Drawing the Line: Fisher's Linear Discriminant vs. Perceptron

Clara Wong

wongc148

400372818

IBEHS 4QZ3

Submitted on: November 9th, 2025

Academic Integrity Statement

As a future member of the engineering profession, I am responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

Executive Summary

Using the Kaggle diabetes dataset and two previously selected features (BMI and glucose), two classical linear classifiers (Fisher's Linear Discriminant (FLD) and the Perceptron) were implemented to compare their predictive performance for diabetes classification. Both models were first manually implemented in Python to explore their differences in learning approaches and visualize the resulting decision boundaries. The FLD computes a projection, aiming to maximize the separation of between-class variance and in-class variance. On the other hand, the perceptron is an iterative algorithm that adjusts weights and bias values when samples are misclassified, with the pocket perceptron keeping the best-performing weights during training.

Based on training results, the FLD had an empirical error of 26.17% compared to 22.66% error for the pocket perceptron. This suggests moderate overlap between diabetic and non-diabetic cases in the glucose-BMI feature space. When validating the models on the test data, the FLD model achieved approximately 69% accuracy, whereas the pocket perceptron model achieved approximately 71%, indicating that both models had similar overall performance. The FLD had higher sensitivity (0.69), meaning it correctly classified more diabetic patients; the perceptron had higher specificity (0.79), meaning it had better success at identifying non-diabetic cases. The accuracy, sensitivity, and specificity values highlight how both models misclassify a significant number of patients and, in turn, demonstrate the limited ability to predict diabetes based on glucose and BMI alone.

In a biomedical context, misclassifications hold severe implications. False Negatives (undiagnosed diabetes in this case) can delay treatment and impact outcome, while False Positives (misdiagnosing diabetes) can lead to unnecessary testing and healthcare costs. These misclassifications highlight the importance of mitigating data and model bias, especially in healthcare, where unrepresentative datasets can heighten biases and disparities that already exist in the healthcare system. Overall, the results of the FLD and perceptron models demonstrate their limited predictive value, underscoring the need for more representative data and models capable of classifying non-linear data.

Manual Implementation FLD

In this section, w and threshold b were computed, the samples were projected, and the training data were classified. From assignment 1, the two selected features were BMI and glucose. First, the feature data were extracted, and any entries with BMI or glucose values of 0 were removed. The features were then combined into a feature matrix, and the within-class and total within-class scatter matrices were computed:

```
# Read the diabetes dataset
df = pd.read_csv('diabetes.csv')
selected_columns = df[['Glucose', 'BMI', 'Outcome']]

# Remove rows where BMI or Glucose is 0
selected_columns = selected_columns[(selected_columns['Glucose'] != 0) &
(selected_columns['BMI'] != 0)]

glucose = selected_columns['Glucose'].values
bmi = selected_columns['BMI'].values
outcome = selected_columns['Outcome'].values

# Combine features into a feature matrix
X = np.column_stack((glucose, bmi))

# Split data into two classes
X_A = X[outcome == 0]  # Non-diabetic
X_B = X[outcome == 1]  # Diabetic

# Compute within-class scatter matrices
S_A, mu_A_hat = scatter_matrix(X_A)
S_B, mu_B_hat = scatter_matrix(X_B)

# Total within-class scatter matrix
S_W = S_A + S_B
```

The fisher_direction function is then used to compute w:

- **delta = mu_B - mu_A**: finds the difference between the class means
- **w = np.linalg.solve(S_W, delta)**: multiply by the inverse of $S_W \rightarrow w = S_w^{-1}(\mu_B - \mu_A)$
- **w = w / np.linalg.norm(w):** scales *w* to unit length

```
def fisher_direction(mu_A, mu_B, S_W):
    """Return a direction proportional to S_W^{-1}(mu_B - mu_A)."""
```

```
    delta = mu_B - mu_A
    w = np.linalg.solve(S_W, delta)
    return w
# Compute Fisher's direction w
w = fisher_direction(mu_A_hat, mu_B_hat, S_W)
w = w / np.linalg.norm(w)
```

The fisher_threshold function was used to compute the threshold b:

- **mA = float(w @ mu_A)**: projects class means onto w (m_A' = $w^T u_A$)
- When class priors are equal (both classes are equally likely), the threshold is the midpoint
- b = -t converts the threshold into the intercept form $w^T x + b = 0$

```
def fisher_threshold(w, mu_A, mu_B, priors_equal=True):
    """Midpoint threshold on the 1-D projection when priors are equal."""
    mA = float(w @ mu_A)
    mB = float(w @ mu_B)
    t = 0.5 * (mA + mB) if priors_equal else None
    return t
# Compute threshold b
t = fisher_threshold(w, mu_A_hat, mu_B_hat, priors_equal=True)
b = -t
```

The classify function was used to classify the training data:

- Mathematically, the decision boundary is: $w^T x + b = 0$. If the data point is greater than or equal to 0, then it is in class B (diabetic); otherwise, the data point is in class A (non-diabetic).
- pred takes the array of predictions (eg. ['A', 'B', 'B', 'A', …]) and turns them into labels of 0/1 values
- A comparison between the pred and outcome elements is completed, and the average value of the boolean array is used to compute the accuracy.

```
def classify(w, b, x):
    return "B" if (w @ x + b) >= 0 else "A"
# Classify training samples
predictions = np.array([classify(w, b, x) for x in X])
pred = (predictions == "B").astype(int)
accuracy = np.mean(pred == outcome) * 100
```

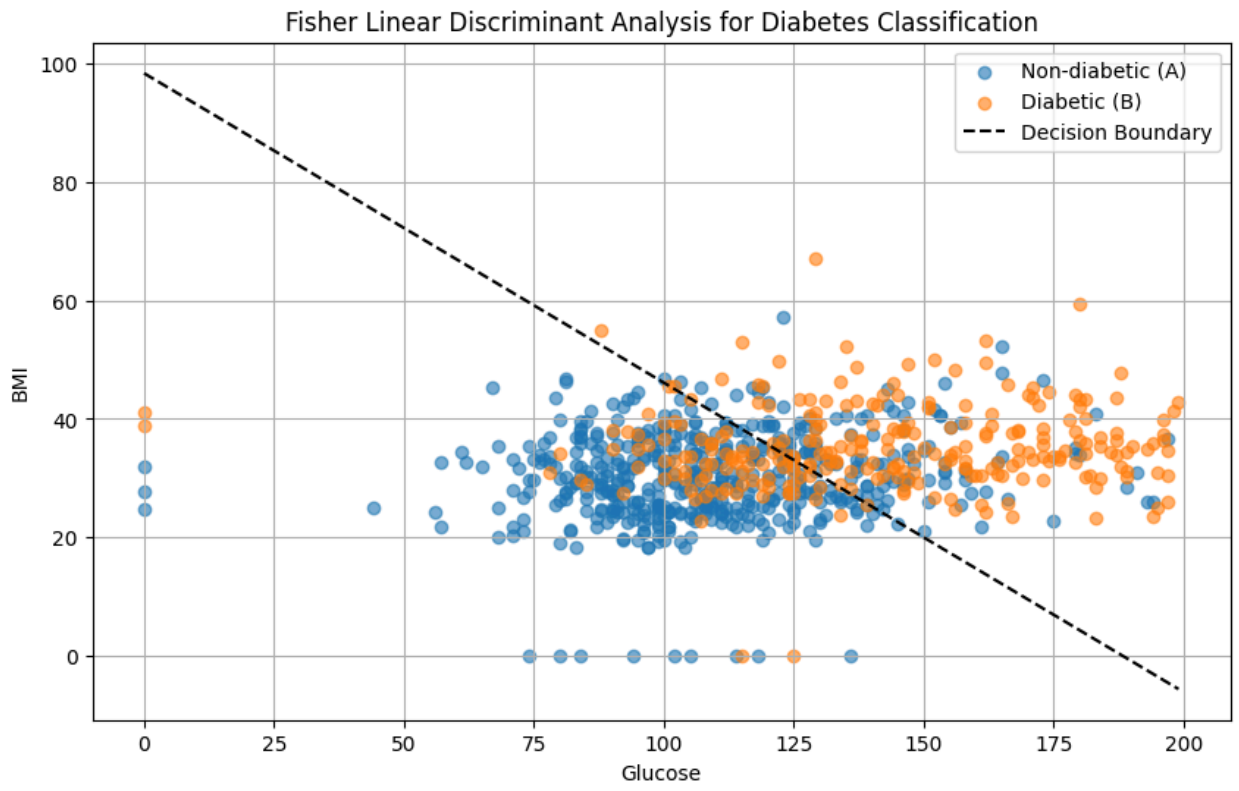

Figure 1. Plot of the decision boundary for diabetes classification

Manual Implementation Perceptron

In this section, the vanilla and pocket perceptron with a learning rate of 1 were implemented.

The vanilla perceptron was implemented to find the final training parameters and accuracy after 500 epochs:

- Bias and weights were initialized to zero. mistakes_per_epoch kept track of the number of misclassifications in each training cycle, which was used to create *Figure 3*.
- For each sample **(x, y)**, the predicted label **yhat** is computed using the current weights and bias.
- If the same is misclassified (yhat is not equal to y), the algorithm updates the bias and weights, moving the decision boundary toward correctly classifying the misclassified sample. The update rules are: $w \leftarrow w + \eta yx$, $b \leftarrow b + \eta y$

```python
def vanilla_perceptron(data, learning_rate, num_epochs):
    b, w1, w2 = 0.0, 0.0, 0.0
    mistakes_per_epoch = []

    for epoch in range(num_epochs):
        mistakes = 0
        for (x, y) in data:
            yhat, _ = predict(x[0], x[1], b, w1, w2)
            if yhat != y:
                # Update weights
                b += learning_rate * y
                w1 += learning_rate * y * x[0]
                w2 += learning_rate * y * x[1]
                mistakes += 1
        mistakes_per_epoch.append(mistakes)
        print(f"Epoch {epoch+1}: {mistakes} misclassified")
        if mistakes == 0:
            break
    acc = accuracy(data, b, w1, w2)
    print(f"new params: b={b:.3f}, w1={w1:.3f}, w2={w2:.3f}")
    print(f"Final Vanilla accuracy: {acc*100:.2f}%")
    return b, w1, w2, mistakes_per_epoch
```

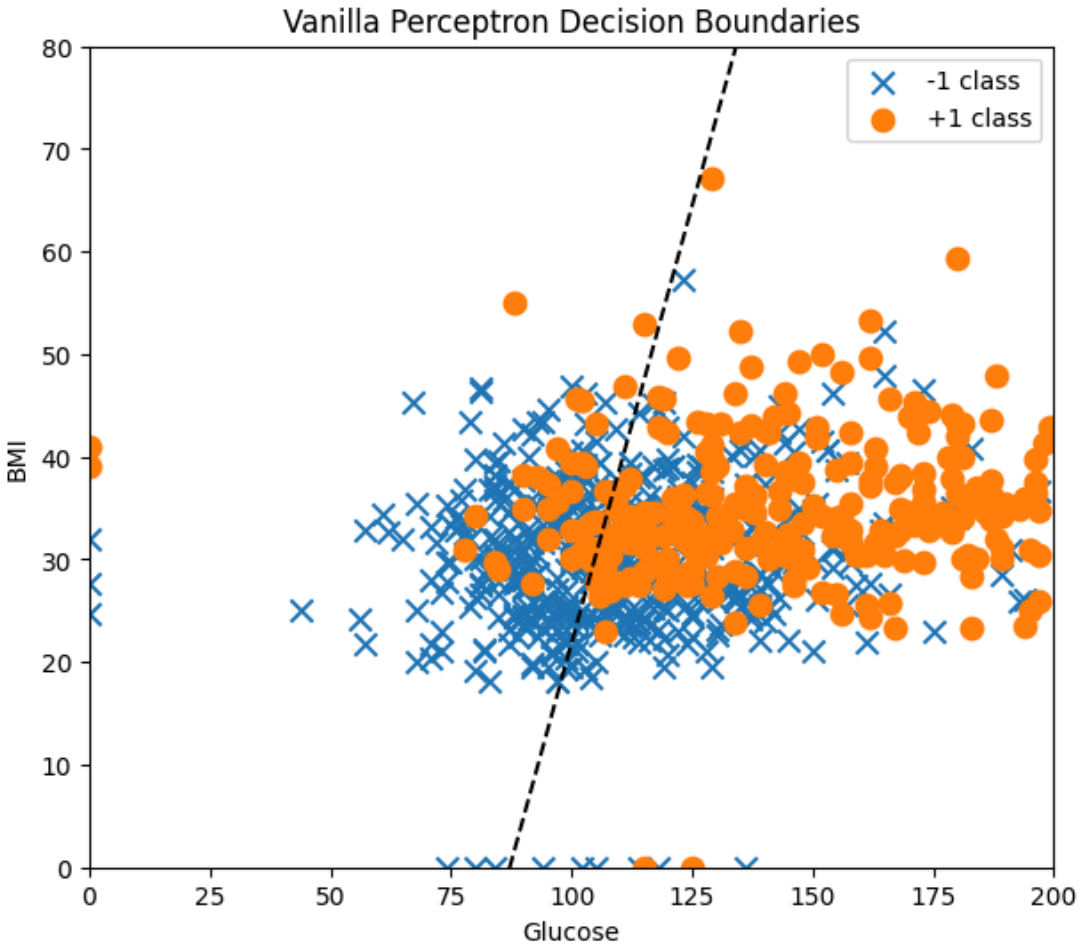

Figure 3. Vanilla perceptron decision boundary.

The pocket perceptron was implemented to find the best parameters over the course of 500 epochs:

- In addition to what was initialized for the vanilla perceptron, variables for the best-performing weights thus far are also initialized (best_b, best_w1, best_w2)
- The update rules are the same as the vanilla perceptron. After each epoch, the accuracy of the new parameters is checked (**current_acc = accuracy(data, b, w1, w2)**). If the current accuracy is better than the previous best, then the parameters are updated with the new best solution. The algorithm stops early if the model perfectly classifies all samples (**if mistakes == 0**).

```
def pocket_perceptron(data, learning_rate, num_epochs):
    b, w1, w2 = 0.0, 0.0, 0.0
    best_b, best_w1, best_w2 = b, w1, w2
    best_acc = accuracy(data, b, w1, w2)
    mistakes_per_epoch = []
```

```
    for epoch in range(num_epochs):
        mistakes = 0
        for (x, y) in data:
            yhat, _ = predict(x[0], x[1], b, w1, w2)
            if yhat != y:
                # Update weights
                b += learning_rate * y
                w1 += learning_rate * y * x[0]
                w2 += learning_rate * y * x[1]
                mistakes += 1

                current_acc = accuracy(data, b, w1, w2)
                if current_acc > best_acc:
                    best_acc = current_acc
                    best_b, best_w1, best_w2 = b, w1, w2

        mistakes_per_epoch.append(mistakes)
        print(f"Epoch {epoch+1}: {mistakes} misclassified")
        if mistakes == 0:
            break

    print(f"Final Pocket accuracy: {best_acc*100:.2f}%")
    print(f"new params: b={best_b:.3f}, w1={best_w1:.3f}, w2={best_w2:.3f}")
    return best_b, best_w1, best_w2, mistakes_per_epoch
```

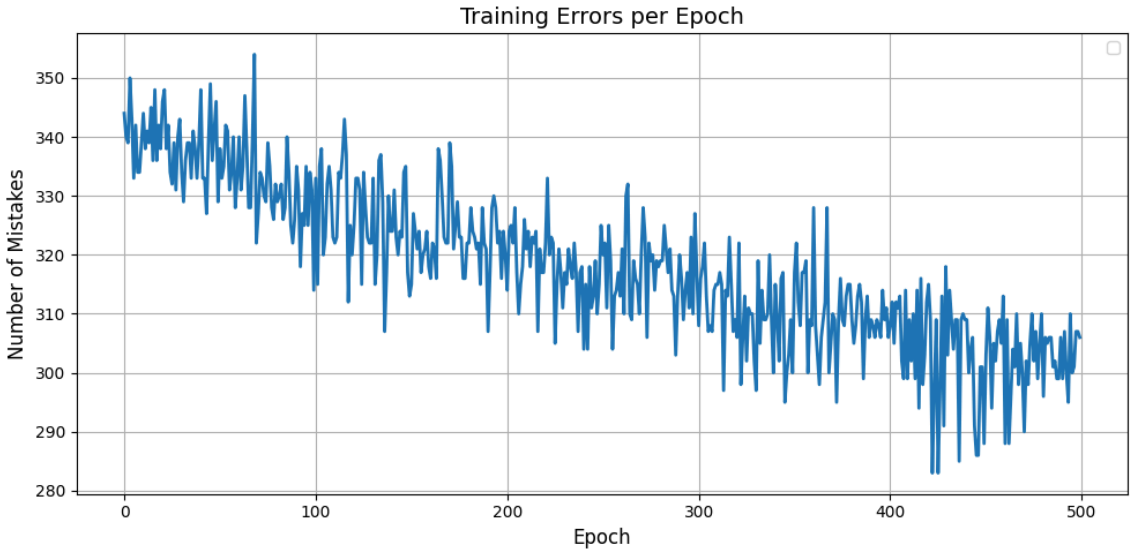

Figure 2. Training errors per epoch.

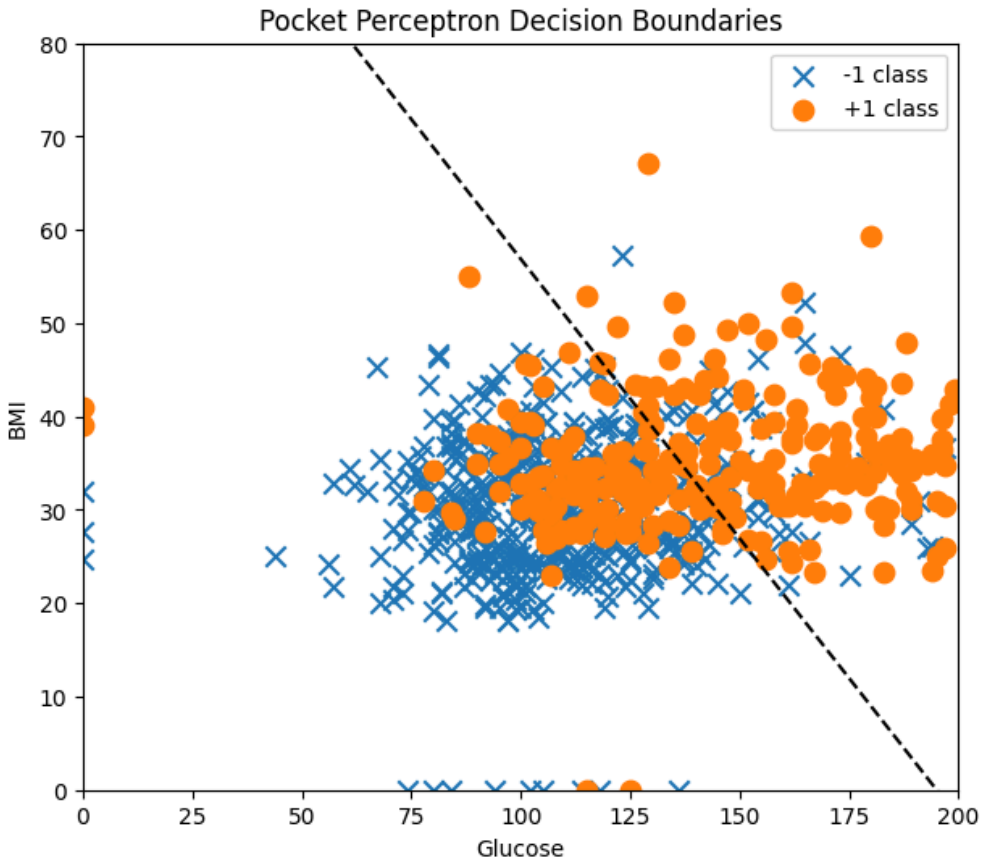

Figure 4. Pocket perceptron decision boundary.

Training Results

The Fisher LDA had an empirical error of 26.17% and the pocket perceptron had an empirical error of 22.66% (*Figure 9)*, indicating the number of patients incorrectly classified in each model. Although the perceptron has a slightly better empirical error, both models show significant misclassifications. The confusion matrices in *Figure 10* also demonstrate that the Fisher model is more likely to misclassify non-diabetic patients as diabetic (more False Positives), while the perceptron model is more likely to misclassify diabetic patients as non-diabetic (more False Negatives).

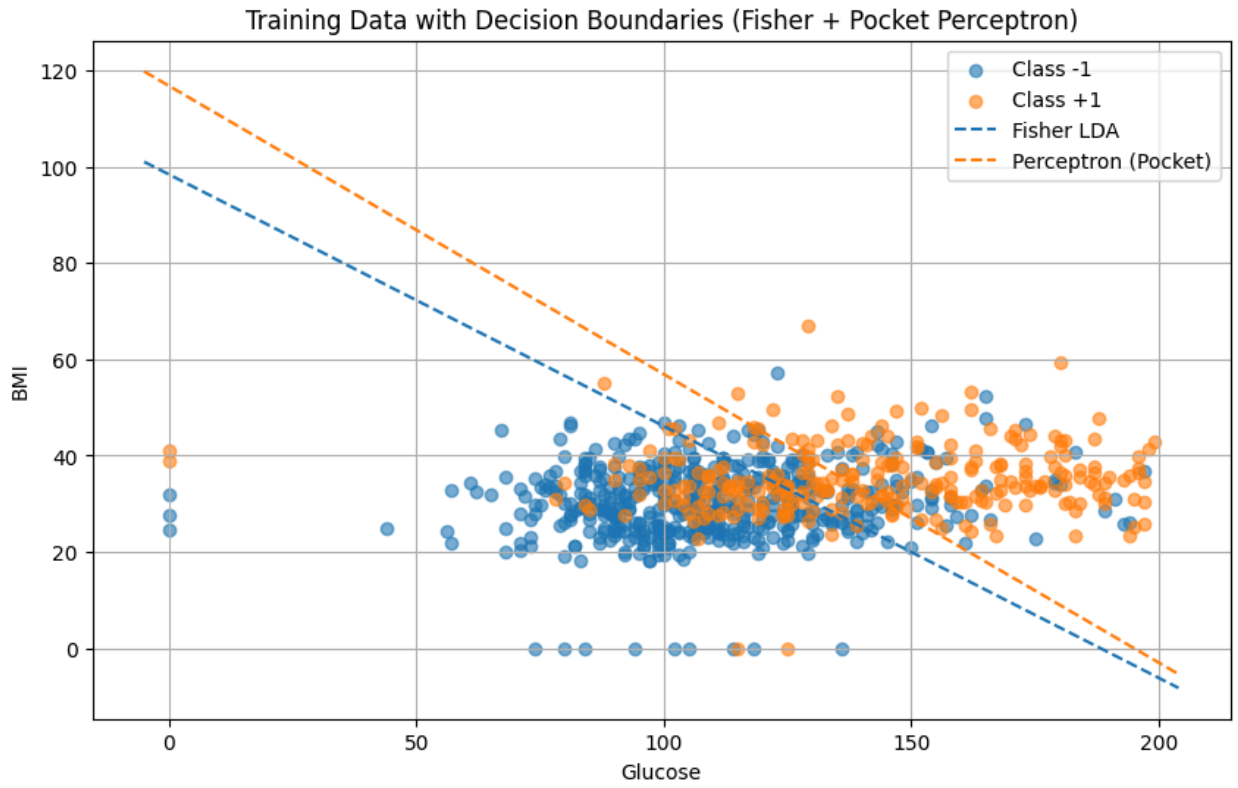

Figure 8. Training data with decision boundaries for Fisher and Pocket perceptron

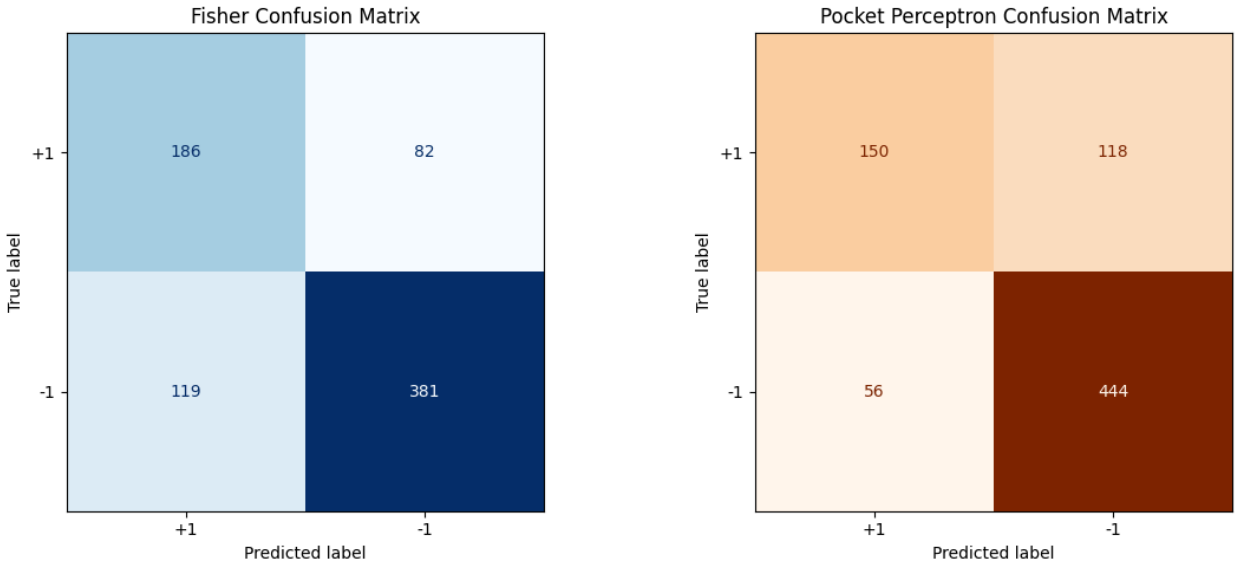

Figure 10. Confusion matrices for Fisher and Pocket Perceptron

Validation Results

To fully compare the Fisher and Pocket Perceptron algorithms, the two models were first trained on training data, then their performance was evaluated on test data.

First, the dataset was split into train/test datasets (80/20 split based on LSD-1 assignment):

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Then, the Fisher and perceptron models were trained on the training set:

```
data = list(zip(X_train, y_train))
b_p, w1_p, w2_p, mistakes = pocket_perceptron(data, learning_rate, num_epochs)
w_f, b_f, X_A_train, X_B_train = fisher_train(X_train, y_train)
```

After, the Fisher and perceptron predict functions apply each model to the test set and output predicted class labels for the test data:

```
yhat_f = fisher_predict(X_test, w_f, b_f)
yhat_p = perceptron_predict(X_test, [w1_p, w2_p], b_p)
```

Finally, the accuracy, sensitivity, and specificity were calculated by comparing the predicted test set outcomes with the actual test set outcomes. Sensitivity describes the model's ability to avoid misclassifying diabetic patients as non-diabetic, while specificity demonstrates the model's ability to avoid misclassifying non-diabetic patients as diabetic.

```
Final Pocket accuracy: 78.83%
new params: b=-11028.000, w1=59.000, w2=96.800
Fisher Test Accuracy: 68.83%, Sensitivity: 0.69, Specificity: 0.69
Perceptron Test Accuracy: 70.78%, Sensitivity: 0.56, Specificity: 0.79
```

Figure 11. Accuracy, sensitivity, and specificity results for the Fisher and Perceptron models on the test dataset.

Visualization of Projection and Learning

To plot the 1D Fisher Projection with class histograms, each data point was first projected onto the Fisher direction:

```
proj = X_train @ w_fisher + b_fisher
```

The projections were then split into diabetic and non-diabetic classes. A threshold was calculated by finding the midpoint between the mean projections of both classes:

```
proj_class_neg = proj[y_train == -1]
proj_class_pos = proj[y_train == 1]
thresh = (np.mean(proj_class_neg) + np.mean(proj_class_pos)) / 2
```

Figure 12 demonstrates the produced histogram. It shows the distribution of projected values for each class, as well as the threshold line/Fisher decision boundary. Ideally, the histogram for both classes will have minimal overlap, demonstrating a clear decision boundary for classification. However, there is significant overlap between the diabetic/non-diabetic classes, meaning that there is poor linear separation and that glucose and BMI alone are not enough to determine if a patient has diabetes.

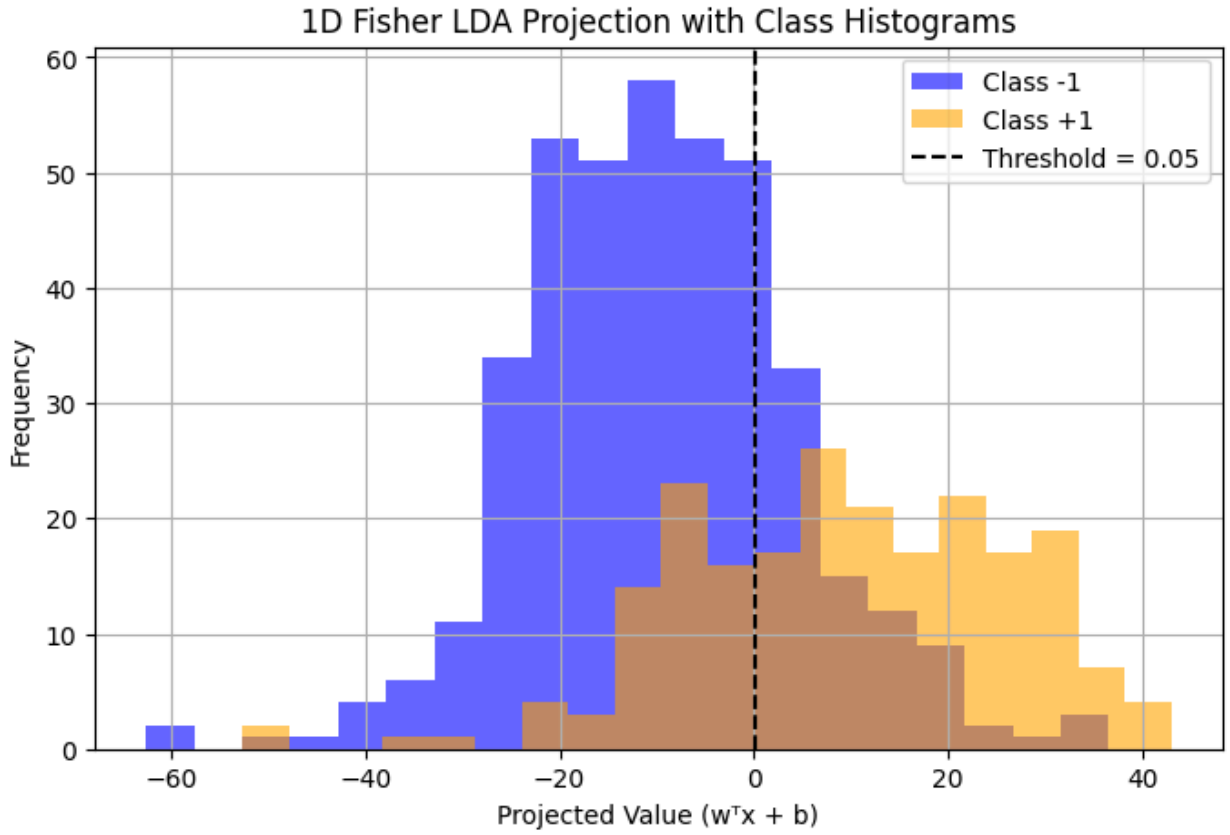

Figure 12. 1-D FLD projections with class histograms and threshold

The number of misclassifications per epoch for the training data was documented in the Validation Results section and then plotted as shown in *Figure 13.* Overall, the number of mistakes per epoch trends downwards, demonstrating that the perceptron model improves overall with each update. However, it does not reach zero mistakes per epoch, indicating that the diabetic/non-diabetic classes cannot be linearly separated in 500 epochs with the current algorithm and learning rate.

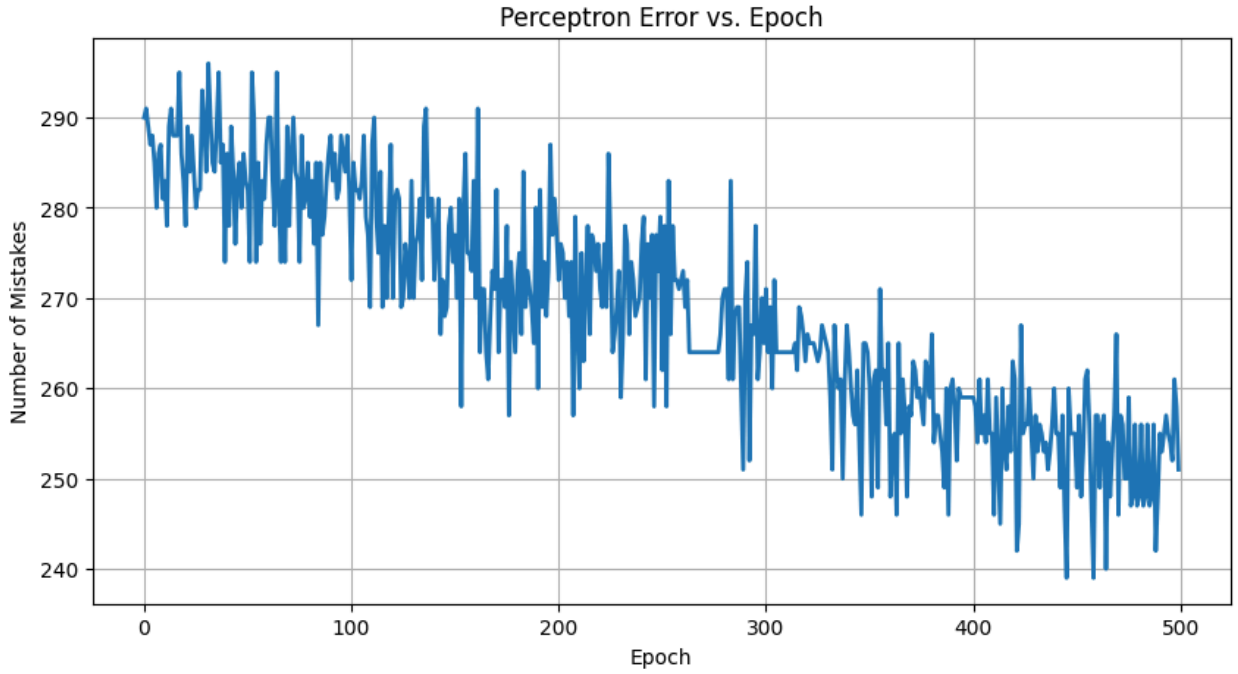

Figure 13. Perceptron Error vs. Epoch for the training data.

Comparison and Interpretation

Learning Behaviours

Fisher Linear Discriminant (FLD) and the Perceptron are both classical linear classifiers. But they differ in how each model adapts to the data, their optimization goals, and how they handle non-linearly separable data. The FLD aims to find a projection that maximizes the ratio of between-class variance to within-class variance. In contrast, the perceptron finds a hyperplane through iterative weight adjustments based on misclassifications during training. Therefore, perceptrons and FLD can have comparable performance on linearly separable classification tasks. Since the pocket perceptron keeps the best-performing weights, the pocket perceptron tends to outperform FLD on non-linear or multi-clustered data, which is typical of biomedical datasets. In *Figure 11*, the perceptron has a slightly higher accuracy on the test data at ~71% as opposed to the ~69% accuracy of the FLD model. This suggests a moderate separation between diabetic and non-diabetic cases, but there is some overlap in the glucose-BMI space, leading to misclassifications. The perceptron test accuracy also has better specificity (0.79 vs 0.69 for FLD), suggesting the computed boundary is better at identifying non-diabetic cases, but at the cost of missing diabetic cases (has worse sensitivity — 0.56 vs 0.69 for FLD). Overall, the accuracy, sensitivity, and specificity values demonstrate that diabetes also depends on non-linear interactions and additional factors other than glucose and BMI, such as blood pressure, age, and family history.

Biomedical Implications of Misclassification

Misclassifications can mean significant clinical implications. As indicated by the moderate sensitivity and specificity values (e.g. 0.79 specificity and 0.56 sensitivity values for perceptron), a significant number of patients are being misclassified. Sensitivity is the proportion of patients who have diabetes and are classified correctly. The low sensitivity value (0.56 for the perceptron) means that 44% of patients will be classified as non-diabetic when they are actually diabetic. This has serious implications, as the lack of an accurate diagnosis might delay their treatment. On the other hand, specificity is the proportion of patients without diabetes who are classified correctly. The moderate specificity value (0.79) has less severe biomedical implications, although misclassifying a patient as diabetic leads to unnecessary stress and testing.

Responsible AI

In a medical context, linear models are susceptible to both data and model bias due to the non-linear nature between factors and diseases. This is demonstrated in this assignment, where diabetes depends on other non-linear factors and interactions aside from glucose and BMI, which is why the FLD and perceptron models were not at all close to achieving perfect accuracy. The implications behind data and model bias are severe because they worsen existing disparities in the healthcare system and can result in misdiagnoses and/or inappropriate treatment recommendations. For one, models are trained on historically biased data (e.g. caucasian males) and may give incorrect assessments on underrepresented groups, such as minorities or women, which reinforces existing healthcare inequalities. Additionally, a model developed and validated in a limited setting may fail to perform accurately in a real-world scenario with a different patient population or region where protocols and healthcare systems vary. To mitigate these two issues, the datasets the models are trained on must be large and representative of the target patient population's demographics (e.g. age, gender, ethnicity, socioeconomic status, etc.). Missing data should also be addressed, especially for variables that are hard to capture but relevant to health outcomes, such as social determinants of health. Moreover, consensus among multiple healthcare experts on the data is essential for reducing individual and cognitive biases when labelling the data. Integrating dashboards and visualizations to help clinicians understand potential biases also helps detect if a model is relying on biased factors [1].

References

[1] J. L. Cross, M. A. Choma, and J. A. Onofrey, “Bias in medical AI: Implications for clinical decision-making,” *PLOS digital health*, vol. 3, no. 11, p. e0000651, Nov. 2024, doi: 10.1371/journal.pdig.0000651.

Appendix

```
Fisher's Linear Discriminant Results:
w (unit): [0.46331806 0.88619207]
Threshold b: -87.2006
Training accuracy: 73.83%
```

Figure A1. W, b, and training accuracy of the Fisher Linear Discriminant (manual implementation of FLD)

```
new params: b=-11684.000, w1=134.000, w2=-78.600
Final Vanilla accuracy: 59.77%
```

Figure A2. Final vanilla perceptron accuracy with updated parameters (manual implementation of perceptron).

```
Final Pocket accuracy: 77.34%
new params: b=-6435.000, w1=33.000, w2=55.100
```

Figure A3. Final pocket perceptron accuracy with updated parameters (manual implementation of perceptron).

```
Empirical Error - Fisher: 26.17%
Empirical Error - Pocket Perceptron: 22.66%
```

Figure A4. Empirical error for Fisher and Pocket perceptron (training results).

Code

Manual Implementation of FLD

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def fisher_direction(mu_A, mu_B, S_W):
    """Return a direction proportional to S_W^{-1}(mu_B - mu_A)."""
    delta = mu_B - mu_A
    w = np.linalg.solve(S_W, delta)
    return w

def fisher_threshold(w, mu_A, mu_B, priors_equal=True):
    """Midpoint threshold on the 1-D projection when priors are equal."""
    mA = float(w @ mu_A)
    mB = float(w @ mu_B)
    t = 0.5 * (mA + mB) if priors_equal else None
    return t

def decision_boundary_line(w, t, x_span):
    """Return x,y for the 2D decision line w^T x = t over x in x_span; handle
vertical case."""
    if abs(w[1]) < 1e-12:
        x_line = np.full_like(x_span, t / w[0])
        y_line = np.linspace(-1.0, 3.0, len(x_span))
        return x_line, y_line, True  # vertical
    else:
        y = (t - w[0] * x_span) / w[1]
        return x_span, y, False

def scatter_matrix(X):
    """Compute scatter matrix S = sum_i (x_i - mu)(x_i - mu)^T for rows of X.
Returns (S, mu)."""
    mu = X.mean(axis=0)
    S = np.zeros((X.shape[1], X.shape[1]))
    for x in X:
        d = (x - mu).reshape(-1,1)
        S += d @ d.T
```

```
    return S, mu

def classify(w, b, x):
    return 1 if (w @ x + b) >= 0 else -1


def fisher_train(X, y):
    # Split data into two classes
    X_A = X[y == -1]  # Non-diabetic
    X_B = X[y == 1]  # Diabetic

    # Compute within-class scatter matrices
    S_A, mu_A_hat = scatter_matrix(X_A)
    S_B, mu_B_hat = scatter_matrix(X_B)

    # Total within-class scatter matrix
    S_W = S_A + S_B

    # Compute Fisher's direction w
    w_fisher = fisher_direction(mu_A_hat, mu_B_hat, S_W)
    w_fisher = w_fisher / np.linalg.norm(w_fisher)

    # Compute threshold b
    t = fisher_threshold(w_fisher, mu_A_hat, mu_B_hat, priors_equal=True)
    b_fisher = -t

    return w_fisher, b_fisher, X_A, X_B

def fisher_predict(X, w, b):
    return np.where(X @ w + b >= 0, 1, -1)

# Read the diabetes dataset
df = pd.read_csv('diabetes.csv')

# Extract only the glucose, BMI, and outcome columns
selected_columns = df[['Glucose', 'BMI', 'Outcome']]

glucose = selected_columns['Glucose'].values
bmi = selected_columns['BMI'].values
```

```
outcome = selected_columns['Outcome'].values

y = np.where(outcome == 0, -1, 1)

# Combine features into a feature matrix
X = np.column_stack((glucose, bmi))

w_fisher, b_fisher, X_A, X_B = fisher_train(X, y)

# Classify all training samples
predictions = np.array([classify(w_fisher, b_fisher, x) for x in X])

# Calculate accuracy
fisher_accuracy = np.mean(predictions == y) * 100

print(f"Fisher's Linear Discriminant Results:")
print(f"w (unit): {w_fisher}")
print(f"Threshold b: {b_fisher:.4f}")
print(f"Training accuracy: {fisher_accuracy:.2f}%")

# Visualize the results
plt.figure(figsize=(10, 6))
plt.scatter(X_A[:, 0], X_A[:, 1], label='Non-diabetic (-1)', alpha=0.6)
plt.scatter(X_B[:, 0], X_B[:, 1], label='Diabetic (1)', alpha=0.6)

# Plot decision boundary
x_span = np.linspace(min(glucose), max(glucose), 100)
x_line, y_line, is_vertical = decision_boundary_line(w_fisher, -b_fisher, x_span)
plt.plot(x_line, y_line, 'k--', label='Decision Boundary')

plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('Fisher Linear Discriminant Analysis for Diabetes Classification')
plt.legend()
plt.grid(True)
plt.show()
```

Manual Implementation of Perceptron

```
import numpy as np
```

```
import matplotlib.pyplot as plt

# parameters / init
learning_rate = 1
num_epochs = 500
AX = (0, 200, 0, 80)  # plot window (xmin, xmax, ymin, ymax)

# Read the diabetes dataset
df = pd.read_csv('diabetes.csv')

# Extract only the glucose, BMI, and outcome columns
selected_columns = df[['Glucose', 'BMI', 'Outcome']]

glucose = selected_columns['Glucose'].values
bmi = selected_columns['BMI'].values
outcome = selected_columns['Outcome'].values
outcome = np.where(outcome == 0, -1, 1)  # Convert to +1/-1

# Combine into a single dataset
data = list(zip(zip(glucose, bmi), outcome))

def predict(x1, x2, b, w1, w2):
    s = b + w1*x1 + w2*x2
    return (1 if s > 0 else -1), s

def plot_points(ax):
    x_n, y_n, x_p, y_p = [], [], [], []
    for (x, y) in data:
        (x_p if y==1 else x_n).append(x[0])
        (y_p if y==1 else y_n).append(x[1])
    ax.scatter(x_n, y_n, marker='x', s=80, label='-1')
    ax.scatter(x_p, y_p, marker='o', s=80, label='+1')

def plot_boundary(ax, b, w1, w2):
    """Draw line w1*x + w2*y + b = 0, clipped to axes."""
    x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
    if np.isclose(w1,0) and np.isclose(w2,0):
        return
    x_vals = np.linspace(x_min, x_max, 200)
```

```
    y_vals = -(b + w1*x_vals) / w2
    ax.plot(x_vals, y_vals, 'k--', label='Decision boundary')

def accuracy(data, b, w1, w2):
    correct = sum(1 for (x, y) in data if predict(x[0], x[1], b, w1, w2)[0] == y)
    return correct / len(data)

def vanilla_perceptron(data, learning_rate, num_epochs):
    b, w1, w2 = 0.0, 0.0, 0.0
    mistakes_per_epoch = []

    for epoch in range(num_epochs):
        mistakes = 0
        for (x, y) in data:
            yhat, _ = predict(x[0], x[1], b, w1, w2)
            if yhat != y:
                # Update weights
                b += learning_rate * y
                w1 += learning_rate * y * x[0]
                w2 += learning_rate * y * x[1]
                mistakes += 1
        mistakes_per_epoch.append(mistakes)
        print(f"Epoch {epoch+1}: {mistakes} misclassified")
        if mistakes == 0:
            break
    acc = accuracy(data, b, w1, w2)
    print(f"new params: b={b:.3f}, w1={w1:.3f}, w2={w2:.3f}")
    print(f"Final Vanilla accuracy: {acc*100:.2f}%")
    return b, w1, w2, mistakes_per_epoch

def pocket_perceptron(data, learning_rate, num_epochs):
    b, w1, w2 = 0.0, 0.0, 0.0
    best_b, best_w1, best_w2 = b, w1, w2
    best_acc = accuracy(data, b, w1, w2)
    mistakes_per_epoch = []

    for epoch in range(num_epochs):
        mistakes = 0
        for (x, y) in data:
```

```
            yhat, _ = predict(x[0], x[1], b, w1, w2)
            if yhat != y:
                # Update weights
                b += learning_rate * y
                w1 += learning_rate * y * x[0]
                w2 += learning_rate * y * x[1]
                mistakes += 1

                current_acc = accuracy(data, b, w1, w2)
                if current_acc > best_acc:
                    best_acc = current_acc
                    best_b, best_w1, best_w2 = b, w1, w2

        mistakes_per_epoch.append(mistakes)
        print(f"Epoch {epoch+1}: {mistakes} misclassified")
        if mistakes == 0:
            break

    print(f"Final Pocket accuracy: {best_acc*100:.2f}%")
    print(f"new params: b={best_b:.3f}, w1={best_w1:.3f}, w2={best_w2:.3f}")
    return best_b, best_w1, best_w2, mistakes_per_epoch

def show_update(update_num, epoch, i, x, y, b, w1, w2):
    fig, ax = plt.subplots(figsize=(4.2, 4.2))
    ax.set_title(f"After update {update_num}\n(epoch={epoch}, sample={i}, x={x},
y={y})")
    ax.set_xlim(AX[0], AX[1]); ax.set_ylim(AX[2], AX[3])
    ax.set_xlabel('x1'); ax.set_ylabel('x2')
    plot_points(ax); plot_boundary(ax, b, w1, w2)
    ax.legend(loc='upper left'); plt.show()

def perceptron_predict(X, w, b):
    w = np.array(w)
    return np.where(X @ w + b >= 0, 1, -1)

# Train both perceptrons
b_v, w1_v, w2_v, vanilla_mistakes = vanilla_perceptron(data, learning_rate,
num_epochs)
```

```
b_p, w1_p, w2_p, pocket_mistakes = pocket_perceptron(data, learning_rate,
num_epochs)

plt.figure(figsize=(10, 5))
plt.plot(vanilla_mistakes, linewidth=2)

# plot training errors per epoch
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Number of Mistakes', fontsize=12)
plt.title('Training Errors per Epoch', fontsize=14)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot vanilla decision boundary
fig, ax = plt.subplots(figsize=(7, 6))
ax.set_xlim(AX[0], AX[1]); ax.set_ylim(AX[2], AX[3])
plot_points(ax)
plot_boundary(ax, b_v, w1_v, w2_v)
ax.legend(['-1 class', '+1 class'])
ax.set_xlabel("Glucose")
ax.set_ylabel("BMI")
ax.set_title("Vanilla Perceptron Decision Boundaries")
plt.show()

# Plot pocket decision boundary
fig, ax = plt.subplots(figsize=(7, 6))
ax.set_xlim(AX[0], AX[1]); ax.set_ylim(AX[2], AX[3])
plot_points(ax)
plot_boundary(ax, b_p, w1_p, w2_p)
ax.legend(['-1 class', '+1 class'])
ax.set_xlabel("Glucose")
ax.set_ylabel("BMI")
ax.set_title("Pocket Perceptron Decision Boundaries")
plt.show()
```

Training Results

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
y_train = outcome
X_train = np.column_stack((glucose, bmi))

# Function to plot decision boundary
def plot_decision_boundary(w, b, ax, label):
    """w can be 2-element array or tuple, b is bias."""
    x_vals = np.linspace(min(glucose)-5, max(glucose)+5, 200)

    if len(w) == 2:
        # General line: w1*x + w2*y + b = 0 => y = -(b + w1*x)/w2
        if np.isclose(w[1],0):
            # Vertical line
            ax.axvline(x=-b/w[0], linestyle='--', label=label)
        else:
            y_vals = -(b + w[0]*x_vals)/w[1]
            ax.plot(x_vals, y_vals, linestyle='--', label=label)

# Prepare figure
plt.figure(figsize=(10,6))
plt.scatter(X_train[y_train==-1, 0], X_train[y_train==-1, 1], label='Class -1',
alpha=0.6)
plt.scatter(X_train[y_train==1,  0], X_train[y_train==1,  1], label='Class +1',
alpha=0.6)

# Plot Fisher decision boundary
plot_decision_boundary(w_fisher, b_fisher, plt.gca(), 'Fisher LDA')

# Plot Perceptron decision boundary
plot_decision_boundary([w1_p, w2_p], b_p, plt.gca(), 'Perceptron (Pocket)')

plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('Training Data with Decision Boundaries (Fisher + Pocket Perceptron)')
plt.legend()
plt.grid(True)
plt.show()

# Fisher predictions
```

```
yhat_fisher = np.array([1 if np.dot(w_fisher, x)+b_fisher >= 0 else -1 for x in
X_train])

# Perceptron predictions
yhat_perc = np.array([1 if np.dot([w1_p, w2_p], x)+b_p >= 0 else -1 for x in
X_train])

emp_error_fisher = np.mean(yhat_fisher != y_train)
emp_error_perc = np.mean(yhat_perc != y_train)

print(f"Empirical Error - Fisher: {emp_error_fisher*100:.2f}%")
print(f"Empirical Error - Pocket Perceptron: {emp_error_perc*100:.2f}%")

cm_fisher = confusion_matrix(y_train, yhat_fisher, labels=[1,-1])
cm_perc   = confusion_matrix(y_train, yhat_perc, labels=[1,-1])

fig, axes = plt.subplots(1, 2, figsize=(12,5))
disp_fisher = ConfusionMatrixDisplay(confusion_matrix=cm_fisher,
display_labels=['+1','-1'])
disp_fisher.plot(ax=axes[0], cmap='Blues', colorbar=False)
axes[0].set_title("Fisher Confusion Matrix")

disp_perc = ConfusionMatrixDisplay(confusion_matrix=cm_perc,
display_labels=['+1','-1'])
disp_perc.plot(ax=axes[1], cmap='Oranges', colorbar=False)
axes[1].set_title("Pocket Perceptron Confusion Matrix")

plt.tight_layout()
plt.show()
```

Validation Results

```
from sklearn.model_selection import train_test_split

y = outcome
X = np.column_stack((glucose, bmi))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# train fisher and perceptron models on training set
num_epochs = 500
learning_rate = 1

data = list(zip(X_train, y_train))

b_p, w1_p, w2_p, mistakes = pocket_perceptron(data, learning_rate, num_epochs)
w_f, b_f, X_A_train, X_B_train = fisher_train(X_train, y_train)

# predictions on test set
yhat_f = fisher_predict(X_test, w_f, b_f)
yhat_p = perceptron_predict(X_test, [w1_p, w2_p], b_p)

# calculate accuracies
acc_fisher = np.mean(yhat_f == y_test) * 100
acc_perceptron = np.mean(yhat_p == y_test) * 100

# calculate sensitivity and specificity
def sensitivity_specificity(y_true, y_pred):
    tp = np.sum((y_true == 1) & (y_pred == 1))
    tn = np.sum((y_true == -1) & (y_pred == -1))
    fp = np.sum((y_true == -1) & (y_pred == 1))
    fn = np.sum((y_true == 1) & (y_pred == -1))

    # people who have the disease
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0

    # people who do not have the disease
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

    return sensitivity, specificity

sens_fisher = sensitivity_specificity(y_test, yhat_f)
sens_perceptron = sensitivity_specificity(y_test, yhat_p)
print(f"Fisher Test Accuracy: {acc_fisher:.2f}%, Sensitivity:
{sens_fisher[0]:.2f}, Specificity: {sens_fisher[1]:.2f}")
print(f"Perceptron Test Accuracy: {acc_perceptron:.2f}%, Sensitivity:
{sens_perceptron[0]:.2f}, Specificity: {sens_perceptron[1]:.2f}")
```

Visualization of Projection and Learning

```
# Project data onto Fisher direction
proj = X_train @ w_fisher + b_fisher

# Separate projections by class
proj_class_neg = proj[y_train == -1]
proj_class_pos = proj[y_train == 1]

# Compute a simple threshold — e.g. midpoint between class means
thresh = (np.mean(proj_class_neg) + np.mean(proj_class_pos)) / 2

plt.figure(figsize=(8,5))
plt.hist(proj_class_neg, bins=20, alpha=0.6, label='Class -1', color='blue')
plt.hist(proj_class_pos, bins=20, alpha=0.6, label='Class +1', color='orange')
plt.axvline(thresh, color='k', linestyle='--', label=f'Threshold = {thresh:.2f}')

plt.title('1D Fisher LDA Projection with Class Histograms')
plt.xlabel('Projected Value (wᵀx + b)')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()

# Plot Perceptron error from training set vs. epoch
plt.figure(figsize=(10,5))
plt.plot(mistakes, linewidth=2)
plt.title('Perceptron Error vs. Epoch')
plt.xlabel('Epoch')
plt.ylabel('Number of Mistakes')
plt.grid(True)
plt.show()
```