Instructions

To do this assignment, save a copy to your own Google drive by going to:

> File --> Save a copy in Drive

Before working on this notebook, please disable inline AI suggestions by going to:

> Settings (top right gear icon) --> AI assistance --> Show AI-powered inline completions (uncheck)

As stated in the syllabus, you're welcome to use AI as an aid, but your answers should be your own.

Submission instructions:

- **Due**: Tuesday 10/7 at 11:59PM AOE
- **Submission**: Turn in both a printed **PDF** and the **editable share link** on MyCourses.

Remember that to make the share link editable, you must **convert the notebook to "anyone with a link can edit"** before copying and pasting the link.

Problem 1

This problem examines the differences between LDA and QDA. Please answer the questions and give a brief justification for your answer.

Problem 1.1

If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

If the Bayes decision boundary is linear, we'd expect QDA to perform better on the training set since its decision boundary is quadratic and it can more cloesly fit the training data, but for the same reason it would perform worse on the test set due to overfitting. Given that the true decisiion boundary is linear LDA would perform better since it produces a linear decision boundary.

Problem 1.2

If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

If the true decision boundary is non-linear we'd expect QDA to perform better on both the training and test set's because it produces a quadratic decision boundary which can better fit the training data and can represent non-linear decision boundaries.

Problem 1.3

In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?

As the sample size increases, we'd expect the accuracy of QDA to improve relative to LDA. With a small n QDA is more sensative to variance and can over fit, but as n increases the impact of variance on QDA decreases and QDA can more closely fit the true decision boundary.

Problem 1.4

True or False: Even if the Bayes decision boundary for a given problem 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. Justify your answer.

False:

While QDA is more flexible than LDA, that flexibility causes QDA to be more sensitive to variance, causing overfitting. If the true decision boundary is linear the best QDA can do is an equal error rate to LDA.

Problem 2

Using the `Boston` data set, fit classification models in order to predict whether a given census tract has a crime rate above or below the median. You will want to use Scikit-Learn for this.

Hint: This page discusses how to download the `Boston` dataset (and discusses some ethical problems with it that we are going to ignore for our purely pedagogical purposes).

Hint: This is a decent example of how to train/compare multiple models on the same dataset using Scikit-Learn. Although I personally am not familiar with the `make_classification` API.

Scikit-Learn implementations of the various models you have learned so far:

- Logistic regression
- LDA
- Naive Bayes
- KNN

Problem 2.1

Explore logistic regression, LDA, naive Bayes, and KNN models using various (at least 3) subsets of the predictors. Describe your findings.

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\\s+", skiprows=22, header=None)
```

```
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
columns = [
  "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE",
  "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"
]
X = pd.DataFrame(data, columns=columns)
median_crime = np.median(X["CRIM"])
y_binary=(X["CRIM"] > median_crime).astype(int)
X = X.drop(columns=["CRIM"])
X_train, X_test, y_train, y_test = train_test_split(
  X, y_binary, test_size=0.3, random_state=42
)

models = {
  "LDA": LinearDiscriminantAnalysis(),
  "Naive Bayes": GaussianNB(),
  "KNN (k=5)": KNeighborsClassifier(n_neighbors=5)
}
i = 0
for name, model in models.items():
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)

  acc = accuracy_score(y_test, y_pred)
  print(f"{name}: Test Accuracy = {acc:.3f}")
```

```
LDA: Test Accuracy = 0.829
Naive Bayes: Test Accuracy = 0.842
KNN (k=5): Test Accuracy = 0.888
```

Problem 2.2

Write code that computes the log-likelihood of the `Boston` dataset for the fitted coefficients β of *logistic regression only*. You should write the code yourself and *not* use a library that does the work for you.

Hint: You will have to create the response variable yourself using the variables that are contained in the Boston data set.

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)
beta = [logreg.intercept_[0]]
dot_prod = np.dot(X_test_scaled, logreg.coef_[0]) + logreg.intercept_[0]
p = 1 / (1 + np.exp(-dot_prod))
epsilon = 1e-9
p = np.clip(p, epsilon, 1 - epsilon)
log_likelihood = np.sum(y_test * np.log(p) + (1 - y_test) * np.log(1 - p))
print(f"{log_likelihood:.3f}")
```

```
-41.061
```

Problem 3

Hint: You can follow the slides or the LAR reference from the class website. See the class website for some recommended linear algebra references.

You will derive the formula used to compute the solution to *linear regression*. The objective in linear regression is:

$$f(\beta) = \|y - A\beta\|_2^2$$

Here, β is the vector of coefficients that we want to optimize, A is the design matrix, y is the target. The notation $\|\cdot\|_2$ represents the Euclidean (or L_2) norm.

Our goal is to find β that solves:

$$\min_{\beta} f(\beta)$$

Follow the next steps to compute it.

Note: We typically use $\hat{y} = f(X)$ to represent the model itself, but now we are using $f(\beta) = \|y - A\beta\|_2^2$ to represent the **error** of the trained model with respect to the training data, as a function of the model parameters β. Don't confuse the two different uses of f.

Problem 3.1

Express the linear regression objective $f(\beta)$ in terms of linear and quadratic terms.

$\|v\|_2^2 = v^\top v$

$f(\beta) = (y - \mathbf{A}\beta)^\top (y - \mathbf{A}\beta)$

$f(\beta) = y^\top y - y^\top \mathbf{A}\beta + \beta^\top \mathbf{A}^\top y + \beta^\top \mathbf{A}^\top \mathbf{A}\beta$

Problem 3.2

Derive the gradient: $\nabla_\beta f(\beta)$ using the linear and quadratic terms above.

$\nabla_\beta f(\beta) = (\nabla_\beta [\beta^\top (A^\top A)\beta]) - (\nabla_\beta [-2\beta^\top A^\top y]) + (\nabla_\beta [y^\top y])$

$\nabla_\beta [\beta^\top (A^\top A)\beta] = 2(A^\top A)\beta$

$\nabla_\beta [-2\beta^\top A^\top y] = -2(A^\top y)$

$\nabla_\beta [y^\top y] = 0$

$\nabla_\beta f(\beta) = 2(A^\top A)\beta - 2(A^\top y)$

Problem 3.3

Since f is convex, its minimal value is attained when

$$\nabla_\beta f(\beta) = 0$$

Derive the expression for the β that satisfies the inequality above. You can adapt the derivation of the similar formula for linear regression from the slides.

$\nabla_\beta f(\beta) = 0$

$2(A^\top A)\beta - 2(A^\top y) = 0$

$A^\top A\beta - A^\top y = 0$

$(A^\top A)\beta = A^\top y$

$\beta = \frac{A^\top y}{A^\top A}$

Problem 3.4

Implement the algorithm for computing β and use it on a small dataset of your choice. Do not forget about the intercept.

```
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 3, 5, 7, 11])
A = np.hstack([np.ones((X.shape[0], 1)), X])
A
beta = np.linalg.inv(A.T @ A) @ (A.T @ y)
print("B0, B1:", beta)
```

```
B0, B1: [-1.   2.2]
```

Problem 3.5

Compare your solution with an standard implementation of linear regression such as `lm`. Are the results the same? Why yes, or no?

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: -1.0000000000000018
Coefficients: [2.2]
```