

CS498 AMO Homework 1

Team :

Minyuan Gu (minyuan3@illinois.edu)

Yanislav Shterev (shterev2@illinois.edu)

Problem 1: Diabetes Classification

We have been using the Pima Indians dataset to train a Naïve Bayes classifier to predict whether given patient has diabetes or not. The dataset contains 8 feature columns containing only continuous values and 1 label column having as values 1 (having diabetes) and 0 (negative diabetes results). There are total of 767 data points. We have used Normal (Gaussian) distribution for estimating the parameters. The classifier's final accuracy has been gathered by the average of 10 random splits the data having 80% for training and 20% for testing.

As a second part to the problem we examined what the effect of clearing the missing data (omitting feature values having 0) will be. For 4 of the feature vectors (attribute 3 (Diastolic blood pressure), attribute 4 (Triceps skinfold thickness), attribute 6 (Body mass index), and attribute 8 (Age)) we replaced the zeros with NaN and in the process of training, parameter prediction and predicting we skipped these features which values is NaN.

Part 1 Accuracies

The accuracies rounded up to the second decimal digit are as follow:

Setup	Cross-validation Accuracy
Unprocessed data	75.45
0-value elements ignored	73.03

Part 1 Code Snippets

1. Calculation of distribution parameters

```
def mean(records):
    return sum(records) / float(len(records))

def standard_deviation(average, records):
    variance = sum([pow(x - average, 2) for x in records]) / float(len(records) - 1)
    return math.sqrt(variance)
```

2. Calculation of naive Bayes predictions

```
# Source: https://stattrek.com/probability-distributions/normal.aspx
def estimate_posterior(x, mean, std):
    exponent = math.exp(-(math.pow(float(x) - mean, 2) / (2 * math.pow(std, 2))))
    return (1 / (math.sqrt(2 * math.pi) * std)) * exponent

def estimate_probabilities_per_class(distribution_parameters, test_vector):
    probabilities = {}
    for label, dist_values in distribution_parameters.items():
        probabilities[label] = 1 #default it to 1
        for i in range(len(dist_values)):
            mean, std = dist_values[i]
            X = test_vector[i]
            if math.isnan(X):
                continue
            probabilities[label] = probabilities[label] * estimate_posterior(X, mean, std)
    return probabilities

def predict(distribution_parameters, test_vector):
    probabilities = estimate_probabilities_per_class(distribution_parameters, test_vector)
    predicted_label = None
    max_probability = -1

    for label, probability in probabilities.items():
        if probability > max_probability:
            max_probability = probability
            predicted_label = label
    return predicted_label
```

3. Test-train split code

```
def split_dataframe(dataframe, train_ratio):
    dataframe = dataframe.iloc[np.random.permutation(len(dataframe))]

    trainset_len = int(len(dataframe)*train_ratio)
    trainset = dataframe.iloc[0:trainset_len, :]
    testset = dataframe.iloc[trainset_len:len(dataframe), :]

    return trainset, testset
```

Part 2 MNIST Accuracies

We used the MNIST dataset located at <https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat> to train Naïve Bayes classifier with Normal

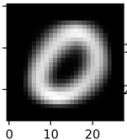
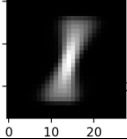
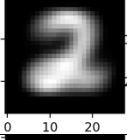
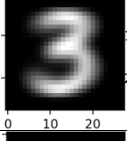
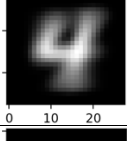
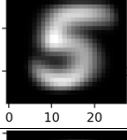
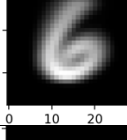
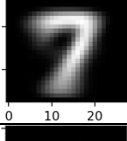
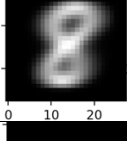
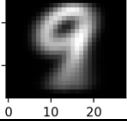
and Bernoulli distributions for estimating the parameters for the posterior probability. As comparison to this we also used the RandomForest classifier from the sklearn library to train and predict over the same dataset using different number of trees 10,30 and different depth 4, 16 as additional parameters.

For each of the classifiers above we used untouched images(no modification or data cleaning over the original image pixels) and bounded box stretched images(cleaned images in 20x20 dimensions) as input. The results show that models using cleaned images have better accuracy.

Results are as follows:

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	80.25%	79.62%
2	Gaussian + stretched	83.95%	83.82%
3	Bernoulli + untouched	83.85%	83.23%
4	Bernoulli + stretched	84.83%	83.15%
5	10 trees + 4 depth + untouched	71.02%	66.31%
6	10 trees + 4 depth + stretched	75.32%	72.01%
7	10 trees + 16 depth + untouched	93.88%	93.56%
8	10 trees + 16 depth + stretched	95.17%	94.50%
9	30 trees + 4 depth + untouched	77.84%	74.32%
10	30 trees + 4 depth + stretched	79.56%	74.17%
11	30 trees + 16 depth + untouched	95.36%	95.42%
12	30 trees + 16 depth + stretched	96.18%	96.04%

Part 2A Digit Images

Digit	Mean Image
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Part 2 Code

- Calculation of the Normal distribution parameters

```
def train(self, train_data, train_labels):
    self.digits = []
    self.p = []
    self.digits_mean = []
    self.digits_var = []
    for i in range(10):
        self.digits.append(train_data[train_labels[:] == i])
        self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])
        self.digits_mean.append(np.mean(self.digits[i], axis=0))
        self.digits_var.append(np.var(self.digits[i], axis=0))
    self.digits_var = np.array(self.digits_var)
    self.digits_var += self.epsilon * self.digits_var.max()
```

- Calculation of the Bernoulli distribution parameters

```
def train(self, train_data, train_labels):
    self.digits = []
    self.p = []
    self.digits_p_ink = []
    for i in range(10):
        self.digits.append(train_data[train_labels[:] == i])
        self.p.append(1.0 * self.digits[i].shape[0] / train_data.shape[0])
        p_ink = (np.sum(self.digits[i], axis=0) + 1) / (self.digits[i].shape[0] + train_data.shape[1])
        self.digits_p_ink.append(p_ink)
    self.digits_p_ink = np.array(self.digits_p_ink)
```

- Calculation of the Naive Bayes predictions

```
def predict(self, test_data):
    self.p_digit_class = []
    self.predicted = []
    if self.digits_mean == [] or self.digits_var == [] or self.p == []:
        print("Fit your model to training data first")
        return []

    for i in range(10):
        normpdf = norm.pdf(test_data, self.digits_mean[i], np.sqrt(self.digits_var[i]))
        p_post = np.sum(np.log(normpdf), axis=1) + np.log(self.p[i])
        self.p_digit_class.append(p_post)
    self.p_digit_class = np.array(self.p_digit_class)
    self.predicted = np.argmax(self.p_digit_class, axis=0)
    return self.predicted
```

- Training of a decision tree: Please see the full code in the end of the pdf.
- Calculation of a decision tree predictions: Please see the full code in the end of the pdf.

Full Code of Problem 1

```

# -*- coding: utf-8 -*-

import numpy as np
import math
import pandas as pd

#Generic enough to work for multiclass datasets
def NaiveBayes(dataset, n):
    accuracies = []
    for i in range(n):

        X_train, X_test = split_dataframe(dataset, 0.8)

        class_division = {}

        for index, feature_vector in X_train.iterrows():
            feature_vector=list(feature_vector)
            if feature_vector[8] not in class_division:
                class_division[feature_vector[8]] = []
            class_division[feature_vector[8]].append(feature_vector)

        mean_std_tuples = {}
        for label, instances in class_division.items():
            mean_std_tuples[label] = group_mean_std_tuples(instances)

        predictions = bulk_predict(mean_std_tuples, X_test)
        #print(predictions)
        accuracy = estimate_accuracy(X_test, predictions)
        accuracies.append(accuracy)
    return accuracies

def group_mean_std_tuples(dataset):
    mean_std_tuples = []
    for vector in zip(*dataset):
        vector = clear_nan_values(vector)
        average = mean(vector)
        std = standard_deviation(average, vector)
        mean_std_tuples.append((average,std))
    del mean_std_tuples[-1] #remove the last label column summaries
    return mean_std_tuples

def mean(records):
    return sum(records) / float(len(records))

def standard_deviation(average, records):
    variance = sum([pow(x - average, 2) for x in records]) / float(len(records) - 1)
    return math.sqrt(variance)

# choose y with largest sum(log(p(Xi|y))) + log(p(y))
# Normal distribution for number
#  $Y = \{ 1 / [ \sigma * \sqrt{2\pi} ] \} * e^{-(x - \mu)^2 / 2\sigma^2}$ 
# Source: https://stattrek.com/probability-distributions/normal.aspx
def estimate_posterior(x, mean, std):
    exponent = math.exp(-(math.pow(float(x) - mean, 2) / (2 * math.pow(std, 2))))
    return (1 / (math.sqrt(2 * math.pi) * std)) * exponent

# for every feature value and the corresponding mean and std use the respective distribution to
# caculate probabilities of X to belong to any of the chosen classes
def estimate_probabilities_per_class(distribution_parameters, test_vector):
    probabilities = {}

```

```

for label, dist_values in distribution_parameters.items():
    probabilities[label] = 1 #default it to 1
    for i in range(len(dist_values)):
        mean, std = dist_values[i]
        X = test_vector[i]
        if math.isnan(X):
            continue
        probabilities[label] = probabilities[label] * estimate_posterior(X, mean, std)
    return probabilities

#shuffle the dataframe and take the ration*data len for training and the rest for testing
def split_dataframe(dataframe, train_ratio):
    dataframe = dataframe.iloc[np.random.permutation(len(dataframe))]

    trainset_len = int(len(dataframe)*train_ratio)
    trainset = dataframe.iloc[0:trainset_len , :]
    testset = dataframe.iloc[trainset_len:len(dataframe) , :]

    return trainset,testset

def clear_nan_values(vector):
    new_vector = []
    for number in vector:
        if not math.isnan(number):
            new_vector.append(number)
    return new_vector

# For given test vector and estimated parameters find the class with the highest probability and assign it to this vector
def predict(distribution_parameters, test_vector):
    probabilities = estimate_probabilities_per_class(distribution_parameters, test_vector)
    predicted_label = None
    max_probability = -1

    for label, probability in probabilities.items():
        if probability > max_probability:
            max_probability = probability
            predicted_label = label
    return predicted_label

#Caclucates the prediction per every row in the test set based on trained data
def bulk_predict(distribution_parameters, test_set):
    predictions = [predict(distribution_parameters, list(feature_test_vector)) for index, feature_test_vector in
test_set.iterrows()]
    return predictions

# Estimate % of correct results
def estimate_accuracy(test_set, results):
    correct = 0
    count = 0
    for index, feature_test_vector in test_set.iterrows():
        feature_test_vector = clear_nan_values(feature_test_vector)
        if feature_test_vector[-1] == results[count]:
            correct += 1
        count = count+ 1
    return (correct / float(len(test_set))) * 100.0

# Part 1
pima_indians = pd.read_csv('pima-indians-diabetes.csv')

print(mean(NaiveBayes(pima_indians, 10)))

```

```
# Part 2
dataset_manipulated = pima_indians.values

for i in [2, 3, 5, 7]:
    for j in range(len(dataset_manipulated)):
        if dataset_manipulated[j][i] == 0:
            dataset_manipulated[j][i] = np.nan

result=NaiveBayes(pd.DataFrame(dataset_manipulated),10)
print(mean(result))

# Making the Confusion Matrix
#from sklearn.metrics import confusion_matrix
#cm = confusion_matrix(y_test, y_pred)
```

Full Code of Problem 2