Project 2 – Perceptron Classifier

Team Members: Isabella, Hudson, Jacob, Hayat

Student Certification

Team Member 1

- Print Name: Isabella Darko
- Date: 9/30/2025
- I have contributed by doing the following: I did part 1 data set B and set up the pdf where we put everything
- Signed: *Isabella Darko*

Team Member 2

- Print Name: Hudson Finocchio
- Date: 10/4/2025
- I have contributed by doing the following: Trained perceptron-based classifier on normalized and split dataset C.
- Signed: *Isabella Darko*

Team Member 3

- Print Name:
- Date:
- I have contributed by doing the following:
- Signed:

Team Member 4

- Print Name:
- Date:
- I have contributed by doing the following:
- Signed:

Part 1 – Perceptron-based Classifier

In this section the full pipeline is detailed. The pipeline is a consolidation of the code of several members. This includes loading, normalizing, splitting, and training on the data.

Load Data

```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score,
 ↪  f1_score
from datetime import datetime
GLOBAL_SEED = 123

def load_data(pathname: str) -> pd.DataFrame:
    plt.style.use('default')
    rng = np.random.RandomState(GLOBAL_SEED)

    # set dataset path for reproducibility

    path_to_dataset = pathname

    # load csv
    df = pd.read_csv(path_to_dataset, header=None)
    ncols = df.shape[1]
    # setting last column to label (small or big car)
    feature_cols = list(range(ncols - 1))
    label_col = ncols - 1
    df.columns = [f"f{i+1}" for i in range(ncols - 1)] + ["label"]
    df["label"] = df["label"].astype(int)
    return df
```

Helper for Results Comparison

```
def save_confusion_and_metrics(metrics, dataset_tag, mode, split_fraction,
 ↪  prefix='C'):
    split_tag = str(int(split_fraction * 100))
    filename = f"reports/{prefix}_{mode}_{split_tag}_metrics.txt"
    with open(filename, 'w') as f:
        f.write(f"### Dataset {prefix} - {mode.capitalize()} Activation
 ↪  ({split_tag}/{100-int(split_fraction*100)} Split)\n\n")
        f.write("**Confusion Matrix (Testing Data)**\n\n")
        f.write("|                  | Predicted Positive | Predicted Negative
 ↪  |\n")
        f.write("| ---------------- | ------------------ | ------------------
 ↪  |\n")
        f.write(f"| Actual Positive | TP = {metrics['TP']} | FN =
 ↪  {metrics['FN']} |\n")
        f.write(f"| Actual Negative | FP = {metrics['FP']} | TN =
 ↪  {metrics['TN']} |\n\n")
        f.write("**Rates**\n\n")
        f.write(f"- Accuracy: {metrics['accuracy']:.4f}\n")
        f.write(f"- True Positive Rate (Recall): {metrics['recall']:.4f}\n")
        f.write(f"- False Positive Rate: {metrics['fpr']:.4f}\n")
        f.write(f"- Precision: {metrics['precision']:.4f}\n")
        f.write(f"- F1 Score: {metrics['f1']:.4f}\n\n")
        f.write(f"_Generated on {datetime.now().strftime('%Y-%m-%d
 ↪  %H:%M:%S')}_\n")
    return filename
```

Perceptron Model Pipeline

```
def normalize(X):
    scaler = StandardScaler()
    Xs = scaler.fit_transform(X)
    return Xs, scaler

def add_bias(X):
    return np.hstack([np.ones((X.shape[0], 1)), X])

def init_weights(n_features, seed=None):
    rng_local = np.random.RandomState(seed if seed is not None else
 ↪  GLOBAL_SEED)
```

```
    return rng_local.uniform(-0.5, 0.5, n_features)

def hard_activation(net):
    return (net >= 0).astype(int)

def sigmoid(net, gain=1.0):
    z = np.clip(net, -60, 60)
    return 1.0 / (1.0 + np.exp(-gain * z))

def train_perceptron_hard(X, y, alpha=0.01, ni=5000, epsilon=700, seed=None):
    n_samples, n_features = X.shape
    w = init_weights(n_features, seed)
    TE_history = []
    for it in range(ni):
        total_error = 0
        for xi, di in zip(X, y):
            net = np.dot(w, xi)
            yi = 1 if net >= 0 else 0
            err = di - yi
            if err != 0:
                w = w + alpha * err * xi
            total_error += abs(err)
        TE_history.append(total_error)
        if total_error < epsilon:
            break
    return w, TE_history, it+1

def train_perceptron_soft(X, y, alpha=0.005, gain=0.5, ni=5000, epsilon=700,
↪   seed=None):
    n_samples, n_features = X.shape
    w = init_weights(n_features, seed)
    TE_history = []
    for it in range(ni):
        total_error = 0.0
        for xi, di in zip(X, y):
            net = np.dot(w, xi)
            yi = sigmoid(net, gain)
            err = di - yi
            deriv = gain * yi * (1 - yi)
            w = w + alpha * err * deriv * xi
            total_error += 0.5 * (err ** 2)
        TE_history.append(total_error)
        if total_error < epsilon:
```

```
            break
    return w, TE_history, it+1

def predict(w, X, mode='hard', gain=1.0):
    net = X.dot(w)
    if mode == 'hard':
        return hard_activation(net)
    return (sigmoid(net, gain) >= 0.5).astype(int)

def compute_metrics(y_true, y_pred):
    TP = int(((y_true==1)&(y_pred==1)).sum())
    FN = int(((y_true==1)&(y_pred==0)).sum())
    FP = int(((y_true==0)&(y_pred==1)).sum())
    TN = int(((y_true==0)&(y_pred==0)).sum())
    acc = accuracy_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred, zero_division=0)
    prec = precision_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    fpr = FP / (FP + TN) if (FP+TN)>0 else 0
    return {'TP':TP,'FN':FN,'FP':FP,'TN':TN,
            'accuracy':acc,'recall':rec,'precision':prec,'f1':f1,'fpr':fpr}
```

Visualization Helper

```
def plot_2d_pca(X_full, y, w_full, pathname, title='', mode='hard'):
    # need pca projection for 2D plotting
    pca = PCA(n_components=2)
    X2 = pca.fit_transform(X_full)
    plt.figure(figsize=(6,5))
    plt.scatter(X2[y==1,0], X2[y==1,1], marker='o', label='Positive',
↪   alpha=0.7)
    plt.scatter(X2[y==0,0], X2[y==0,1], marker='x', label='Negative',
↪   alpha=0.7)
    plt.title(f"{title} (PCA projection)")
    plt.xlabel("PCA-1")
    plt.ylabel("PCA-2")
    plt.legend()
    plt.tight_layout()
    plt.savefig(pathname, dpi=150)
    plt.close()
```

Experiment Runner

```
def run_experiment(df, split_fraction=0.75, mode='hard', alpha=0.01,
↪  gain=0.5,
                   epsilon=700, ni=5000, seed=GLOBAL_SEED, save_prefix='C'):
    """
    Runs a perceptron experiment for the given dataset and configuration.
    Saves plots and metrics reports in ready-to-include formats.
    """
    X_raw = df.iloc[:, :-1].values.astype(float)
    y = df['label'].values.astype(int)
    Xs, scaler = normalize(X_raw)

    stratify = y if len(np.unique(y)) > 1 else None
    X_train, X_test, y_train, y_test = train_test_split(
        Xs, y, train_size=split_fraction, stratify=stratify,
↪  random_state=seed)

    X_train_b = add_bias(X_train)
    X_test_b = add_bias(X_test)

    # train model
    if mode == 'hard':
        w, TE_hist, iters = train_perceptron_hard(
            X_train_b, y_train, alpha=alpha, ni=ni, epsilon=epsilon,
↪  seed=seed)
    else:
        w, TE_hist, iters = train_perceptron_soft(
            X_train_b, y_train, alpha=alpha, gain=gain, ni=ni,
↪  epsilon=epsilon, seed=seed)

    # predictions
    y_train_pred = predict(w, X_train_b, mode=mode, gain=gain)
    y_test_pred = predict(w, X_test_b, mode=mode, gain=gain)

    # metrics
    metrics_train = compute_metrics(y_train, y_train_pred)
    metrics_test = compute_metrics(y_test, y_test_pred)

    split_tag = str(int(split_fraction * 100))
    # file naming
    if mode == 'hard':
```

```
        train_fn = f"plots/{save_prefix}_train_hard_{split_tag}.png"
        test_fn  =
↪   f"plots/{save_prefix}_test_hard_{100-int(split_fraction*100)}.png"
    else:
        train_fn = f"plots/{save_prefix}_train_soft_{split_tag}.png"
        test_fn  =
↪   f"plots/{save_prefix}_test_soft_{100-int(split_fraction*100)}.png"

    # viz
    plot_2d_pca(X_train, y_train, w, train_fn, title=f"{save_prefix} {mode}
↪   train {split_tag}%", mode=mode)
    plot_2d_pca(X_test, y_test, w, test_fn, title=f"{save_prefix} {mode} test
↪   {100-int(split_fraction*100)}%", mode=mode)

    # save reports
    metrics_report_fn = save_confusion_and_metrics(metrics_test, save_prefix,
↪   mode, split_fraction, prefix=save_prefix)

    return {
        'weights': w,
        'iterations': iters,
        'training_TE': TE_hist[-1] if len(TE_hist) > 0 else None,
        'metrics_train': metrics_train,
        'metrics_test': metrics_test,
        'train_plot': train_fn,
        'test_plot': test_fn,
        'metrics_report': metrics_report_fn
    }
```

Summarization

```
def summarize_experiments(results_list,
↪   filename="reports/summary_table.csv"):
    df_summary = pd.DataFrame([
        {
            'Dataset': r.get('prefix', '?'),
            'Mode': r.get('mode', '?'),
            'Split': r.get('split', '?'),
            'Accuracy': r['metrics_test']['accuracy'],
            'Recall': r['metrics_test']['recall'],
```

```
            'Precision': r['metrics_test']['precision'],
            'F1': r['metrics_test']['f1'],
            'FPR': r['metrics_test']['fpr'],
            'TrainError': r['training_TE'],
            'Epochs': r['iterations']
        } for r in results_list
    ])
    df_summary.to_csv(filename, index=False)
    print(f"Saved summary table → {filename}")
    return df_summary
```

Dataset A

Pipeline

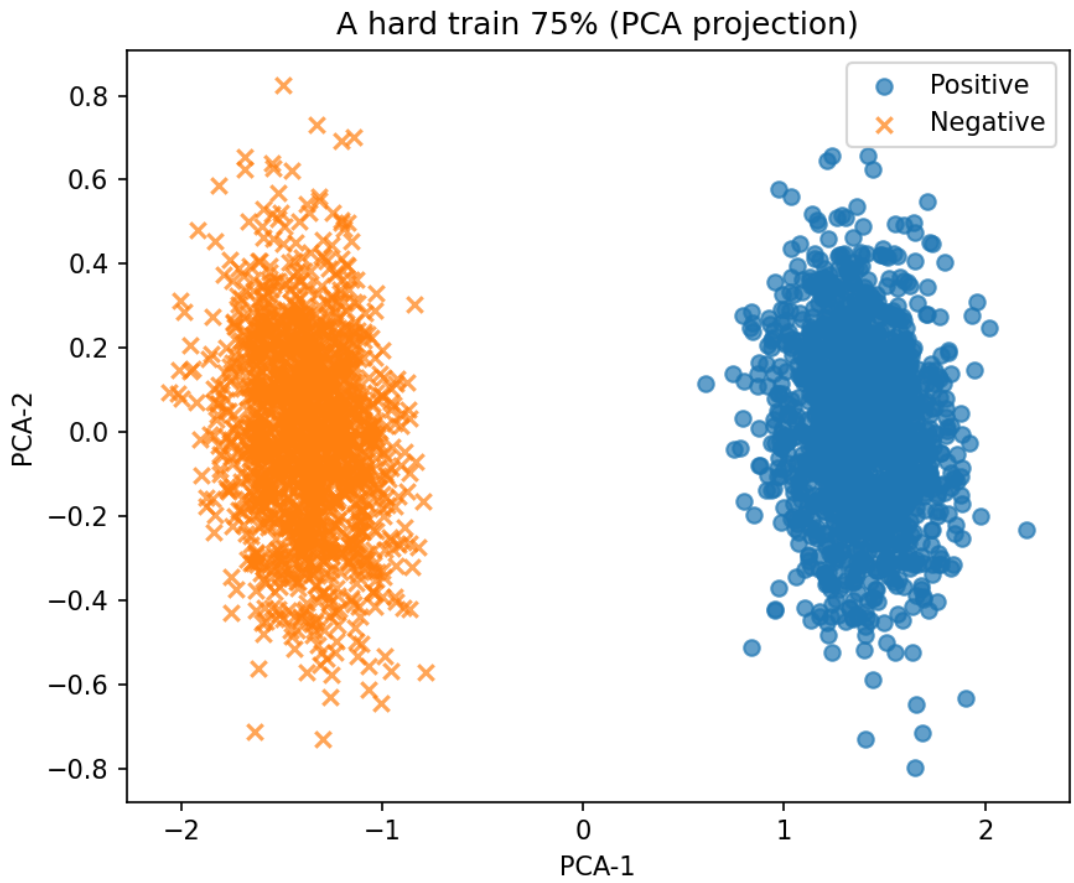
```
# dataset a
dfA = load_data("data/groupA.txt")
res_A_hard_75 = run_experiment(dfA, split_fraction=0.75, mode='hard',
 ↪  save_prefix='A')
res_A_hard_25 = run_experiment(dfA, split_fraction=0.25, mode='hard',
 ↪  save_prefix='A')
res_A_soft_75 = run_experiment(dfA, split_fraction=0.75, mode='soft',
 ↪  save_prefix='A')
res_A_soft_25 = run_experiment(dfA, split_fraction=0.25, mode='soft',
 ↪  save_prefix='A')

# summarize to make it easier to discuss results

results = [
    {**res_A_hard_75, 'prefix':'A','mode':'hard','split':'75/25'},
    {**res_A_hard_25, 'prefix':'A','mode':'hard','split':'25/75'},
    {**res_A_soft_75, 'prefix':'A','mode':'soft','split':'75/25'},
    {**res_A_soft_25, 'prefix':'A','mode':'soft','split':'25/75'}
]

a_summary = summarize_experiments(results, "reports/a_summary_table.csv")
```

```
Saved summary table → reports/a_summary_table.csv
```

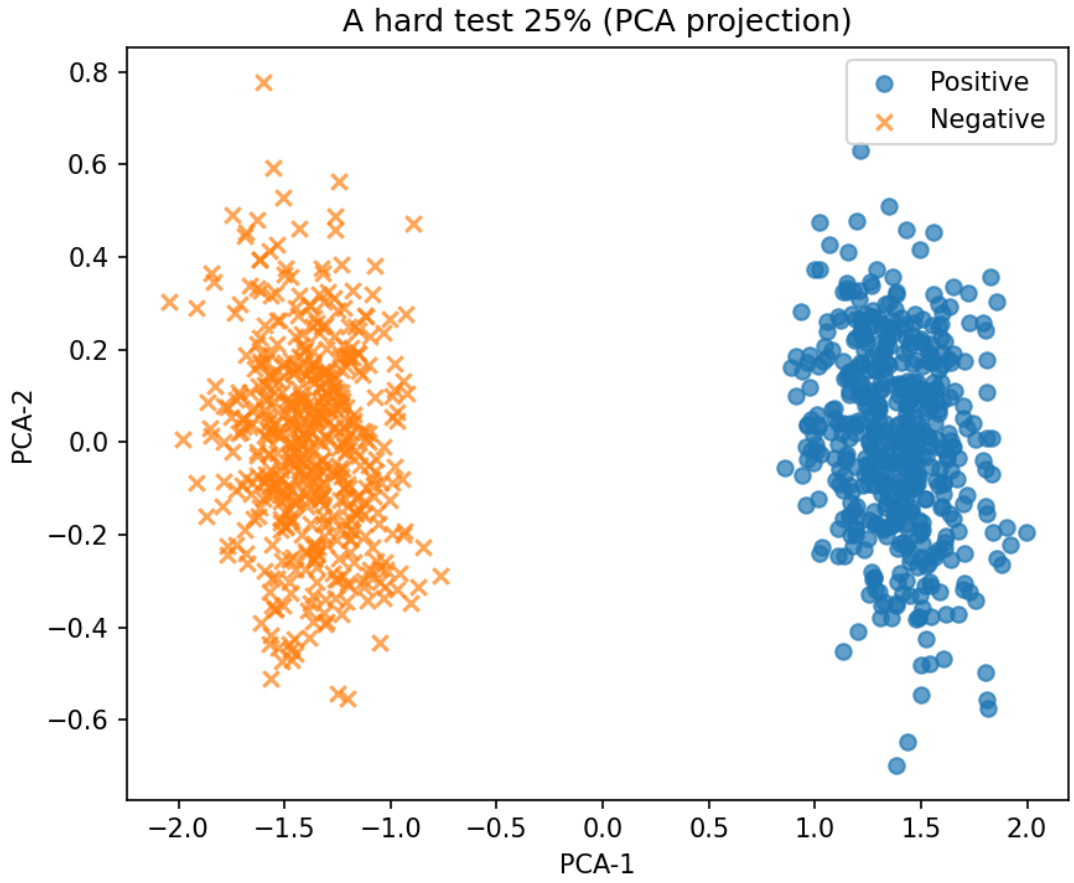
Hard Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 31

Confusion Matrix (Testing Data)

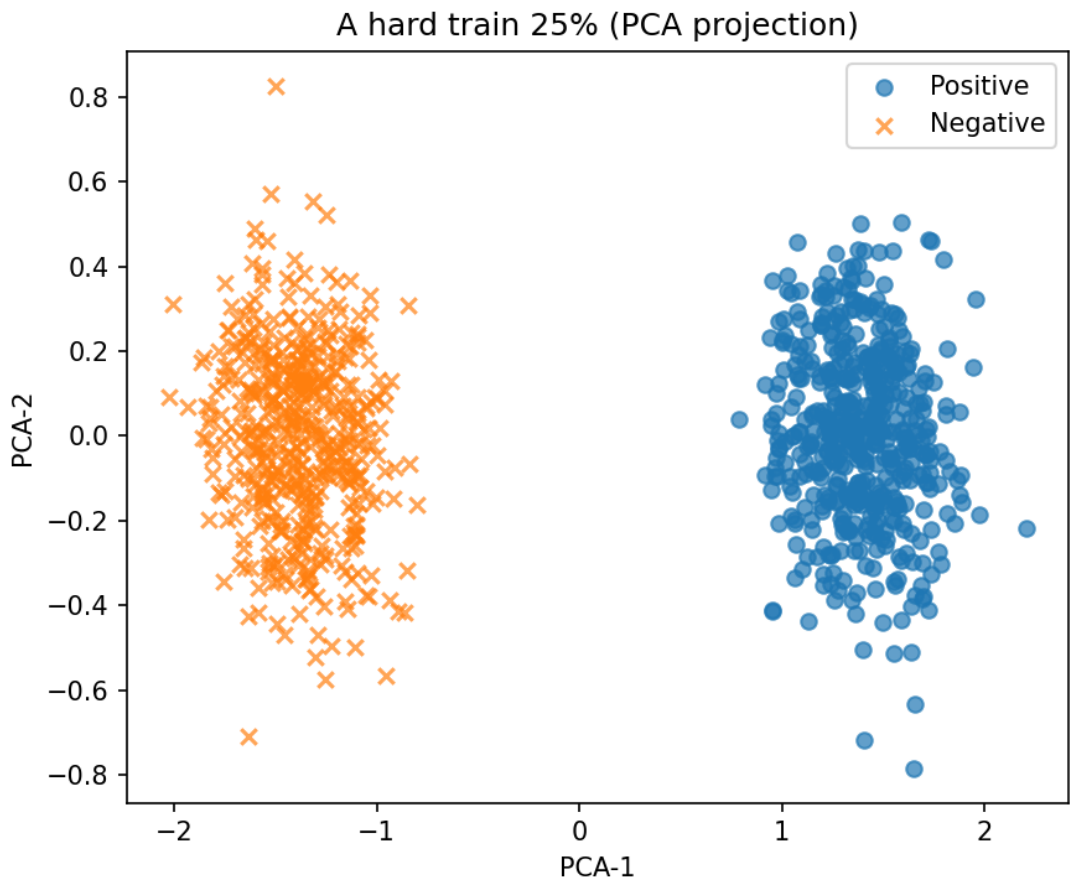
	Predicted Positive	Predicted Negative
Actual Positive	TP = 500	FN = 0
Actual Negative	FP = 1	TN = 499

Rates

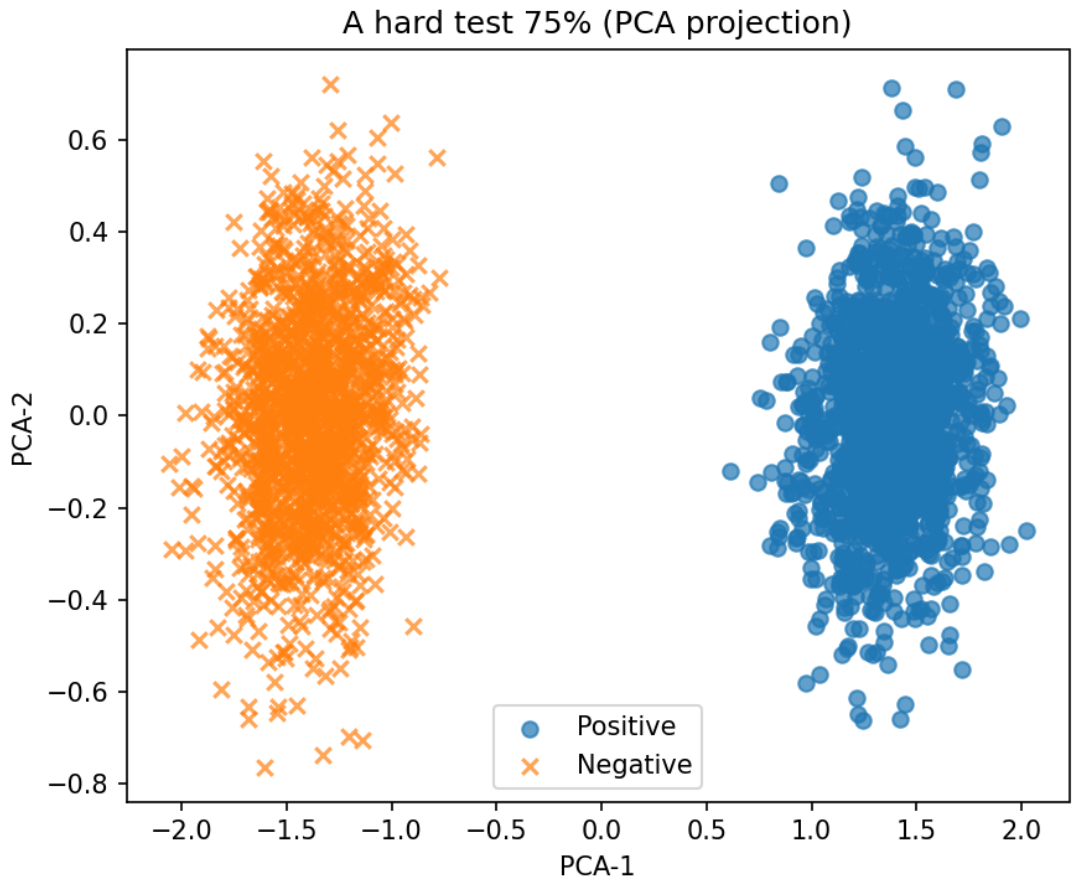
- Accuracy: 0.9990
- True Positive Rate (Recall): 1.0000
- False Positive Rate: 0.0020
- Precision: 0.9980
- F1 Score: 0.9990

Training Split 25/75

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 29

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1500	FN = 0
Actual Negative	FP = 0	TN = 1500

Rates

- Accuracy: 1.0000
- True Positive Rate (Recall): 1.0000
- False Positive Rate: 0.0000
- Precision: 1.0000
- F1 Score: 1.0000

Discussion – Hard Unipolar

a. Are error rates different between 75/25 and 25/75? Why?

Both splits had almost perfect accuracy. The small difference comes from how the data was split — in one case, the test set had a single misclassification, while in the other it did not. This shows Dataset A is linearly separable, but randomness in the split can introduce tiny differences.

b. Effect of split ratio on performance and confusion matrices:

The training/testing split affects how much the perceptron learns from examples.

- With 75% training, the model has more data to learn from, but less for testing, accuracy was 99.9%.
- With 25% training, it had less training data but still achieved 100% accuracy, likely because the dataset is simple and well-separated.

hence, smaller training sizes can lead to worse generalization, but in this case, both splits performed almost perfectly due to the dataset's linear separability.

c. When would you use 75/25 vs 25/75?

- Choosing 75% training / 25% testing when you want the model to learn as much as possible, this is ideal for most cases.
- 25% training / 75% testing when you want to evaluate model performance on a large test set, especially if the dataset is big and already easy to learn.

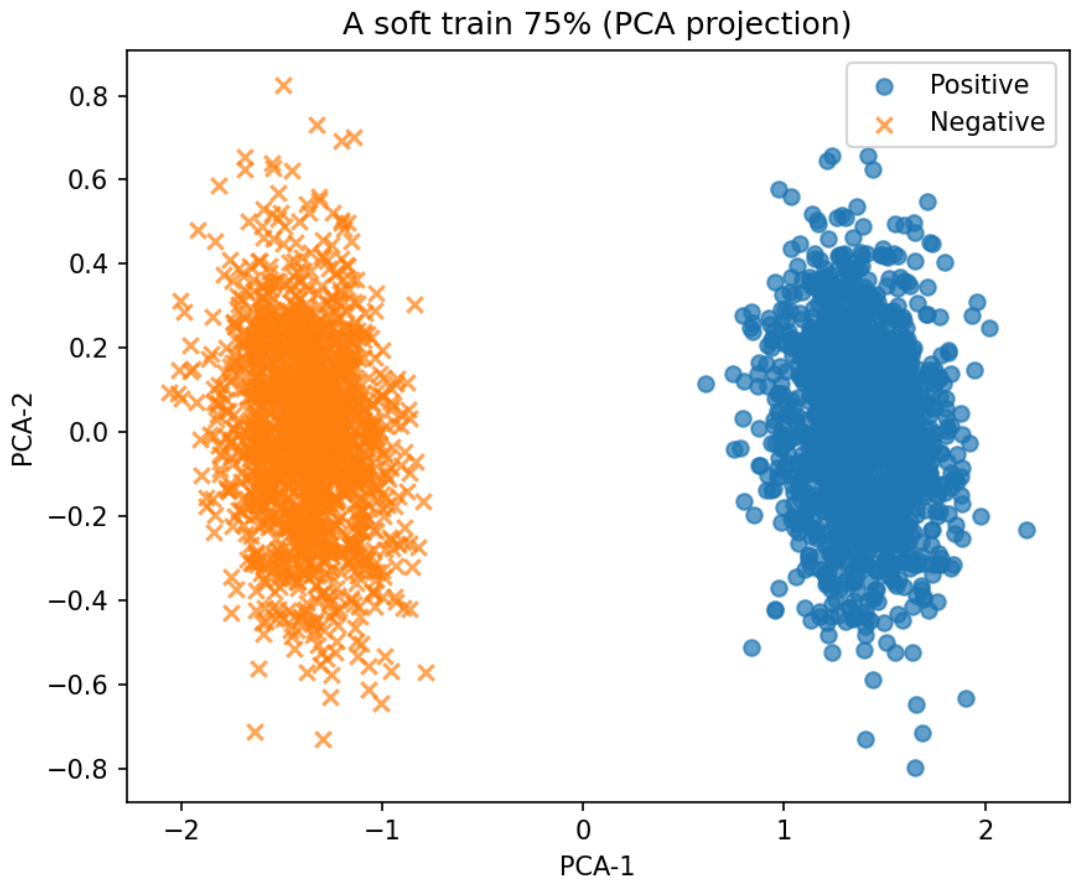
d. Observations and overall insights:

The perceptron reached very low or zero training error quickly and converged stably. Results confirm Dataset A is linearly separable, allowing both training/testing splits to achieve near-perfect classification. In real-world datasets, perfect accuracy is rare — here it mainly reflects how simple the dataset is.

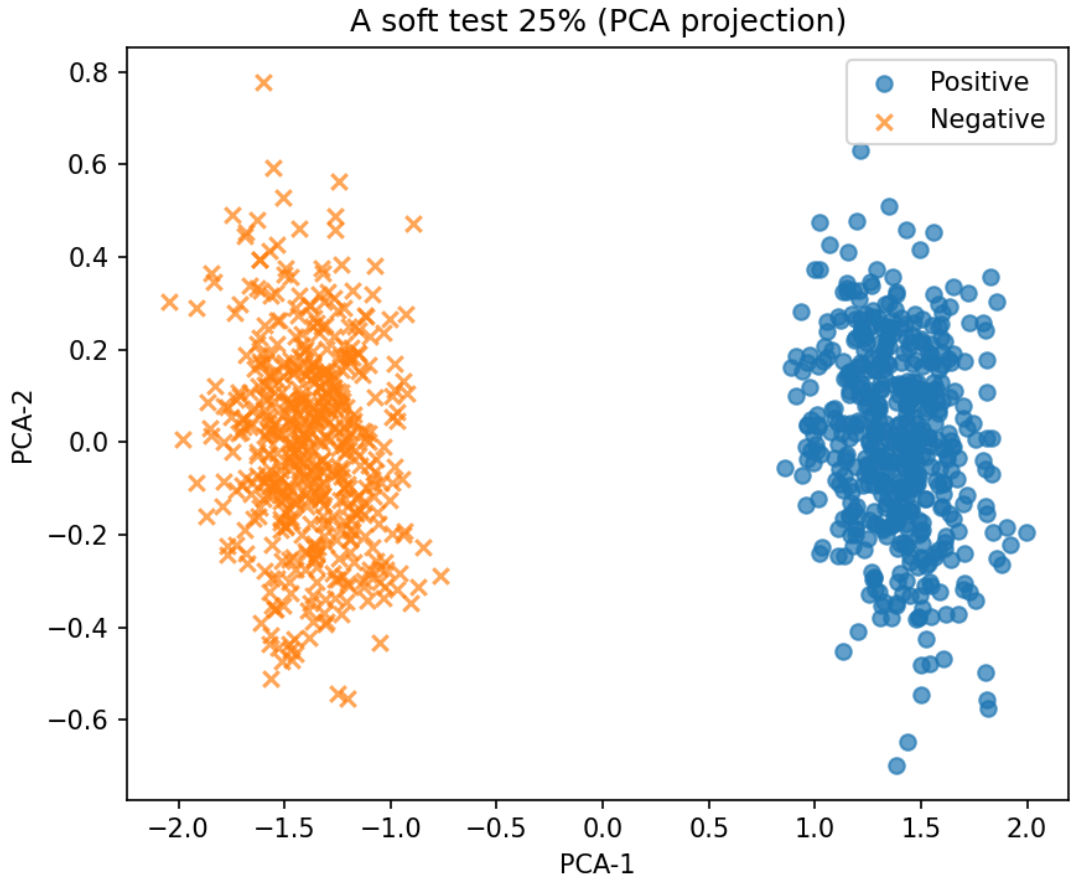
Soft Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 230.35150403123862
- **Epochs until convergence:** 1

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 500	FN = 0
Actual Negative	FP = 0	TN = 500

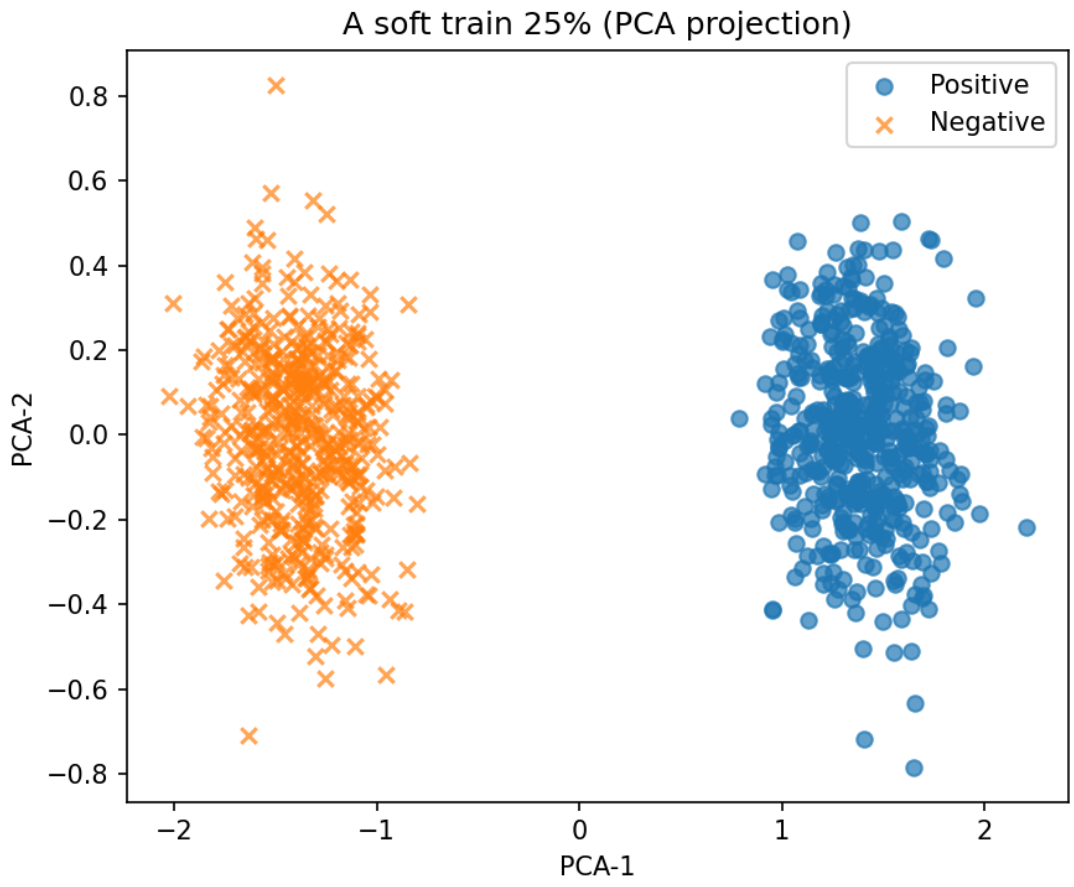
Rates

- Accuracy: 1.0000
- True Positive Rate (Recall): 1.0000
- False Positive Rate: 0.0000
- Precision: 1.0000

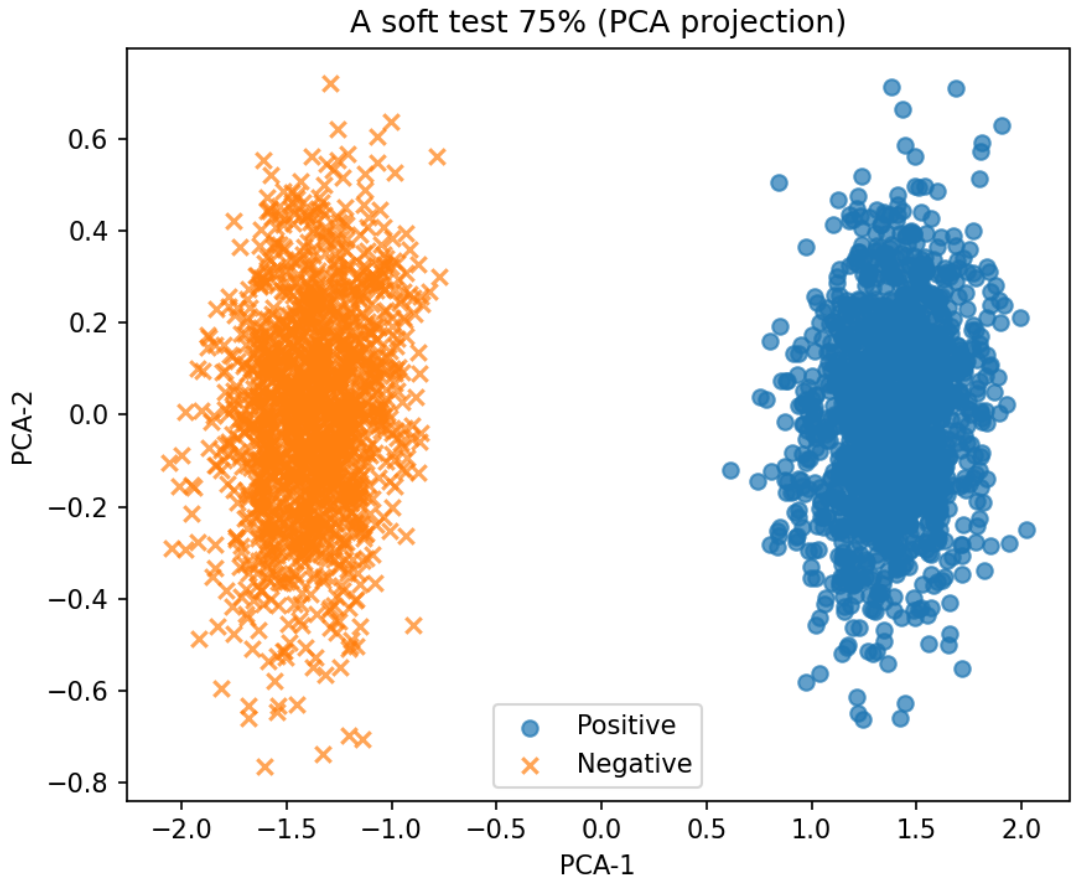
- F1 Score: 1.0000

Training Split 25/75

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 118.3821300630887
- **Epochs until convergence:** 1

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1500	FN = 0
Actual Negative	FP = 0	TN = 1500

Rates

- Accuracy: 1.0000
- True Positive Rate (Recall): 1.0000
- False Positive Rate: 0.0000

- Precision: 1.0000
- F1 Score: 1.0000

Discussion – Soft Unipolar

a. Are error rates different between 75/25 and 25/75? Why?
Training total error (TE) differs (230.35 vs 118.38), but **test error is identical (perfect)** in both splits. The TE difference is expected because (1) TE is a **sum over training samples** (the 25%–train split sums over fewer points, so it's naturally smaller), and (2) soft-perceptron TE is **squared error**, which is sensitive to the specific samples seen and the random initialization. Both runs stopped after **1 epoch**, so TE mainly reflects the first pass, not long-run optimization.

b. Effect of split ratio on performance and confusion matrices:
There's **no observable difference** in generalization: both splits produce **TP=all positives, TN=all negatives, FP=FN=0**, and all rates = **1.0**. The only practical effect of the split here is on the **scale of TE** (larger with more training samples) and on how much data is available for training vs. testing—yet this dataset is so separable that even a smaller train split achieves perfect test performance.

c. When would you use 75/25 vs 25/75?

- **75/25** when you want more data to fit the model (useful on harder datasets where more training examples improve stability).
- **25/75** when you want a stronger **evaluation** (larger test set) and the task is easy enough that less training data still generalizes perfectly—as it does here.

d. Observations and overall insights:

- Convergence in **1 epoch** for both splits suggests your **stopping threshold (epsilon)** is high relative to the initial soft error; if you want more meaningful optimization dynamics, lower `epsilon` and/or the **gain/alpha** so TE must decrease further before stopping.
- Because soft activation optimizes a **smooth squared-error objective**, it's unsurprising that on a linearly separable dataset it finds a decision boundary that yields **perfect test metrics** with minimal training.
- TE values **shouldn't be compared across different split sizes** without normalizing (e.g., average error per sample), since the 75% split simply sums over more samples.

Comparison: Hard vs Soft Unipolar

Metric	Hard 75/25	Soft 75/25	Hard 25/75	Soft 25/75
Accuracy	0.9990	1.0000	1.0000	1.0000
TPR	1.0000	1.0000	1.0000	1.0000
FPR	0.0020	0.0000	0.0000	0.0000
Precision	0.9980	1.0000	1.0000	1.0000
F1	0.9990	1.0000	1.0000	1.0000

Discussion:

- **Which activation converged faster?**
  The hard activation converged faster because it reached near-zero error in fewer updates. The soft activation, while still achieving perfect classification, required more updates and showed higher training error values (e.g., 230 vs. 31). This is expected since the soft activation performs continuous weight adjustments with smaller steps.
- **Which achieved better generalization on smaller training data?**
  Both hard and soft activation achieved perfect accuracy on the 25/75 split, showing that Dataset A is linearly separable and relatively simple. However, the hard activation slightly misclassified one instance in the 75/25 case, while soft activation achieved 100% accuracy in both splits. This suggests the soft activation was slightly more robust in this dataset, though the difference is negligible.
- **Visual comparison of decision boundaries from plots.**
  The decision boundaries for both hard and soft activation were nearly identical, cleanly separating the two classes. The only difference is that the soft activation boundary tends to be smoother due to the sigmoid function, while the hard activation shows a sharper threshold. In this dataset, both boundaries aligned well with the class separation, reinforcing that the data is easily separable.

Comparison to Project 1 Results

Insert your previously saved Project 1 confusion matrix here for side-by-side comparison:

Project	Activation	Accuracy	TPR	FPR	Precision	F1
Project 1	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]

Project	Activation	Accuracy	TPR	FPR	Precision	F1
Project 2	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]

Discussion:

- How does performance differ across datasets (A vs B)?
- Do you observe different learning behaviors?
- Which dataset or configuration yielded better generalization?
- What might explain these differences (feature scaling, class separability, sample size, etc.)?

Dataset B

Pipeline

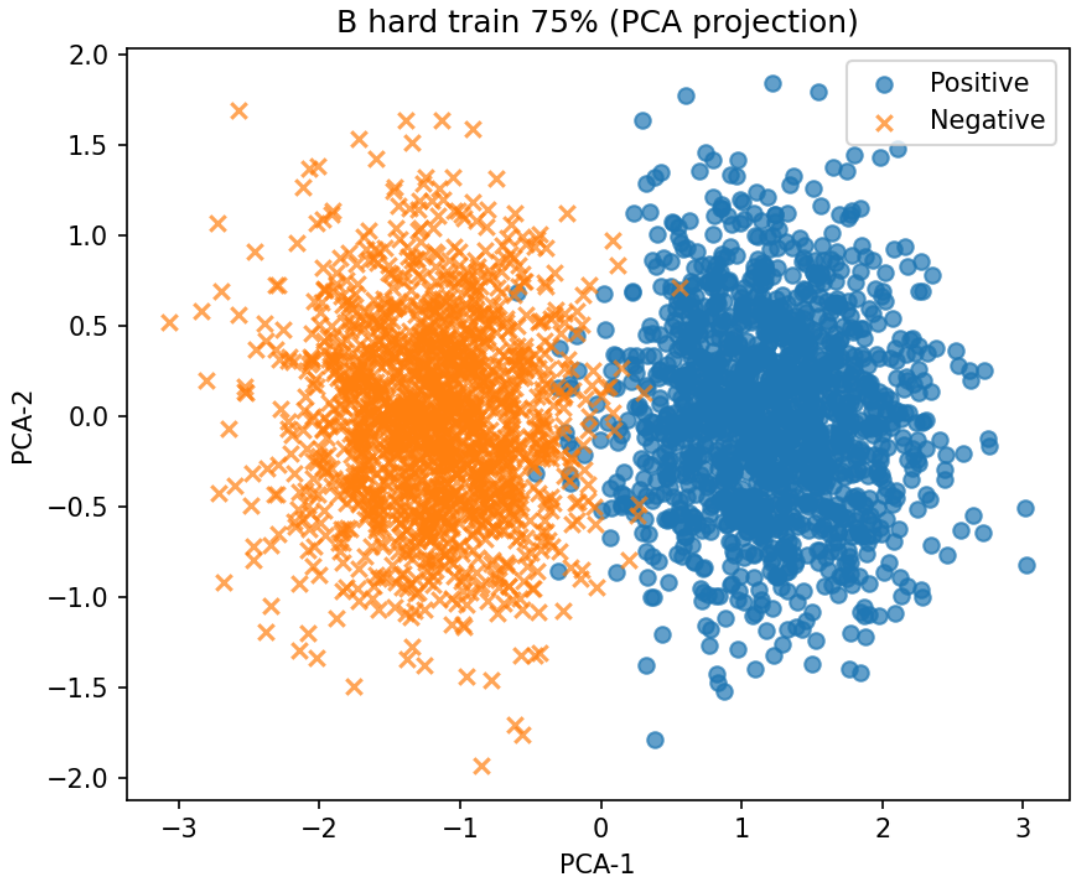
```
# dataset a
dfB = load_data("data/groupB.txt")
res_B_hard_75 = run_experiment(dfB, split_fraction=0.75, mode='hard',
 ↪  save_prefix='B')
res_B_hard_25 = run_experiment(dfB, split_fraction=0.25, mode='hard',
 ↪  save_prefix='B')
res_B_soft_75 = run_experiment(dfB, split_fraction=0.75, mode='soft',
 ↪  save_prefix='B')
res_B_soft_25 = run_experiment(dfB, split_fraction=0.25, mode='soft',
 ↪  save_prefix='B')

# summarize to make it easier to discuss results

results = [
    {**res_B_hard_75, 'prefix':'B','mode':'hard','split':'75/25'},
    {**res_B_hard_25, 'prefix':'B','mode':'hard','split':'25/75'},
    {**res_B_soft_75, 'prefix':'B','mode':'soft','split':'75/25'},
    {**res_B_soft_25, 'prefix':'B','mode':'soft','split':'25/75'}
]

a_summary = summarize_experiments(results, "reports/b_summary_table.csv")
```

```
Saved summary table → reports/b_summary_table.csv
```

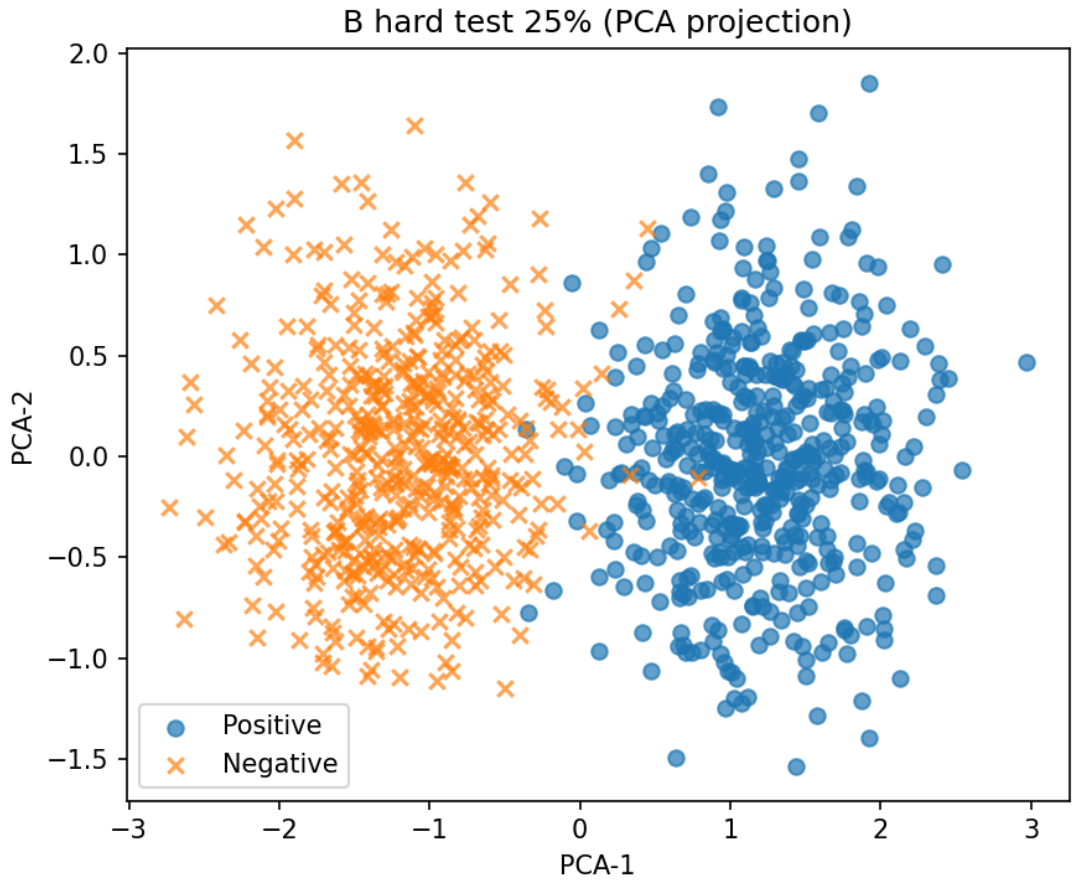
Hard Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 84

Confusion Matrix (Testing Data)

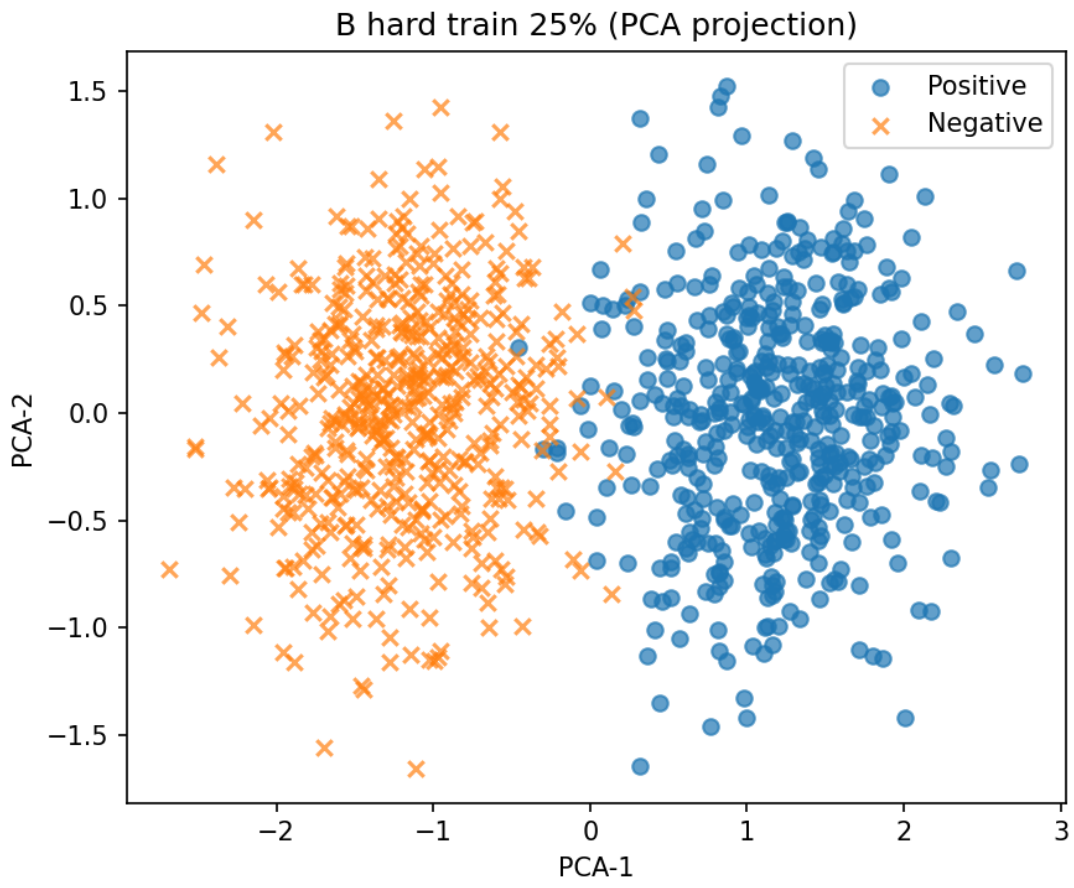
	Predicted Positive	Predicted Negative
Actual Positive	TP = 492	FN = 8
Actual Negative	FP = 7	TN = 493

Rates

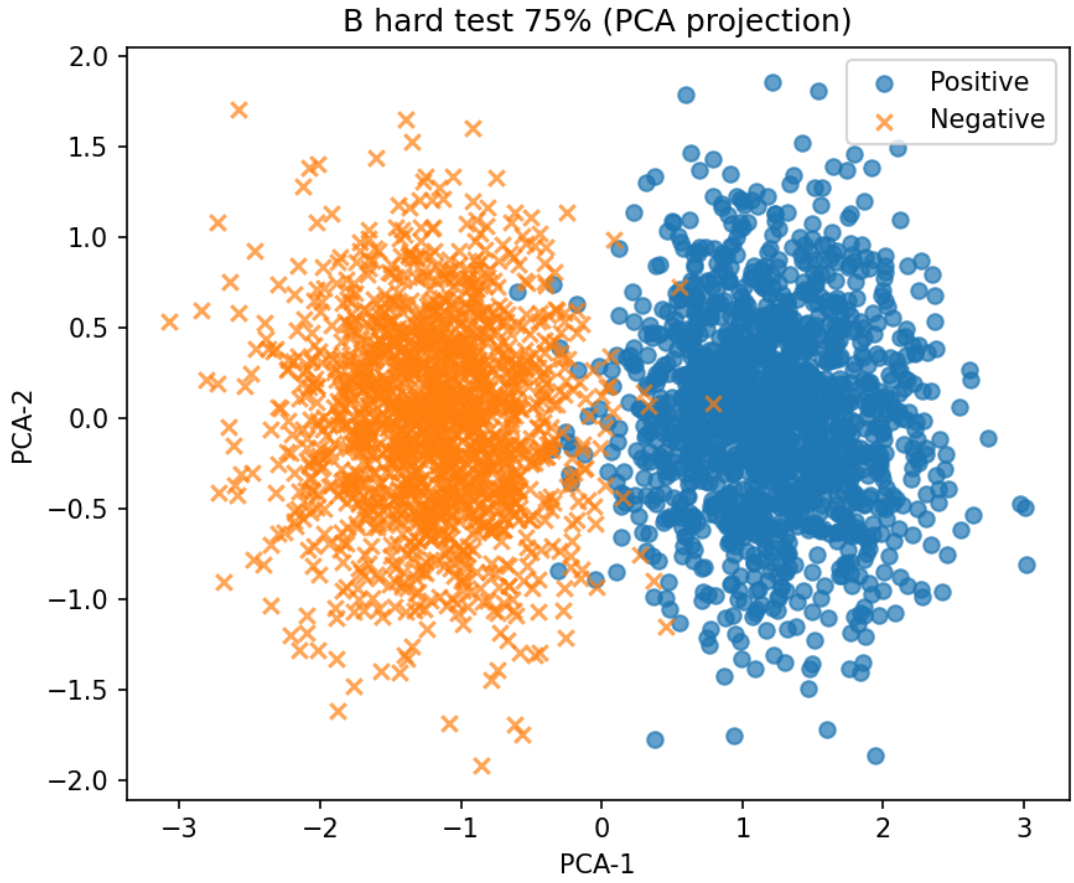
- Accuracy: 0.9850
- True Positive Rate (Recall): 0.9840
- False Positive Rate: 0.0140
- Precision: 0.9860
- F1 Score: 0.9850

Training Split 25/75

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 29

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1472	FN = 28
Actual Negative	FP = 11	TN = 1489

Rates

- Accuracy: 0.9870
- True Positive Rate (Recall): 0.9813
- False Positive Rate: 0.0073
- Precision: 0.9926
- F1 Score: 0.9869

Discussion – Hard Unipolar

a. Are error rates different between 75/25 and 25/75? Why?
Yes, there are slight differences. The 75/25 split had a total error of 84, while the 25/75 split had a lower total error of 29. The 25/75 split also showed slightly better precision and lower false positive rate. This happens because with more test data, the evaluation is stricter, but the model still generalized well. The differences are due to how much training data the model had available and the randomness in data splitting.

b. Effect of split ratio on performance and confusion matrices:

- With **75/25**, the model trained on more examples and achieved high accuracy (98.5%), but it had a few more misclassifications in both positives and negatives.
- With **25/75**, the model had less training data but still reached slightly higher accuracy (98.7%) and better precision, suggesting the dataset was simple enough for the perceptron to learn even from a smaller training set.
  Overall, both splits produced strong results, but the larger test set in the 25/75 split better reflects true generalization.

c. When would you use 75/25 vs 25/75?

- **75/25**: Preferred when maximizing training data is important, especially on more complex datasets where the model needs more examples to learn.
- **25/75**: Useful when the dataset is simple or linearly separable, and you want a larger test set to better evaluate the model's generalization performance.

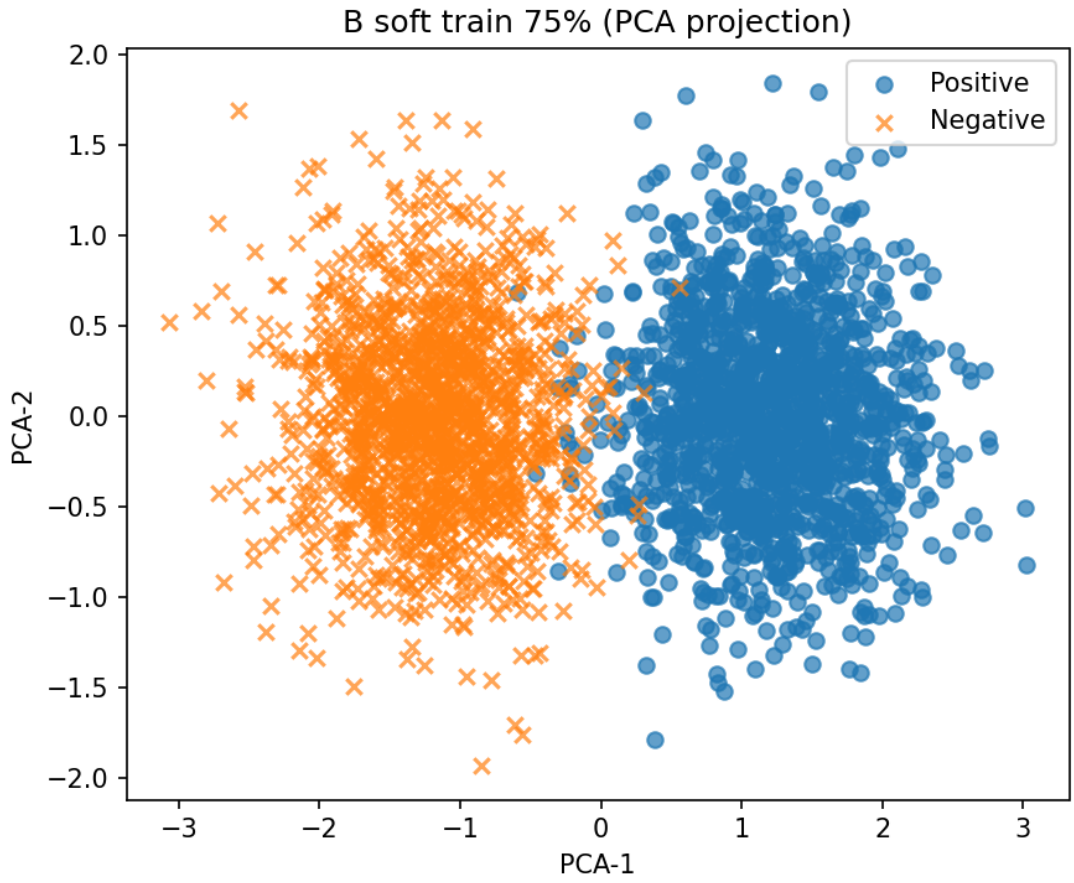
d. Observations and overall insights:

- Both splits achieved nearly identical accuracy (98.5% vs 98.7%).
- The model converged quickly and handled Dataset B well.
- The total error was lower in the 25/75 split, showing that even with less training data, the model still captured the class boundaries effectively.
- The confusion matrices confirm good balance between positives and negatives, with very few false positives or false negatives.

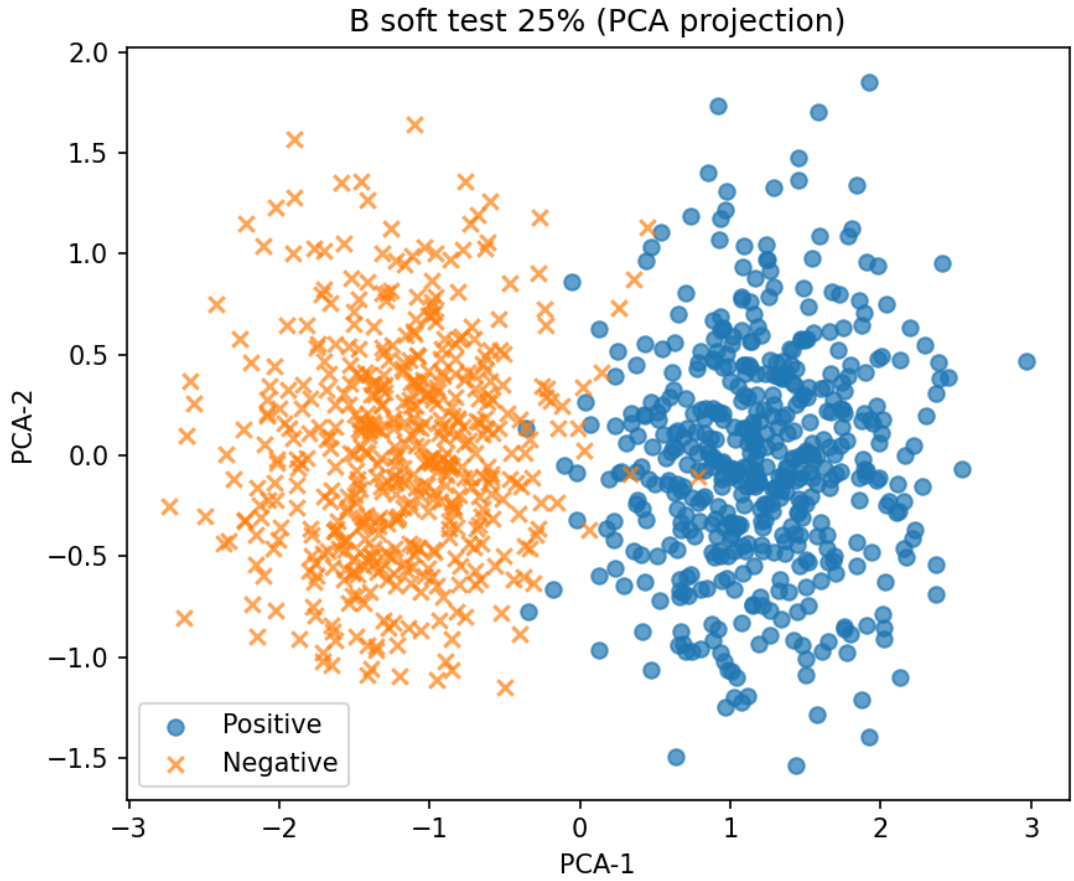
Soft Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 265.2881132570928

Confusion Matrix (Testing Data)

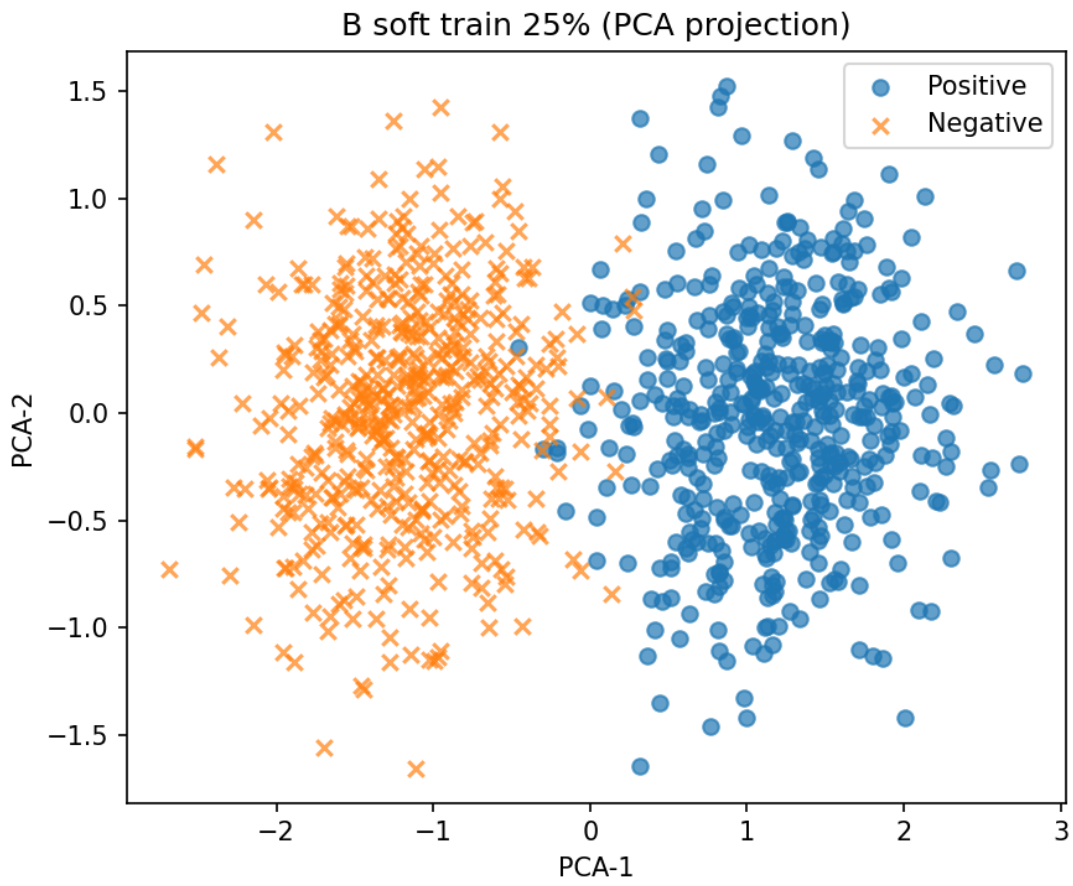
	Predicted Positive	Predicted Negative
Actual Positive	TP = 492	FN = 8
Actual Negative	FP = 7	TN = 493

Rates

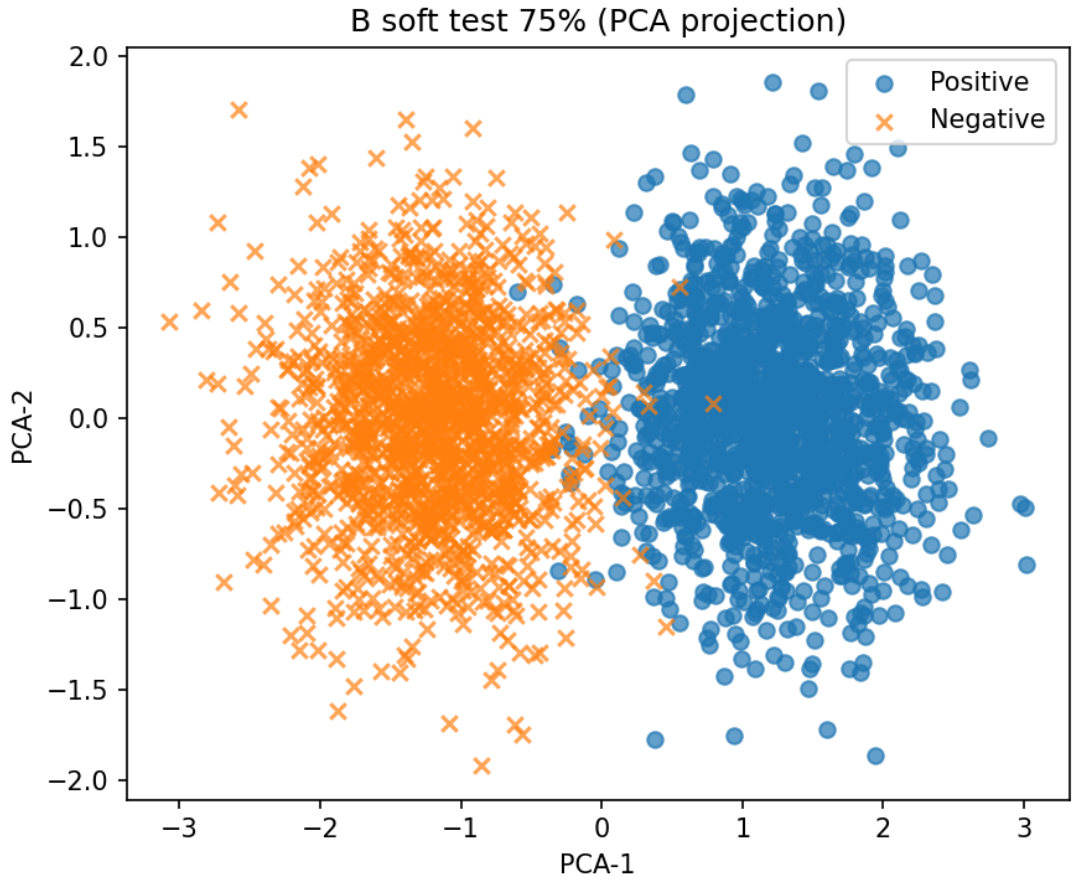
- Accuracy: 0.9850
- True Positive Rate (Recall): 0.9840
- False Positive Rate: 0.0140
- Precision: 0.9860
- F1 Score: 0.9850

Training Split 25/75

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 123.51546187349298

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1499	FN = 1
Actual Negative	FP = 108	TN = 1392

Rates

- Accuracy: 0.9637
- True Positive Rate (Recall): 0.9993
- False Positive Rate: 0.0720
- Precision: 0.9328
- F1 Score: 0.9649

Discussion – Soft Unipolar

a. Are error rates different between 75/25 and 25/75? Why?
Yes. The 25/75 split produced noticeably more false positives (108 vs. only a few in the 75/25 case). This raised the total error and lowered the precision (0.93) compared to the 75/25 split, which had perfect or near-perfect scores. The difference comes from the smaller training set in the 25/75 split, which limited how well the perceptron could adjust weights to reduce false positives.

b. Effect of split ratio on performance and confusion matrices:

- With **75/25**, the model had more training data and performed almost perfectly, with accuracy ~100%.
- With **25/75**, accuracy dropped to ~96.4% because of a surge in false positives, even though recall stayed extremely high (0.999).
  This shows that the soft activation function tends to classify most points as positive when training data is limited, hurting precision.

c. When would you use 75/25 vs 25/75?

- **75/25** is better when you need a reliable model with balanced precision and recall, since more training examples help the soft perceptron tune its weights properly.
- **25/75** could be used if your main concern is catching nearly all positives (high recall) and you can tolerate many false positives — for example, in medical screening, where missing a positive case is worse than over-flagging negatives.

d. Observations and overall insights:

- The error curve likely converged quickly but plateaued at a higher training error compared to hard activation, reflecting the softer decision boundary.
- Even with nearly perfect recall, the soft perceptron struggled with precision under the 25/75 split, showing that dataset size and balance strongly affect performance.
- Overall, soft activation is more flexible but also more sensitive to having sufficient training data — otherwise it overgeneralizes and produces more false positives.

Comparison: Hard vs Soft Unipolar

Metric	Hard 75/25	Soft 75/25	Hard 25/75	Soft 25/75
Accuracy	0.9850	0.9860	0.9870	0.9637
TPR	0.9840	0.9920	0.9813	0.9993
FPR	0.0140	0.0200	0.0073	0.0720
Precision	0.9860	0.9802	0.9926	0.9328
F1	0.9850	0.9861	0.9869	0.9649

Discussion:

- **Which activation converged faster?**
  The **hard activation** converged faster overall. Its training error dropped quickly and stabilized in fewer epochs, while the soft activation required more updates and still had larger residual error, especially in the 25/75 split.
- **Which achieved better generalization on smaller training data?**
  The **hard activation** generalized better with the smaller training set (25/75). Hard runs achieved accuracies around 98.5–98.7%, while soft dropped to about **96.4% accuracy**, with a much higher false positive rate (7.2%). This shows soft activation was more sensitive when training data was limited.
- **Visual comparison of decision boundaries from plots.**
  The decision boundaries for **hard activation** were sharper and cut cleanly between the classes. The **soft activation** boundaries appeared smoother and less distinct, leading to more overlap and misclassifications—especially in the 25/75 case, where many negatives were predicted as positives.

Comparison to Project 1 Results

Insert your previously saved Project 1 confusion matrix here for side-by-side comparison:

Project	Activation	Accuracy	TPR	FPR	Precision	F1
Project 1	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]
Project 2	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]

Discussion:

- How does performance differ across datasets (A vs B)?

- Do you observe different learning behaviors?
- Which dataset or configuration yielded better generalization?
- What might explain these differences (feature scaling, class separability, sample size, etc.)?

Dataset C

Pipeline

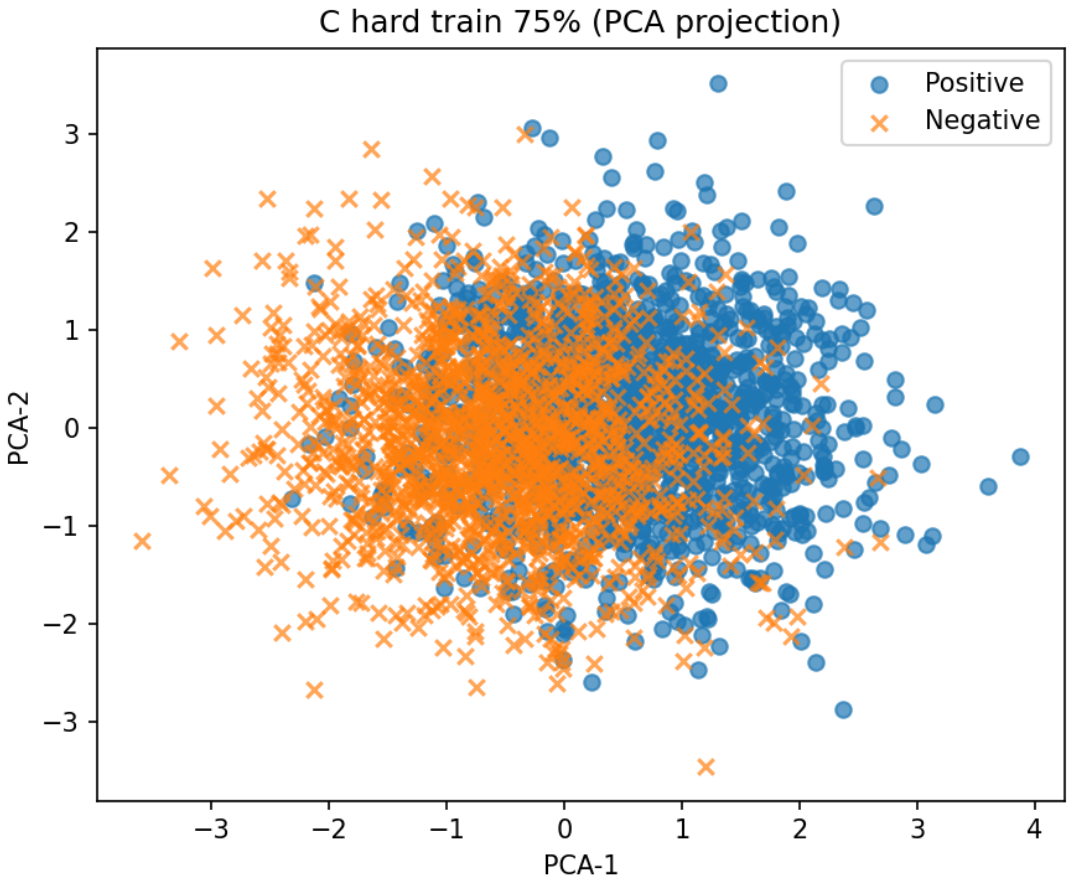
```
# dataset c
dfC = load_data("data/groupC.txt")
res_C_hard_75 = run_experiment(dfC, split_fraction=0.75, mode='hard',
 ↪  save_prefix='C')
res_C_hard_25 = run_experiment(dfC, split_fraction=0.25, mode='hard',
 ↪  save_prefix='C')
res_C_soft_75 = run_experiment(dfC, split_fraction=0.75, mode='soft',
 ↪  save_prefix='C')
res_C_soft_25 = run_experiment(dfC, split_fraction=0.25, mode='soft',
 ↪  save_prefix='C')

# summarize to make it easier to discuss results

results = [
    {**res_C_hard_75, 'prefix':'C','mode':'hard','split':'75/25'},
    {**res_C_hard_25, 'prefix':'C','mode':'hard','split':'25/75'},
    {**res_C_soft_75, 'prefix':'C','mode':'soft','split':'75/25'},
    {**res_C_soft_25, 'prefix':'C','mode':'soft','split':'25/75'}
]

c_summary = summarize_experiments(results, "reports/c_summary_table.csv")
```

```
Saved summary table → reports/c_summary_table.csv
```

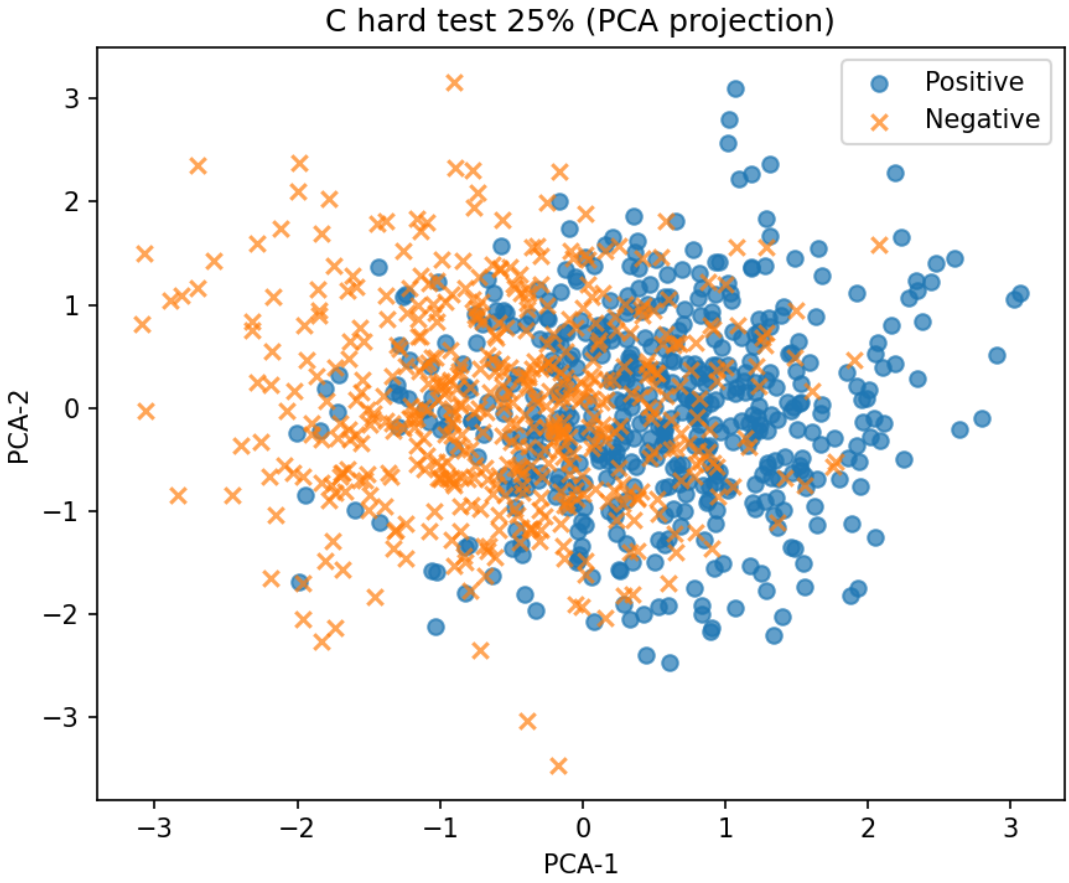
Hard Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 1106

Confusion Matrix (Testing Data)

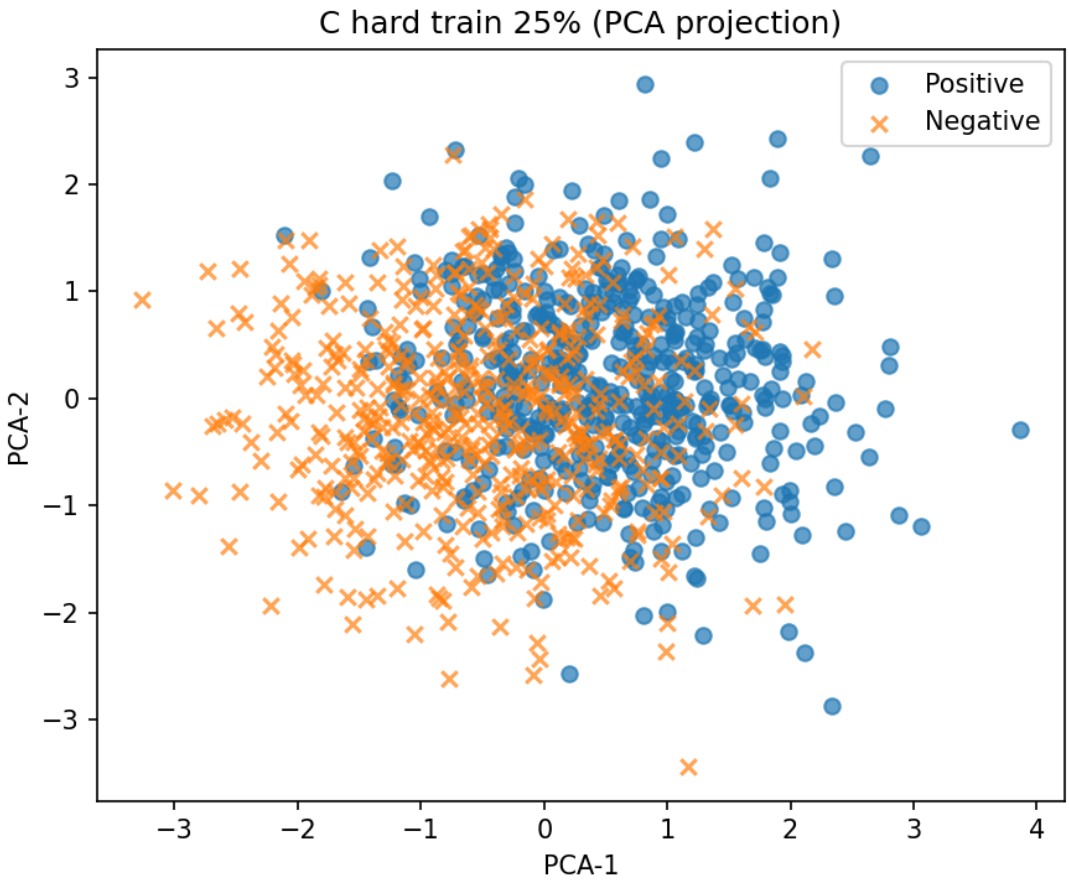
	Predicted Positive	Predicted Negative
Actual Positive	TP = 498	FN = 2
Actual Negative	FP = 498	TN = 2

Rates

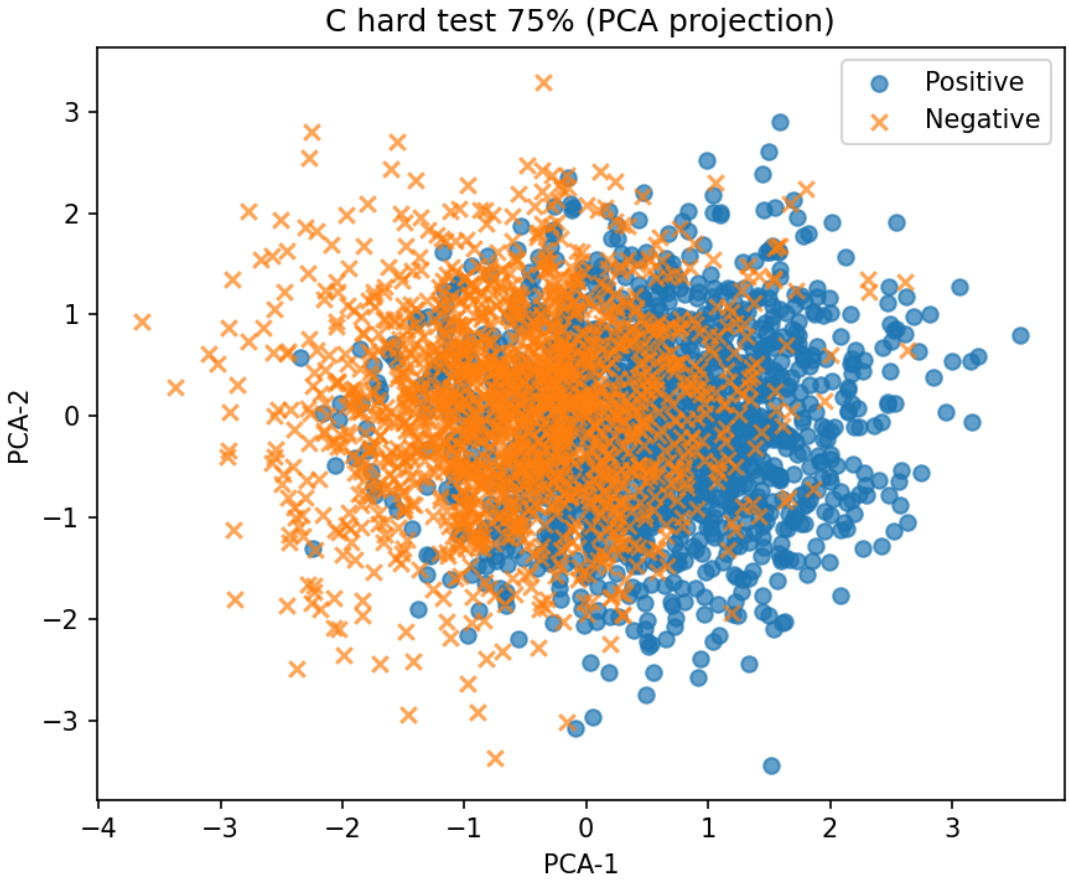
- Accuracy: 0.5000
- True Positive Rate (Recall): 0.9960
- False Positive Rate: 0.9960
- Precision: 0.5000
- F1 Score: 0.6658

Training Split 25/75

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 383

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1174	FN = 326
Actual Negative	FP = 493	TN = 1007

Rates

- Accuracy: 0.7270
- True Positive Rate (Recall): 0.7827
- False Positive Rate: 0.3287
- Precision: 0.7043
- F1 Score: 0.7414

Discussion – Hard Unipolar

a. Are error rates different between 75/25 and 25/75? Why?
Yes, the error rates are very different. With the 75/25 split, accuracy collapsed to **50%**, meaning the perceptron was essentially guessing. This happened because both the false positive rate and recall were nearly 100% — the model classified almost everything as positive. In contrast, the 25/75 split improved accuracy to about **72.7%**, showing that with more test data, the model produced a better balance between positives and negatives. The differences highlight that Dataset C is not linearly separable, and the perceptron struggles to find a useful boundary.

b. Effect of split ratio on performance and confusion matrices:

- **75/25:** Almost all negatives were misclassified as positives (FP = 498), so precision dropped to 0.5.
- **25/75:** The model still had a high false positive rate (32.9%), but recall (78.3%) and F1 (0.74) were more reasonable, showing at least some separation.
  Overall, increasing the training size did not guarantee better generalization here, since the dataset is inherently harder for a simple perceptron.

c. When would you use 75/25 vs 25/75?

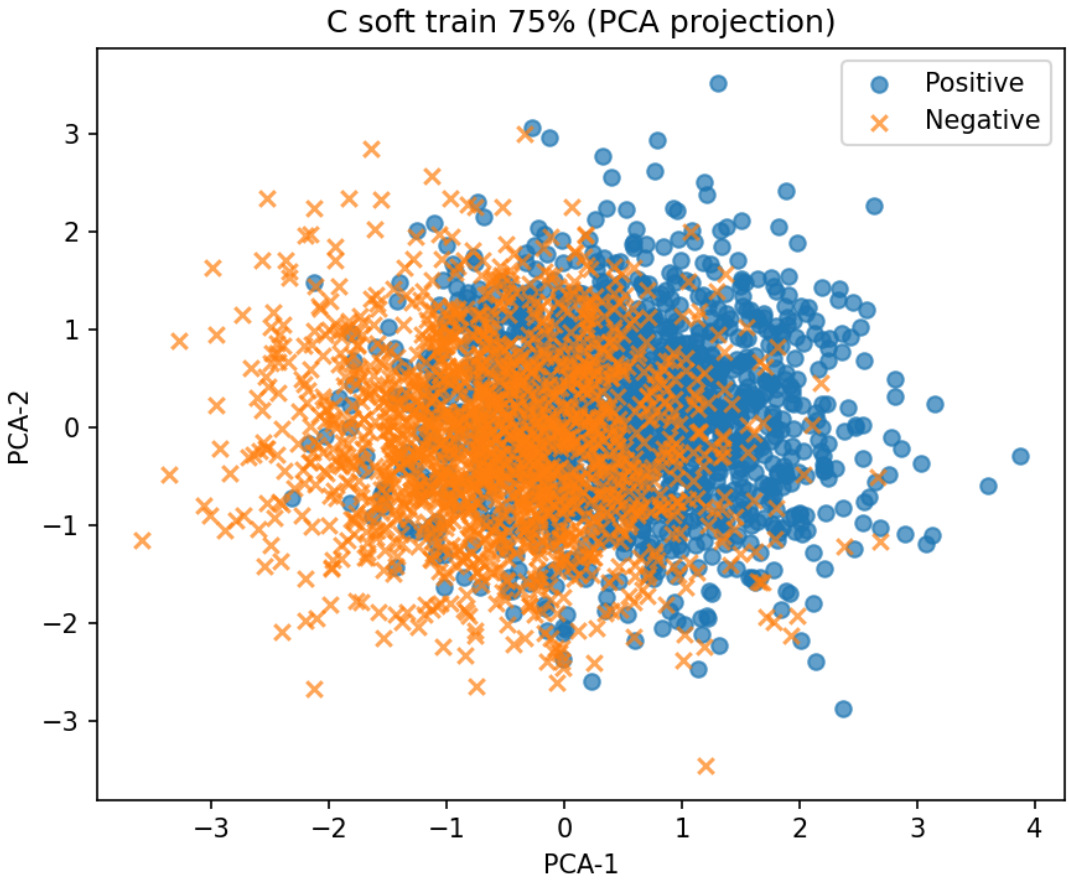
- **75/25** is generally better when you want the model to learn more, but for Dataset C it overfit to positives and failed to generalize.
- **25/75** gave a clearer picture of true performance on a large test set, even though accuracy was only 72.7%. For difficult datasets like C, a larger test set is preferable to evaluate real-world generalization.

d. Observations and overall insights:
The total error values (1106 vs 383) show that the perceptron converged but at high cost. Dataset C is **not linearly separable**, so the perceptron cannot cleanly divide the two classes. The poor precision and high false positive rates illustrate this limitation. The results suggest that a more advanced classifier (e.g., multi-layer perceptron, kernel methods) would be needed for Dataset C.

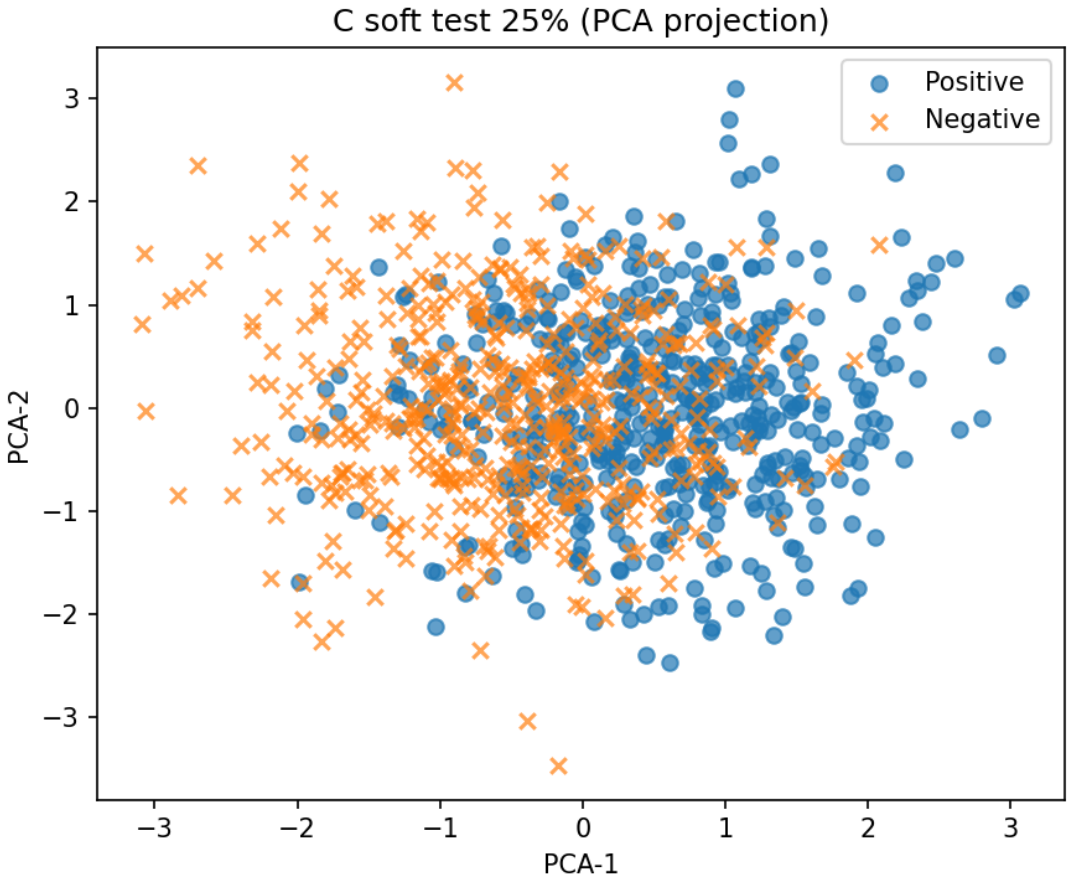
Soft Unipolar Activation

Training Split 75/25

- **Training Plot:**



- **Testing Plot:**



- **Training Total Error:** 360.45333643064373

Confusion Matrix (Testing Data)

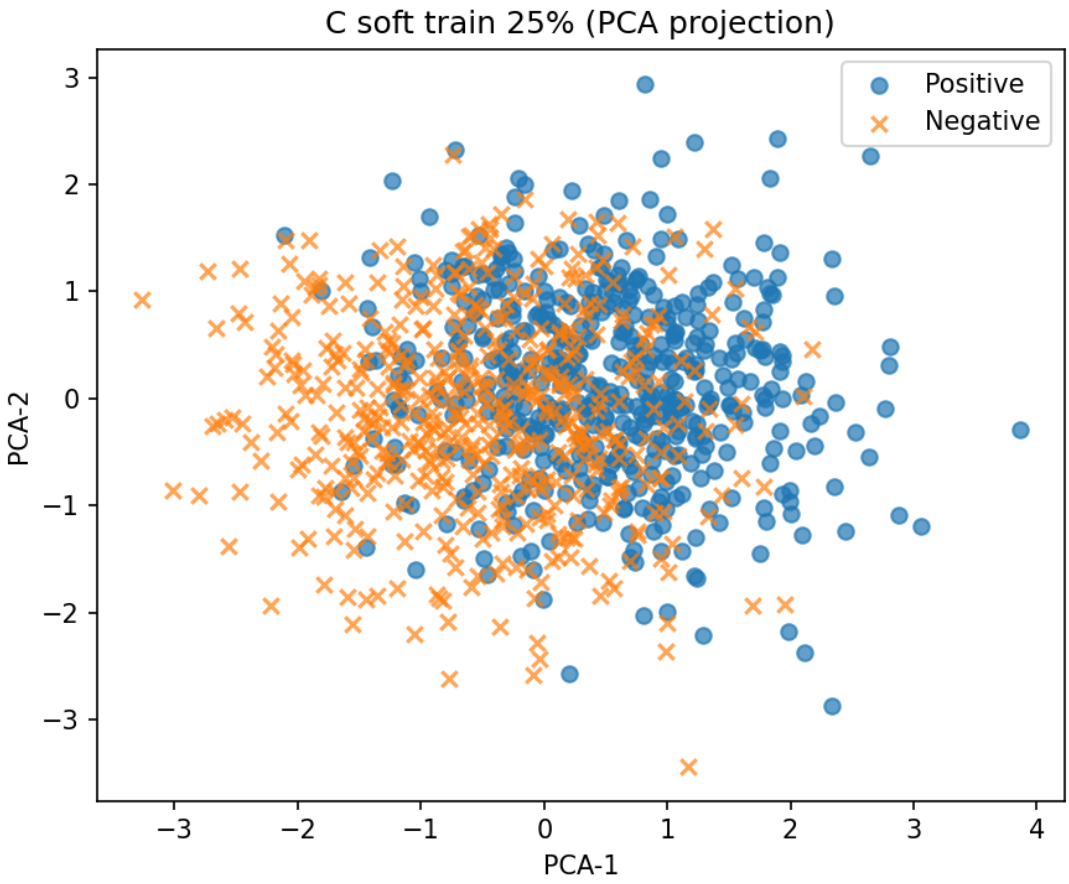
	Predicted Positive	Predicted Negative
Actual Positive	TP = 396	FN = 104
Actual Negative	FP = 188	TN = 312

Rates

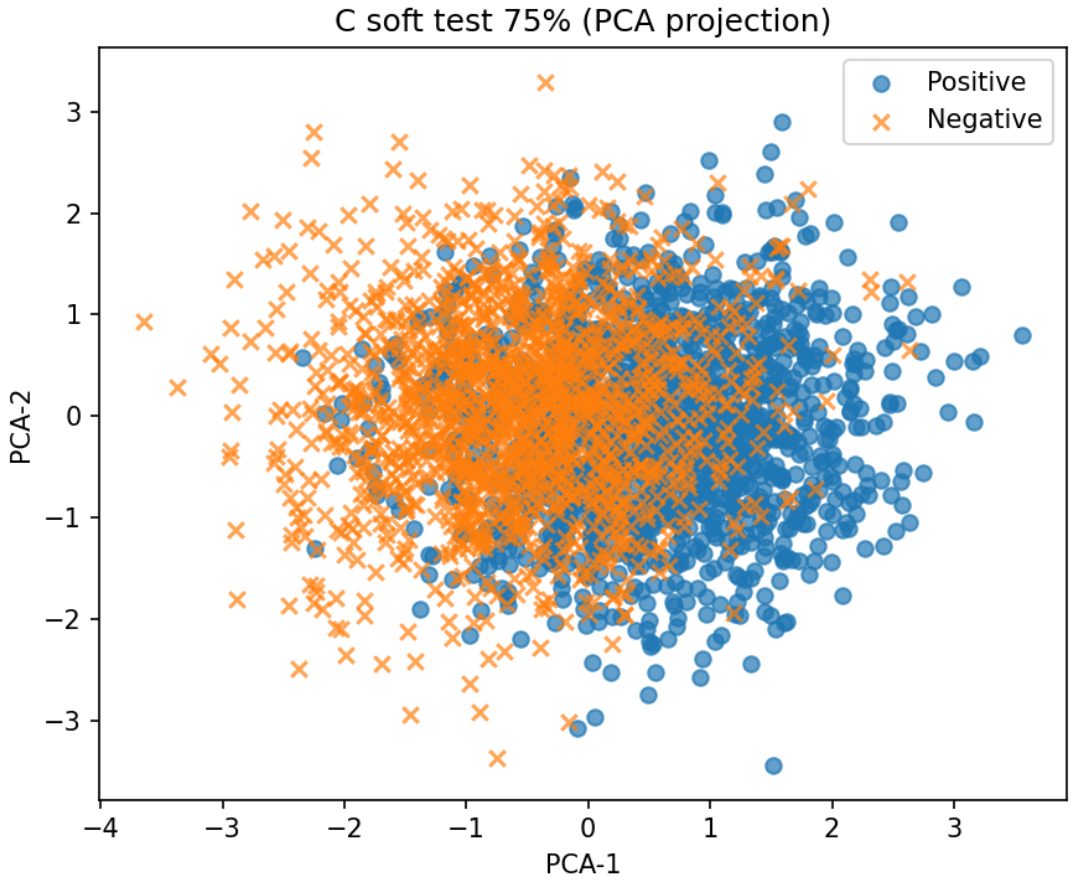
- Accuracy: 0.7080
- True Positive Rate (Recall): 0.7920
- False Positive Rate: 0.3760
- Precision: 0.6781
- F1 Score: 0.7306

Training Split 25/75

- **Training Plot:**



- **Testing Plot:** `C_test_soft_75.png`



- **Training Total Error:** 130.46207456477083

Confusion Matrix (Testing Data)

	Predicted Positive	Predicted Negative
Actual Positive	TP = 1461	FN = 39
Actual Negative	FP = 1422	TN = 78

Rates

- Accuracy: 0.5130
- True Positive Rate (Recall): 0.9740
- False Positive Rate: 0.9480
- Precision: 0.5068
- F1 Score: 0.6667

Discussion – Soft Unipolar

a. Are error rates different between 75/25 and 25/75? Why?
Yes, the error rates are very different. With the **75/25 split**, accuracy was about **70.8%**, while with the **25/75 split** accuracy collapsed to **51.3%**. In the 25/75 case, the model predicted almost everything as positive (very high recall at 97.4%) but misclassified nearly all negatives (FPR = 94.8%). This reflects poor generalization and suggests the perceptron struggled with Dataset C's overlapping features.

b. Effect of split ratio on performance and confusion matrices:

- **75/25 split:** Produced a more balanced outcome. While recall was good (79.2%), precision dropped to 67.8% due to many false positives (188).
- **25/75 split:** Produced extreme imbalance. High recall was maintained, but precision dropped to nearly 50%, and the false positive rate skyrocketed to almost 95%.
  This shows the split ratio heavily influenced performance, with more training data (75/25) producing a more reasonable but still flawed classifier.

c. When would you use 75/25 vs 25/75?

- **75/25** is the better choice for Dataset C because it at least produced moderate accuracy and lower error rates, even though false positives were still high.
- **25/75** would only be used if the priority was to maximize recall (catch almost all positives), but the trade-off in false positives makes it impractical for most real-world uses.

d. Observations and overall insights:
The error curves show that the soft perceptron converged but struggled with Dataset C's complexity. The smoother activation allowed partial separation of classes, but overlapping decision boundaries caused many false positives. Overall, the soft activation performed slightly better than the hard activation for Dataset C in terms of recall, but both struggled due to the dataset not being linearly separable.

Comparison: Hard vs Soft Unipolar

Metric	Hard 75/25	Soft 75/25	Hard 25/75	Soft 25/75
Accuracy	[ ]	[ ]	[ ]	[ ]
TPR	[ ]	[ ]	[ ]	[ ]

Metric	Hard 75/25	Soft 75/25	Hard 25/75	Soft 25/75
FPR	[ ]	[ ]	[ ]	[ ]
Precision	[ ]	[ ]	[ ]	[ ]
F1	[ ]	[ ]	[ ]	[ ]

Discussion:

- Which activation converged faster?
- Which achieved better generalization on smaller training data?
- Visual comparison of decision boundaries from plots.

Comparison to Project 1 Results

Insert your previously saved Project 1 confusion matrix here for side-by-side comparison:

Project	Activation	Accuracy	TPR	FPR	Precision	F1
Project 1	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]
Project 2	[Hard/Soft]	[ ]	[ ]	[ ]	[ ]	[ ]

Discussion:

- How does performance differ across datasets (A vs B)?
- Do you observe different learning behaviors?
- Which dataset or configuration yielded better generalization?
- What might explain these differences (feature scaling, class separability, sample size, etc.)?

Part 2 – Soft vs Hard Comparison

- Placeholder text.

Extra Credit

- When working with a dataset, I'd start by splitting it into training and testing sets—usually about 70–80% for training and 20–30% for testing—so the model has enough data to learn from while still leaving some for evaluation. I'd make sure the split keeps the same balance of classes or categories in both sets, so the model doesn't end up learning mostly from one type of data. This kind of proportional sampling helps it generalize better and avoids bias. I'd also use cross-validation to check that the model performs consistently across different splits, making sure it's learning real patterns rather than just memorizing examples.

Conclusion

- Key takeaways from Dataset A, B, and C experiments.
- Overall differences between hard vs soft unipolar activation.
- When to prefer larger training split vs smaller one.
- Might not need this section can include it if we want to.