math


def loadCsv(filename):

lines = csv.reader(open(filename, "rb"))

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [float(x) for x in dataset[i]]

return dataset


def splitDataset(dataset, splitRatio):

trainSize = int(len(dataset) * splitRatio)

trainSet = []

copy = list(dataset)

while len(trainSet) < trainSize:

index = random.randrange(len(copy))

trainSet.append(copy.pop(index))

return [trainSet, copy]
```

Mohd. Aftab Alam
02013302717

```
def separateByClass(dataset):

separated = {}

for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated):

separated[vector[-1]] = []

\ separated[vector[-1]].append(vector)

return separated


def mean(numbers):

return sum(numbers)/float(len(numbers))


def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)


def summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)]

del summaries[-1]

return summaries


def summarizeByClass(dataset):

separated = separateByClass(dataset)

summaries = {}

for classValue, instances in separated.iteritems():

summaries[classValue] = summarize(instances)

return summaries
```

```
def calculateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))

return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent


def calculateClassProbabilities(summaries, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.iteritems():

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i]

x = inputVector[i]

probabilities[classValue] *= calculateProbability(x, mean, stdev)

return probabilities


def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.iteritems():

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel


def getPredictions(summaries, testSet):

predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i])

predictions.append(result)

return predictions
```

Mohd. Aftab Alam
02013302717

```
def getAccuracy(testSet, predictions):

correct = 0

for i in range(len(testSet)):

if testSet[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testSet))) * 100.0


def main():

filename = 'pima-indians-diabetes.data.csv'

splitRatio = 0.67

dataset = loadCsv(filename)

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('Split {0} rows into train={1} and test={2} rows').format(len(dataset), len(trainingSet), len(testSet))


# prepare model

summaries = summarizeByClass(trainingSet)


# test model

predictions = getPredictions(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)

print('Accuracy: {0}%').format(accuracy)


main()
```