

Name: Vikas Mane
Class: BE-B
Roll Number: B1921152

PRN Number: 72000291F

ASSIGNMENT 2

Problem Statement : Download Pima Indians Diabetes dataset. Use Naive Bayes" Algorithm for classification

- Load the data from CSV file and split it into training and test datasets.
- summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
- Classify samples from a test dataset and a summarized training dataset.

Libraries Required for Code Implementation :
Installation Steps for Library

```
install numpy
$sudo apt-get install python-numpy
check installation
$pip list
```

```
install scipy
$sudo apt-get install python-scipy
check installation
$pip list
```

```
install matplotlib
$sudo apt-get install python-matplotlib
check installation
$pip list
```

```
install pandas
$sudo apt-get install python2-pandas
Test installation...on python terminal run
>>> import pandas as pd
>>> pd.test()
```

```
install sklearn
$pip install -U scikit-learn --user
```

Theory :

The test problem we will use in this tutorial is the Pima Indians Diabetes problem.

This problem is comprised of 768 observations of medical details for Pima indians patents. The records describe instantaneous measurements taken from the patient

such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.

Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0).

This is a standard dataset that has been studied a lot in machine learning literature. A good prediction accuracy is 70%-76%.

Datasets for training, testing and validation of Algorithms :

In machine learning, the study and construction of algorithms that can learn from and make predictions on data is a common task. Such algorithms work by making data-driven predictions or decisions, through building a mathematical model from input data.

The data used to build the final model usually comes from multiple datasets. In particular, three data sets are commonly used in different stages of the creation of the model.

The model is initially fit on a **training dataset**, that is a set of examples used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model. The model (e.g. a neural net or a naive Bayes classifier) is trained on the training dataset using a supervised learning method (e.g. gradient descent or stochastic gradient descent). In practice, the training dataset often consist of pairs of an input vector and the corresponding *answer* vector or scalar, which is commonly denoted as the *target*. The current model is run with the training dataset and produces a result, which is then compared with the *target*, for each input vector in the training dataset. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the **validation dataset**. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyper parameters (e.g. the number of hidden units in a neural network). Validation datasets can be used for regularization by early stopping: stop training when the error on the validation dataset increases, as this is a sign of over fitting to the training dataset. This simple procedure is complicated in practice by the fact that the validation dataset's error may fluctuate during training, producing multiple local minima. This complication has led to the creation of many ad-hoc rules for deciding when over fitting has truly begun.

Finally, the **test dataset** is a dataset used to provide an unbiased evaluation of a *final* model fit on the training dataset.

NAIVE BAYES

What is Naive Bayes Classifier?

Have you ever wondered how your mail provider implements spam filtering or how

online news channels perform news text classification or even how companies perform sentiment analysis of their audience on social media? All of this and more are done through a machine learning algorithm called Naive Bayes Classifier. Named after Thomas Bayes from the 1700s who first coined this in the Western literature. Naive Bayes classifier works on the principle of conditional probability as given by the Bayes theorem.

Advantages of Naive Bayes Classifier

Listed below are six benefits of Naive Bayes Classifier.

- Very simple and easy to implement
- Needs less training data
- Handles both continuous and discrete data
- Highly scalable with the number of predictors and data points
- As it is fast, it can be used in real-time predictions
- Not sensitive to irrelevant features

Bayes Theorem

We will understand Bayes Theorem in detail from the points mentioned below.

- According to the Bayes model, the conditional probability $P(Y|X)$ can be calculated as:
$$P(Y|X) = P(X|Y)P(Y) / P(X)$$
- This means you have to estimate a very large number of $P(X|Y)$ probabilities for a relatively small vector space X .
- For example, for a Boolean Y and 30 possible Boolean attributes in the X vector, you will have to estimate 3 billion probabilities $P(X|Y)$.
- To make it practical, a Naïve Bayes classifier is used, which assumes conditional independence of $P(X)$ to each other, with a given value of Y .
- This reduces the number of probability estimates to $2*30=60$ in the above example.

Code

```
import csv
import random
import math
```

```
def loadCsv(filename):
    lines = csv.reader(open(filename, "r"))
    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]
```

```
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.items():
        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.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
```

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

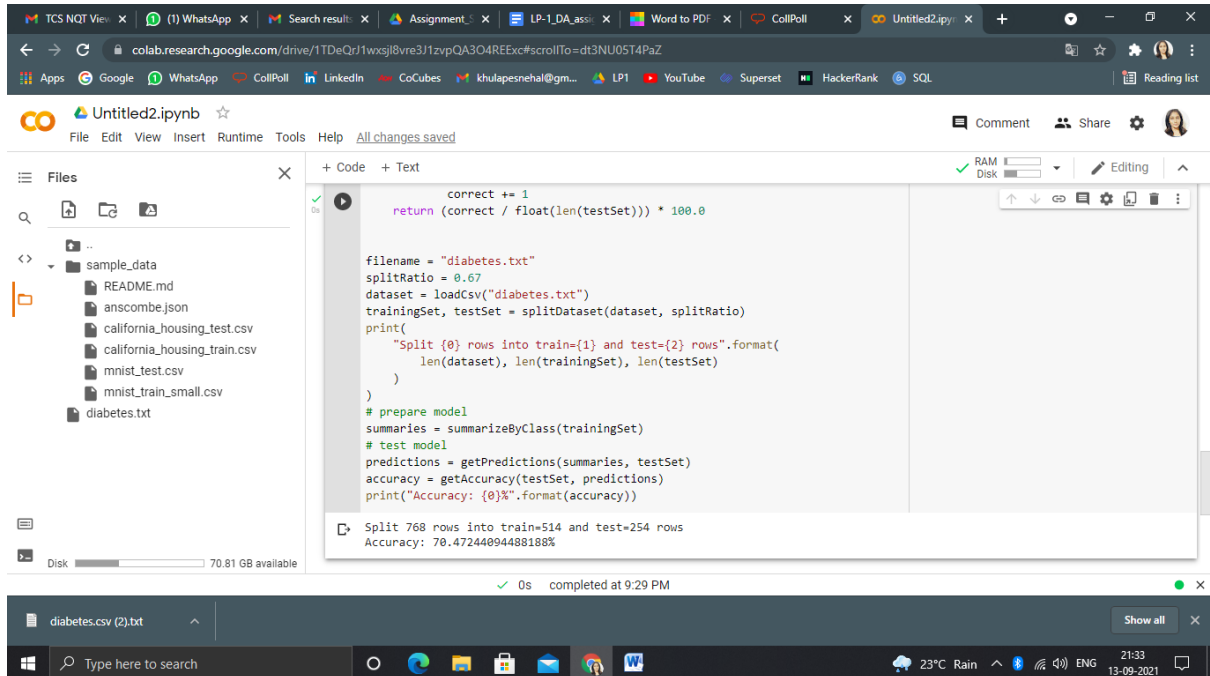
```
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            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
```

```
def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][0] == predictions[i]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0
```

```
filename = "diabetes.txt"
splitRatio = 0.67
dataset = loadCsv("diabetes.txt")
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))
```

Output :



```
correct += 1
return (correct / float(len(testSet))) * 100.0

filename = "diabetes.txt"
splitRatio = 0.67
dataset = loadCsv("diabetes.txt")
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))
```

Split 768 rows into train=514 and test=254 rows
Accuracy: 70.47244094488188%

Conclusion :

We learned to classification of data based on Bayes' probability theorem.