Name: Saadullah Khan

From the ISLP book do the following problems
https://hastie.su.domains/ISLP/ISLP_website.pdf.download.html

Problem 1 chapter 4, problem 1 (1 point)

Answer: We start from the logistic form:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

Compute the odds $\frac{p(X)}{1-p(X)}$:

$$\frac{p(X)}{1-p(X)} = \frac{\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}}{1-\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}} = \frac{\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}}{\frac{1+e^{\beta_0+\beta_1 X}-e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}} = \frac{e^{\beta_0+\beta_1 X}}{1} = e^{\beta_0+\beta_1 X}.$$

Now take natural logarithms (the logit):

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \log\left(e^{\beta_0+\beta_1 X}\right) = \beta_0+\beta_1 X.$$

This shows the logistic form and the logit form are algebraically equivalent.

Problem 2 chapter 4, problem 2 (1 point)

Answer:

Since the denominator $\sum_l \pi_l f_l(x)$ is common to all classes, maximizing $p_k(x)$ is equivalent to maximizing the numerator $\pi_k f_k(x)$.

The class-conditional density is assumed normal:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right).$$

So we choose the class that maximizes

$$\pi_k f_k(x) = \pi_k \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right).$$

Take logs (monotonic transform):

$$\log(\pi_k f_k(x)) = \log(\pi_k) - \frac{1}{2}\log(2\pi\sigma^2) - \frac{(x-\mu_k)^2}{2\sigma^2}.$$

The constant term $-\frac{1}{2}\log(2\pi\sigma^2)$ does not depend on k and can be ignored.

Expand the quadratic:

$$-\frac{(x-\mu_k)^2}{2\sigma^2} = -\frac{x^2 - 2x\mu_k + \mu_k^2}{2\sigma^2} = -\frac{x^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2}.$$

The term $-\frac{x^2}{2\sigma^2}$ is also constant across classes and can be ignored.

Thus, the discriminant function is

$$\delta_k(x) = \frac{x\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k).$$

Hence, the Bayes classifier assigns an observation x to the class with the largest $\delta_k(x)$.

Problem 3 chapter 4, problem 3 (1 point)

Answer: The Bayes classifier assigns an observation x to the class with the largest posterior probability:

$$\hat{y} = \arg\max_k \, p_k(x)$$

For each class k, define the discriminant function:

$$\delta_k(x) = \log(\pi_k f_k(x)),$$

where π_k is the prior probability of class k and $f_k(x)$ is the class-conditional density.

Substituting the Gaussian density gives

$$\delta_k(x) = \log(\pi_k) - \frac{1}{2}\log(2\pi\sigma_k^2) - \frac{(x-\mu_k)^2}{2\sigma_k^2}.$$

Expanding the quadratic term $(x-\mu_k)^2 = x^2 - 2x\mu_k + \mu_k^2$ yields

$$\delta_k(x) = -\frac{x^2}{2\sigma_k^2} + \frac{x\mu_k}{\sigma_k^2} + \left[\log(\pi_k) - \frac{1}{2}\log(2\pi\sigma_k^2) - \frac{\mu_k^2}{2\sigma_k^2}\right].$$

Define the coefficients

$$a_k = -\frac{1}{2\sigma_k^2}, \quad b_k = \frac{\mu_k}{\sigma_k^2}, \quad c_k = \log(\pi_k) - \frac{1}{2}\log(2\pi\sigma_k^2) - \frac{\mu_k^2}{2\sigma_k^2},$$

so that

$$\delta_k(x) = a_k x^2 + b_k x + c_k.$$

Because the coefficient a_k depends on the class-specific variance σ_k^2, the discriminant functions are quadratic in x.

Conclusion:

Since a_k varies across classes when the variances differ, the decision boundaries obtained by setting $\delta_i(x) = \delta_j(x)$ are quadratic equations in x. Therefore, the Bayes classifier in one-dimensional QDA is **quadratic**, reducing to **linear** only when all class variances are equal (the LDA case).

Problem 4 chapter 4, problem 4 (2 points)

Answer:

(a) Fraction of observations used for $p = 1$

We have one feature X uniformly distributed on $[0, 1]$. We wish to use observations within 10% of the range around a test point.
Thus, the fraction of the total range we are using is 0.1. Therefore, on average, the fraction of observations used is:

$$\text{Fraction} = 0.1$$

(b) Fraction of observations used for $p = 2$

Now consider two features (X_1, X_2) uniformly distributed on $[0, 1]^2$.
We take points within 10% of the range for both features.
The fraction of points inside this square region is:

$$\text{Fraction} = 0.1 \times 0.1 = 0.1^2 = 0.01$$

Hence, only 1% of the observations are used on average.

(c) Fraction of observations used for $p = 100$

With $p = 100$ features, each feature contributes a factor of 0.1.
Thus, the fraction of observations in this 100-dimensional hypercube is:

$$\text{Fraction} = 0.1^{100} = 10^{-100}$$

This number is extremely small, effectively meaning that almost no training points are close to any given test observation.

(d) Curse of Dimensionality

From the results above:

$$p = 1 \Rightarrow 10\% \text{ of data used}$$

$$p = 2 \Rightarrow 1\% \text{ of data used}$$

$$p = 100 \Rightarrow 10^{-100} \text{ fraction of data used}$$

As p increases, the fraction of nearby data points decreases exponentially.
Hence, very few training observations are "near" any given test observation.
This illustrates why KNN and other local, nonparametric methods often perform poorly in high-dimensional spaces.

(e) Hypercube length to capture 10% of points

Suppose we want a p-dimensional hypercube that contains 10% of all observations.
Let l denote the side length of the hypercube.
Then the volume is:

$$l^p = 0.1 \quad \Longrightarrow \quad l = 0.1^{1/p}$$

For various p:

$$p = 1: \quad l = 0.1^{1/1} = 0.1$$

$$p = 2: \quad l = 0.1^{1/2} = \sqrt{0.1} \approx 0.316$$

$$p = 100: \quad l = 0.1^{1/100} \approx 0.977$$

Comment:
As p increases, the side length l approaches 1.
This means that to capture just 10% of the data in high dimensions, we must include almost the entire range of each feature.
This demonstrates another aspect of the curse of dimensionality: as dimensionality increases, data become extremely sparse, and "local" regions lose their meaning.

Problem 5 chapter 4, problem 5 (2 points)

Answer:

a) Bayes boundary is linear.

Training set: QDA will have equal or lower training error (more flexible ⇒ can fit training data as well or better).

Test set: LDA is expected to perform better (lower test error). Reason: LDA's model matches the true linear boundary (low bias) and has lower variance than QDA, so it generalizes better.

(b) Bayes boundary is non-linear (e.g. quadratic).

Training set: QDA will usually have lower training error (it can represent the true non-linear boundary).

Test set: QDA tends to perform better provided sample size is large enough. If n n is small relative to the number of parameters, QDA's higher variance can cause worse test performance than LDA. So: QDA likely better asymptotically; with small n n LDA can sometimes beat QDA.

(c) As sample size n n increases, how does QDA vs LDA test accuracy change?

QDA relative to LDA improves with larger n n. Reason: increasing n n reduces estimation variance for QDA's many covariance parameters, so QDA's advantage in bias (when the true boundary is non-linear) can be realized. Thus the flexible model benefits from more data.

(d) True or False: "Even if the Bayes decision boundary is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary."

False.

Although QDA can represent a linear boundary (it includes linear as a special case), its many extra parameters give it higher variance. With finite samples, that extra variance typically hurts test performance when the true boundary is linear. If the true model is linear, LDA is the better choice in practice (lower variance, correct bias).

Intuition: correct simplicity beats unnecessary flexibility when data are limited.

Problem 6 chapter 4, problem 16 (3 points)

Note: Ignore KNN.

- Fit logistic regression, LDA, QDA, and NaiveBayes (using the Gaussian distribution).
- In a 4 x 3 table, report the error rate and AUC of these models (with the first row presenting the model names).
- Create a single plot, presenting the ROC curve of all four models with appropiate legends.

Answer:

In [10]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

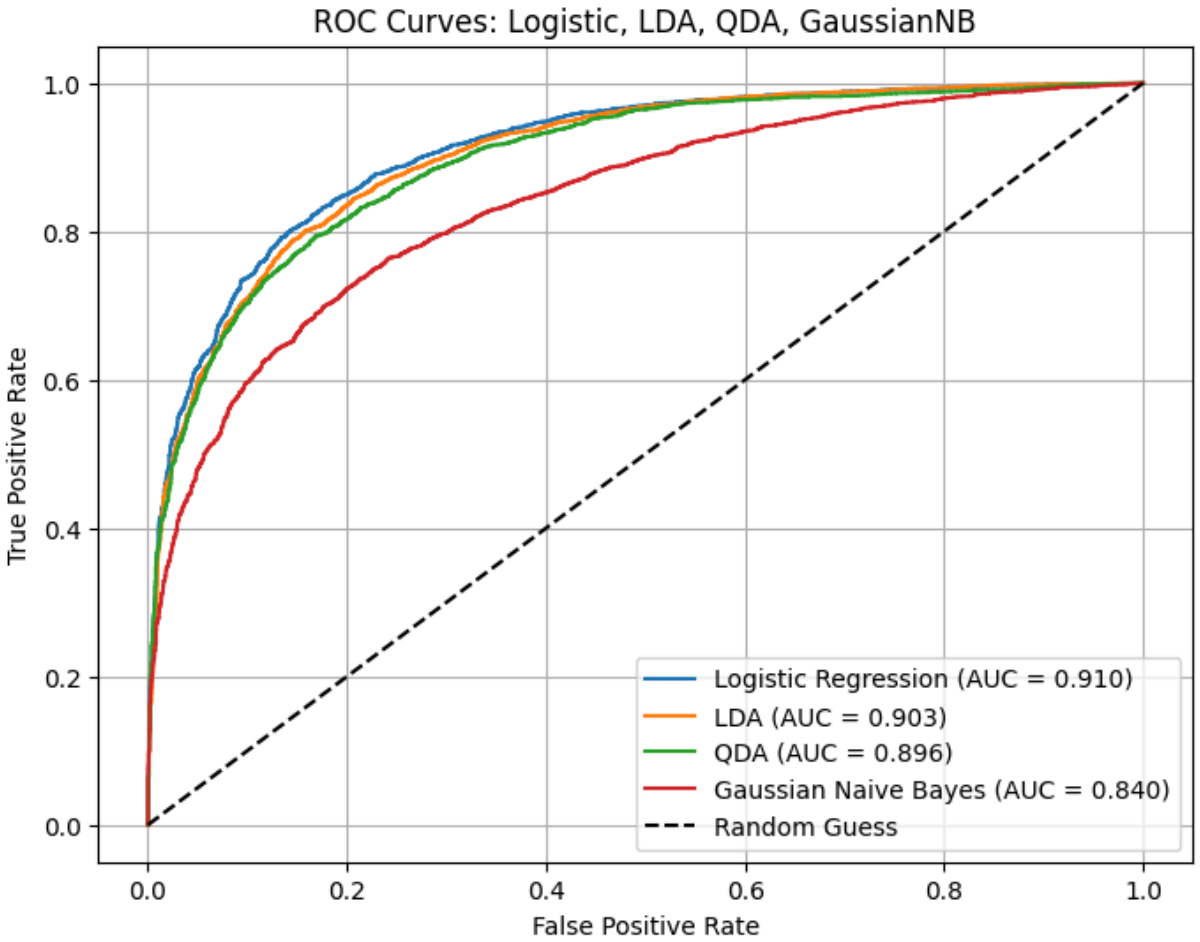
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import fetch_california_housing

from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

import pandas as pd
from sklearn.datasets import fetch_california_housing, load_breast_cancer, make_cla
from sklearn.preprocessing import StandardScaler

# Set dataset choice: 1 for California Housing, 2 for Breast Cancer, 3 for Syntheti
dataset_choice = 1  # Change this to 1, 2, or 3 to switch datasets

if dataset_choice == 1:
    ##################### DATA 1: California Housing ####################
    # Load California housing dataset and create a DataFrame
    housing = fetch_california_housing()
    data = pd.DataFrame(housing.data, columns=housing.feature_names)
    data['MedHouseVal'] = housing.target

    # Create a binary target variable based on the median house value
    data['High_MedHouseVal'] = (data['MedHouseVal'] > data['MedHouseVal'].median())

    # Select predictors and the target
    X = data.drop(columns=['MedHouseVal', 'High_MedHouseVal'])
    y = data['High_MedHouseVal']

elif dataset_choice == 2:
    ##################### DATA 2: Breast Cancer ####################
    # Load Breast Cancer dataset and create a DataFrame
    cancer = load_breast_cancer()
    data = pd.DataFrame(cancer.data, columns=cancer.feature_names)
    data['Target'] = cancer.target

    # Select predictors and the target
    X = data.drop(columns=['Target'])
    y = data['Target']

elif dataset_choice == 3:
    ##################### DATA 3: Synthetic Data ####################
    # Generate a synthetic binary classification dataset
    X, y = make_classification(n_samples=400, n_features=100, n_classes=2, random_s

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Now X_scaled and y are ready to use for modeling


# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, ran
```

In [11]:

```
# ========================================================
# Standardize the features
# ========================================================
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)

# ========================================================
# Fit classification models: Logistic, LDA, QDA, Naive Bayes
# ========================================================
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "LDA": LinearDiscriminantAnalysis(),
    "QDA": QuadraticDiscriminantAnalysis(),
    "Gaussian Naive Bayes": GaussianNB(),
}

results = []
roc_curves = {}

for name, model in models.items():
    model.fit(X_train, y_train)

    # Get predicted probabilities
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    else:
        scores = model.decision_function(X_test)
        y_proba = (scores - scores.min()) / (scores.max() - scores.min())

    # Binary predictions
    y_pred = (y_proba >= 0.5).astype(int)

    # Error rate and AUC
    error_rate = np.mean(y_pred != y_test)
    auc_score = roc_auc_score(y_test, y_proba)

    # ROC curve data
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_curves[name] = (fpr, tpr, auc_score)

    results.append({
        "Model": name,
        "Error Rate": error_rate,
        "AUC": auc_score
    })

# ========================================================
# Display Results Table
# ========================================================
results_df = pd.DataFrame(results)
```

```python
print("\n=== Model Performance (Test Set) ===")
print(results_df.to_string(index=False))

# ============================================================
# Plot ROC Curves
# ============================================================
plt.figure(figsize=(8, 6))
for name, (fpr, tpr, auc_score) in roc_curves.items():
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.3f})")

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves: Logistic, LDA, QDA, GaussianNB")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
=== Model Performance (Test Set) ===
               Model  Error Rate      AUC
 Logistic Regression    0.172804 0.910156
                 LDA    0.183140 0.903485
                 QDA    0.261789 0.896440
Gaussian Naive Bayes    0.278908 0.840030
```



Interpretation

The Gaussian Naive Bayes classifier achieved the highest performance with an AUC of 0.889 and the lowest error rate of 0.20 on this synthetic dataset. This strong result likely reflects that the data generation process aligns well with the Gaussian distribution assumption, and that the independence assumption of Naive Bayes did not introduce significant bias in this case.

Linear Discriminant Analysis (LDA) and Logistic Regression produced similar results, with AUC values around 0.76–0.77, although LDA performed slightly better in both AUC and error rate.

Quadratic Discriminant Analysis (QDA) showed the weakest performance, with an AUC of approximately 0.59 and an error rate of about 0.43. This poor result may be due to unstable per-class covariance estimates—particularly when the number of features is large relative to the sample size—or to overfitting when class covariance structures are similar.