Olimov Bekhzod (올리모브 벡조드)

Kyungpook National University

Naive Bayes Classifier From Scratch in Python

In [1]:

```
# Import libraries
from csv import reader
from random import seed
from random import randrange
from math import sqrt
from math import exp
from math import pi

# Tool Functions ######################################################################
# Load a CSV file
def load_csv(filename):
 dataset = list()
 with open(filename, 'r') as file:
  csv_reader = reader(file)
  for row in csv_reader:
   if not row:
    continue
   dataset.append(row)
 return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
 for row in dataset:
  row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
 class_values = [row[column] for row in dataset]
 unique = set(class_values)
 lookup = dict()
 for i, value in enumerate(unique):
  lookup[value] = i
  print('[%s] => %d' % (value, i))
 for row in dataset:
  row[column] = lookup[row[column]]
 return lookup

# Calculate the mean of a list of numbers
def mean(numbers):
 return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
 avg = mean(numbers)
 variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
 return sqrt(variance)
```

Functions for training and testing

In [2]:

```
# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
 dataset_split = list()
 dataset_copy = list(dataset)
 fold_size = int(len(dataset) / n_folds)
 for _ in range(n_folds):
  fold = list()
  while len(fold) < fold_size:
   index = randrange(len(dataset_copy))
   fold.append(dataset_copy.pop(index))
```

```
   fold.append(dataset_copy.pop(index))
  dataset_split.append(fold)
 return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
 correct = 0
 for i in range(len(actual)):
  if actual[i] == predicted[i]:
   correct += 1
 return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
 folds = cross_validation_split(dataset, n_folds)
 scores = list()
 for fold in folds:
  train_set = list(folds)
  train_set.remove(fold)
  train_set = sum(train_set, [])
  test_set = list()
  for row in fold:
   row_copy = list(row)
   test_set.append(row_copy)
   row_copy[-1] = None
  predicted = algorithm(train_set, test_set, *args)
  actual = [row[-1] for row in fold]
  accuracy = accuracy_metric(actual, predicted)
  scores.append(accuracy)
 return scores
```

Functions for Naive Bayes Classification

In [3]:

```
# Split dataset by class then calculate statistics for each row
# Naive Bayes Algorithm
def naive_bayes(train, test):
    summary = summarize(train)
    predictions = list()
    for row in test:
        output = predict(summary, row)
        predictions.append(output)
    return(predictions)


# Calculate the Gaussian probability distribution function for x
def calculate_gaussian_prob(x, mean, stdev):

    gaussian_exp = exp(-(1/2) * ((x - mean) / (stdev)) ** 2)
    result = 1 / (stdev * sqrt(2 * pi)) * gaussian_exp

    return result


def summarize(dataset):

    # get data with separated classes
    data = {}
    for i in range(len(dataset)):
        sample = dataset[i]
        label = sample[-1]
        if (label not in data):
            data[label] = []
        data[label].append(sample)

    # create summary of the data
    summary = {}
    for labels, samples in data.items():
        stats = [(mean(feature), stdev(feature), len(feature)) for feature in zip(*samples)]
        # set statistics of each label into summary dictionary
        # we do not take the last column since it contains mean, std of the labels. They are
useless in our case.
        summary[labels] = stats[:-1]

    return summary
```

```python
    return summary


# Calculate the probabilities of predicting each class for a given row
def calculate_class_prob(summary, row):

    # get total num of samples. We need it to compute prior prob (p(c)) later
    num_samples = sum([summary[label][0][2] for label in summary])

    # probabilities computation
    probabilities = dict()
    for labels, stats in summary.items():
        # compute initial probabilities for each class (sum of samples belonging to one class / sum of total samples)
        probabilities[labels] = summary[labels][0][2]/float(num_samples)
        for index in range(len(stats)):
            # get mean and standard deviation for each feature
            mean, stdev, length = stats[index]
            # compute class probabilities given features:

            # P(C=0|X1,X2,X3,X4) = P(X1|C=0) * P(X2|C=0) * P(X3|C=0)* P(X4|C=0) * P(C=0))
            # P(C=1|X1,X2,X3,X4) = P(X1|C=1) * P(X2|C=1) * P(X3|C=1)* P(X4|C=1) * P(C=1))
            # P(C=2|X1,X2,X3,X4) = P(X1|C=2) * P(X2|C=2) * P(X3|C=2)* P(X4|C=2) * P(C=2))

            probabilities[labels] *= calculate_gaussian_prob(row[index], mean, stdev)

    return probabilities

# Predict the class for a given row
def predict(summary, row):

    # obtain calculated probabilities
    probabilities = calculate_class_prob(summary, row)
    best_label, best_label_prob = None, 0
    for label, prob in probabilities.items():
        if best_label is None or prob > best_label_prob:
            best_label_prob = prob
            best_label = label

    return best_label
```

Experiments

In [4]:

```python
def step1():
    print("Start step 1: Gaussian Probability Density Function")
    print(calculate_gaussian_prob(1.0, 1.0, 1.0))
    print(calculate_gaussian_prob(2.0, 1.0, 1.0))
    print(calculate_gaussian_prob(0.0, 1.0, 1.0))
    print("\n")

def step2():
    print("Start step 2: Class Probabilities")
    # Data set format = [feature1, Feature2, class]
    dataset = [[3.393533211,2.331273381,0],
    [3.110073483,1.781539638,0],
    [1.343808831,3.368360954,0],
    [3.582294042,4.67917911,0],
    [2.280362439,2.866990263,0],
    [7.423436942,4.696522875,1],
    [5.745051997,3.533989803,1],
    [9.172168622,2.511101045,1],
    [7.792783481,3.424088941,1],
    [7.939820817,0.791637231,1]]
    summary = summarize(dataset)
    probabilities = calculate_class_prob(summary, dataset[0])
    print(probabilities)
    print("\n")

def k_fold_cross_validation():
    print("Start Iris study: K-fold Cross Validation")
    # Test Naive Bayes on Iris Dataset
    seed(1)
    filename = 'iris.csv'
    dataset = load_csv(filename)
```

```
    for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
    # convert class column to integers
    str_column_to_int(dataset, len(dataset[0])-1)
    # evaluate algorithm
    n_folds = 5
    scores = evaluate_algorithm(dataset, naive_bayes, n_folds)
    print('Scores: %s' % scores)
    print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
    print("\n")

def iris_study():
    print("Start Iris study: Prediction")
    # Make a prediction with Naive Bayes on Iris Dataset
    filename = 'iris.csv'
    dataset = load_csv(filename)
    for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
    # convert class column to integers
    str_column_to_int(dataset, len(dataset[0])-1)
    # fit model
    model = summarize(dataset)
    # define a new record
    row = [5.7,2.9,4.2,1.3]
    # predict the label
    label = predict(model, row)
    print('Data=%s, Predicted: %s' % (row, label))
    print("\n")
```

Run

In [5]:

```
#step 1
step1()
#step 2
step2()
#Iris Flower Species Case Study
k_fold_cross_validation()
iris_study()
```

```
Start step 1: Gaussian Probability Density Function
0.3989422804014327
0.24197072451914337
0.24197072451914337


Start step 2: Class Probabilities
{0: 0.05032427673372074, 1: 0.00011557718379945776}


Start Iris study: K-fold Cross Validation
[Iris-setosa] => 0
[Iris-versicolor] => 1
[Iris-virginica] => 2
Scores: [93.33333333333333, 96.66666666666667, 100.0, 93.33333333333333, 93.33333333333333]
Mean Accuracy: 95.333%


Start Iris study: Prediction
[Iris-setosa] => 0
[Iris-versicolor] => 1
[Iris-virginica] => 2
Data=[5.7, 2.9, 4.2, 1.3], Predicted: 1
```