# HW01_Q1-Q2_SVW2112

September 27, 2022

### 0.0.1 Problem 1

Let's start with a two class classification problem. You are asked to

1. Create your own dataset using real data. This data can be found by you online or gathered and measured by you. You are not to use datasets that others have put together, instead you are looking for raw data for a two class classification problem. The data does not have to come from images, but it is ok if it does. The data should contain 100 or more samples (more is better) from each class and the features should be at least 2 dimensional. It is ok to use Python libs, e.g. OpenCV, to extract features from your raw data. And it is ok if the labels for the classes are given by the data, or you can annotate this data yourself.
2. Divide this data up into a training set (80%) and a test set (20%). Experiment with plotting this data in some of the raw feature dimensions much like we did in class. Hand-draw possible classifiers in these plots.
3. Compute the prior for both classes from the data you gathered. Or if you do not believe you can determine this, assume both classes are equally likely. If you only have a couple of features, you can try to fit to a multivariate Gaussian class conditional density functions– this simply means you need to find the sample mean and sample covariance matrix for both classes from the samples. Alternatively, you can use a naive Bayes algorithm that assumes each feature is independent. But again assume a Gaussian distribution for each and find the joint probability as the product of the per feature probabilities. Plot the decision boundary for a minimum error-rate classifier assuming your fits are correct. Are these good fits to your data? How does this decision boundary differ from the one you drew by hand?

To make things more consistent, let's all use the same notation. Let the class conditional density functions (or likelihoods) be given by $\rho(\mathbf{x}|y_i)$ where $\mathbf{x}$ is your feature and $y_i$ specifies the class. Let's also assume that the priors $P(y_i)$ are the same for both classes. Hint: you can find the decision boundary by densely generating sample points $\mathbf{x}_j \in X$ according to your ccd's, classifying them, and then plotting and coloring them according to their predicted label.

Remember all you need to do is to use Bayes theorem to get the expression for ( | ) , then for each sample choose the with the highest aposteriori probability. Do not use any statisical packages to do this other than numpy to fit to your Gaussian distribution(s).

---

README:

For this exercise, I have chosen two separate datasets - the first is an image dataset comprised of horses and flowers that has been processed and converted to a CSV file containing 412 entries, the two features are image lightness and levels of redness; the second dataset includes BMI and

Glucose data as the two features from a larger diagnostics dataset observing more than 700 diabetic patients, which did not require CSV conversion.

The image dataset was sourced through Google images.

The diabetes dataset was sourced from Kaggle for the sake of a supplemental excercise, the diabetes dataset was provided by the National Institute of Diabetes and Digestive and Kidney Diseases:

https://www.kaggle.com/datasets/mathchi/diabetes-data-set

All images and CSV files are included at the following Google Drive link, under individual folders titled "Image Dataset" and "Diabetes Dataset." All images collected are included in the Image Dataset folder, but only the contents included in the "Horses" and "Flowers" folders have been used for this exercise.

https://drive.google.com/drive/folders/1JgOW9_IlpV3x_8q9ftkPsHeGhd3NqVzY?usp=sharing

The hand-drawn classifiers are included in a separate pdf file as part of my complete submission.

Any pip install packages have been labeled in the Import Libraries cell.

The program compiles as instructed. Problem 02 is included below and separately as a pdf.

---

[26]:
```
# Import libraries
import numpy as np # pip install numpy
import math # pip install python-math
import csv  # pip install python-csv
import cv2 as cv # pip install opencv-python
import os, glob  # pip install glob2
import matplotlib.pyplot as plt # pip install matplotlib
import pandas as pd # pip install pandas
```

[27]:
```
# Process image dataset
# Diabetes dataset is already a CSV
pathRoot = '/Users/stephen/Desktop/DL LAB 01/Datasets/Converted Images'
classLabels = ['Flowers', 'Horses']

featRedness = []
featLightness = []
label = []

for k in range(len(classLabels)):
    for file in glob.glob(pathRoot + '/' + classLabels[k] + "/*.jpeg"):
        norm = np.zeros((128, 128))
        pic = cv.imread(file, cv.IMREAD_COLOR)

        featRedness.append(np.average(pic[:,:,2]))
        featLightness.append(np.average(pic))
        label.append(k)
```

```
# Write CSV
df = pd.DataFrame({"Lightness": featLightness, "Redness": featRedness, "Label":␣
↪label})
df.to_csv('dataset_flowers_horses.csv', index=False)
```

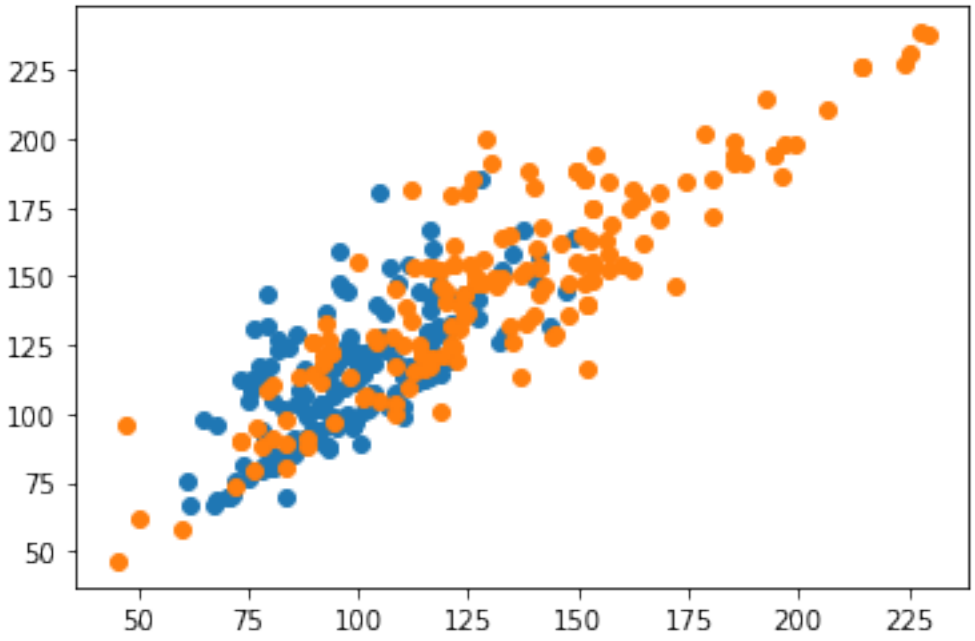
[28]:
```
# Plot features by label
def plot_scatter(df, x1_name, x2_name):
    plt.figure()
    plt.scatter(df.loc[df['Label'] == 0][x1_name], df.loc[df['Label'] ==␣
↪0][x2_name], label="0")
    plt.scatter(df.loc[df['Label'] == 1][x1_name], df.loc[df['Label'] ==␣
↪1][x2_name], label="1")

# Gaussian distribution
def get_likelihood(x, mu, sigma):
    x_mu = np.array([[x[0] - mu[0]], [x[1] - mu[1]]])
    inv = np.linalg.inv(sigma)
    Q = np.matmul(np.transpose(x_mu), np.matmul(inv, x_mu))
    exponent = math.pow(math.e, -0.5 * Q)

    norm_const = 1.0 / ( math.pow((2*math.pi), float(len(x))/2) * math.pow(np.
↪linalg.det(sigma), 0.5) )
    return norm_const * exponent

def classify(csv_path):

    # Create dataset
    x1 = []
    x2 = []
    label = []

    with open(csv_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            if line_count == 0:
                x1_name = str(row[0])
                x2_name = str(row[1])
            else:
                x1.append(float(row[0]))
                x2.append(float(row[1]))
                label.append(int(row[2]))

            # Next line
            line_count += 1

    df = pd.DataFrame({x1_name: x1, x2_name: x2, "Label": label})
```

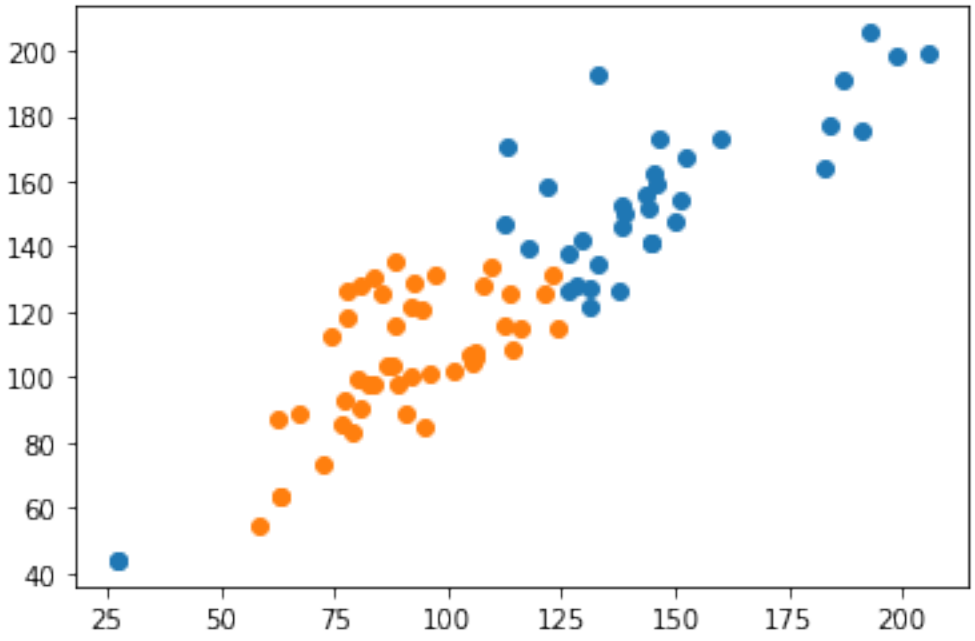
```
# Split data into training and test
sample_random_state = 200  # Use None for random
df_train = df.sample(frac = 0.8, random_state = sample_random_state)
df_test = df.drop(df_train.index)
plot_scatter(df_train, x1_name, x2_name)

# Means, standard deviations, and counts
x1_mean = []
x1_std = []
x2_mean = []
x2_std = []
count = []

for k in range(2):
    x1_class = df_train.loc[df_train['Label'] == k][x1_name]
    x2_class = df_train.loc[df_train['Label'] == k][x2_name]

    x1_mean.append(np.average(x1_class))
    x2_mean.append(np.average(x2_class))

    x1_std.append(np.std(x1_class, ddof = 1))
    x2_std.append(np.std(x2_class, ddof = 1))

    count.append(len(x1_class))

# Prior probabilities
prior_prob = []
for k in range(2):
    prior_prob.append(count[k] / np.sum(count))

# Loop over all rows in the test set
n_total = 0
n_correct = 0

x1 = []
x2 = []
label_pred = []

for index, row in df_test.iterrows():
    x = [row[0], row[1]]
    label_true = row[2]

    # Get likelihoods and probability of evidence
    L = []
    prob_evidence = 0
    for k in range(2):
```

```
            mu = [x1_mean[k], x2_mean[k]]
            sigma = np.array([[x1_std[k] ** 2, 0], [0, x2_std[k] ** 2]])
            L_class = get_likelihood(x, mu, sigma)
            L.append(L_class)
            prob_evidence += L_class * prior_prob[k]

        # Get posterior probabilities
        posterior_prob = []
        for k in range(2):
            posterior_prob.append(L[k] * prior_prob[k] / prob_evidence)

        # Make the prediction
        if posterior_prob[0] <= posterior_prob[1]:
            label_pred_value = 0
        else:
            label_pred_value = 1

        # Append results
        x1.append(row[0])
        x2.append(row[1])
        label_pred.append(label_pred_value)

        # Track accuracy
        n_total += 1
        if (label_pred_value == label_true):
            n_correct += 1

    # Assemble into dataframe
    df_pred = pd.DataFrame({x1_name: x1, x2_name: x2, "Label": label_pred})
    plot_scatter(df_pred, x1_name, x2_name)

    # Print accuracy
    acc = 100 * n_correct / n_total
    print(f"Accuracy: {acc:.1f}%")
```

[29]:
```
# Image Dataset
classify("/Users/stephen/Desktop/DL LAB 01/Datasets/dataset_flowers_horses.csv")
```

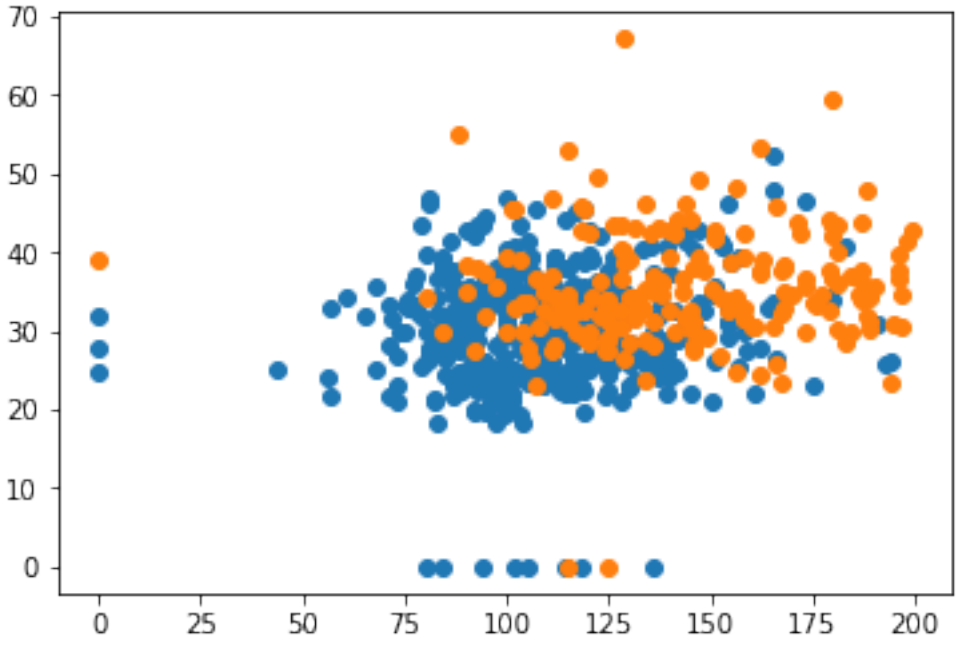
```
Accuracy: 22.0%
```

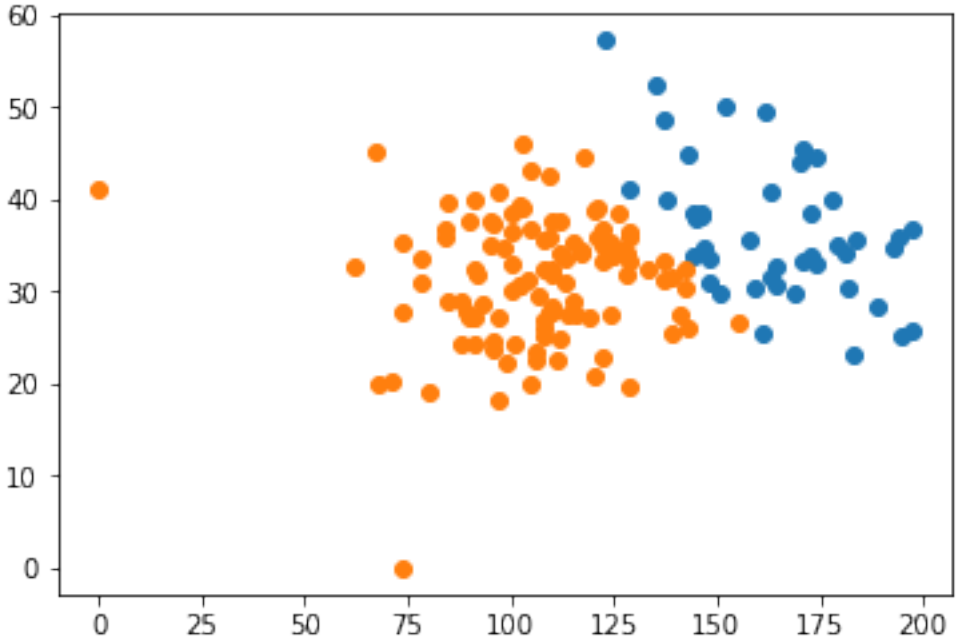



[30]:
```
# Diabetes Dataset
classify("/Users/stephen/Desktop/DL LAB 01/Datasets/dataset_diabetes.csv")
```

```
Accuracy: 27.3%
```




### 0.0.2 Problem 2

Research an application for computer vision, describe the underlying problem and the methods used to solve it. If the application was presented in popular media (e.g., NY Times), track down

the scientific methods used in the academic literature. Include references. It is ok if this is related to what you are thinking about for your final project! This write-up should be a two to three pages including images and diagrams as needed.

---

Deep learning technology has been successfully developed and deployed in computer vision applications in a multitude of ways across major industries that includes retail, manufacturing, agriculture, farming, smart cities, education, healthcare, energy, automotive, insurance, sports, aviation, and construction. The intention of these many applications typically falls under the categories of efficiency, effectiveness, and safety improvements in each respective sector. For the sake of this course, it is worth defining what we mean by Deep Learning and Computer Vision in order to correctly identify applications that are of particular interest to our immediate training and research goals.

Within the broader field of Artificial Intelligence is the imbedded subtopic, Machine Learning. And where Machine Learning is a technique by which a computer learns from and draws inference from patterns in data without requiring specific instructions, Deep Learning is a type of Machine Learning based on artificial neural networks that seeks to mimic the human brain through multiple processing layers that extract higher level features from data so that a computer can learn in a manner that is natural to humans. The link between Computer Vision and Deep Learning is somewhat imprecise, but Computer Vision is another subset of Machine Learning and an interdisciplinary field of Artificial Intelligence that enables computers to process, analyze, and interpret the visual world. Traditional applications of computer vision are solved by invoking Deep Learning methods in order to assist computers or machines in understanding the actions, behaviors, and languages of human beings.

Nevertheless, the most common applications of Computer Vision include defect detection, image labeling, facial recognition, object detection, image classification, object tracking, movement analysis, and cell classification. And the top Deep Learning applications are self-driving cars, natural language processing, visual recognition, image and search recognition, virtual assistants, chatbots, and fraud detection. For this assignment and likely for my broader research interests this semester, I would like to focus on the Deep Learning and Computer Vision applications specific to self-driving cars, which includes defect detection, image labeling, facial recognition, object detection, image classification, object tracking, and movement analysis – the most complex and biggest topic currently concerning Computer Vision.

The first work on computer image processing started in 1957, as two neurophysiologists became interested in how the brain interprets visual stimuli. It wasn't until 2012, when a neural network called AlexNet won a visual recognition competition and paved the way for neural networks to dominate the field of Artificial Intelligence. Much earlier in 1994, however, the first fully autonomous vehicle demonstration took place in Paris, and later in 1995 a team from Carnegie Mellon drove a minivan from Pittsburgh to San Diego without manually steering. Today, Tesla is regarded as the pioneer of autonomous driving, offering various self-driving packages powered by an 8-camera system that gives the vehicle a 360-degree view of its surroundings up to 250-meters away. But despite the rapid growth of Computer Vision in the automotive industry and groundbreaking developments in Advanced Driver Assistance Systems, there are still no fully-autonomous vehicles on the road today. While there are certainly many issues at the policy and infrastructure level, the primary roadblock holding back fully autonomous vehicles is the lack of reliable object detection models – specifically, current object detection models are typically trained using datasets and static imagery that presents a significant learning curve when researchers attempt to introduce the same

models to vehicle-generated video feeds and real-time imagery.

Human beings enter their vehicles every day and experience a variety of circumstances, both expected and unexpected or completely new in real-time, enroute to their final destination. The experience and intelligence of human beings allows them to carefully navigate these potential threats and obstacles in a manner that is relatively effective and safe. This pattern of behavior, of course, is incongruent and far more advanced than the level of sophistication that autonomous vehicles are capable. And while there are currently many known faults and limitations of machine vision technologies that observe the environment, such as LIDAR that is prone to failure in inclement weather or poor lighting, we can instead define the overarching difficulty in developing fully autonomous vehicles as something concerned with how the Artificial Intelligence interacts with the Computer Vision signals. For example, models are often developed within constrained environments – meaning, a particular locale that is being observed repeatedly in order to train a model and its vehicle within only that environment. This is problematic, of course, because there is no convergence on the basic approach of machine vision and once the vehicle exits its geographically defined region it will enter a new, unconstrained environment that it cannot effectively process and adapt to as quickly.

Other immediate machine vision problems include the unsolved problems of validation, the many limitations of LIDAR that have given way to hybrid approaches pairing cameras or radar with LIDAR technologies to develop more optimized vision processing strategies, and the lack of agreement around how vertically integrated vision systems should be or the broader openness and collaboration of the automotive industry and vision researchers. One particular problem that I am interested in is the applications of hyperdimensional computing (vector symbolic architectures) and its applications in computer vision and pattern recognition, specifically in how image data is processed in a more sophisticated way of encapsulating spatial information of multiple objects in semantic vectors of fixed length through the exponentiation of high-dimensional vectors based on the binding process that is specific to vector symbolic architectures. The basic idea behind this family of cognitive modeling is to represent structures, concepts, symbols, or language in a high-dimensional vector space by mapping each entity to be represented by a vector, which can be manipulated through algebraic operations. Why does this matter to Computer Vision and autonomous driving? Because many image and video descriptors are numerical vectors with a potentially very large number of dimensions that ordinarily require considerable memory consumption or time for comparison and processing, and hyperdimensional computing offers a new approach to systematically combine information from a set of vectors in a single vector of the same dimensionality to process descriptors together with simple and fast vector operations with a 20% performance improvement and more than two times the better worst-case performance compared to the next best solution.

Work Cited:

Xiao et al., 2017. L. Xiao, R. Wang, B. Dai, Y. Fang, D. Liu, T. Wu; Hybrid Conditional Random Field Based Camera-Lidar Fusion for Road Detection; Inf. Sci., 432 (2017), pp. 543-558, 10.1016/j.ins.2017.04.048

Neubert et al., 2021. S. Schubert; Hyperdimensional Computing as a Framework for Systematic Aggregation of Image Descriptors; IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 16938-16947

Schlegel et al., 2021. P. Neubert, P. Protzel; A Comparison of Vector Symbolic Architectures; Artif Intell Rev 55, 4523–4555 (2022)

Mirus et al., 2019. P. Blouw, TC. Stewart, J. Conradt; An Investigation of Vehicle Behavior Prediction Using a Vector Power Representation to Encode Spatial Positions of Multiple Objects and Neural Networks; Frontiers in Neurorobotics, Vol 13; DOI=10.3389/fnbot.2019.00084

Glace, 2022. Autonomous Vehicles Are Driving Computer Vision Into the Future

Eglash, 2019. Progress Toward Safe and Reliable AI; Stanford University

Stetic, 2022. The History of Computer Vision and the Evolution of Autonomous Vehicles

Rizzoli, 2022. 27+ Most Popular Computer Vision Applications and Use Cases in 2022

Khillar, 2021. Difference Between Computer Vision and Deep Learning

Boesch, 2022. Top Applications of Computer Vision in the Automotive Industry

Meel, 2022. The 87 Most Popular Computer Vision Applications for 2023