NYU CS-GY 6923: Machine Learning Fall 2024

Homework 2

Name: Rohan Mahesh Gore (N19332535)

Problem 1. Impacts of Regularization

Collaborators: None.

This problem is related to ridge regularization, a technique which involves penalizing the parameters in order to prevent over-fitting of the ML model.

We have following least squares regression problems that have been ridge regularized :

1. For λ_1:
$$\beta_1^* = \arg\min_{\beta} \left(\|X\beta - y\|_2^2 + \lambda_1 \|\beta\|_2^2 \right)$$

2. For λ_2:
$$\beta_2^* = \arg\min_{\beta} \left(\|X\beta - y\|_2^2 + \lambda_2 \|\beta\|_2^2 \right)$$

The main condition we have over here is that: $\lambda_1 \geq \lambda_2$.

Before moving on to questions we need to dissect the objective function to easily solve the questions.

The objective function for ridge regression consists of two parts:
1. Data fitting term $\|X\beta - y\|_2^2$: This term encourages β to fit the data as well as possible.
2. Regularization term $\lambda\|\beta\|_2^2$: This penalizes large values of β to prevent over-fitting.

a. Proving that increasing penalty λ always increases the loss $\boldsymbol{\beta^*}$

We have to prove that if $\lambda_1 \geq \lambda_2$, then the norm of the optimal parameter vector $\|\boldsymbol{\beta}_1^*\|_2^2$ is less than or equal to $\|\boldsymbol{\beta}_2^*\|_2^2$ for the ridge regularized least squares regression problem.

Step 1: Framing **Inequality 1** for $\lambda = \lambda_1$

β_1^* is the minimizer of the objective function for λ_1. Hence, no other β, even including β_2^*, will give a lower value of the objective function for λ_1. Mathematically, this means:

$$\|X\beta_1^* - y\|_2^2 + \lambda_1 \|\beta_1^*\|_2^2 \leq \|X\beta_2^* - y\|_2^2 + \lambda_1 \|\beta_2^*\|_2^2$$

This is because β_1^* minimizes the objective function when $\lambda = \lambda_1$.

Step 2: Framing **Inequality 2** for $\lambda = \lambda_2$

Similarly, β_2^* is the minimizer of the objective function for λ_2 and no other β, including β_1^*, will give a lower value of the objective function for λ_2. Mathematically, this means:

$$\|X\beta_2^* - y\|_2^2 + \lambda_2\|\beta_2^*\|_2^2 \leq \|X\beta_1^* - y\|_2^2 + \lambda_2\|\beta_1^*\|_2^2$$

This is because β_2^* minimizes the objective function when $\lambda = \lambda_2$.

Step 3: Subtracting Inequality 2 from Inequality 1:

$$(\|X\beta_1^*-y\|_2^2+\lambda_1\|\beta_1^*\|_2^2)-(\|X\beta_2^*-y\|_2^2+\lambda_1\|\beta_2^*\|_2^2) \leq (\|X\beta_2^*-y\|_2^2+\lambda_2\|\beta_2^*\|_2^2)-(\|X\beta_1^*-y\|_2^2+\lambda_2\|\beta_1^*\|_2^2)$$

Simplifying:

$$(\lambda_1 - \lambda_2)\|\boldsymbol{\beta}_1^*\|_2^2 \leq (\lambda_1 - \lambda_2)\|\boldsymbol{\beta}_2^*\|_2^2$$

Therefore,

$$(\lambda_1 - \lambda_2)(\|\boldsymbol{\beta}_1^*\|_2^2 - \|\boldsymbol{\beta}_2^*\|_2^2) \leq 0$$

But we know that $\lambda_1 \geq \lambda_2$, therefore $(\lambda_1 - \lambda_2)$ should be greater than or equal to zero. Therefore, from the above inequality we can have

$$(\|\boldsymbol{\beta}_1^*\|_2^2 - \|\boldsymbol{\beta}_2^*\|_2^2) \leq 0$$

Therefore,

$$\|\boldsymbol{\beta}_1^*\|_2^2 \leq \|\boldsymbol{\beta}_2^*\|_2^2$$

This proves that the The norm of the optimal parameter vector decreases with increasing regularization.

Hence, Proved.

b. Proving that: increasing penalty λ always increases training loss

WE have to prove that, if $\lambda_1 \geq \lambda_2$, then the training loss for λ_1 is at least as large as that for λ_2.

That is: $\|X\beta_1^* - y\|_2^2 \geq \|X\beta_2^* - y\|_2^2$, given that $\lambda_1 \geq \lambda_2$.

This aligns with the knowledge gained from part(a) of the question that larger the λ, smaller the parameter vector β. The main part is that this happens potentially at the expense of fitting the training data less accurately for the model.

We can prove in a few steps

Step 1: Framing **Inequality 1** for $\lambda = \lambda_1$

β_1^* is the minimizer of the objective function for λ_1. Hence, no other β, even including β_2^*, will give a lower value of the objective function for λ_1. Mathematically, this means:

$$\|X\beta_1^* - y\|_2^2 + \lambda_1\|\beta_1^*\|_2^2 \leq \|X\beta_2^* - y\|_2^2 + \lambda_1\|\beta_2^*\|_2^2$$

This is because β_1^* minimizes the objective function when $\lambda = \lambda_1$.

Step 2: Framing **Inequality 2** for $\lambda = \lambda_2$

Similarly, β_2^* is the minimizer of the objective function for λ_2 and no other β, including β_1^*, will give a lower value of the objective function for λ_2. Mathematically, this means:

$$\|X\beta_2^* - y\|_2^2 + \lambda_2\|\beta_2^*\|_2^2 \leq \|X\beta_1^* - y\|_2^2 + \lambda_2\|\beta_1^*\|_2^2$$

This is because β_2^* minimizes the objective function when $\lambda = \lambda_2$.

Step 3: Subtracting Inequality 2 from Inequality 1:

$$(\|X\beta_1^*-y\|_2^2+\lambda_1\|\beta_1^*\|_2^2)-(\|X\beta_2^*-y\|_2^2+\lambda_1\|\beta_2^*\|_2^2) \leq (\|X\beta_2^*-y\|_2^2+\lambda_2\|\beta_2^*\|_2^2)-(\|X\beta_1^*-y\|_2^2+\lambda_2\|\beta_1^*\|_2^2)$$

Simplifying:

$$\|X\beta_1^* - y\|_2^2 - \|X\beta_2^* - y\|_2^2 + \lambda_1(\|\beta_1^*\|_2^2 - \|\beta_2^*\|_2^2) \leq \|X\beta_2^* - y\|_2^2 - \|X\beta_1^* - y\|_2^2 + \lambda_2(\|\beta_2^*\|_2^2 - \|\beta_1^*\|_2^2)$$

Rearranging terms:

$$2(\|X\beta_1^* - y\|_2^2 - \|X\beta_2^* - y\|_2^2) + (\lambda_1 - \lambda_2)(\|\beta_1^*\|_2^2 - \|\beta_2^*\|_2^2) \leq 0$$

Step 4: Using the result of part(a):

WE know that $|\beta_1^*\|_2^2 \leq \|\beta_2^*\|_2^2$, therefore the term $(\lambda_1 - \lambda_2)(\|\beta_1^*\|_2^2 - \|\beta_2^*\|_2^2) \leq 0$.

Therefore we can interpret and rewrite the earlier inequality as:

$$2(\|X\beta_1^* - y\|_2^2 - \|X\beta_2^* - y\|_2^2) \leq -(\lambda_1 - \lambda_2)(\|\beta_1^*\|_2^2 - \|\beta_2^*\|_2^2) \geq 0$$

This implies:

$$2(\|X\beta_1^* - y\|_2^2 - \|X\beta_2^* - y\|_2^2) \geq 0$$

Therefore:

$$\|X\beta_1^* - y\|_2^2 - \|X\beta_2^* - y\|_2^2 \geq 0$$

Therefore:

$$\|X\beta_1^* - y\|_2^2 \geq \|X\beta_2^* - y\|_2^2$$

Hence Proved.

c. Comments on same conditions for Lasso Regularization

In this part, the problem asks us to explore what happens if we switch from ridge regularization (which uses the ℓ_2-norm) to LASSO regularization (which uses the ℓ_1-norm).

The first two parts (a and b) of the problem were w.r.t ridge regularization (also called ℓ_2-regularization) the objective function was as:

$$\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_2^2$$

Here in LASSO (Least Absolute Shrinkage and Selection Operator) regularization, the objective function changes to:

$$\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_1$$

where,
- $\|\beta\|_1 = \sum_i |\beta_i|$ is the ℓ_1-norm of the vector β.
- $\lambda \geq 0$ is the regularization parameter as before.

So the first question we have to answer is that, "Do the conclusions from part (a) still hold with LASSO regularization?"

In part (a), we proved for ridge regression that increasing the regularization parameter resulted in a smaller ℓ_2-norm of the coefficient vector β. Essentially, larger λ forces the coefficients to shrink.

Yes, this conclusion still holds for LASSO regularization.

- In LASSO, the regularization term is $\lambda\|\beta\|_1 = \lambda \sum_i |\beta_i|$ which acts similar to ridge regression as increasing the weight of the regularization term, leads to the model focusing on reducing the magnitude of the coefficients to minimize the overall objective function.
- Hence, As λ increases:
  - The model will shrink the coefficients β .
  - The solution β^* will have a smaller ℓ_1-norm ($\|\beta^*\|_1$)

But there are 2 key differences:

- Unlike ridge regression (which shrinks all coefficients but never fully zeroes them out), LASSO has the ability to set some coefficients exactly to zero which introduces sparsity in the model. This is because the ℓ_1-norm is non-differentiable at zero, which makes it possible for LASSO to force some β_i's to become exactly zero, effectively performing feature selection.
- Due to the sparsity property: Though the ℓ_1 norm decreases, it does not necessarily decrease monotonically with increasing λ.

The second question is: Do the conclusions from part (b) still hold with LASSO regularization?

In part (a), we proved for ridge regression that increasing λ leads to a larger training loss. Essentially, increasing the regularization parameter sacrifices the model's ability to fit the training data perfectly, as it prioritizes shrinking the coefficients.

Yes, this conclusion still holds for LASSO regularization

- In LASSo, the regularization term $\lambda\|\beta\|_1$ also introduces a trade-off between fitting the data and shrinking the coefficients.
- As we increase λ, the model gives more weight to reducing the coefficients and less weight to minimizing the residual error $\|X\beta - y\|_2^2$.

- This results in a poor model fit because the stronger regularization enforces smaller (or even zero) coefficients, which might not capture the true relationships in the data, resulting in a worse fit to the training data.

Similar to the norm, due to the sparsity property of lasso which enables it to set coefficients to zero, the training loss may not increase monotonically with increasing λ as it does with ridge regularization.

In conclusion, the overall behaviour of the lasso regularization is similar to that of ridge in terms of the effects of increasing the regularization parameter, with some minor differences in the rate of change of norm and training loss.

Problem 2. Gaussian Naive Bayes

Collaborators: None.

a. Estimating Model Parameters

The task is to estimate the following parameters:
- $\mu_{0,j}$ and $\sigma^2_{0,j}$ for each feature j when $y = 0$.
- $\mu_{1,j}$ and $\sigma^2_{1,j}$ for each feature j when $y = 1$.

Step 1: Estimating the Mean $\mu_{y,j}$

To estimate the mean for feature j in class y, we calculate the average value of that feature across all training examples where the class label $y_i = y$.

Therefore, for class $y = 0$, the mean fora feature j is:

$$\mu_{0,j} = \frac{1}{n_0} \sum_{i:y_i=0} x_{i,j}$$

where,
- n_0 is the number of training examples where $y_i = 0$ (class 0).
- $x_{i,j}$ is the j-th feature of the i-th training example.

Similarly, for class $y = 1$, the mean for feature j is:

$$\mu_{1,j} = \frac{1}{n_1} \sum_{i:y_i=1} x_{i,j}$$

where,
- n_1 is the number of training examples where $y_i = 1$ (class 1).

Step 2: Estimating the Variance $\sigma^2_{y,j}$

To estimate the variance for feature j in class y, we calculate the average squared deviation from the mean for that feature across all training examples where the class label $y_i = y$.

For class $y = 0$, the variance for feature j is:

$$\sigma^2_{0,j} = \frac{1}{n_0} \sum_{i:y_i=0} (x_{i,j} - \mu_{0,j})^2$$

Similarly, for class $y = 1$, the variance for feature j is:

$$\sigma^2_{1,j} = \frac{1}{n_1} \sum_{i:y_i=1} (x_{i,j} - \mu_{1,j})^2$$

Hence, Demonstrated.

b. Predicting Class Label Using Maximum A Posteriori (MAP) Estimate

We have to predict the most likely class y_{new} for a new feature vector x_{new} by maximizing the posterior probability $P(y_{\text{new}}|x_{\text{new}})$, using Bayes' theorem.

Bayes' Theorem gives us the posterior probability $P(y_{\text{new}}|x_{\text{new}})$ as:

$$P(y_{\text{new}} = y|x_{\text{new}}) = \frac{P(x_{\text{new}}|y_{\text{new}} = y)P(y_{\text{new}} = y)}{P(x_{\text{new}})}$$

where,
- $P(x_{\text{new}}|y_{\text{new}} = y)$ is the likelihood
- $P(y_{\text{new}} = y)$ is the prior probability
- $P(x_{\text{new}})$ is the evidence

For classification, we do not need the exact value of $P(x_{\text{new}})$ because we are interested in which class has the largest posterior probability. So, we only need to maximize the numerator of Bayes' theorem. This means we select the class y_{new} that maximizes the product of the likelihood $P(x_{\text{new}}|y)$ and the prior $P(y)$.

$$y_{\text{new}} = \arg \max_{y \in \{0,1\}} P(x_{\text{new}}|y)P(y)$$

We can predict that in a few steps as belows:

Step 1: Calculate the Prior Probability $P(y)$:
From the training data, we can estimate the prior probability of each class. Let $P(y = 0)$ and $P(y = 1)$ represent the prior probabilities of classes 0 and 1, respectively. These can be computed as:

$$P(y = 0) = \frac{n_0}{n}, \quad P(y = 1) = \frac{n_1}{n}$$

where,
- n_0 is the number of training samples in class 0.
- n_1 is the number of training samples in class 1.
- $n = n_0 + n_1$ is the total number of training samples.

Step 2: Calculate the Likelihood $P(x_{\text{new}}|y)$:
For each class y, the likelihood is modeled using a Gaussian distribution foor each feature. For a given feature $x_{\text{new},j}$, the likelihood for class y is:

using respective "j" mean X_new, j and U_y,j

$$P(x_{\text{new},j}|y) = \frac{1}{\sqrt{2\pi\sigma_{y,j}^2}} \exp\left(-\frac{(x_{\text{new},j} - \mu_{y,j})^2}{2\sigma_{y,j}^2}\right)$$

The overall likelihood for x_{new} given class y (assuming independence across features) is the product of the individual feature likelihoods:

$$P(x_{\text{new}}|y) = \prod_{j=1}^{d} P(x_{\text{new},j}|y)$$

Step 3: Combine Prior and Likelihood : - For each class $y \in \{0, 1\}$, compute:

$$P(x_{\text{new}}|y)P(y)$$

Step 4: Choose and predict the Class with the Maximum Posterior that is:

$$y_{\text{new}} = \arg\max_{y \in \{0,1\}} P(x_{\text{new}}|y)P(y)$$

The above process can be showcased in the form of a pseudocode as below:

```
def predict_class(x_new, mu_0, sigma_0, mu_1, sigma_1, P_y0, P_y1):

    # Step 1: Calculate the likelihood for class 0
    likelihood_y0 = 1
    for j = 1 to d:  # Loop over each feature j
        likelihood_y0 *= (1 / sqrt(2 * pi * sigma_0[j]^2)) *
                         exp(- (x_new[j] - mu_0[j])^2 / (2 * sigma_0[j]^2))

    # Multiply by the prior for class 0
    posterior_y0 = likelihood_y0 * P_y0

    # Step 2: Calculate the likelihood for class 1
    likelihood_y1 = 1
    for j = 1 to d:  # Loop over each feature j
        likelihood_y1 *= (1 / sqrt(2 * pi * sigma_1[j]^2)) *
                         exp(- (x_new[j] - mu_1[j])^2 / (2 * sigma_1[j]^2))

    # Multiply by the prior for class 1
    posterior_y1 = likelihood_y1 * P_y1

    # Step 3: Choose the class with the highest posterior probability
    if posterior_y0 > posterior_y1:
        y_new = 0
    else:
        y_new = 1

    return y_new
```

For the above function predictclass the input output parameters have been listed below:

```
# Input:
# x_new: the new feature vector for which we want to predict the class
# mu_0, sigma_0: means and variances of features for class 0
# mu_1, sigma_1: means and variances of features for class 1
# P_y0, P_y1: prior probabilities for classes 0 and 1

# Output:
# y_new: predicted class label (0 or 1)
```

- Step 1 Replication: We compute the likelihood $P(x_{\text{new}}|y = 0)$ for class 0 by looping through each feature j and calculating the Gaussian probability for that feature.
  We multiply the likelihoods of all features together (since the Naive Bayes assumption treats

the features as independent).
Similarly, we compute the likelihood $P(x_{\text{new}}|y = 1)$ for class 1.

- Step 2 and 3 Replication: Multiply each likelihood by the prior probability for that class to get the posterior probability.
- Step 4 Replication: We then compare the two posterior probabilities. If the posterior for class 0 is larger, we predict class 0. Otherwise, we predict class 1.

c. Jupyter Notebook Task

The code from the jupyter notebook is attached below.

Question 2 c

Gaussian Naive Bayes

In this notebook, you will implement your Homework 2 generalization of Naive Bayes Classification for classification on the breast cancer dataset from our logistic regression demo. The data set is described here: https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)

More information on the problem can be found in the demo.

Loading and Visualizing the Data

We first load the packages and data as in the demo.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets, linear_model, preprocessing
%matplotlib inline
names = ['id','thick','size_unif','shape_unif','marg','cell_size','bare',
         'chrom','normal','mit','class']
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/' +
                 'breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                names=names,na_values='?',header=None)
df = df.dropna()

# Get the response.  Convert to a zero-one indicator
yraw = np.array(df['class'])
BEN_VAL = 2   # value in the 'class' label for benign samples
MAL_VAL = 4   # value in the 'class' label for malignant samples
y = (yraw == MAL_VAL).astype(int) # now y has values of 0,1
Iben = (y==0)
Imal = (y==1)
```

For this problem we are going to use all predictors we have at our disposal, as was done at the end of the demo. The code below places all of the predictor data in a matrix `Xfull` of dimension (n x d).

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 683 entries, 0 to 698
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          683 non-null    int64
 1   thick       683 non-null    int64
 2   size_unif   683 non-null    int64
 3   shape_unif  683 non-null    int64
 4   marg        683 non-null    int64
 5   cell_size   683 non-null    int64
 6   bare        683 non-null    float64
 7   chrom       683 non-null    int64
 8   normal      683 non-null    int64
```

```
 9   mit         683 non-null    int64
 10  class       683 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 64.0 KB
```

```
df["class"].value_counts()
```

class	count
2	444
4	239

dtype: int64

```
df.shape
```

```
(683, 11)
```

```
# Iben
len(Iben)
```

```
683
```

```
# Imal
len(Imal)
```

```
683
```

```
xnames = names[:-1]
Xfull = np.array(df[xnames])
n = Xfull.shape[0]
d = Xfull.shape[1]
```

```
xnames
```

```
['id',
 'thick',
 'size_unif',
 'shape_unif',
 'marg',
 'cell_size',
 'bare',
 'chrom',
 'normal',
 'mit']
```

```
print(n)
print(d)
```

```
683
10
```

Naive Bayes Classification

The first step in using the Gaussian Naive Bayes method is to estimate all parameters of the probabilistic model. These include:

- `p` -- the probability that a data example is malignant (label 1)
- `mu0 = [mu0[0], ..., mu0[d-1]]` -- the expected value of each predictor variable for benign examples.
- `sig0 = [sig0[0], ..., sig0[d-1]]` -- the variance value of each predictor variable for benign examples.
- `mu1 = [mu1[0], ..., mu1[d-1]]` -- the expected value of each predictor variable for malignant examples.
- `sig1 = [sig1[0], ..., sig1[d-1]]` -- the variance value of each predictor variable for maglignant examples.

Compute estimates for these values below using the data in `Xfull`.

Hint: For a compact approach, you might want to use the boolean arrays `Imal` and `Iben` created above for "mask indexing" (see these docs).

```
  # TODO
# p = ...
# mu0 = ..
# sig0 = ...
# mu1 = ..
# sig1 = ...
```

```
n_malignant = np.sum(Imal)  # Number of malignant cases
n_total = n  # Total number of cases
p = n_malignant / n_total  # Prior probability of malignant cases
```

```
mu0 = np.mean(Xfull[Iben], axis=0)  # Mean for benign cases (class 0)
mu1 = np.mean(Xfull[Imal], axis=0)  # Mean for malignant cases (class 1)
```

```
sig0 = np.var(Xfull[Iben], axis=0)  # Variance for benign cases (class 0)
sig1 = np.var(Xfull[Imal], axis=0)  # Variance for malignant cases (class 1)
```

```
# # Print the results to check
# print("Prior probability (p):", p)
# print("Means for benign class (mu0):", mu0)
# print("Means for malignant class (mu1):", mu1)
# print("Variances for benign class (sig0):", sig0)
# print("Variances for malignant class (sig1):", sig1)
```

Next, for every row $\mathbf{x}$ in `Xfull`, use the model parameters determined above to compute the maximum a posterior (MAP) estimate for the class label y. You should use the equations derived in your solution to Problem 3 (b) on the homework.

You will need to be thoughtful in how you do this computation to avoid numerical underflow or overflow in your computations: keep in mind that you just need to determine which of $p(y = 0|\mathbf{x})$ or $p(y = 1|\mathbf{x})$ is **larger** -- you don't necessarily need to compute both values explicitly.

Store your MAP estimates for the data examples in an integer vector `yhat` (with 0 indicating benign, 1 indicating malignant).

```
# TODO
# ...
# yhat =
```

```
# Initialize yhat to store the MAP estimates
yhat = np.zeros(n, dtype=int)

log_p_y0 = np.log(1 - p)  # log P(y = 0) (benign)
log_p_y1 = np.log(p)      # log P(y = 1) (malignant)
```

```
for i in range(n):
    x = Xfull[i, :]

    log_likelihood_y0 = log_p_y0
    log_likelihood_y1 = log_p_y1

    for j in range(d):
        log_likelihood_y0 += -0.5 * np.log(2 * np.pi * sig0[j]) - ((x[j] -
                              mu0[j])**2) / (2 * sig0[j])
        log_likelihood_y1 += -0.5 * np.log(2 * np.pi * sig1[j]) - ((x[j] -
                             mu1[j])**2) / (2 * sig1[j])

    # Choose and predict the class with the larger log-posterior
    if log_likelihood_y1 > log_likelihood_y0:
        yhat[i] = 1  # malignant
    else:
        yhat[i] = 0  # benign
```

Compute the accuracy of your estimates using the code below. You should see a result which is very competitive with logistic regression, which is pretty cool given how simple this algorithm is!

Note: we would ideally want to do a proper train/test split to evaluate the performance of both logistic regression and Naive Bayes classification. We're just looking at training set loss to keep things simple, and because the number of features is relatively small, so we're not super worried about overfitting.

```
acc = np.mean(yhat == y)
print("Accuracy on training data = %f" % acc)
recall = np.sum((yhat == 1)*(y == 1))/np.sum(y == 1)
precision = np.sum((yhat == 1)*(y == 1))/np.sum(yhat == 1)
print("Recall: " + str(recall))
print("Precision: " + str(precision))
```

```
Accuracy on training data = 0.963397
Recall: 0.9790794979079498
Precision: 0.9212598425196851
```

d. Numerical: Calculating the Misclassification Probability

We have been given two Gaussian distributions for the two classes (malignant and benign) and we have to calculate the probability of making a mistake when classifying.

- For class 0 (benign): we have a Gaussian distribution with mean $\mu_0 = 5$ and standard deviation $\sigma_0 = 3$.
- For class 1 (malignant): we have a Gaussian distribution with mean $\mu_1 = 2$ and standard deviation $\sigma_1 = 3$.

We can calculate the probability of misclassificationin few steps as below:

Step 1. Find the Decision Boundary:
Set the two likelihoods equal:

$$\frac{1}{\sqrt{2\pi \cdot 3^2}} \exp\left(-\frac{(x-5)^2}{2 \cdot 3^2}\right) = \frac{1}{\sqrt{2\pi \cdot 3^2}} \exp\left(-\frac{(x-2)^2}{2 \cdot 3^2}\right)$$

Cancelling out the common terms and therefore simplifying:

$$(x-5)^2 = (x-2)^2$$

Expanding both sides:

$$(x^2 - 10x + 25) = (x^2 - 4x + 4)$$

After simplifying we get x as:

$$-10x + 25 = -4x + 4$$

$$x = 3.5$$

Thus, the decision boundary is at $x = 3.5$.

Step 2: Compute the Misclassification Probabilities

Now that we know the decision boundary is at $x = 3.5$, we can calculate the probability of misclassification.

For class 0 (benign) : The misclassification probability is the probability that a benign sample is classified as malignant. This is given by the CDF of the normal distribution:

$$P(x < 3.5|y = 0) = P\left(Z < \frac{3.5 - 5}{3}\right) = P(Z < -0.5)$$

Therefore,

$$P(Z < -0.5) \approx 0.3085$$

For class 1 (malignant) : The misclassification probability is the probability that a malignant sample is classified as benign. This given by:

$$P(x > 3.5|y = 1) = 1 - P\left(Z < \frac{3.5 - 2}{3}\right) = 1 - P(Z < 0.5)$$

Therefore,

$$P(Z < 0.5) \approx 0.6915$$

Therefore,

$$P(x > 3.5|y = 1) = 1 - 0.6915 = 0.3085$$

Step 3: Total Misclassification Probability

Since the priors are equal, the overall probability of misclassification is the average of the two misclassification probabilities:

$$P(\text{misclassify}) = \frac{1}{2}\left(P(x < 3.5|y = 0) + P(x > 3.5|y = 1)\right)$$

$$P(\text{misclassify}) = \frac{1}{2}\left(0.3085 + 0.3085\right) = 0.3085$$

Thus, the total probability of misclassification is approximately 0.3085 .

In cocnlusion, the probability of misclassification for the Gaussian Naive Bayes classifier in this numerical is approximately 30.85% . This result means that, on average, the classifier will misclassify about 30.85% of the data points when making predictions based on the given Gaussian distributions for benign and malignant cases.

The result can also be verified by the using the "scipy.stats.norm.cdf()" library as follows:

Question 2 d

```
from scipy.stats import norm

mu0 = 5   # mean for benign (class 0)
sigma0 = 3  # std dev for benign (class 0)
mu1 = 2   # mean for malignant (class 1)
sigma1 = 3  # std dev for malignant (class 1)


# Decision boundary --> calculated by equating likelihoods for both the classes
decision_boundary = 3.5


# CDF function to calculate misclassification probability
# Using scipy.stats.norm.cdf to calculate cumulative distribution
# For benign (class 0): P(x < 3.5 | y = 0)
P_x_less_3_5_given_y0 = norm.cdf(decision_boundary, loc=mu0, scale=sigma0)
P_x_greater_3_5_given_y1 = 1 - norm.cdf(decision_boundary, loc=mu1, scale=sigma1)


# Compute total misclassification probability
P_misclassification = 0.5 * (P_x_less_3_5_given_y0 + P_x_greater_3_5_given_y1)


# Print out the results
print(f"Total probability of misclassification: {P_misclassification:.4f}")
```

```
Total probability of misclassification: 0.3085
```

Problem 3. More Practice with MAP Calculations

Collaborators: None.

We are asked to determine the values of $x \in [0, 7]$ where the MAP estimate will predict $y = 1$. In other words, we need to find the regions where the posterior probability of $y = 1$ is higher than that of $y = 0$.

Using Bayes' Theorem , the posterior probability of $y = 1$ given x is:

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x)}$$

Similarly, the posterior probability of $y = 0$ given x is:

$$P(y = 0|x) = \frac{P(x|y = 0)P(y = 0)}{P(x)}$$

Since $P(x)$ is common in both expressions, we don't need to compute it explicitly for comparison. We just can compare the numerator of these expressions:

$$P(y = 1|x) \propto P(x|y = 1)P(y = 1)$$

$$P(y = 0|x) \propto P(x|y = 0)P(y = 0)$$

IMPORTANT: Finding the Threshold for MAP estimate. Therfore, we will choose $y = 1$ when:

$$\boldsymbol{P}(\boldsymbol{x} \mid \boldsymbol{y} = 1)\boldsymbol{P}(\boldsymbol{y} = 1) > \boldsymbol{P}(\boldsymbol{x} \mid \boldsymbol{y} = 0)\boldsymbol{P}(\boldsymbol{y} = 0)$$

Or equivalently:

$$\frac{P(x|y = 1)}{P(x|y = 0)} > \frac{P(y = 0)}{P(y = 1)}$$

Substituting the prior probabilities:

$$\frac{P(x|y = 1)}{P(x|y = 0)} > \frac{0.4}{0.6} = \frac{2}{3}$$

Thus, we need to find for which values of x, the ratio of $P(x|y = 1)$ to $P(x|y = 0)$ is greater than $\frac{2}{3}$.

Stepwise soltion for all Piecewise Intervals:

Interval 1: $0 \leq x \leq 1$ For this range, we have:

- $P(x|y = 0) = \frac{1}{5}$ (since $0 \leq x \leq 2$).
- $P(x|y = 1) = \frac{1}{6}$.

The ratio is:

$$\frac{P(x|y=1)}{P(x|y=0)} = \frac{1/6}{1/5} = \frac{5}{6}$$

Since $\frac{5}{6} > \frac{2}{3}$, the MAP estimate will predict $y = 1$ in this Interval.

Interval 2: $1 < x \leq 2$ For this range, we have:

- $P(x|y=0) = \frac{1}{5}$
- $P(x|y=1) = \frac{1}{8}$.

The ratio is:

$$\frac{P(x|y=1)}{P(x|y=0)} = \frac{1/8}{1/5} = \frac{5}{8}$$

Since $\frac{5}{8} < \frac{2}{3}$, the MAP estimate will predict $y = 0$ in this interval.

Interval 3: $2 < x \leq 3$ For this range, we have:

- $P(x|y=0) = \frac{1}{3}$
- $P(x|y=1) = \frac{1}{8}$

The ratio is:

$$\frac{P(x|y=1)}{P(x|y=0)} = \frac{1/8}{1/3} = \frac{3}{8}$$

Since $\frac{3}{8} < \frac{2}{3}$, the MAP estimate will predict $y = 0$ in this interval.

Interval 4: $3 < x \leq 5$ For this range, we have:

- $P(x|y=0) = \frac{1}{15}$
- $P(x|y=1) = \frac{1}{8}$

The ratio is:

$$\frac{P(x|y=1)}{P(x|y=0)} = \frac{1/8}{1/15} = \frac{15}{8}$$

Since $\frac{15}{8} > \frac{2}{3}$, the MAP estimate will predict $y = 1$ in this interval.

Interval 5: $5 < x \leq 7$ For this range, we have:

- $P(x|y=0) = \frac{1}{15}$
- $P(x|y=1) = \frac{1}{6}$.

The ratio is:

$$\frac{P(x|y=1)}{P(x|y=0)} = \frac{1/6}{1/15} = \frac{15}{6} = 2.5$$

Since $2.5 > \frac{2}{3}$, the MAP estimate will predict $y = 1$ in this interval.

Therefore, in conclusion, the MAP estimate will predict $y = 1$ for:

$$x \in [0, 1] \cup (3, 7]$$

and it will predict $y = 0$ for:

$$x \in (1, 3]$$

Hence, Solved.

Problem 4. Bayesian Central Tendency

Collaborators: None.

a. Maximum Likelihood Estimation for the Mean of a Gaussian Distribution

We are supposed to prove that the sample mean $\hat{\mu}$ is the MLE for the unknown mean μ in a **Gaussian distribution**.

This can be proved in few steps as below:

Step 1: Likelihood Function for the Gaussian Distribution

The probability density function (PDF) of a Gaussian distribution with mean μ and variance σ^2 is given by:

$$p(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

Since the data points $x_1, x_2, \ldots, x_n$ are assumed to be independent and identically distributed (i.i.d.), the likelihood of the entire dataset $x_1, x_2, \ldots, x_n$ is the product of the individual likelihoods:

$$L(\mu, \sigma^2|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

We need to find the value of μ that maximizes this likelihood function.

Maximizing the likelihood directly is often complex because it involves a product of exponentials. To simplify the calculations, we take the logarithm of the likelihood function. Since the logarithm is a monotonic function, maximizing the log-likelihood is equivalent to maximizing the likelihood.

Step 2: Calculatting Log-Likelihood Function

The log-likelihood function $\ell(\mu, \sigma^2)$ is:

$$\ell(\mu, \sigma^2|x_1, \ldots, x_n) = \log L(\mu, \sigma^2|x_1, \ldots, x_n)$$

Therefore,

$$\ell(\mu, \sigma^2) = \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)\right)$$

Simplifying using log properties,

$$\ell(\mu, \sigma^2) = \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \sum_{i=1}^{n} \log\left(\exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)\right)$$

The logarithm of the Gaussian density components after managing and removing exponential terms, can be written as:

$$\ell(\mu, \sigma^2) = \sum_{i=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma^2)\right) + \sum_{i=1}^{n} \left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

Simplifying further:

$$\ell(\mu, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Since σ^2 is unknown but fixed, we focus only on maximizing the log-likelihood with respect to μ.

Step 3: Differentiating the Log-Likelihood with Respect to μ

To find the value of μ that maximizes the log-likelihood, we take the derivative of the log-likelihood $\ell(\mu, \sigma^2)$ with respect to μ and set it equate it to zero.

$$\frac{\partial \ell(\mu, \sigma^2)}{\partial \mu} = \frac{\partial}{\partial \mu}\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2\right)$$

using chain rule to find the derivative of $(x_i - \mu)^2$ with respect to μ :

$$\frac{\partial}{\partial \mu}(x_i - \mu)^2 = -2(x_i - \mu)$$

Thus, the derivative of the log-likelihood is:

$$\frac{\partial \ell(\mu, \sigma^2)}{\partial \mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)$$

Set this derivative equal to zero to find the maximum:

$$\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) = 0$$

Simplifying:

$$\sum_{i=1}^{n}(x_i - \mu) = 0$$

$$\sum_{i=1}^{n} x_i - n\mu = 0$$

Solving for μ:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Thus we have shown that the value of μ that maximizes the log-likelihood is the sample mean :

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Thus, the sample mean is the Maximum Likelihood Estimator (MLE) for the unknown mean μ when the data is drawn from a **Gaussian distribution** with unknown mean and variance.

Hence, Proved.

b. Maximum Likelihood Estimation for the Mean of a Laplace Distribution

We are to prove that the sample median is the Maximum Likelihood Estimate (MLE) for the mean μ when the data is drawn from a **Laplace distribution**.

This can be proved in a few steps as follows:

Step 1: Likelihood Function for the Laplace Distribution

As the data points $x_1, x_2, \ldots, x_n$ are are i.i.d., the likelihood function is the product of the individual probability densities:

$$L(\mu, b|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{2b} \exp\left(-\frac{|x_i - \mu|}{b}\right)$$

Since b is a constant (and we're maximizing with respect to μ), as of now we can ignore the constant $\frac{1}{2b}$ and onyl consider the exponential terms.

Thus, the likelihood function simplifies to:

$$L(\mu|x_1, x_2, \ldots, x_n) = \exp\left(-\frac{1}{b}\sum_{i=1}^{n} |x_i - \mu|\right)$$

The goal is to maximize this likelihood with respect to μ.

To simplify the maximization, we take the logarithm of the likelihood function as it turns the product of exponentials into a sum, which is easier to differentiate.

Step 2: Log-Likelihood Function

.The log of the likelihood (called the log-likelihood) is:

$$\ell(\mu|x_1, \ldots, x_n) = \log L(\mu|x_1, \ldots, x_n) = -\frac{1}{b}\sum_{i=1}^{n} |x_i - \mu|$$

Again we can ig nore b. Therefore, we need to minimize the following expression:

$$\sum_{i=1}^{n} |x_i - \mu|$$

This is the sum of absolute deviations between the data points x_i and the estimate μ.

And from Homework1, we already know that the value of μ that minimizes the sum of absolute deviations is the sample median.
Hence, we have shown that the value of μ that maximizes the log-likelihood (or equivalently, minimizes the sum of absolute deviations) is the sample median.
Thus, the sample median is the Maximum Likelihood Estimator (MLE) for the mean μ when the data is drawn from a **Laplace distribution**.

Hence, Proved.

c. Maximum Likelihood Estimation for the Mean of a Bernoulli Distribution

We are supposed to prove that the sample mean is the Maximum Likelihood Estimate (MLE) for μ when the data $x_1, x_2, \ldots, x_n$ are drawn from a **Bernoulli distribution** with parameter μ.

This can be solved in a few steps as below:

Step 1: Likelihood Function for the Bernoulli Distribution

For the Bernoulli distribution, each data point $x_i \in \{0, 1\}$ is drawn independently, and the likelihood of observing the dataset $x_1, x_2, \ldots, x_n$ is the product of the individual probabilities.

Therefore, the likelihood function is:

$$L(\mu|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|\mu)$$

Using the Bernoulli PMF:

$$P(x_i|\mu) = \mu^{x_i}(1-\mu)^{1-x_i}$$

After using Bernoulli PMF, the likelihood function will become:

$$L(\mu|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \mu^{x_i}(1-\mu)^{1-x_i}$$

Expand as:

$$L(\mu|x_1, x_2, \ldots, x_n) = \mu^{\sum_{i=1}^{n} x_i}(1-\mu)^{n-\sum_{i=1}^{n} x_i}$$

This is the likelihood function that we need to maximize with respect to μ.

Taking the logarithm of the likelihood function makes it easier to differentiate and find the maximum. Therefore we do the same in the next step.

Step 2: Log-Likelihood Function

$$\ell(\mu|x_1, x_2, \ldots, x_n) = \log L(\mu|x_1, x_2, \ldots, x_n)$$

Substituting the expression for $L(\mu)$ from above:

$$\ell(\mu) = \log\left(\mu^{\sum_{i=1}^{n} x_i}(1-\mu)^{n-\sum_{i=1}^{n} x_i}\right)$$

Using the properties of logarithms:

$$\ell(\mu) = \sum_{i=1}^{n} x_i \log(\mu) + \left(n - \sum_{i=1}^{n} x_i\right) \log(1-\mu)$$

Simplifying:

$$\ell(\mu) = \left(\sum_{i=1}^{n} x_i\right) \log(\mu) + \left(n - \sum_{i=1}^{n} x_i\right) \log(1-\mu)$$

This is the log-likelihood function that we need to maximize.

Step 3: Differentiating Log-Likelihood with Respect to μ

To find the value of μ that maximizes the log-likelihood, we take the derivative of $\ell(\mu)$ with respect to μ and set it equal to zero.

Therefore:

$$\frac{d\ell(\mu)}{d\mu} = \frac{\sum_{i=1}^{n} x_i}{\mu} - \frac{n - \sum_{i=1}^{n} x_i}{1 - \mu}$$

After Equating to zero:

$$\frac{\sum_{i=1}^{n} x_i}{\mu} = \frac{n - \sum_{i=1}^{n} x_i}{1 - \mu}$$

Simplifying

$$\sum_{i=1}^{n} x_i(1 - \mu) = (n - \sum_{i=1}^{n} x_i)\mu$$

Expanding both sides:

$$\sum_{i=1}^{n} x_i - \mu \sum_{i=1}^{n} x_i = n\mu - \mu \sum_{i=1}^{n} x_i$$

Simplifying further:

$$\sum_{i=1}^{n} x_i = n\mu$$

Solving for μ:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Hence, we have shown that the value of μ that maximizes the log-likelihood is the sample mean :

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Thus, we can conclude that, the sample mean is the Maximum Likelihood Estimator (MLE) for μ when the data are drawn from a **Bernoulli distribution**.

Hence, Proved.

Problem 5. Unexpected Mean Estimators

Collaborators: None.

The Rayleigh distribution is often used to model the magnitude of a 2D vector $v = (x, y)$, where the components x and y are independent Gaussian random variables with zero mean and variance σ^2.
The magnitude $r = \sqrt{x^2 + y^2}$ follows a Rayleigh distribution with parameter σ.

The probability density function (PDF) of the Rayleigh distribution can be writteng as:

$$p(x|\sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right), \quad \text{for } x \geq 0$$

where,
- x is the observed data (magnitude of the vector)
- σ is the unknown scale parameter that we need to estimate using the MLE.

a. Maximum Likelihood Estimation for the Rayleigh Distribution

We have to find the Maximum Likelihood Estimator (MLE) for the parameter σ of a **Rayleigh distribution** where the data points $x_1, x_2, \ldots, x_n$ are drawn independently.

This can be done in few steps as belows:

Step 1: Likelihood Function

The Likelihood function is the product of the individual probability densities for each observation:

$$L(\sigma|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|\sigma)$$

Substituting the PDF of the Rayleigh distribution into this expression:

$$L(\sigma|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \left(\frac{x_i}{\sigma^2} \exp\left(-\frac{x_i^2}{2\sigma^2}\right)\right)$$

Expanding:

$$L(\sigma|x_1, x_2, \ldots, x_n) = \frac{1}{\sigma^{2n}} \prod_{i=1}^{n} x_i \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} x_i^2\right)$$

This is the likelihood function for the Rayleigh distribution, and we want to maximize this with respect to σ. To simplify the maximization, we take the logarithm of the likelihood function.

Step 2: Log-Likelihood Function

Log-Likelihood Function $\ell(\sigma|x_1, \ldots, x_n)$ is given by:

$$\ell(\sigma|x_1, \ldots, x_n) = \log L(\sigma|x_1, \ldots, x_n)$$

Taking the logarithm of the likelihood function:

$$\ell(\sigma) = \log\left(\frac{1}{\sigma^{2n}}\prod_{i=1}^{n} x_i \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2\right)\right)$$

properties I'll used for expansion: $\log(ab) = \log(a) + \log(b)$ and $\log(a^b) = b\log(a)$

$$\ell(\sigma) = \log\left(\frac{1}{\sigma^{2n}}\right) + \log\left(\prod_{i=1}^{n} x_i\right) + \log\left(\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2\right)\right)$$

This simplifies to:

$$\ell(\sigma) = -2n\log(\sigma) + \sum_{i=1}^{n}\log(x_i) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2$$

Therefore, the log-likelihood function for the Rayleigh distribution is:

$$\ell(\sigma) = -2n\log(\sigma) + \sum_{i=1}^{n}\log(x_i) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2$$

This expression gives the log-likelihood of the data $x_1, x_2, \ldots, x_n$ as a function of the unknown parameter σ.

Hence, Solved.

b. Maximum Likelihood Estimation for the Rayleigh Distribution – Finding the MLE for σ

In part (a), we derived the log-likelihood function for σ, and now we will maximize it to obtain the MLE for σ in the Rayleigh distribution.

We can do it in few steps as follows:

Step 1: Differentiate the Log-Likelihood Function obtained in Part (a)

differentiating $\ell(\sigma)$ with respect to σ.

- The derivative of $-2n\log(\sigma)$ with respect to σ is:

$$\frac{d}{d\sigma}\left(-2n\log(\sigma)\right) = -\frac{2n}{\sigma}$$

- The derivative of $\sum_{i=1}^{n}\log(x_i)$ with respect to σ is zero , because this term does not depend on σ.

- The derivative of $-\frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2$ with respect to σ is:

$$\frac{d}{d\sigma}\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} x_i^2\right) = \frac{1}{\sigma^3}\sum_{i=1}^{n} x_i^2$$

Now, putting all these pieces together, the derivative of the log-likelihood function is:

$$\frac{d\ell(\sigma)}{d\sigma} = -\frac{2n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n} x_i^2$$

Step 2: Set the Derivative Equals to Zero

we need to set derivative equal to zero, in order to find the value of σ that maximizes the log-likelihood

$$-\frac{2n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n} x_i^2 = 0$$

Multiply through by σ^3 to eliminate the denominators:

$$-2n\sigma^2 + \sum_{i=1}^{n} x_i^2 = 0$$

Rearranging the equation:

$$2n\sigma^2 = \sum_{i=1}^{n} x_i^2$$

therefore, solving for σ^2:

$$\sigma^2 = \frac{1}{2n}\sum_{i=1}^{n} x_i^2$$

Taking square root to find σ:

$$\sigma_{\text{MLE}} = \sqrt{\frac{1}{2n}\sum_{i=1}^{n} x_i^2}$$

Thus, the Maximum Likelihood Estimate (MLE) for σ is:

$$\sigma_{\text{MLE}} = \sqrt{\frac{1}{2n}\sum_{i=1}^{n} x_i^2}$$

Therefore, this is the MLE for the σ of the Rayleigh Distribution.

We can also verify this by checking the second derivative of the log-likelihood function. The second derivative is:

$$\frac{d^2\ell(\sigma)}{d\sigma^2} = \frac{2n}{\sigma^2} - \frac{3}{\sigma^4}\sum_{i=1}^{n} x_i^2$$

Since the second derivative is negative (for $\sigma > 0$), the log-likelihood function is concave at this point, confirming that the solution we found is a maximum .

Therefore the Maximum Likelihood Estimate (MLE) for the parameter σ of the Rayleigh distribution is:

$$\sigma_{\text{MLE}} = \sqrt{\frac{1}{2n}\sum_{i=1}^{n} x_i^2}$$

Hence, Proved.