114-1 Machine Learning
Week 6 Assignment

Ming Hsun Wu

October 14, 2025

Problem 1. Unanswered Questions

Nope!

Problem 2.

Classification Dataset

Format: (Longitude, Latitude, Label)

Rules:

- If the temperature observation value is an invalid value of -999, then label $= 0$.
- If the temperature observation value is valid, then label $= 1$.
- Use Gaussian Discriminant Analysis (GDA) to build a classification model.

Note of Problem 2.

Gaussian Discriminant Analysis (GDA) is a generative classification model that assumes data are generated from two Gaussian distributions:

$$p(x \mid y=0) = \mathcal{N}(\mu_0, \Sigma), \qquad p(x \mid y=1) = \mathcal{N}(\mu_1, \Sigma)$$

with class priors

$$p(y=1) = \phi, \qquad p(y=0) = 1-\phi$$

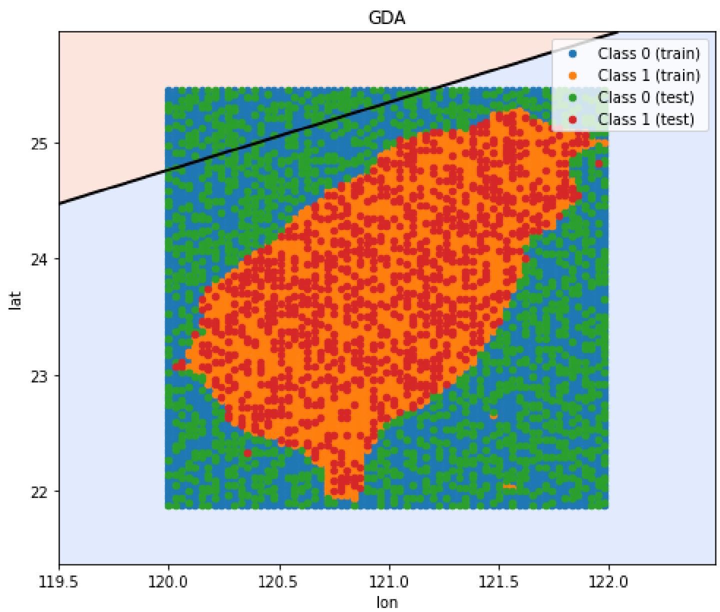
Using Bayes' theorem, we derive the posterior probability:

$$p(y=1 \mid x) = \frac{p(x \mid y=1)\, p(y=1)}{p(x \mid y=1)\, p(y=1) + p(x \mid y=0)\, p(y=0)}$$

The idea of Gaussian Discriminant Analysis (GDA) is to model the conditional density $p(x \mid y=k)$ for each class and use Bayes' theorem to compute $p(y \mid x)$.

In this dataset , GDA learns the mean centers and spread of each class and estimates the probability that a point belongs to a class under a Gaussian assumption. If the data distribution is approximately Gaussian, GDA can classify effectively. However, if the data structure is nonlinear , a linear boundary will not fit well.

The dataset was split into 70% training and 30% testing. Model performance was evaluated by accuracy on the test set. An accuracy close to **0.5** indicates that the linear boundary cannot effectively separate the two classes. I evaluated the test set performance using the mean of the predicted and true values, i.e., mean(pred, true).



```python
#%% source code
#%%
import json, re
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
#%%
'load data'
path = r"./O-A0038-003.json"
with open(path, "r", encoding="utf-8") as f:
    data = json.load(f)

content = data['cwaopendata']["dataset"]["Resource"]["Content"]

nums = re.findall(r'[-+]?\d+\.\d+E[+-]\d+', content)
vals = np.array([float(x) for x in nums], dtype=float)


ncols, nrows = 67, 120
grid = vals.reshape((nrows, ncols))

```

```
bl_lon, bl_lat = 120.0, 21.88
res_lon, res_lat = 0.03, 0.03


lon = bl_lon + np.arange(ncols) * res_lon
lat = bl_lat + np.arange(nrows) * res_lat
lon_grid, lat_grid = np.meshgrid(lon, lat)


lon_flat = lon_grid.ravel()
lat_flat = lat_grid.ravel()
val_flat = grid.ravel()


labels = np.where(val_flat == -999.0, 0, 1)
df_classification = pd.DataFrame({"lon": lon_flat, "lat":
    lat_flat, "label": labels})


mask = val_flat != -999.0
df_regression = pd.DataFrame({
    "lon": lon_flat[mask],
    "lat": lat_flat[mask],
    "value": val_flat[mask]
})


```

```
#%%
'data preprocessing'
x_cls = df_classification[["lon", "lat"]].values
y_cls = df_classification["label"].values

x_cls_train, x_cls_test, y_cls_train, y_cls_test =
    train_test_split(x_cls, y_cls, test_size=0.3, random_state=40)

x_reg = df_regression[["lon", "lat"]].values
y_reg = df_regression["value"].values

x_reg_train, x_reg_test, y_reg_train, y_reg_test =
    train_test_split(x_reg, y_reg, test_size=0.3, random_state=42)

#%%

phi = np.mean(y_cls_train)
mu0 = x_cls_train[y_cls_train == 0].mean(axis=0)
mu1 = x_cls_train[y_cls_train == 1].mean(axis=0)

# Conv matrix
X0 = x_cls_train[y_cls_train == 0]
X1 = x_cls_train[y_cls_train == 1]
Sigma = ((X0 - mu0).T @ (X0 - mu0) + (X1 - mu1).T @ (X1 - mu1)) /
    len(x_cls_train)

# Define predict function
```

```python
def predict_proba(X):
    invS = np.linalg.inv(Sigma)
    detS = np.linalg.det(Sigma)
    def gaussian(x, mu):
        diff = x - mu
        return np.exp(-0.5*diff@invS@diff.T) /
            np.sqrt((2*np.pi)**X.shape[1]*detS)
    probs = []
    for x in X:
        p1 = gaussian(x, mu1)*phi
        p0 = gaussian(x, mu0)*(1-phi)
        probs.append(p1/(p1+p0))
    return np.array(probs)

def predict(X):
    return (predict_proba(X) >= 0.5).astype(int)

# Evaluate
y_pred = predict(x_cls_test)
acc = np.mean(y_pred == y_cls_test)
```