Parametric Classification

CS554 – Introduction to Machine Learning and Artificial Neural Networks

Osman Furkan Kınlı – S002969

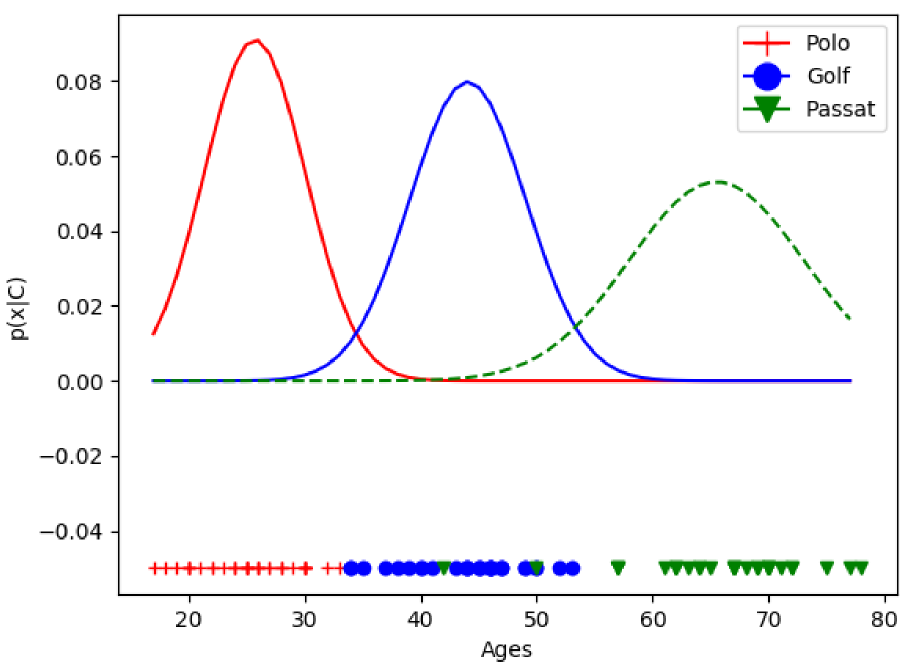
1. Introduction

In this project, we implement Bayesian classification task that estimates parametric Gaussian densities. The data has one-dimensional feature (the ages of the customers) and class labels ("Polo", "Golf", "Passat"). To achieve this, we first use the training data that have 100 unique samples to estimate the mean and variance for each class. Then, we calculate the class-conditional probabilities $p(x|C_i)$ and priors $p(x)$. According to Bayesian Rule formula, we calculate the posterior probability $p(C_i|x)$ of a sample for each class.

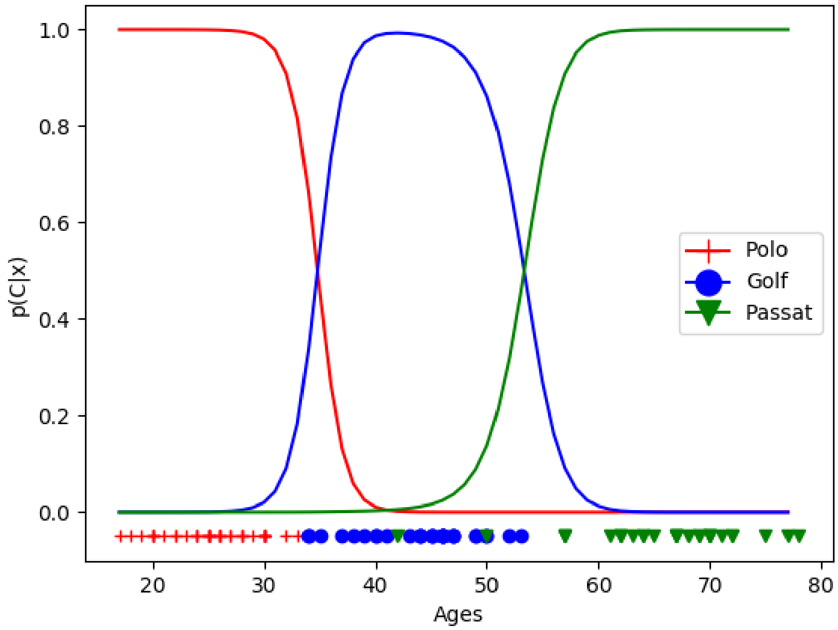
2. Results

The plots for estimated $p(x|C_i)$ and $p(C_i|x)$ and the data points can be seen as follows:

$p(x|C_i)$:

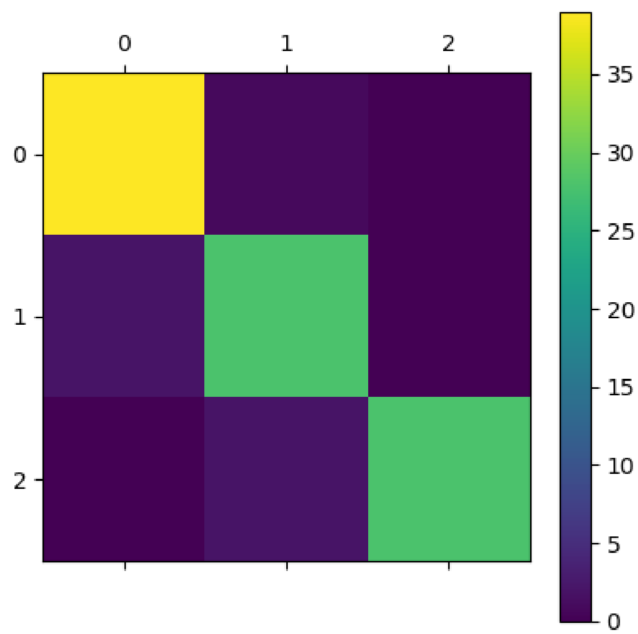


$p(C_i|x)$:

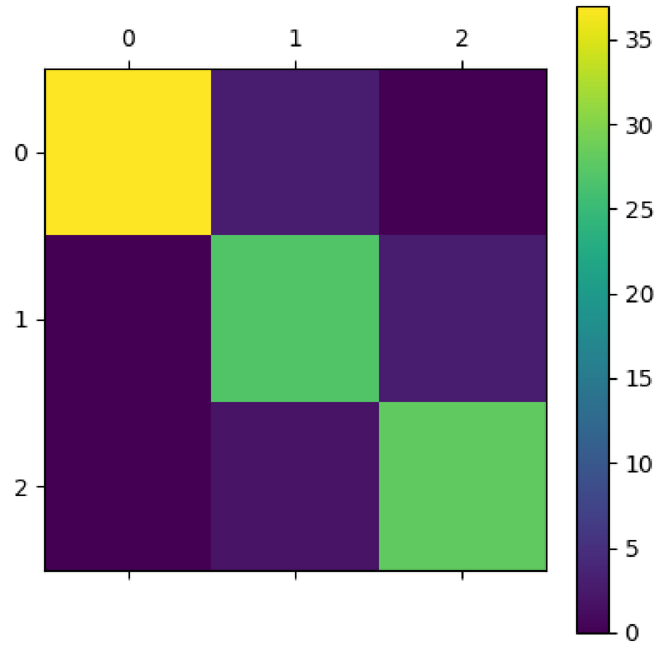


The accuracy on test set: **92%**

Confusion matrix for <u>training</u> set:



Confusion matrix for test set:



3. Code snippets

```
import csv
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from itertools import chain

# Initialization

train_fname = "training.csv"
test_fname = "testing.csv"

class_dict = {"Polo": 0, "Golf": 1, "Passat": 2}

data, test_data = dict(), list()
class_priors, means, variances = dict(), dict(), dict()
posteriors, likelihoods = dict(), dict()

for k in class_dict.keys():
    data[k] = list()
    posteriors[k] = list()
```

```python
    likelihoods[k] = list()

# Reading the data

with open(train_fname) as file:
    reader = csv.reader(file, delimiter=',')
    for i, r in enumerate(reader):
        if i > 0:
            data[r[1]].append(int(r[0]))

with open(test_fname) as file:
    reader = csv.reader(file, delimiter=",")
    for i, r in enumerate(reader):
        if i > 0:
            test_data.append((int(r[0]), r[1]))

# Calculating the relevant stuffs

for k in class_dict.keys():
    class_priors[k] = len(data[k]) / sum(len(lst) for lst in data.values())
    means[k] = sum(data[k]) / len(data[k])
    variances[k] = sum(list(map(lambda x: (x - means[k]) ** 2, data[k]))) /
len(data[k])

print("Data set mean: {}".format(sum(means.values()) / len(means.values())))
print("Data set variance: {}".format((sum(variances.values()) /
len(variances.values()))))
print("Class-based means: {}".format(means))
print("Class-based variances: {}".format(variances))


def calculate_posterior(x):
    likelihood_lst, prior_lst = list(), list()
    posteriors_dict = dict()
    for k_ in class_dict.keys():
        likelihood = (1. / math.sqrt(2 * math.pi * variances[k_])) * math.exp(-
((x - means[k_]) ** 2 / (2 * variances[k_])))
        likelihood_lst.append(likelihood)
        prior = class_priors[k_]
        prior_lst.append(prior)

    evidence = sum([l * p for l, p in zip(likelihood_lst, prior_lst)])

    for k_ in class_dict.keys():
        likelihood = (1. / math.sqrt(2 * math.pi * variances[k_])) * math.exp(-
((x - means[k_]) ** 2 / (2 * variances[k_])))
        prior = class_priors[k_]
        posteriors_dict[k_] = likelihood * prior / (evidence + 1e-7)

    return posteriors_dict
```

```python
def calculate_likelihood(x):
    likelihood_dict = dict()
    for k_ in class_dict.keys():
        likelihood = (1. / math.sqrt(2 * math.pi * variances[k_])) * math.exp(-
((x - means[k_]) ** 2 / (2 * variances[k_])))
        likelihood_dict[k_] = likelihood
    return likelihood_dict


def calculate_discriminant(x):
    discriminant = dict()
    for k_ in class_dict.keys():
        discriminant[k_] = math.log((1. / math.sqrt(2 * math.pi * variances[k_]))
*
                                    math.exp(-((x - means[k_]) ** 2 / (2 *
variances[k_])))) + math.log(class_priors[k_])
    return discriminant


# Evaluation

def predict(x):
    return np.argmax(list(calculate_discriminant(x).values()))


def evaluate(dataset):
    tp = 0
    for x, y in dataset:
        if predict(x) == class_dict[y]:
            tp += 1
    accuracy = tp / len(test_data)
    print("Accuracy: {}%".format(accuracy * 100))


print(evaluate(test_data))


# Confusion matrix

def compute_confusion_matrix(y_true, y_pred):
    K = len(np.unique(y_true))
    result = np.zeros((K, K))

    for i in range(len(y_true)):
        result[y_true[i]][y_pred[i]] += 1

    return result


train_data = [(v_, k) for k, v in data.items() for v_ in v]
train_cm = compute_confusion_matrix([class_dict[y] for x, y in train_data],
                                    [predict(x) for x, y in train_data])
```

```python
print("Confusion matrix for training data: \n{}".format(train_cm))
plt.matshow(train_cm)
plt.colorbar()
# plt.savefig('training_cm.png')
plt.show()
test_cm = compute_confusion_matrix([class_dict[y] for x, y in test_data],
                                    [predict(x) for x, y in test_data])
plt.matshow(test_cm)
plt.colorbar()
# plt.savefig('test_cm.png')
plt.show()
print("Confusion matrix for test data: \n{}".format(test_cm))


# Plotting

ages = dict()

for k in class_dict.keys():
    ages[k] = sorted([x for x, y in train_data if k == y])

class_color_dict = {"Polo": 'r+', "Golf": "bo", "Passat": "gv"}
class_line_dict = {"Polo": 'r-', "Golf": "b-", "Passat": "g-"}

fig_1 = plt.figure()

for k in class_dict.keys():
    plot_x = data[k]
    plot_y = [-0.05 for _ in range(len(data[k]))]
    ax = fig_1.add_subplot(111)
    ax.plot(plot_x, plot_y, class_color_dict[k])

for k in class_dict.keys():
    plot_x = range(min(set(chain(*data.values()))),
max(set(chain(*data.values()))))
    plot_y = [calculate_posterior(x)[k] for x in
range(min(set(chain(*data.values()))),

max(set(chain(*data.values()))))]
    ax = fig_1.add_subplot(111)
    ax.plot(plot_x, plot_y, class_line_dict[k])

plt.xlabel('Ages')
plt.ylabel('p(C|x)')
plt.legend(handles=[mlines.Line2D([], [], color='r', marker='+',
                                  markersize=12, label='Polo'),
                    mlines.Line2D([], [], color='b', marker='o',
                                  markersize=12, label='Golf'),
                    mlines.Line2D([], [], color='g', marker='v',
                                  markersize=12, label='Passat')])
# plt.savefig('posterior_graph.png')
plt.show()
```

```
class_line_dict = {"Polo": 'r-', "Golf": "b-", "Passat": "g--"}

fig_2 = plt.figure()

for k in class_dict.keys():
    plot_x = data[k]
    plot_y = [-0.05 for _ in range(len(data[k]))]
    ax = fig_2.add_subplot(111)
    ax.plot(plot_x, plot_y, class_color_dict[k])

for k in class_dict.keys():
    plot_x = range(min(set(chain(*data.values()))),
max(set(chain(*data.values()))))
    plot_y = [calculate_likelihood(x)[k] for x in
range(min(set(chain(*data.values()))),

max(set(chain(*data.values()))))]
    ax = fig_2.add_subplot(111)
    ax.plot(plot_x, plot_y, class_line_dict[k])

plt.xlabel('Ages')
plt.ylabel('p(x|C)')
plt.legend(handles=[mlines.Line2D([], [], color='r', marker='+',
                                  markersize=12, label='Polo'),
                    mlines.Line2D([], [], color='b', marker='o',
                                  markersize=12, label='Golf'),
                    mlines.Line2D([], [], color='g', marker='v',
                                  markersize=12, label='Passat')])
# plt.savefig('likelihood_graph.png')
plt.show()
```