guyvitelson_mmn11_ml_latest

March 19, 2025

1 11 - - 2025 - 203379706

##**If you run this within Google Collab, Dont Worry!** all the missing python files/directories/modules will be automatically feteched from my github repository

My GitHub Profile : https://github.com/v1t3ls0n

The Repository: https://github.com/v1t3ls0n/ml_intro_course_mmn11

Student ID: 203379706

1.1 Fetch Resources

1.1.1 External Code Imports (pip packages)

[1]:
```
import os
import shutil
import sys
import logging
import numpy as np # type: ignore
import matplotlib.pyplot as plt # type: ignore
import seaborn as sns # type: ignore
import time
import pandas as pd
```

1.1.2 Fetch Missing Files For Google Colab Env

[2]:
```
# %%capture run_output
# %matplotlib inline

if sys.platform != 'win32': # check if we are running on google collab
  repo_url = "https://github.com/v1t3ls0n/ml_intro_course_mmn11"
  repo_name = "ml_intro_course_mmn11"
  from tqdm.notebook import tqdm # type: ignore


  # Clone the repository if it doesn't exist
  if not os.path.exists(repo_name):
    os.system(f"git clone {repo_url}")
```

```
  # Construct the path to the repository directory
  repo_path = os.path.join(os.getcwd(), repo_name)

  # Add the repository directory to the Python path
  if repo_path not in sys.path:
    sys.path.insert(0, repo_path)

  # --- Extract 'core' and 'notebooks' directories ---
  def extract_directories(source_dir, destination_dir, dir_names):
      for dir_name in dir_names:
          source_path = os.path.join(source_dir, dir_name)
          destination_path = os.path.join(destination_dir, dir_name)
          if os.path.exists(source_path):
              shutil.copytree(source_path, destination_path, dirs_exist_ok=True)

  destination_path = "."
  # Extract the directories
  extract_directories(repo_path, destination_path, ["core"])
  project_root = os.path.abspath(os.path.join(os.getcwd(), '..'))
  sys.path.insert(0, project_root)
  if os.path.exists("ml_intro_course_mmn11"):
    shutil.rmtree("ml_intro_course_mmn11")
  if os.path.exists("sample_data"):
    shutil.rmtree("sample_data")
else:
  from tqdm import tqdm  # type: ignore
  current_dir = os.getcwd()  # Current working directory
  project_root = os.path.abspath(os.path.join(current_dir, '..'))  # Root␣
 ↪directory of the project
  sys.path.insert(0, project_root)
```

1.1.3 Internal Code Imports (original code)

[3]:
```
# ========== Internal Code Imports ==========

#Logger
from core.logger.config import logger

# Data Preprocessing
from core.data.mnist_loader import load_mnist
from core.data.data_preprocessing import preprocess_data

# Models
from core.models.perceptron.multi_class_perceptron import MultiClassPerceptron
from core.models.logistic_regression.softmax_lregression import␣
 ↪SoftmaxRegression
from core.models.linear_regression.linear_regression import  LinearRegression
```

```
# Performance & Plotting
from core.analysis.evaluation_functions import (
    evaluate_model,
    aggregate_iteration_losses,
    aggregate_iteration_losses_softmax
)

from core.analysis.plotting import (
    plot_confusion_matrix_annotated,
    plot_error_curves,
    plot_accuracy_vs_max_iter,
    plot_runtime_vs_max_iter,
    plot_performance_summary_extended,
    plot_train_curves_three_models,
    plot_metric_vs_learning_rate,
    plot_accuracy_vs_max_iter_4models,
    plot_runtime_vs_max_iter_4models,
    plot_accuracy_vs_runtime,
    plot_performance_summary_extended_by_runtime,
    plot_performance_summary_4models_by_runtime,
    plot_accuracy_vs_runtime_4models
)

logger = logging.getLogger("MyGlobalLogger") # configured in core/logger/config.
 ↪py
```

2 Overview

2.1 MNIST Digit Classification Report

2.1.1 Approach

Data Preprocessing The MNIST dataset was prepared by: - Splitting into training (60,000 samples) and test sets (10,000 samples). - Normalizing pixel values to the [0,1] range. - Flattening images into vectors (784 pixels plus 1 bias term). - Encoding labels into one-hot vectors.

Model Implementation

- **Multi-Class Perceptron:**
  - One-vs-all strategy implemented with standard Perceptron and Pocket Perceptron algorithms.
- **Softmax Regression:**
  - Implemented using cross-entropy loss and adaptive learning rates (AdaGrad).
  - Included early stopping based on loss improvement.
- **Linear Regression:**
  - Utilized mean squared error loss with gradient descent.
  - AdaGrad adaptive learning rate and early stopping were applied.

2.1.2 Results

- **Accuracy:**
  - Softmax Regression achieved the highest accuracy.
  - Multi-class Pocket Perceptron showed good performance, surpassing standard Perceptron.
  - Linear Regression exhibited relatively lower accuracy due to its limitations for classification tasks.

Confusion Matrices and Metrics

- Softmax Regression demonstrated the lowest misclassification rates across digits.
- Pocket Perceptron reduced errors compared to standard Perceptron, indicating improved robustness.
- Sensitivity and accuracy clearly highlighted Softmax Regression as superior for multi-class digit classification.

2.1.3 Discussion

- Softmax Regression proved best for digit classification, providing reliable probability estimations and stable convergence.
- Pocket Perceptron algorithm offered notable improvements over standard Perceptron, highlighting its utility in non-linearly separable scenarios.
- Linear Regression's limitations in classification tasks were evident, reaffirming theoretical expectations.

2.1.4 Conclusions

- Softmax Regression is the most suitable algorithm for multi-class digit recognition problems.
- Pocket Perceptron serves as an effective alternative, offering a balance between simplicity and performance.
- Linear Regression, while straightforward, is suboptimal for classification due to its inherent limitations.

3 Choose Run Parameters (Significant Effect On Model's Runtime!)

[4]:
```
######################################################################
# SEPARATE RUN PARAMETERS FOR PERCEPTRONS vs. REGRESSIONS
######################################################################

# Perceptrons (Clean & Pocket) iteration-based run
perceptron_max_iter_values = [100,1000]  # for Clean PLA & Pocket PLA
# Logging the run parameters
logger.info(f"=== Perceptron Run Parameters ===")
logger.info(f"max_iter_values = {perceptron_max_iter_values}")
```

```
# Regression (Softmax & Linear) run parameters.
learning_rates = [0.1]  # for Softmax & Linear Regression
iteration_counts = [1000,2000]
regression_run_configs = [
    {
        "label": f"LR={lr}/Iter={it}",
        "learning_rate": lr,
        "max_iter": it
    }
    for lr in learning_rates
    for it in iteration_counts
]

logger.info(f"=== Regression Run Parameters ===")
for cfg in regression_run_configs:
    logger.info(f"{cfg['label']} -> learning_rate={cfg['learning_rate']},␣
 ↪max_iter={cfg['max_iter']}")
```

```
INFO - === Perceptron Run Parameters ===
INFO - max_iter_values = [100, 1000]
INFO - === Regression Run Parameters ===
INFO - LR=0.1/Iter=1000 -> learning_rate=0.1, max_iter=1000
INFO - LR=0.1/Iter=2000 -> learning_rate=0.1, max_iter=2000
INFO - max_iter_values = [100, 1000]
INFO - === Regression Run Parameters ===
INFO - LR=0.1/Iter=1000 -> learning_rate=0.1, max_iter=1000
INFO - LR=0.1/Iter=2000 -> learning_rate=0.1, max_iter=2000
```

4 Load and Preprocess the MNIST Dataset

[5]:
```
'''
We'll load the MNIST dataset using our custom loader (`mnist_loader`) and then␣
 ↪apply preprocessing (`data_preprocessing`).
The preprocessing step normalizes each image to the range [0, 1] and adds a␣
 ↪bias term, resulting in input samples with 785 features.
This setup ensures that the training set contains 60,000 samples and the test␣
 ↪set 10,000 samples, preparing the data for the subsequent classification␣
 ↪tasks.
'''

# New section
# Load raw MNIST data (X: images, y: labels)
X_raw, y_raw = load_mnist()
```

```
logger.info("Raw MNIST data shapes: X_raw: %s, y_raw: %s", X_raw.shape, y_raw.
 ↪shape)

# Preprocess (normalize & add bias = True)
X = preprocess_data(X_raw, add_bias=True, normalize=True)
logger.info("Preprocessed shape: %s", X.shape)

# Split into train/test manually or with 60k/10k as the task suggests
X_train, y_train = X[:60000], y_raw[:60000]
X_test,  y_test  = X[60000:], y_raw[60000:]

logger.info("Train set: X_train: %s, y_train: %s", X_train.shape, y_train.shape)
logger.info("Test set: X_test: %s, y_test: %s", X_test.shape, y_test.shape)
```

```
INFO - Raw MNIST data shapes: X_raw: (70000, 784), y_raw: (70000,)
INFO - Preprocessed shape: (70000, 785)
INFO - Train set: X_train: (60000, 785), y_train: (60000,)
INFO - Test set: X_test: (10000, 785), y_test: (10000,)
```

5 Train

[6]:
```
# ============================================================
# TRAINING CELL
# ============================================================

# 1) Dictionaries to store trained models
trained_models_clean   = {}
trained_models_pocket  = {}
trained_models_softmax = {}
trained_models_linear  = {}

# 2) Train Regression Models (Softmax & Linear)
logger.info("=== TRAINING REGRESSION MODELS (Softmax & Linear) ===")
for cfg in tqdm(regression_run_configs, desc="Train Regressions"):
    lr_val = cfg["learning_rate"]
    max_iter_val = cfg["max_iter"]
    label = cfg["label"]  # e.g. "LR=0.001/Iter=1000"

    # --- Softmax ---
    logger.info(f"--- Softmax {label} ---")
    s_model = SoftmaxRegression(
        num_classes=10,
        max_iter=max_iter_val,
        learning_rate=lr_val,
        adaptive_lr=True
    )
    s_model.fit(X_train, y_train)
```

```
    trained_models_softmax[(lr_val, max_iter_val)] = s_model

    # --- Linear ---
    logger.info(f"--- Linear Regression {label} ---")
    lin_model = LinearRegression(
        num_classes=10,
        max_iter=max_iter_val,
        learning_rate=lr_val,
        adaptive_lr=True,
        early_stopping=False
    )
    lin_model.fit(X_train, y_train)
    trained_models_linear[(lr_val, max_iter_val)] = lin_model

logger.info("Training complete for Softmax and Linear.")

# 3) Train Perceptron Models (Clean & Pocket)
logger.info("=== TRAINING PERCEPTRON MODELS (Clean & Pocket) ===")
for max_iter in tqdm(perceptron_max_iter_values, desc="Train Clean & Pocket"):
    logger.info(f"--- Clean PLA, max_iter={max_iter} ---")
    clean_perc = MultiClassPerceptron(num_classes=10, max_iter=max_iter,␣
 ↪use_pocket=False)
    clean_perc.fit(X_train, y_train)
    trained_models_clean[max_iter] = clean_perc

    logger.info(f"--- Pocket PLA, max_iter={max_iter} ---")
    pocket_perc = MultiClassPerceptron(num_classes=10, max_iter=max_iter,␣
 ↪use_pocket=True)
    pocket_perc.fit(X_train, y_train)
    trained_models_pocket[max_iter] = pocket_perc

logger.info("Training complete for Clean PLA and Pocket PLA.")
logger.info("=== ALL TRAINING COMPLETE ===")
```

```
INFO - === TRAINING REGRESSION MODELS (Softmax & Linear) ===
Train Regressions:   0%|          | 0/2 [00:00<?, ?it/s]INFO - --- Softmax
LR=0.1/Iter=1000 ---
Train Regressions:   0%|          | 0/2 [00:00<?, ?it/s]INFO - --- Softmax
LR=0.1/Iter=1000 ---
INFO - Iter 1/1000, Loss: 2.3908, Avg Adaptive LR: 12.669847
INFO - Iter 11/1000, Loss: 0.5140, Avg Adaptive LR: 2.937770
INFO - Iter 21/1000, Loss: 0.3770, Avg Adaptive LR: 2.876440
INFO - Iter 31/1000, Loss: 0.3518, Avg Adaptive LR: 2.875163
INFO - Iter 41/1000, Loss: 0.3359, Avg Adaptive LR: 2.874354
INFO - Iter 51/1000, Loss: 0.3247, Avg Adaptive LR: 2.873774
INFO - Iter 61/1000, Loss: 0.3161, Avg Adaptive LR: 2.873330
INFO - Iter 71/1000, Loss: 0.3093, Avg Adaptive LR: 2.872977
```

```
INFO - Iter 81/1000, Loss: 0.3036, Avg Adaptive LR: 2.872685
INFO - Iter 91/1000, Loss: 0.2989, Avg Adaptive LR: 2.872440
INFO - Iter 101/1000, Loss: 0.2949, Avg Adaptive LR: 2.872229
INFO - Iter 111/1000, Loss: 0.2913, Avg Adaptive LR: 2.872046
INFO - Iter 121/1000, Loss: 0.2882, Avg Adaptive LR: 2.871884
INFO - Iter 131/1000, Loss: 0.2854, Avg Adaptive LR: 2.871739
INFO - Iter 141/1000, Loss: 0.2830, Avg Adaptive LR: 2.871610
INFO - Iter 151/1000, Loss: 0.2807, Avg Adaptive LR: 2.871492
INFO - Iter 161/1000, Loss: 0.2787, Avg Adaptive LR: 2.871385
INFO - Iter 171/1000, Loss: 0.2768, Avg Adaptive LR: 2.871287
INFO - Iter 181/1000, Loss: 0.2750, Avg Adaptive LR: 2.871196
INFO - Iter 191/1000, Loss: 0.2734, Avg Adaptive LR: 2.871112
INFO - Iter 201/1000, Loss: 0.2719, Avg Adaptive LR: 2.871034
INFO - Iter 211/1000, Loss: 0.2705, Avg Adaptive LR: 2.870961
INFO - Iter 221/1000, Loss: 0.2692, Avg Adaptive LR: 2.870893
INFO - Iter 231/1000, Loss: 0.2680, Avg Adaptive LR: 2.870829
INFO - Iter 241/1000, Loss: 0.2668, Avg Adaptive LR: 2.870768
INFO - Iter 251/1000, Loss: 0.2658, Avg Adaptive LR: 2.870711
INFO - Iter 261/1000, Loss: 0.2647, Avg Adaptive LR: 2.870657
INFO - Iter 271/1000, Loss: 0.2637, Avg Adaptive LR: 2.870606
INFO - Iter 281/1000, Loss: 0.2628, Avg Adaptive LR: 2.870557
INFO - Iter 291/1000, Loss: 0.2619, Avg Adaptive LR: 2.870511
INFO - Iter 301/1000, Loss: 0.2611, Avg Adaptive LR: 2.870466
INFO - Iter 311/1000, Loss: 0.2603, Avg Adaptive LR: 2.870424
INFO - Iter 321/1000, Loss: 0.2595, Avg Adaptive LR: 2.870383
INFO - Iter 331/1000, Loss: 0.2587, Avg Adaptive LR: 2.870345
INFO - Iter 341/1000, Loss: 0.2580, Avg Adaptive LR: 2.870307
INFO - Iter 351/1000, Loss: 0.2573, Avg Adaptive LR: 2.870271
INFO - Iter 361/1000, Loss: 0.2567, Avg Adaptive LR: 2.870237
INFO - Iter 371/1000, Loss: 0.2560, Avg Adaptive LR: 2.870203
INFO - Iter 381/1000, Loss: 0.2554, Avg Adaptive LR: 2.870171
INFO - Iter 391/1000, Loss: 0.2548, Avg Adaptive LR: 2.870140
INFO - Iter 401/1000, Loss: 0.2543, Avg Adaptive LR: 2.870110
INFO - Iter 411/1000, Loss: 0.2537, Avg Adaptive LR: 2.870081
INFO - Iter 421/1000, Loss: 0.2532, Avg Adaptive LR: 2.870053
INFO - Iter 431/1000, Loss: 0.2527, Avg Adaptive LR: 2.870026
INFO - Iter 441/1000, Loss: 0.2521, Avg Adaptive LR: 2.869999
INFO - Iter 451/1000, Loss: 0.2517, Avg Adaptive LR: 2.869974
INFO - Iter 461/1000, Loss: 0.2512, Avg Adaptive LR: 2.869949
INFO - Iter 471/1000, Loss: 0.2507, Avg Adaptive LR: 2.869925
INFO - Iter 481/1000, Loss: 0.2503, Avg Adaptive LR: 2.869901
INFO - Iter 491/1000, Loss: 0.2498, Avg Adaptive LR: 2.869878
INFO - Iter 501/1000, Loss: 0.2494, Avg Adaptive LR: 2.869856
INFO - Iter 511/1000, Loss: 0.2490, Avg Adaptive LR: 2.869835
INFO - Iter 521/1000, Loss: 0.2486, Avg Adaptive LR: 2.869813
INFO - Iter 531/1000, Loss: 0.2482, Avg Adaptive LR: 2.869793
INFO - Iter 541/1000, Loss: 0.2478, Avg Adaptive LR: 2.869773
INFO - Iter 551/1000, Loss: 0.2474, Avg Adaptive LR: 2.869753
```

```
INFO - Iter 561/1000, Loss: 0.2471, Avg Adaptive LR: 2.869734
INFO - Iter 571/1000, Loss: 0.2467, Avg Adaptive LR: 2.869715
INFO - Iter 581/1000, Loss: 0.2464, Avg Adaptive LR: 2.869697
INFO - Iter 591/1000, Loss: 0.2460, Avg Adaptive LR: 2.869679
INFO - Iter 601/1000, Loss: 0.2457, Avg Adaptive LR: 2.869661
INFO - Iter 611/1000, Loss: 0.2454, Avg Adaptive LR: 2.869644
INFO - Iter 621/1000, Loss: 0.2450, Avg Adaptive LR: 2.869628
INFO - Iter 631/1000, Loss: 0.2447, Avg Adaptive LR: 2.869611
INFO - Iter 641/1000, Loss: 0.2444, Avg Adaptive LR: 2.869595
INFO - Iter 651/1000, Loss: 0.2441, Avg Adaptive LR: 2.869579
INFO - Iter 661/1000, Loss: 0.2438, Avg Adaptive LR: 2.869564
INFO - Iter 671/1000, Loss: 0.2435, Avg Adaptive LR: 2.869549
INFO - Iter 681/1000, Loss: 0.2433, Avg Adaptive LR: 2.869534
INFO - Iter 691/1000, Loss: 0.2430, Avg Adaptive LR: 2.869519
INFO - Iter 701/1000, Loss: 0.2427, Avg Adaptive LR: 2.869505
INFO - Iter 711/1000, Loss: 0.2424, Avg Adaptive LR: 2.869491
INFO - Iter 721/1000, Loss: 0.2422, Avg Adaptive LR: 2.869477
INFO - Iter 731/1000, Loss: 0.2419, Avg Adaptive LR: 2.869464
INFO - Iter 741/1000, Loss: 0.2417, Avg Adaptive LR: 2.869451
INFO - Iter 751/1000, Loss: 0.2414, Avg Adaptive LR: 2.869437
INFO - Iter 761/1000, Loss: 0.2412, Avg Adaptive LR: 2.869425
INFO - Iter 771/1000, Loss: 0.2409, Avg Adaptive LR: 2.869412
INFO - Iter 781/1000, Loss: 0.2407, Avg Adaptive LR: 2.869400
INFO - Iter 791/1000, Loss: 0.2405, Avg Adaptive LR: 2.869387
INFO - Iter 801/1000, Loss: 0.2402, Avg Adaptive LR: 2.869375
INFO - Iter 811/1000, Loss: 0.2400, Avg Adaptive LR: 2.869364
INFO - Iter 821/1000, Loss: 0.2398, Avg Adaptive LR: 2.869352
INFO - Iter 831/1000, Loss: 0.2396, Avg Adaptive LR: 2.869341
INFO - Iter 841/1000, Loss: 0.2394, Avg Adaptive LR: 2.869329
INFO - Iter 851/1000, Loss: 0.2391, Avg Adaptive LR: 2.869318
INFO - Iter 861/1000, Loss: 0.2389, Avg Adaptive LR: 2.869307
INFO - Iter 871/1000, Loss: 0.2387, Avg Adaptive LR: 2.869297
INFO - Iter 881/1000, Loss: 0.2385, Avg Adaptive LR: 2.869286
INFO - Iter 891/1000, Loss: 0.2383, Avg Adaptive LR: 2.869275
INFO - Iter 901/1000, Loss: 0.2381, Avg Adaptive LR: 2.869265
INFO - Iter 911/1000, Loss: 0.2379, Avg Adaptive LR: 2.869255
INFO - Iter 921/1000, Loss: 0.2377, Avg Adaptive LR: 2.869245
INFO - Iter 931/1000, Loss: 0.2376, Avg Adaptive LR: 2.869235
INFO - Iter 941/1000, Loss: 0.2374, Avg Adaptive LR: 2.869225
INFO - Iter 951/1000, Loss: 0.2372, Avg Adaptive LR: 2.869216
INFO - Iter 961/1000, Loss: 0.2370, Avg Adaptive LR: 2.869206
INFO - Iter 971/1000, Loss: 0.2368, Avg Adaptive LR: 2.869197
INFO - Iter 981/1000, Loss: 0.2367, Avg Adaptive LR: 2.869188
INFO - Iter 991/1000, Loss: 0.2365, Avg Adaptive LR: 2.869179
INFO - SoftmaxRegression training completed in 39.22 seconds.
INFO - --- Linear Regression LR=0.1/Iter=1000 ---
INFO - Iter 100/1000, Loss: 0.9795, Gradient Norm: 19.0261, Avg Adaptive LR:
1.3953058746703297
```

```
INFO - Iter 200/1000, Loss: 0.4939, Gradient Norm: 13.2273, Avg Adaptive LR:
0.9907771626676626
INFO - Iter 300/1000, Loss: 0.3175, Gradient Norm: 10.3436, Avg Adaptive LR:
0.8105121597887128
INFO - Iter 400/1000, Loss: 0.2320, Gradient Norm: 8.6050, Avg Adaptive LR:
0.7027197517330492
INFO - Iter 500/1000, Loss: 0.1824, Gradient Norm: 7.4104, Avg Adaptive LR:
0.6290330163111517
INFO - Iter 600/1000, Loss: 0.1520, Gradient Norm: 6.5727, Avg Adaptive LR:
0.5745438847898712
INFO - Iter 700/1000, Loss: 0.1328, Gradient Norm: 5.9872, Avg Adaptive LR:
0.532162511232368
INFO - Iter 800/1000, Loss: 0.1182, Gradient Norm: 5.4959, Avg Adaptive LR:
0.49793527286140227
INFO - Iter 900/1000, Loss: 0.1077, Gradient Norm: 5.1169, Avg Adaptive LR:
0.469602984759602
INFO - Iter 1000/1000, Loss: 0.0997, Gradient Norm: 4.8082, Avg Adaptive LR:
0.44559667277505965
INFO - LinearRegressionClassifier training completed in 31.86 seconds.
Train Regressions:  50%|        | 1/2 [01:11<01:11, 71.09s/it]INFO - ---
Softmax LR=0.1/Iter=2000 ---
INFO - Iter 1/2000, Loss: 2.4012, Avg Adaptive LR: 13.713582
INFO - Iter 11/2000, Loss: 0.4419, Avg Adaptive LR: 2.999741
INFO - Iter 21/2000, Loss: 0.3784, Avg Adaptive LR: 2.996271
INFO - Iter 31/2000, Loss: 0.3530, Avg Adaptive LR: 2.994877
INFO - Iter 41/2000, Loss: 0.3370, Avg Adaptive LR: 2.993987
INFO - Iter 51/2000, Loss: 0.3255, Avg Adaptive LR: 2.993346
INFO - Iter 61/2000, Loss: 0.3167, Avg Adaptive LR: 2.992854
INFO - Iter 71/2000, Loss: 0.3097, Avg Adaptive LR: 2.992461
INFO - Iter 81/2000, Loss: 0.3040, Avg Adaptive LR: 2.992137
INFO - Iter 91/2000, Loss: 0.2991, Avg Adaptive LR: 2.991864
INFO - Iter 101/2000, Loss: 0.2950, Avg Adaptive LR: 2.991631
INFO - Iter 111/2000, Loss: 0.2914, Avg Adaptive LR: 2.991428
INFO - Iter 121/2000, Loss: 0.2882, Avg Adaptive LR: 2.991249
INFO - Iter 131/2000, Loss: 0.2854, Avg Adaptive LR: 2.991090
INFO - Iter 141/2000, Loss: 0.2829, Avg Adaptive LR: 2.990948
INFO - Iter 151/2000, Loss: 0.2806, Avg Adaptive LR: 2.990819
INFO - Iter 161/2000, Loss: 0.2786, Avg Adaptive LR: 2.990702
INFO - Iter 171/2000, Loss: 0.2767, Avg Adaptive LR: 2.990595
INFO - Iter 181/2000, Loss: 0.2749, Avg Adaptive LR: 2.990497
INFO - Iter 191/2000, Loss: 0.2733, Avg Adaptive LR: 2.990406
INFO - Iter 201/2000, Loss: 0.2718, Avg Adaptive LR: 2.990321
INFO - Iter 211/2000, Loss: 0.2704, Avg Adaptive LR: 2.990242
INFO - Iter 221/2000, Loss: 0.2691, Avg Adaptive LR: 2.990169
INFO - Iter 231/2000, Loss: 0.2679, Avg Adaptive LR: 2.990100
INFO - Iter 241/2000, Loss: 0.2668, Avg Adaptive LR: 2.990035
INFO - Iter 251/2000, Loss: 0.2657, Avg Adaptive LR: 2.989973
INFO - Iter 261/2000, Loss: 0.2647, Avg Adaptive LR: 2.989915
```

```
INFO - Iter 271/2000, Loss: 0.2637, Avg Adaptive LR: 2.989860
INFO - Iter 281/2000, Loss: 0.2628, Avg Adaptive LR: 2.989808
INFO - Iter 291/2000, Loss: 0.2619, Avg Adaptive LR: 2.989758
INFO - Iter 301/2000, Loss: 0.2611, Avg Adaptive LR: 2.989711
INFO - Iter 311/2000, Loss: 0.2603, Avg Adaptive LR: 2.989666
INFO - Iter 321/2000, Loss: 0.2595, Avg Adaptive LR: 2.989622
INFO - Iter 331/2000, Loss: 0.2588, Avg Adaptive LR: 2.989581
INFO - Iter 341/2000, Loss: 0.2581, Avg Adaptive LR: 2.989541
INFO - Iter 351/2000, Loss: 0.2574, Avg Adaptive LR: 2.989503
INFO - Iter 361/2000, Loss: 0.2567, Avg Adaptive LR: 2.989466
INFO - Iter 371/2000, Loss: 0.2561, Avg Adaptive LR: 2.989430
INFO - Iter 381/2000, Loss: 0.2555, Avg Adaptive LR: 2.989396
INFO - Iter 391/2000, Loss: 0.2549, Avg Adaptive LR: 2.989363
INFO - Iter 401/2000, Loss: 0.2544, Avg Adaptive LR: 2.989331
INFO - Iter 411/2000, Loss: 0.2538, Avg Adaptive LR: 2.989301
INFO - Iter 421/2000, Loss: 0.2533, Avg Adaptive LR: 2.989271
INFO - Iter 431/2000, Loss: 0.2528, Avg Adaptive LR: 2.989242
INFO - Iter 441/2000, Loss: 0.2523, Avg Adaptive LR: 2.989214
INFO - Iter 451/2000, Loss: 0.2518, Avg Adaptive LR: 2.989187
INFO - Iter 461/2000, Loss: 0.2514, Avg Adaptive LR: 2.989160
INFO - Iter 471/2000, Loss: 0.2509, Avg Adaptive LR: 2.989134
INFO - Iter 481/2000, Loss: 0.2505, Avg Adaptive LR: 2.989109
INFO - Iter 491/2000, Loss: 0.2500, Avg Adaptive LR: 2.989085
INFO - Iter 501/2000, Loss: 0.2496, Avg Adaptive LR: 2.989062
INFO - Iter 511/2000, Loss: 0.2492, Avg Adaptive LR: 2.989038
INFO - Iter 521/2000, Loss: 0.2488, Avg Adaptive LR: 2.989016
INFO - Iter 531/2000, Loss: 0.2484, Avg Adaptive LR: 2.988994
INFO - Iter 541/2000, Loss: 0.2481, Avg Adaptive LR: 2.988973
INFO - Iter 551/2000, Loss: 0.2477, Avg Adaptive LR: 2.988952
INFO - Iter 561/2000, Loss: 0.2473, Avg Adaptive LR: 2.988932
INFO - Iter 571/2000, Loss: 0.2470, Avg Adaptive LR: 2.988912
INFO - Iter 581/2000, Loss: 0.2466, Avg Adaptive LR: 2.988892
INFO - Iter 591/2000, Loss: 0.2463, Avg Adaptive LR: 2.988873
INFO - Iter 601/2000, Loss: 0.2460, Avg Adaptive LR: 2.988854
INFO - Iter 611/2000, Loss: 0.2457, Avg Adaptive LR: 2.988836
INFO - Iter 621/2000, Loss: 0.2453, Avg Adaptive LR: 2.988818
INFO - Iter 631/2000, Loss: 0.2450, Avg Adaptive LR: 2.988801
INFO - Iter 641/2000, Loss: 0.2447, Avg Adaptive LR: 2.988784
INFO - Iter 651/2000, Loss: 0.2444, Avg Adaptive LR: 2.988767
INFO - Iter 661/2000, Loss: 0.2441, Avg Adaptive LR: 2.988751
INFO - Iter 671/2000, Loss: 0.2439, Avg Adaptive LR: 2.988734
INFO - Iter 681/2000, Loss: 0.2436, Avg Adaptive LR: 2.988719
INFO - Iter 691/2000, Loss: 0.2433, Avg Adaptive LR: 2.988703
INFO - Iter 701/2000, Loss: 0.2430, Avg Adaptive LR: 2.988688
INFO - Iter 711/2000, Loss: 0.2428, Avg Adaptive LR: 2.988673
INFO - Iter 721/2000, Loss: 0.2425, Avg Adaptive LR: 2.988658
INFO - Iter 731/2000, Loss: 0.2423, Avg Adaptive LR: 2.988644
INFO - Iter 741/2000, Loss: 0.2420, Avg Adaptive LR: 2.988629
```

```
INFO - Iter 751/2000, Loss: 0.2418, Avg Adaptive LR: 2.988615
INFO - Iter 761/2000, Loss: 0.2415, Avg Adaptive LR: 2.988602
INFO - Iter 771/2000, Loss: 0.2413, Avg Adaptive LR: 2.988588
INFO - Iter 781/2000, Loss: 0.2411, Avg Adaptive LR: 2.988575
INFO - Iter 791/2000, Loss: 0.2408, Avg Adaptive LR: 2.988562
INFO - Iter 801/2000, Loss: 0.2406, Avg Adaptive LR: 2.988549
INFO - Iter 811/2000, Loss: 0.2404, Avg Adaptive LR: 2.988536
INFO - Iter 821/2000, Loss: 0.2402, Avg Adaptive LR: 2.988524
INFO - Iter 831/2000, Loss: 0.2399, Avg Adaptive LR: 2.988512
INFO - Iter 841/2000, Loss: 0.2397, Avg Adaptive LR: 2.988499
INFO - Iter 851/2000, Loss: 0.2395, Avg Adaptive LR: 2.988488
INFO - Iter 861/2000, Loss: 0.2393, Avg Adaptive LR: 2.988476
INFO - Iter 871/2000, Loss: 0.2391, Avg Adaptive LR: 2.988464
INFO - Iter 881/2000, Loss: 0.2389, Avg Adaptive LR: 2.988453
INFO - Iter 891/2000, Loss: 0.2387, Avg Adaptive LR: 2.988442
INFO - Iter 901/2000, Loss: 0.2385, Avg Adaptive LR: 2.988431
INFO - Iter 911/2000, Loss: 0.2383, Avg Adaptive LR: 2.988420
INFO - Iter 921/2000, Loss: 0.2381, Avg Adaptive LR: 2.988409
INFO - Iter 931/2000, Loss: 0.2379, Avg Adaptive LR: 2.988398
INFO - Iter 941/2000, Loss: 0.2378, Avg Adaptive LR: 2.988388
INFO - Iter 951/2000, Loss: 0.2376, Avg Adaptive LR: 2.988377
INFO - Iter 961/2000, Loss: 0.2374, Avg Adaptive LR: 2.988367
INFO - Iter 971/2000, Loss: 0.2372, Avg Adaptive LR: 2.988357
INFO - Iter 981/2000, Loss: 0.2370, Avg Adaptive LR: 2.988347
INFO - Iter 991/2000, Loss: 0.2369, Avg Adaptive LR: 2.988337
INFO - Iter 1001/2000, Loss: 0.2367, Avg Adaptive LR: 2.988328
INFO - Iter 1011/2000, Loss: 0.2365, Avg Adaptive LR: 2.988318
INFO - Iter 1021/2000, Loss: 0.2364, Avg Adaptive LR: 2.988309
INFO - Iter 1031/2000, Loss: 0.2362, Avg Adaptive LR: 2.988299
INFO - Iter 1041/2000, Loss: 0.2360, Avg Adaptive LR: 2.988290
INFO - Iter 1051/2000, Loss: 0.2359, Avg Adaptive LR: 2.988281
INFO - Iter 1061/2000, Loss: 0.2357, Avg Adaptive LR: 2.988272
INFO - Iter 1071/2000, Loss: 0.2356, Avg Adaptive LR: 2.988263
INFO - Iter 1081/2000, Loss: 0.2354, Avg Adaptive LR: 2.988254
INFO - Iter 1091/2000, Loss: 0.2353, Avg Adaptive LR: 2.988246
INFO - Iter 1101/2000, Loss: 0.2351, Avg Adaptive LR: 2.988237
INFO - Iter 1111/2000, Loss: 0.2350, Avg Adaptive LR: 2.988229
INFO - Iter 1121/2000, Loss: 0.2348, Avg Adaptive LR: 2.988220
INFO - Iter 1131/2000, Loss: 0.2347, Avg Adaptive LR: 2.988212
INFO - Iter 1141/2000, Loss: 0.2345, Avg Adaptive LR: 2.988204
INFO - Iter 1151/2000, Loss: 0.2344, Avg Adaptive LR: 2.988196
INFO - Iter 1161/2000, Loss: 0.2342, Avg Adaptive LR: 2.988188
INFO - Iter 1171/2000, Loss: 0.2341, Avg Adaptive LR: 2.988180
INFO - Iter 1181/2000, Loss: 0.2340, Avg Adaptive LR: 2.988172
INFO - Iter 1191/2000, Loss: 0.2338, Avg Adaptive LR: 2.988164
INFO - Iter 1201/2000, Loss: 0.2337, Avg Adaptive LR: 2.988157
INFO - Iter 1211/2000, Loss: 0.2336, Avg Adaptive LR: 2.988149
INFO - Iter 1221/2000, Loss: 0.2334, Avg Adaptive LR: 2.988141
```

```
INFO - Iter 1231/2000, Loss: 0.2333, Avg Adaptive LR: 2.988134
INFO - Iter 1241/2000, Loss: 0.2332, Avg Adaptive LR: 2.988127
INFO - Iter 1251/2000, Loss: 0.2330, Avg Adaptive LR: 2.988119
INFO - Iter 1261/2000, Loss: 0.2329, Avg Adaptive LR: 2.988112
INFO - Iter 1271/2000, Loss: 0.2328, Avg Adaptive LR: 2.988105
INFO - Iter 1281/2000, Loss: 0.2327, Avg Adaptive LR: 2.988098
INFO - Iter 1291/2000, Loss: 0.2325, Avg Adaptive LR: 2.988091
INFO - Iter 1301/2000, Loss: 0.2324, Avg Adaptive LR: 2.988084
INFO - Iter 1311/2000, Loss: 0.2323, Avg Adaptive LR: 2.988077
INFO - Iter 1321/2000, Loss: 0.2322, Avg Adaptive LR: 2.988070
INFO - Iter 1331/2000, Loss: 0.2321, Avg Adaptive LR: 2.988064
INFO - Iter 1341/2000, Loss: 0.2319, Avg Adaptive LR: 2.988057
INFO - Iter 1351/2000, Loss: 0.2318, Avg Adaptive LR: 2.988050
INFO - Iter 1361/2000, Loss: 0.2317, Avg Adaptive LR: 2.988044
INFO - Iter 1371/2000, Loss: 0.2316, Avg Adaptive LR: 2.988037
INFO - Iter 1381/2000, Loss: 0.2315, Avg Adaptive LR: 2.988031
INFO - Iter 1391/2000, Loss: 0.2314, Avg Adaptive LR: 2.988024
INFO - Iter 1401/2000, Loss: 0.2313, Avg Adaptive LR: 2.988018
INFO - Iter 1411/2000, Loss: 0.2311, Avg Adaptive LR: 2.988012
INFO - Iter 1421/2000, Loss: 0.2310, Avg Adaptive LR: 2.988006
INFO - Iter 1431/2000, Loss: 0.2309, Avg Adaptive LR: 2.988000
INFO - Iter 1441/2000, Loss: 0.2308, Avg Adaptive LR: 2.987994
INFO - Iter 1451/2000, Loss: 0.2307, Avg Adaptive LR: 2.987988
INFO - Iter 1461/2000, Loss: 0.2306, Avg Adaptive LR: 2.987982
INFO - Iter 1471/2000, Loss: 0.2305, Avg Adaptive LR: 2.987976
INFO - Iter 1481/2000, Loss: 0.2304, Avg Adaptive LR: 2.987970
INFO - Iter 1491/2000, Loss: 0.2303, Avg Adaptive LR: 2.987964
INFO - Iter 1501/2000, Loss: 0.2302, Avg Adaptive LR: 2.987958
INFO - Iter 1511/2000, Loss: 0.2301, Avg Adaptive LR: 2.987952
INFO - Iter 1521/2000, Loss: 0.2300, Avg Adaptive LR: 2.987947
INFO - Iter 1531/2000, Loss: 0.2299, Avg Adaptive LR: 2.987941
INFO - Iter 1541/2000, Loss: 0.2298, Avg Adaptive LR: 2.987936
INFO - Iter 1551/2000, Loss: 0.2297, Avg Adaptive LR: 2.987930
INFO - Iter 1561/2000, Loss: 0.2296, Avg Adaptive LR: 2.987925
INFO - Iter 1571/2000, Loss: 0.2295, Avg Adaptive LR: 2.987919
INFO - Iter 1581/2000, Loss: 0.2294, Avg Adaptive LR: 2.987914
INFO - Iter 1591/2000, Loss: 0.2293, Avg Adaptive LR: 2.987908
INFO - Iter 1601/2000, Loss: 0.2292, Avg Adaptive LR: 2.987903
INFO - Iter 1611/2000, Loss: 0.2291, Avg Adaptive LR: 2.987898
INFO - Iter 1621/2000, Loss: 0.2290, Avg Adaptive LR: 2.987893
INFO - Iter 1631/2000, Loss: 0.2290, Avg Adaptive LR: 2.987887
INFO - Iter 1641/2000, Loss: 0.2289, Avg Adaptive LR: 2.987882
INFO - Iter 1651/2000, Loss: 0.2288, Avg Adaptive LR: 2.987877
INFO - Iter 1661/2000, Loss: 0.2287, Avg Adaptive LR: 2.987872
INFO - Iter 1671/2000, Loss: 0.2286, Avg Adaptive LR: 2.987867
INFO - Iter 1681/2000, Loss: 0.2285, Avg Adaptive LR: 2.987862
INFO - Iter 1691/2000, Loss: 0.2284, Avg Adaptive LR: 2.987857
INFO - Iter 1701/2000, Loss: 0.2283, Avg Adaptive LR: 2.987852
```

```
INFO - Iter 1711/2000, Loss: 0.2283, Avg Adaptive LR: 2.987847
INFO - Iter 1721/2000, Loss: 0.2282, Avg Adaptive LR: 2.987843
INFO - Iter 1731/2000, Loss: 0.2281, Avg Adaptive LR: 2.987838
INFO - Iter 1741/2000, Loss: 0.2280, Avg Adaptive LR: 2.987833
INFO - Iter 1751/2000, Loss: 0.2279, Avg Adaptive LR: 2.987828
INFO - Iter 1761/2000, Loss: 0.2278, Avg Adaptive LR: 2.987824
INFO - Iter 1771/2000, Loss: 0.2278, Avg Adaptive LR: 2.987819
INFO - Iter 1781/2000, Loss: 0.2277, Avg Adaptive LR: 2.987814
INFO - Iter 1791/2000, Loss: 0.2276, Avg Adaptive LR: 2.987810
INFO - Iter 1801/2000, Loss: 0.2275, Avg Adaptive LR: 2.987805
INFO - Iter 1811/2000, Loss: 0.2274, Avg Adaptive LR: 2.987801
INFO - Iter 1821/2000, Loss: 0.2274, Avg Adaptive LR: 2.987796
INFO - Iter 1831/2000, Loss: 0.2273, Avg Adaptive LR: 2.987792
INFO - Iter 1841/2000, Loss: 0.2272, Avg Adaptive LR: 2.987787
INFO - Iter 1851/2000, Loss: 0.2271, Avg Adaptive LR: 2.987783
INFO - Iter 1861/2000, Loss: 0.2270, Avg Adaptive LR: 2.987779
INFO - Iter 1871/2000, Loss: 0.2270, Avg Adaptive LR: 2.987774
INFO - Iter 1881/2000, Loss: 0.2269, Avg Adaptive LR: 2.987770
INFO - Iter 1891/2000, Loss: 0.2268, Avg Adaptive LR: 2.987766
INFO - Iter 1901/2000, Loss: 0.2267, Avg Adaptive LR: 2.987762
INFO - Iter 1911/2000, Loss: 0.2267, Avg Adaptive LR: 2.987757
INFO - Iter 1921/2000, Loss: 0.2266, Avg Adaptive LR: 2.987753
INFO - Iter 1931/2000, Loss: 0.2265, Avg Adaptive LR: 2.987749
INFO - Iter 1941/2000, Loss: 0.2264, Avg Adaptive LR: 2.987745
INFO - Iter 1951/2000, Loss: 0.2264, Avg Adaptive LR: 2.987741
INFO - Iter 1961/2000, Loss: 0.2263, Avg Adaptive LR: 2.987737
INFO - Iter 1971/2000, Loss: 0.2262, Avg Adaptive LR: 2.987733
INFO - Iter 1981/2000, Loss: 0.2262, Avg Adaptive LR: 2.987729
INFO - Iter 1991/2000, Loss: 0.2261, Avg Adaptive LR: 2.987725
INFO - SoftmaxRegression training completed in 78.71 seconds.
INFO - --- Linear Regression LR=0.1/Iter=2000 ---
INFO - Iter 100/2000, Loss: 0.7434, Gradient Norm: 16.4314, Avg Adaptive LR:
1.397649924413671
INFO - Iter 200/2000, Loss: 0.3673, Gradient Norm: 11.2073, Avg Adaptive LR:
0.9918329254702632
INFO - Iter 300/2000, Loss: 0.2350, Gradient Norm: 8.6471, Avg Adaptive LR:
0.8111994843788692
INFO - Iter 400/2000, Loss: 0.1670, Gradient Norm: 6.9753, Avg Adaptive LR:
0.7032171365457824
INFO - Iter 500/2000, Loss: 0.1305, Gradient Norm: 5.8862, Avg Adaptive LR:
0.629413012128739
INFO - Iter 600/2000, Loss: 0.1078, Gradient Norm: 5.0944, Avg Adaptive LR:
0.5748706895919814
INFO - Iter 700/2000, Loss: 0.0926, Gradient Norm: 4.4842, Avg Adaptive LR:
0.5324476225488672
INFO - Iter 800/2000, Loss: 0.0823, Gradient Norm: 4.0262, Avg Adaptive LR:
0.4982238690962341
INFO - Iter 900/2000, Loss: 0.0747, Gradient Norm: 3.6480, Avg Adaptive LR:
```

```
0.46984265772905426
INFO - Iter 1000/2000, Loss: 0.0688, Gradient Norm: 3.3269, Avg Adaptive LR:
0.44583839977808687
INFO - Iter 1100/2000, Loss: 0.0646, Gradient Norm: 3.0739, Avg Adaptive LR:
0.42517222371384156
INFO - Iter 1200/2000, Loss: 0.0612, Gradient Norm: 2.8596, Avg Adaptive LR:
0.4071361723692215
INFO - Iter 1300/2000, Loss: 0.0583, Gradient Norm: 2.6612, Avg Adaptive LR:
0.3912209707987767
INFO - Iter 1400/2000, Loss: 0.0561, Gradient Norm: 2.4996, Avg Adaptive LR:
0.3770407038987045
INFO - Iter 1500/2000, Loss: 0.0543, Gradient Norm: 2.3669, Avg Adaptive LR:
0.364298727908763
INFO - Iter 1600/2000, Loss: 0.0527, Gradient Norm: 2.2349, Avg Adaptive LR:
0.35276909915206506
INFO - Iter 1700/2000, Loss: 0.0516, Gradient Norm: 2.1373, Avg Adaptive LR:
0.34226757586741535
INFO - Iter 1800/2000, Loss: 0.0504, Gradient Norm: 2.0295, Avg Adaptive LR:
0.3326542533733307
INFO - Iter 1900/2000, Loss: 0.0495, Gradient Norm: 1.9534, Avg Adaptive LR:
0.32380681690084534
INFO - Iter 2000/2000, Loss: 0.0486, Gradient Norm: 1.8655, Avg Adaptive LR:
0.31563179686776166
INFO - LinearRegressionClassifier training completed in 64.35 seconds.
Train Regressions: 100%|      | 2/2 [03:34<00:00, 107.07s/it]
INFO - Training complete for Softmax and Linear.
INFO - === TRAINING PERCEPTRON MODELS (Clean & Pocket) ===
Train Clean & Pocket:   0%|          | 0/2 [00:00<?, ?it/s]INFO - --- Clean PLA,
max_iter=100 ---
INFO - Training for digit 0…
INFO - Training for digit 1…
INFO - Training for digit 2…
INFO - Training for digit 3…
INFO - Training for digit 4…
INFO - Training for digit 5…
INFO - Training for digit 6…
INFO - Training for digit 7…
INFO - Training for digit 8…
INFO - Training for digit 9…
INFO - --- Pocket PLA, max_iter=100 ---
INFO - Training for digit 0…
INFO - Training for digit 1…
INFO - Training for digit 2…
INFO - Training for digit 3…
INFO - Training for digit 4…
INFO - Training for digit 5…
INFO - Training for digit 6…
INFO - Training for digit 7…
```

```
INFO - Training for digit 8…
INFO - Training for digit 9…
Train Clean & Pocket:  50%|        | 1/2 [01:23<01:23, 83.55s/it]INFO - ---
Clean PLA, max_iter=1000 ---
INFO - Training for digit 0…
INFO - Training for digit 1…
INFO - Training for digit 2…
INFO - Training for digit 3…
INFO - Training for digit 4…
INFO - Training for digit 5…
INFO - Training for digit 6…
INFO - Training for digit 7…
INFO - Training for digit 8…
INFO - Training for digit 9…
INFO - --- Pocket PLA, max_iter=1000 ---
INFO - Training for digit 0…
INFO - Training for digit 1…
INFO - Training for digit 2…
INFO - Training for digit 3…
INFO - Training for digit 4…
INFO - Training for digit 5…
INFO - Training for digit 6…
INFO - Training for digit 7…
INFO - Training for digit 8…
INFO - Training for digit 9…
Train Clean & Pocket: 100%|      | 2/2 [12:43<00:00, 381.65s/it]
INFO - Training complete for Clean PLA and Pocket PLA.
INFO - === ALL TRAINING COMPLETE ===
```

6 Evaluate

[7]:
```python
##################################################
# EVALUATION CELL (with pandas DataFrame)
##################################################


# 1) Evaluate Perceptrons: Clean & Pocket
accuracies_clean, accuracies_pocket = [], []
runtimes_clean,   runtimes_pocket   = [], []
sensitivities_clean, sensitivities_pocket = [], []
selectivities_clean, selectivities_pocket = [], []

conf_clean, conf_pocket = [], []
meta_clean, meta_pocket = [], []

for max_iter in tqdm(perceptron_max_iter_values, desc="Evaluate Clean &␣
 ↪Pocket"):
```

```
    # === Evaluate Clean PLA ===
    c_model = trained_models_clean[max_iter]
    cm_c, acc_c, s_c, sp_c, rt_c, ex_c = evaluate_model(
        c_model, X_test, y_test, classes=range(10), model_name="Clean PLA"
    )
    accuracies_clean.append(acc_c)
    runtimes_clean.append(rt_c)
    sensitivities_clean.append(np.mean(s_c))
    selectivities_clean.append(np.mean(sp_c))
    conf_clean.append(cm_c)

    cdict = {
        "max_iter": max_iter,
        "accuracy": acc_c,
        "runtime": rt_c,
        "avg_sensitivity": np.mean(s_c),
        "avg_selectivity": np.mean(sp_c),
        "method": "Clean PLA"
    }
    cdict.update(ex_c)
    meta_clean.append(cdict)

    # === Evaluate Pocket PLA ===
    p_model = trained_models_pocket[max_iter]
    cm_p, acc_p, s_p, sp_p, rt_p, ex_p = evaluate_model(
        p_model, X_test, y_test, classes=range(10), model_name="Pocket PLA"
    )
    accuracies_pocket.append(acc_p)
    runtimes_pocket.append(rt_p)
    sensitivities_pocket.append(np.mean(s_p))
    selectivities_pocket.append(np.mean(sp_p))
    conf_pocket.append(cm_p)

    pdict = {
        "max_iter": max_iter,
        "accuracy": acc_p,
        "runtime": rt_p,
        "avg_sensitivity": np.mean(s_p),
        "avg_selectivity": np.mean(sp_p),
        "method": "Pocket PLA"
    }
    pdict.update(ex_p)
    meta_pocket.append(pdict)

# Aggregated iteration-level training curves for Perceptrons
clean_train_curve = aggregate_iteration_losses(
    [trained_models_clean[m] for m in perceptron_max_iter_values]
```

```
)
pocket_train_curve = aggregate_iteration_losses(
    [trained_models_pocket[m] for m in perceptron_max_iter_values]
)

# 2) Evaluate Regression Models: Softmax & Linear
accuracies_softmax = []
runtimes_softmax   = []
sensitivities_soft = []
selectivities_soft = []
conf_soft          = []
meta_soft          = []

accuracies_linear = []
runtimes_linear   = []
sensitivities_lin = []
selectivities_lin = []
conf_linear       = []
meta_linear       = []

for cfg in tqdm(regression_run_configs, desc="Evaluate Regressions"):
    lr_val = cfg["learning_rate"]
    max_iter_val = cfg["max_iter"]
    label = cfg["label"]

    # === Evaluate Softmax ===
    s_model = trained_models_softmax[(lr_val, max_iter_val)]
    cm_s, a_s, se_s, sp_s, r_s, ex_s = evaluate_model(
        s_model, X_test, y_test, classes=range(10),
        model_name=f"Softmax ({label})"
    )
    accuracies_softmax.append(a_s)
    runtimes_softmax.append(r_s)
    sensitivities_soft.append(np.mean(se_s))
    selectivities_soft.append(np.mean(sp_s))
    conf_soft.append(cm_s)

    ms = {
        "label": label,
        "learning_rate": lr_val,
        "max_iter": max_iter_val,
        "accuracy": a_s,
        "runtime": r_s,
        "avg_sensitivity": np.mean(se_s),
        "avg_selectivity": np.mean(sp_s),
        "method": "Softmax"
    }
```

```
    ms.update(ex_s)
    meta_soft.append(ms)

    # === Evaluate Linear ===
    lin_model = trained_models_linear[(lr_val, max_iter_val)]
    cm_l, a_l, se_l, sp_l, r_l, ex_l = evaluate_model(
        lin_model, X_test, y_test, classes=range(10),
        model_name=f"Linear ({label})"
    )
    accuracies_linear.append(a_l)
    runtimes_linear.append(r_l)
    sensitivities_lin.append(np.mean(se_l))
    selectivities_lin.append(np.mean(sp_l))
    conf_linear.append(cm_l)

    ml = {
        "label": label,
        "learning_rate": lr_val,
        "max_iter": max_iter_val,
        "accuracy": a_l,
        "runtime": r_l,
        "avg_sensitivity": np.mean(se_l),
        "avg_selectivity": np.mean(sp_l),
        "method": "Linear Regression"
    }
    ml.update(ex_l)
    meta_linear.append(ml)


logger.info("Evaluation complete for Perceptrons & Regressions.")
```

```
Evaluate Clean & Pocket:   0%|          | 0/2 [00:00<?, ?it/s]INFO - Built-in
Confusion Matrix:
[[ 964    0    3    2    1    1    6    2    1    0]
 [   0 1107   10    6    0    2    5    2    3    0]
 [  18   12  914    9   13    1   23   20   15    7]
 [  12    1   26  910    2   20    6   18    3   12]
 [   2    1    5    0  930    0   11    3    2   28]
 [  25    6   13   44   33  703   30   18    8   12]
 [  12    3    5    2   10    6  920    0    0    0]
 [   5    8   29    5    7    0    2  951    0   21]
 [  30   14  105   81   59   44   29   32  542   38]
 [  12    7   10   17   93    5    1   60    0  804]]
INFO - Overall Accuracy: 87.45%
INFO - Built-in Confusion Matrix:
[[ 964    0    3    2    1    1    6    2    1    0]
 [   0 1107   10    6    0    2    5    2    3    0]
 [  18   12  914    9   13    1   23   20   15    7]
```

```
 [  12    1   26  910    2   20    6   18    3   12]
 [   2    1    5    0  930    0   11    3    2   28]
 [  25    6   13   44   33  703   30   18    8   12]
 [  12    3    5    2   10    6  920    0    0    0]
 [   5    8   29    5    7    0    2  951    0   21]
 [  30   14  105   81   59   44   29   32  542   38]
 [  12    7   10   17   93    5    1   60    0  804]]
INFO - Overall Accuracy: 87.45%
INFO - Class '0': TPR=0.98, TNR=0.99
INFO - Class '1': TPR=0.98, TNR=0.99
INFO - Class '2': TPR=0.89, TNR=0.98
INFO - Class '3': TPR=0.90, TNR=0.98
INFO - Class '4': TPR=0.95, TNR=0.98
INFO - Class '5': TPR=0.79, TNR=0.99
INFO - Class '6': TPR=0.96, TNR=0.99
INFO - Class '7': TPR=0.93, TNR=0.98
INFO - Class '8': TPR=0.56, TNR=1.00
INFO - Class '9': TPR=0.80, TNR=0.99
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 2015.91it/s]
INFO - Built-in Confusion Matrix:
[[ 963    0    3    3    1    0    5    2    3    0]
 [   0 1097    9    6    0    1    4    1   17    0]
 [   8    3  906   20   12    0   16   17   43    7]
 [   6    0   21  921    1   18    4   13   19    7]
 [   2    0    8    2  916    1    9    2   11   31]
 [  21    4   10   65   24  664   22   14   58   10]
 [  12    3    9    3   10    7  909    0    5    0]
 [   5    7   32    9    6    0    2  943    2   22]
 [  13    3   24   51   14   12   14   17  821    5]
 [  10    7   11   20   70   10    0   46   11  824]]
INFO - Overall Accuracy: 89.64%
INFO - Class '0': TPR=0.98, TNR=0.99
INFO - Class '1': TPR=0.97, TNR=1.00
INFO - Class '2': TPR=0.88, TNR=0.99
INFO - Class '3': TPR=0.91, TNR=0.98
INFO - Class '4': TPR=0.93, TNR=0.98
INFO - Class '5': TPR=0.74, TNR=0.99
INFO - Class '6': TPR=0.95, TNR=0.99
INFO - Class '7': TPR=0.92, TNR=0.99
INFO - Class '8': TPR=0.84, TNR=0.98
INFO - Class '9': TPR=0.82, TNR=0.99
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 2158.12it/s]
INFO - Built-in Confusion Matrix:
[[ 949    0    7    3    0    5   12    3    1    0]
 [   0 1100   24    2    0    2    4    2    1    0]
 [   9    2  968   10   10    2   13    8    7    3]
 [   4    2   46  915    1   24    3   12    3    0]
 [   2    3   14    4  934    0    8    6    3    8]
```

```
 [  15    4   22   37   12  773   13    6    9    1]
 [  12    3   23    1    4   19  894    2    0    0]
 [   5    5   47    6    9    2    1  949    0    4]
 [  43   29  148   50   38   74    7   34  551    0]
 [  24   13   18   21  132   20    0  181    0  600]]
INFO - Overall Accuracy: 86.33%
INFO - Class '0': TPR=0.97, TNR=0.99
INFO - Class '1': TPR=0.97, TNR=0.99
INFO - Class '2': TPR=0.94, TNR=0.96
INFO - Class '3': TPR=0.91, TNR=0.99
INFO - Class '4': TPR=0.95, TNR=0.98
INFO - Class '5': TPR=0.87, TNR=0.98
INFO - Class '6': TPR=0.93, TNR=0.99
INFO - Class '7': TPR=0.92, TNR=0.97
INFO - Class '8': TPR=0.57, TNR=1.00
INFO - Class '9': TPR=0.59, TNR=1.00
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 3411.67it/s]
INFO - Built-in Confusion Matrix:
[[ 961    0    0    2    0    4    7    3    3    0]
 [   0 1110    3    2    0    3    5    2   10    0]
 [  12    5  926   18    7    6   17   14   22    5]
 [   6    2   21  914    2   29    6   11   12    7]
 [   3    2    7    4  908    2    9    6    5   36]
 [  14    3    5   33    9  776   19    4   23    6]
 [  13    3    6    1    9   18  906    1    1    0]
 [   5    8   24    8    7    3    1  943    1   28]
 [  22   21   14   35   30   49   17   26  754    6]
 [  19   10    1   14   48   14    0   54    2  847]]
INFO - Overall Accuracy: 90.45%
INFO - Class '0': TPR=0.98, TNR=0.99
INFO - Class '1': TPR=0.98, TNR=0.99
INFO - Class '2': TPR=0.90, TNR=0.99
INFO - Class '3': TPR=0.90, TNR=0.99
INFO - Class '4': TPR=0.92, TNR=0.99
INFO - Class '5': TPR=0.87, TNR=0.99
INFO - Class '6': TPR=0.95, TNR=0.99
INFO - Class '7': TPR=0.92, TNR=0.99
INFO - Class '8': TPR=0.77, TNR=0.99
INFO - Class '9': TPR=0.84, TNR=0.99
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 2259.38it/s]
Evaluate Clean & Pocket: 100%|     | 2/2 [00:00<00:00, 44.68it/s]
Aggregating train losses across Perceptron models: 100%|     | 2/2
[00:00<00:00, 1957.21it/s]
Aggregating train losses across Perceptron models: 100%|     | 2/2
[00:00<00:00, 1833.98it/s]
Evaluate Regressions:   0%|          | 0/2 [00:00<?, ?it/s]INFO - Built-in
Confusion Matrix:
[[ 961    0    0    2    0    5    8    2    2    0]
```

```
 [   0 1113    4    2    0    1    4    2    9    0]
 [   7    9  921   20    9    3   11   11   38    3]
 [   4    0   17  926    1   24    2    9   21    6]
 [   1    1    3    2  918    0   12    4    9   32]
 [   8    4    1   35    9  772   15   10   33    5]
 [  11    3    7    1    7   14  912    2    1    0]
 [   1    7   23    5    4    0    0  954    4   30]
 [   7    9    5   20    9   21    8   13  876    6]
 [  10    7    1   11   25    6    0   22    7  920]]
INFO - Overall Accuracy: 92.73%
INFO - Class '0': TPR=0.98, TNR=0.99
INFO - Class '1': TPR=0.98, TNR=1.00
INFO - Class '2': TPR=0.89, TNR=0.99
INFO - Class '3': TPR=0.92, TNR=0.99
INFO - Class '4': TPR=0.93, TNR=0.99
INFO - Class '5': TPR=0.87, TNR=0.99
INFO - Class '6': TPR=0.95, TNR=0.99
INFO - Class '7': TPR=0.93, TNR=0.99
INFO - Class '8': TPR=0.90, TNR=0.99
INFO - Class '9': TPR=0.91, TNR=0.99
Evaluating class metrics: 100%|      | 10/10 [00:00<00:00, 3761.37it/s]
INFO - Built-in Confusion Matrix:
[[ 955    0    8    1    0    3    4    1    8    0]
 [   0 1102    7    0    1    0    3    0   22    0]
 [  13   49  896   11   13    0    6    6   38    0]
 [   8   27   57  849    2    7    3   12   43    2]
 [   1   33   15    0  890    2    3    1   22   15]
 [  36   20   26   95   26  543   12   10  119    5]
 [  47   22   42    0   22   13  789    0   23    0]
 [  13   61   51    9   35    0    0  815   14   30]
 [  15   54   21   15   13    8    7    1  840    0]
 [  25   30   20   14  123    3    0   44   54  696]]
INFO - Overall Accuracy: 83.75%
INFO - Class '0': TPR=0.97, TNR=0.98
INFO - Class '1': TPR=0.97, TNR=0.97
INFO - Class '2': TPR=0.87, TNR=0.97
INFO - Class '3': TPR=0.84, TNR=0.98
INFO - Class '4': TPR=0.91, TNR=0.97
INFO - Class '5': TPR=0.61, TNR=1.00
INFO - Class '6': TPR=0.82, TNR=1.00
INFO - Class '7': TPR=0.79, TNR=0.99
INFO - Class '8': TPR=0.86, TNR=0.96
INFO - Class '9': TPR=0.69, TNR=0.99
Evaluating class metrics: 100%|      | 10/10 [00:00<00:00, 3802.29it/s]
INFO - Built-in Confusion Matrix:
[[ 959    0    1    2    0    7    7    2    2    0]
 [   0 1112    6    3    0    1    3    2    8    0]
 [   6    9  926   17    8    4   11    9   39    3]
```

```
 [   3    1   16  924    1   23    3    9   24    6]
 [   1    1    4    2  917    0   12    5   10   30]
 [   7    4    1   34    8  775   14   11   32    6]
 [  12    3    7    1    7   15  910    1    2    0]
 [   1    7   24    5    5    1    0  951    4   30]
 [   6    7    5   21    8   22    9   13  876    7]
 [  10    9    1    8   24    7    0   24    6  920]]
INFO - Overall Accuracy: 92.70%
INFO - Class '0': TPR=0.98, TNR=0.99
INFO - Class '1': TPR=0.98, TNR=1.00
INFO - Class '2': TPR=0.90, TNR=0.99
INFO - Class '3': TPR=0.91, TNR=0.99
INFO - Class '4': TPR=0.93, TNR=0.99
INFO - Class '5': TPR=0.87, TNR=0.99
INFO - Class '6': TPR=0.95, TNR=0.99
INFO - Class '7': TPR=0.93, TNR=0.99
INFO - Class '8': TPR=0.90, TNR=0.99
INFO - Class '9': TPR=0.91, TNR=0.99
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 3941.64it/s]
INFO - Built-in Confusion Matrix:
[[ 963    0    0    0    0    1    3    3   10    0]
 [   0 1044    0    1    1    0    5    6   78    0]
 [  95   25  615   11    9    0   30   97  150    0]
 [  58    3    1  743    0    0    5  100   99    1]
 [  27   15    1    1  780    0   10   47   76   25]
 [ 142    6    0   55    1  342   19   82  240    5]
 [  99    6    1    0    6    3  814    3   26    0]
 [   7   19    2    5    4    0    2  968   15    6]
 [  34   11    2    4    5    1    6   34  877    0]
 [  49    5    1    5   19    0    0  352   87  491]]
INFO - Overall Accuracy: 76.37%
INFO - Class '0': TPR=0.98, TNR=0.94
INFO - Class '1': TPR=0.92, TNR=0.99
INFO - Class '2': TPR=0.60, TNR=1.00
INFO - Class '3': TPR=0.74, TNR=0.99
INFO - Class '4': TPR=0.79, TNR=1.00
INFO - Class '5': TPR=0.38, TNR=1.00
INFO - Class '6': TPR=0.85, TNR=0.99
INFO - Class '7': TPR=0.94, TNR=0.92
INFO - Class '8': TPR=0.90, TNR=0.91
INFO - Class '9': TPR=0.49, TNR=1.00
Evaluating class metrics: 100%|     | 10/10 [00:00<00:00, 3552.69it/s]
Evaluate Regressions: 100%|     | 2/2 [00:00<00:00, 55.61it/s]
INFO - Evaluation complete for Perceptrons & Regressions.
```

7 Visualize (Generate Plots, Confusion Matricies, etc.)

[8]:
```
#####################################################
# 1) CREATE A SINGLE PANDAS DATAFRAME FOR ALL RESULTS
#####################################################
all_rows = []

# (A) Clean PLA
for i, max_iter in tqdm(
    enumerate(perceptron_max_iter_values),
    desc="Collecting Clean PLA",
    total=len(perceptron_max_iter_values)
):
    all_rows.append({
        'model': 'Clean PLA',
        'max_iter': max_iter,
        'runtime': runtimes_clean[i],
        'accuracy': accuracies_clean[i],
        'sensitivity': sensitivities_clean[i],
        'selectivity': selectivities_clean[i]
    })

# (B) Pocket PLA
for i, max_iter in tqdm(
    enumerate(perceptron_max_iter_values),
    desc="Collecting Pocket PLA",
    total=len(perceptron_max_iter_values)
):
    all_rows.append({
        'model': 'Pocket PLA',
        'max_iter': max_iter,
        'runtime': runtimes_pocket[i],
        'accuracy': accuracies_pocket[i],
        'sensitivity': sensitivities_pocket[i],
        'selectivity': selectivities_pocket[i]
    })

# (C) Softmax
for i, row_meta in tqdm(
    enumerate(meta_soft),
    desc="Collecting Softmax",
    total=len(meta_soft)
):
    all_rows.append({
        'model': 'Softmax',
        'max_iter': row_meta['max_iter'],
        'runtime': runtimes_softmax[i],
```

```
        'accuracy': accuracies_softmax[i],
        'sensitivity': sensitivities_soft[i],
        'selectivity': selectivities_soft[i]
    })

# (D) Linear
for i, row_meta in tqdm(
    enumerate(meta_linear),
    desc="Collecting Linear",
    total=len(meta_linear)
):
    all_rows.append({
        'model': 'Linear',
        'max_iter': row_meta['max_iter'],
        'runtime': runtimes_linear[i],
        'accuracy': accuracies_linear[i],
        'sensitivity': sensitivities_lin[i],
        'selectivity': selectivities_lin[i]
    })

df_results = pd.DataFrame(all_rows)
logger.info("Combined Results DataFrame:\n%s", df_results)
display(df_results.head(20))

################################################################################
# 2) CONFUSION MATRICES FOR ALL MODELS (GROUPED BY PLOT TYPE)
################################################################################

logger.info("=== Plotting ALL Confusion Matrices ===")

# 2A) Perceptron: Clean
for idx, meta in tqdm(enumerate(meta_clean), total=len(meta_clean),␣
 ↪desc="Confusions: Clean PLA"):
    title = f"Clean PLA (max_iter={meta['max_iter']}, Acc={meta['accuracy']*100:
 ↪.2f}%)"
    plot_confusion_matrix_annotated(
        conf_clean[idx],
        classes=range(10),
        title=title,
        method=meta["method"],
        max_iter=meta["max_iter"]
    )

# 2B) Perceptron: Pocket
for idx, meta in tqdm(enumerate(meta_pocket), total=len(meta_pocket),␣
 ↪desc="Confusions: Pocket PLA"):
```

```
    title = f"Pocket PLA (max_iter={meta['max_iter']},␣
 ↪Acc={meta['accuracy']*100:.2f}%)"
    plot_confusion_matrix_annotated(
        conf_pocket[idx],
        classes=range(10),
        title=title,
        method=meta["method"],
        max_iter=meta["max_iter"]
    )

# 2C) Softmax
for idx, meta in tqdm(enumerate(meta_soft), total=len(meta_soft),␣
 ↪desc="Confusions: Softmax"):
    title = f"Softmax ({meta['label']}, Acc={meta['accuracy']*100:.2f}%)"
    plot_confusion_matrix_annotated(
        conf_soft[idx],
        classes=range(10),
        title=title,
        method=meta["method"],
        max_iter=meta["max_iter"]
    )

# 2D) Linear
for idx, meta in tqdm(enumerate(meta_linear), total=len(meta_linear),␣
 ↪desc="Confusions: Linear"):
    title = f"Linear ({meta['label']}, Acc={meta['accuracy']*100:.2f}%)"
    plot_confusion_matrix_annotated(
        conf_linear[idx],
        classes=range(10),
        title=title,
        method=meta["method"],
        max_iter=meta["max_iter"]
    )


##############################################################################
# 3) ITERATION-LEVEL PLOTS (ALL MODELS)
##############################################################################

logger.info("=== Iteration-Level Visualization (All Models) ===")

# 3A) Perceptron: Clean & Pocket
for max_iter, c_model in trained_models_clean.items():
    df_iter = c_model.get_iteration_df()
    if not df_iter.empty and "train_error" in df_iter.columns:
        title = f"Clean PLA max_iter={max_iter}: Train Error vs. Iteration"
```

```
        df_iter.plot(x="iteration", y="train_error", marker='o', figsize=(8,5),␣
 ↪title=title)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.show()

for max_iter, p_model in trained_models_pocket.items():
    df_iter = p_model.get_iteration_df()
    if not df_iter.empty and "train_error" in df_iter.columns:
        title = f"Pocket PLA max_iter={max_iter}: Train Error vs. Iteration"
        df_iter.plot(x="iteration", y="train_error", marker='o', figsize=(8,5),␣
 ↪title=title)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.show()

# 3B) Softmax
for (lr_val, max_iter_val), s_model in trained_models_softmax.items():
    df_iter = s_model.get_iteration_df()  # Must be implemented in your␣
 ↪SoftmaxRegression
    if not df_iter.empty:
        title = f"Softmax LR={lr_val}, max_iter={max_iter_val}: Train Loss vs.␣
 ↪Iteration"
        df_iter.plot(x="iteration", y="train_loss", marker='o', figsize=(8,5),␣
 ↪title=title)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.show()

        if "test_loss" in df_iter.columns:
            title = f"Softmax LR={lr_val}, max_iter={max_iter_val}: Train &␣
 ↪Test Loss"
            df_iter.plot(x="iteration", y=["train_loss","test_loss"],␣
 ↪marker='o', figsize=(8,5), title=title)
            plt.grid(True, linestyle='--', alpha=0.7)
            plt.show()

        if "avg_adaptive_lr" in df_iter.columns:
            title = f"Softmax LR={lr_val}, max_iter={max_iter_val}: Avg␣
 ↪Adaptive LR vs. Iteration"
            df_iter.plot(x="iteration", y="avg_adaptive_lr", marker='x',␣
 ↪figsize=(8,5), title=title)
            plt.grid(True, linestyle='--', alpha=0.7)
            plt.show()

# 3C) Linear
for (lr_val, max_iter_val), lin_model in trained_models_linear.items():
    df_iter = lin_model.get_iteration_df()  # Must be implemented in your␣
 ↪LinearRegression
```

```
    if not df_iter.empty:
        title = f"Linear LR={lr_val}, max_iter={max_iter_val}: Train Loss vs.␣
↪Iteration"
        df_iter.plot(x="iteration", y="train_loss", marker='o', figsize=(8,5),␣
↪title=title)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.show()

        if "test_loss" in df_iter.columns:
            title = f"Linear LR={lr_val}, max_iter={max_iter_val}: Train & Test␣
↪Loss"
            df_iter.plot(x="iteration", y=["train_loss","test_loss"],␣
↪marker='o', figsize=(8,5), title=title)
            plt.grid(True, linestyle='--', alpha=0.7)
            plt.show()

        if "avg_adaptive_lr" in df_iter.columns:
            title = f"Linear LR={lr_val}, max_iter={max_iter_val}: Avg Adaptive␣
↪LR vs. Iteration"
            df_iter.plot(x="iteration", y="avg_adaptive_lr", marker='x',␣
↪figsize=(8,5), title=title)
            plt.grid(True, linestyle='--', alpha=0.7)
            plt.show()


##############################################################################
# 4) PANDAS + SEABORN PLOTS
##############################################################################

logger.info("=== Pandas + Seaborn Plots ===")

# 4A) LINE PLOT: Accuracy vs. max_iter (Perceptrons Only)
df_perc = df_results[df_results['model'].isin(['Clean PLA','Pocket PLA'])].
↪copy()
df_perc.sort_values(['model','max_iter'], inplace=True)

plt.figure(figsize=(6,4))
sns.lineplot(
    data=df_perc,
    x='max_iter', y='accuracy',
    hue='model', marker='o'
)
plt.title("Perceptrons: Accuracy vs. max_iter (Pandas/Seaborn)")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
# 4B) BAR CHART: Average Accuracy by Model
df_mean = df_results.groupby('model', as_index=False)['accuracy'].mean()

plt.figure(figsize=(6,4))
sns.barplot(data=df_mean, x='model', y='accuracy')
plt.title("Average Accuracy by Model (Pandas/Seaborn)")
plt.ylim(0.7, 1.0)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.show()

# 4C) SCATTER PLOT: Accuracy vs. Runtime, colored by model
plt.figure(figsize=(6,4))
sns.scatterplot(
    data=df_results,
    x='runtime', y='accuracy',
    hue='model', style='model',
    s=100
)
plt.title("Accuracy vs. Runtime (All Models) (Pandas/Seaborn)")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()


##############################################################################
# 5) CUSTOM SUMMARY PLOTS (AGGREGATED CURVES, ETC.)
##############################################################################

logger.info("=== Custom Summaries (Aggregated Curves, etc.) ===")

# 5A) Aggregated Perceptron Curves
plot_train_curves_three_models(
    clean_train_curve=clean_train_curve,
    pocket_train_curve=pocket_train_curve,
    softmax_train_curve=None,  # no Softmax aggregator
    title="Aggregated Perceptron Train Curves (Clean vs. Pocket)",
    max_iter=perceptron_max_iter_values[-1]
)

# 5B) Summaries for Perceptron
plot_accuracy_vs_max_iter(
    max_iter_values=perceptron_max_iter_values,
    accuracies_clean=accuracies_clean,
    accuracies_pocket=accuracies_pocket,
    accuracies_softmax=None
)

plot_runtime_vs_max_iter(
```

```
    max_iter_values=perceptron_max_iter_values,
    runtimes_clean=runtimes_clean,
    runtimes_pocket=runtimes_pocket,
    runtimes_softmax=None
)

plot_accuracy_vs_runtime(
    runtimes_clean=runtimes_clean,
    accuracies_clean=accuracies_clean,
    runtimes_pocket=runtimes_pocket,
    accuracies_pocket=accuracies_pocket,
    title="Perceptrons: Accuracy vs. Runtime"
)

plot_performance_summary_extended_by_runtime(
    runtimes_clean=runtimes_clean,
    accuracies_clean=accuracies_clean,
    sensitivities_clean=sensitivities_clean,
    selectivities_clean=selectivities_clean,
    runtimes_pocket=runtimes_pocket,
    accuracies_pocket=accuracies_pocket,
    sensitivities_pocket=sensitivities_pocket,
    selectivities_pocket=selectivities_pocket,
    title="Perceptrons: Performance vs. Runtime"
)

# 5C) Summaries for Softmax & Linear
plot_accuracy_vs_runtime(
    runtimes_clean=runtimes_softmax,
    accuracies_clean=accuracies_softmax,
    title="Softmax: Accuracy vs. Runtime"
)
plot_accuracy_vs_runtime(
    runtimes_clean=runtimes_linear,
    accuracies_clean=accuracies_linear,
    title="Linear: Accuracy vs. Runtime"
)
plot_accuracy_vs_runtime(
    runtimes_clean=runtimes_softmax,
    accuracies_clean=accuracies_softmax,
    runtimes_pocket=runtimes_linear,
    accuracies_pocket=accuracies_linear,
    title="Softmax vs. Linear: Accuracy vs. Runtime"
)
plot_performance_summary_extended_by_runtime(
    runtimes_clean=runtimes_softmax,
    accuracies_clean=accuracies_softmax,
```

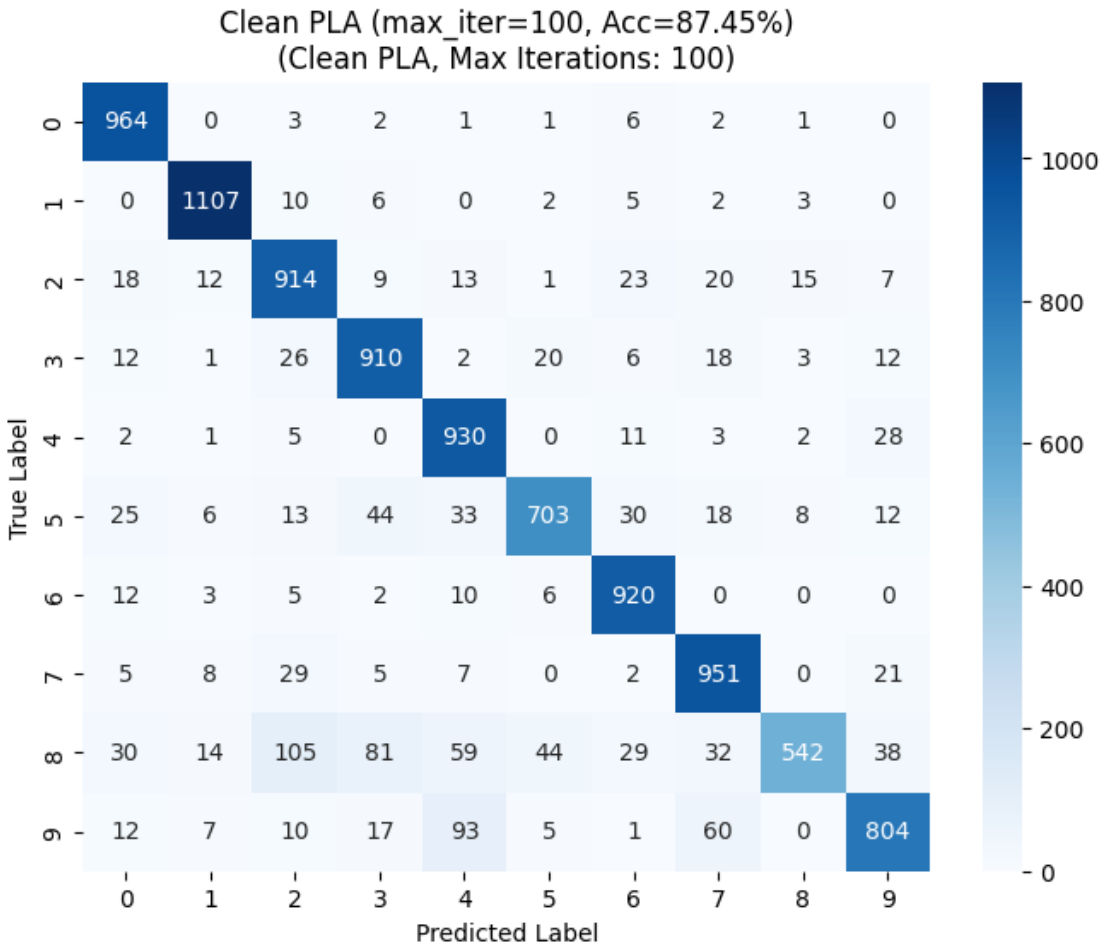
```
    sensitivities_clean=sensitivities_soft,
    selectivities_clean=selectivities_soft,
    runtimes_pocket=runtimes_linear,
    accuracies_pocket=accuracies_linear,
    sensitivities_pocket=sensitivities_lin,
    selectivities_pocket=selectivities_lin,
    title="Softmax vs. Linear: TPR/TNR vs. Runtime"
)

# 5D) 4-Model Comparison
plot_performance_summary_4models_by_runtime(
    runtimes_clean, accuracies_clean, sensitivities_clean, selectivities_clean,
    runtimes_pocket, accuracies_pocket, sensitivities_pocket,␣
 ↪selectivities_pocket,
    runtimes_softmax, accuracies_softmax, sensitivities_soft,␣
 ↪selectivities_soft,
    runtimes_linear, accuracies_linear, sensitivities_lin, selectivities_lin,
    title="Performance vs. Runtime (4-Model Comparison)"
)

plot_accuracy_vs_runtime_4models(
    rt_clean=runtimes_clean,
    acc_clean=accuracies_clean,
    rt_pocket=runtimes_pocket,
    acc_pocket=accuracies_pocket,
    rt_softmax=runtimes_softmax,
    acc_softmax=accuracies_softmax,
    rt_linear=runtimes_linear,
    acc_linear=accuracies_linear,
    title="Accuracy vs. Runtime (4 Models)"
)

logger.info("=== All Visualizations Complete ===")
```

```
Collecting Clean PLA: 100%|      | 2/2 [00:00<00:00, 57456.22it/s]
Collecting Pocket PLA: 100%|      | 2/2 [00:00<00:00, 53773.13it/s]
Collecting Softmax: 100%|     | 2/2 [00:00<00:00, 16070.13it/s]
Collecting Linear: 100%|     | 2/2 [00:00<00:00, 74235.47it/s]
INFO - Combined Results DataFrame:
        model  max_iter     runtime  accuracy  sensitivity  selectivity
0   Clean PLA       100   41.638159    0.8745     0.871954     0.986055
1   Clean PLA      1000  340.500284    0.8633     0.861587     0.984807
2  Pocket PLA       100   41.915633    0.8964     0.894188     0.988493
3  Pocket PLA      1000  339.249425    0.9045     0.903203     0.989398
4     Softmax      1000   39.220767    0.9273     0.926198     0.991928
5     Softmax      2000   78.707570    0.9270     0.925926     0.991895
6      Linear      1000   31.863395    0.8375     0.833789     0.981906
```

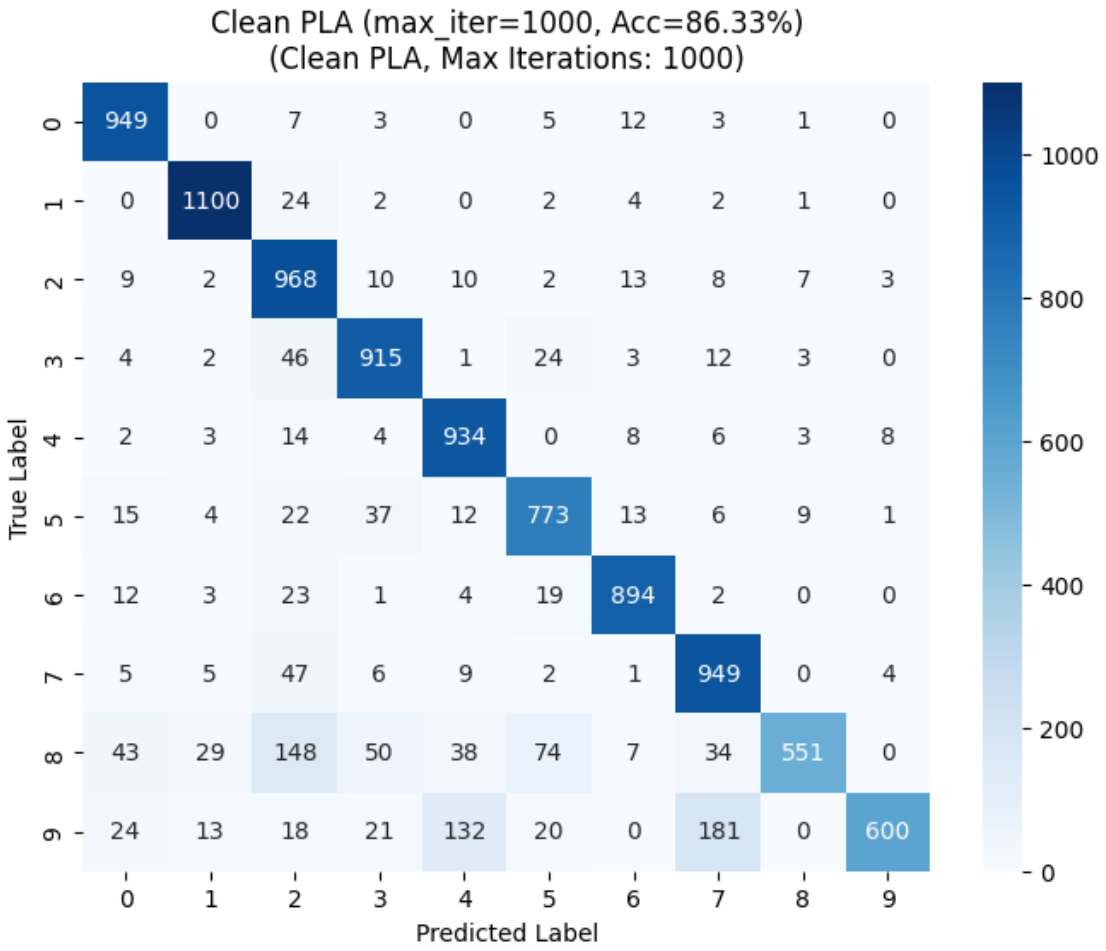
```
7      Linear       2000   64.348078    0.7637     0.759011     0.973746
```

	model	max_iter	runtime	accuracy	sensitivity	selectivity
0	Clean PLA	100	41.638159	0.8745	0.871954	0.986055
1	Clean PLA	1000	340.500284	0.8633	0.861587	0.984807
2	Pocket PLA	100	41.915633	0.8964	0.894188	0.988493
3	Pocket PLA	1000	339.249425	0.9045	0.903203	0.989398
4	Softmax	1000	39.220767	0.9273	0.926198	0.991928
5	Softmax	2000	78.707570	0.9270	0.925926	0.991895
6	Linear	1000	31.863395	0.8375	0.833789	0.981906
7	Linear	2000	64.348078	0.7637	0.759011	0.973746

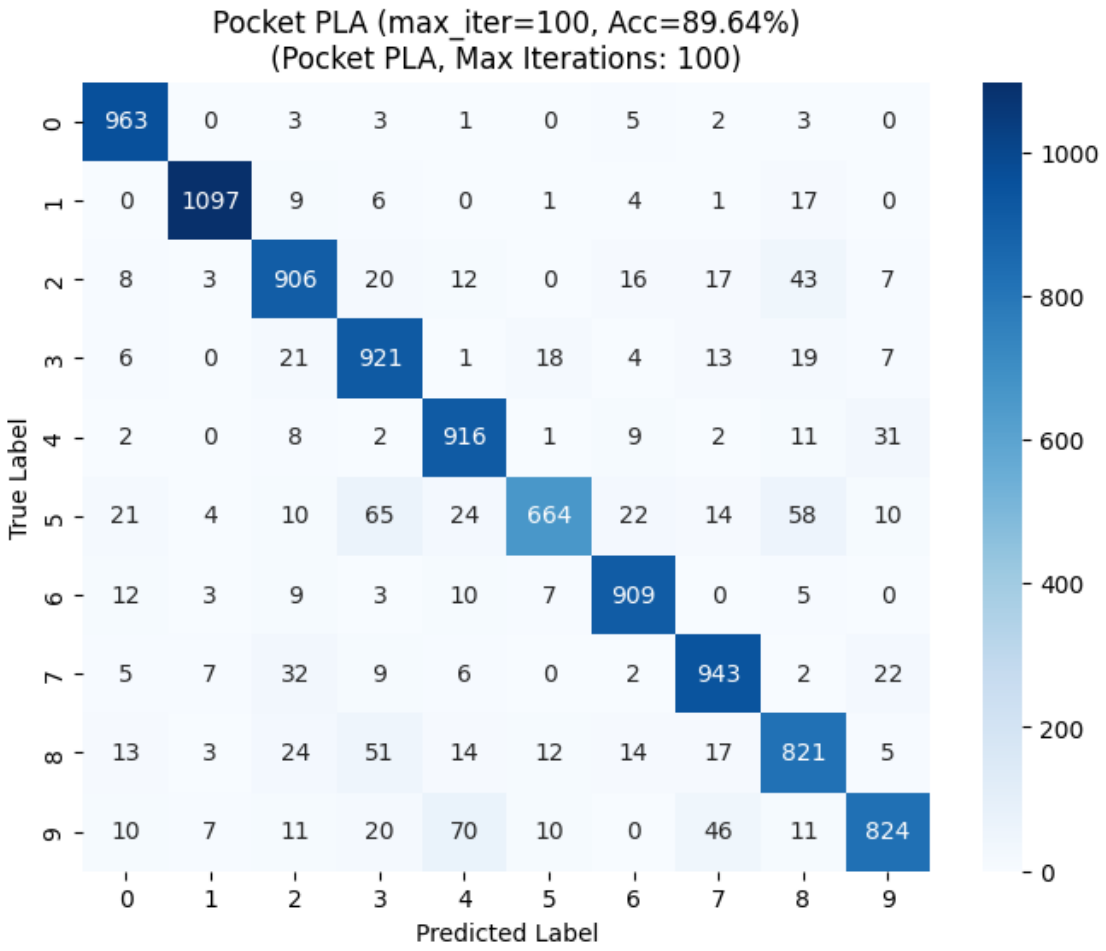
```
INFO - === Plotting ALL Confusion Matrices ===
Confusions: Clean PLA:   0%|          | 0/2 [00:00<?, ?it/s]
```



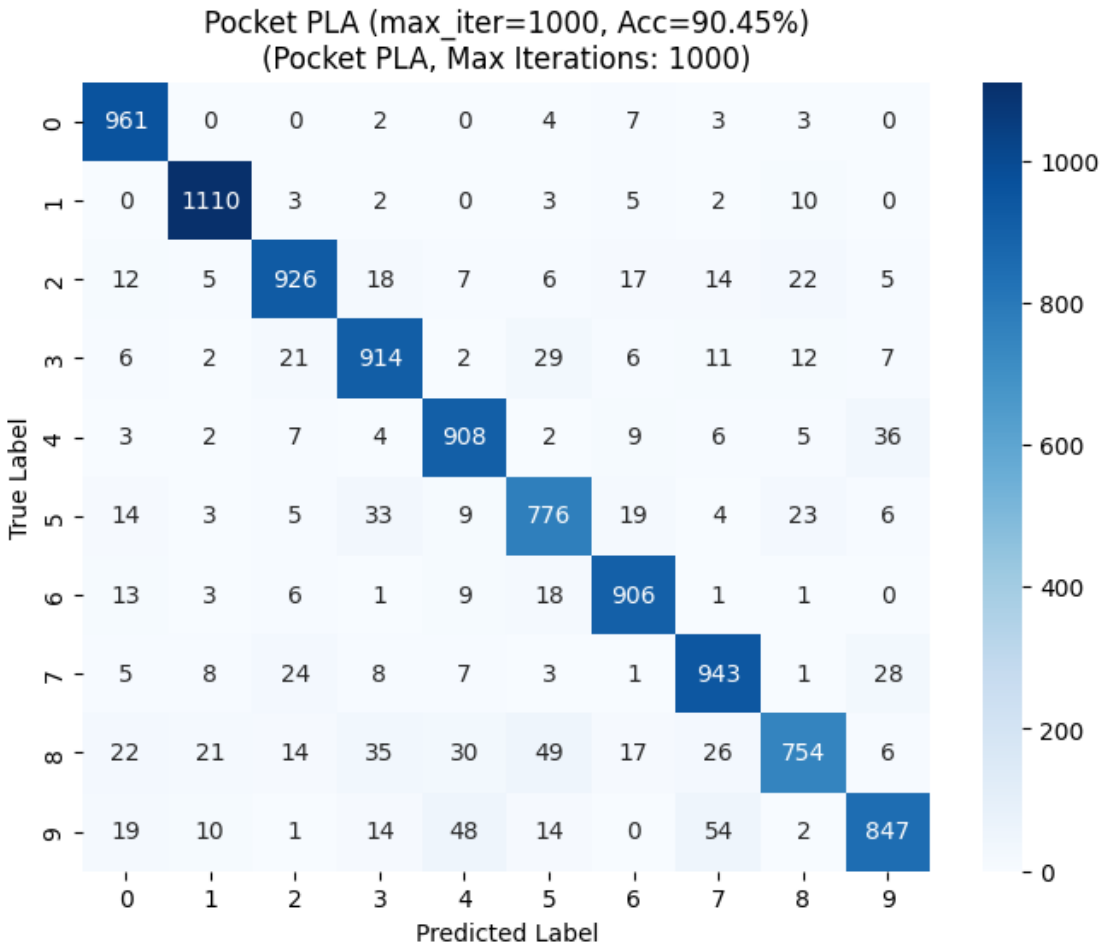

```
Confusions: Clean PLA:  50%|          | 1/2 [00:00<00:00,  7.62it/s]
```



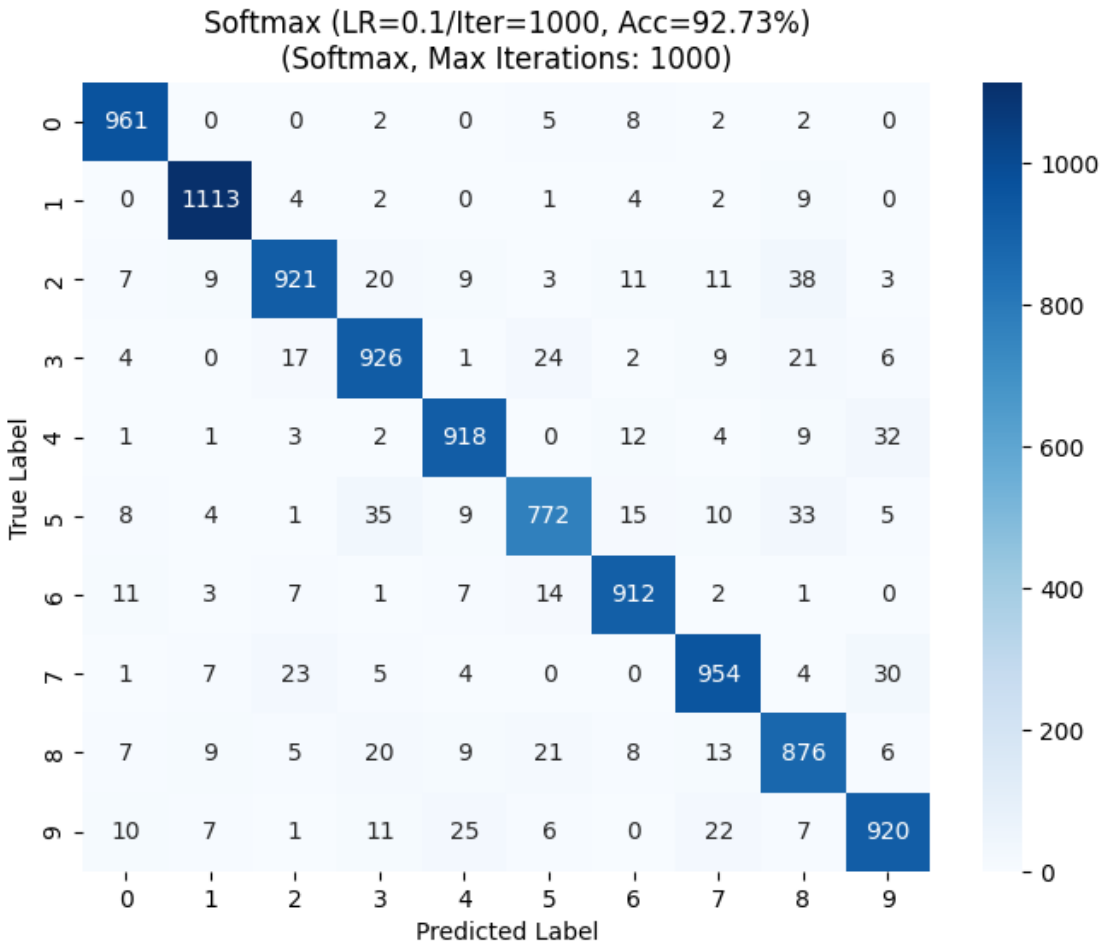

```
Confusions: Clean PLA: 100%|      | 2/2 [00:00<00:00,  8.16it/s]
Confusions: Pocket PLA:   0%|          | 0/2 [00:00<?, ?it/s]
```



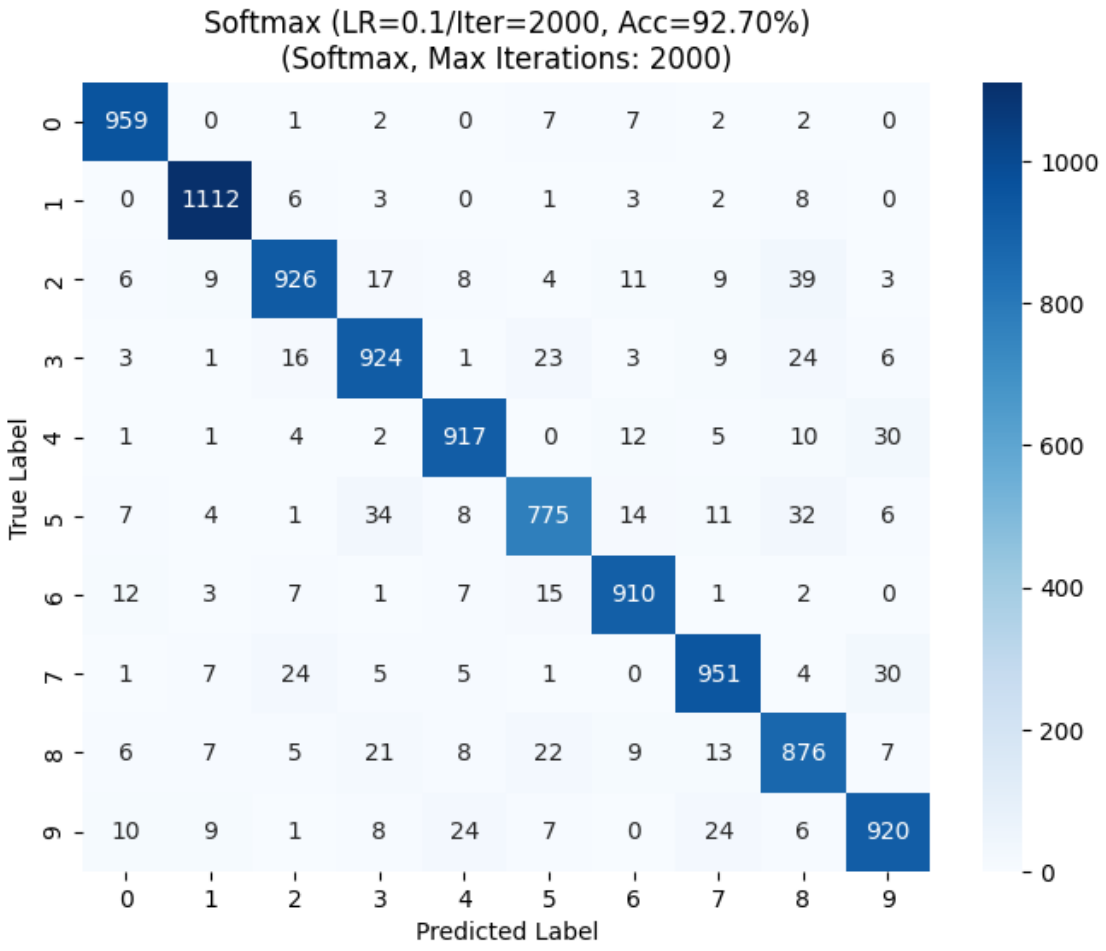

```
Confusions: Pocket PLA:  50%|          | 1/2 [00:00<00:00,  9.26it/s]
```



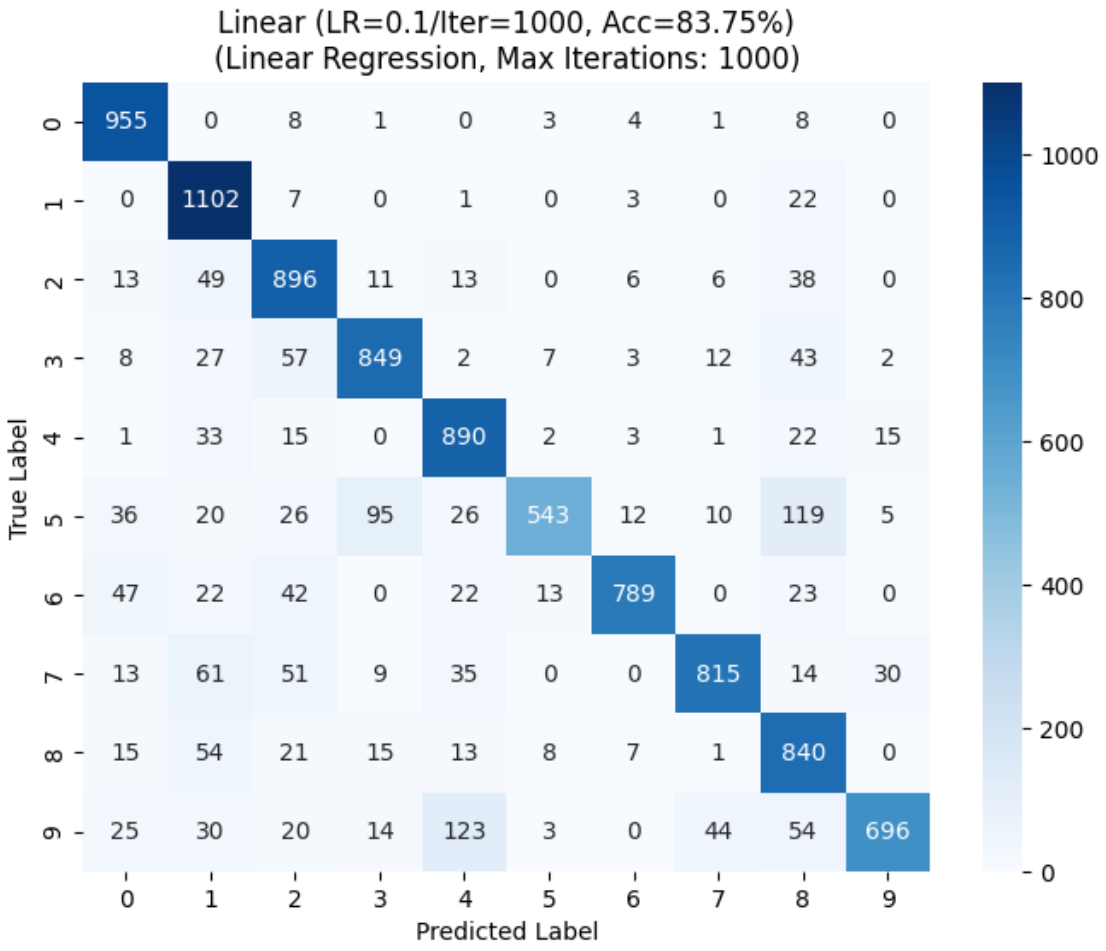

```
Confusions: Pocket PLA: 100%|      | 2/2 [00:00<00:00,  9.19it/s]
Confusions: Softmax:   0%|          | 0/2 [00:00<?, ?it/s]
```



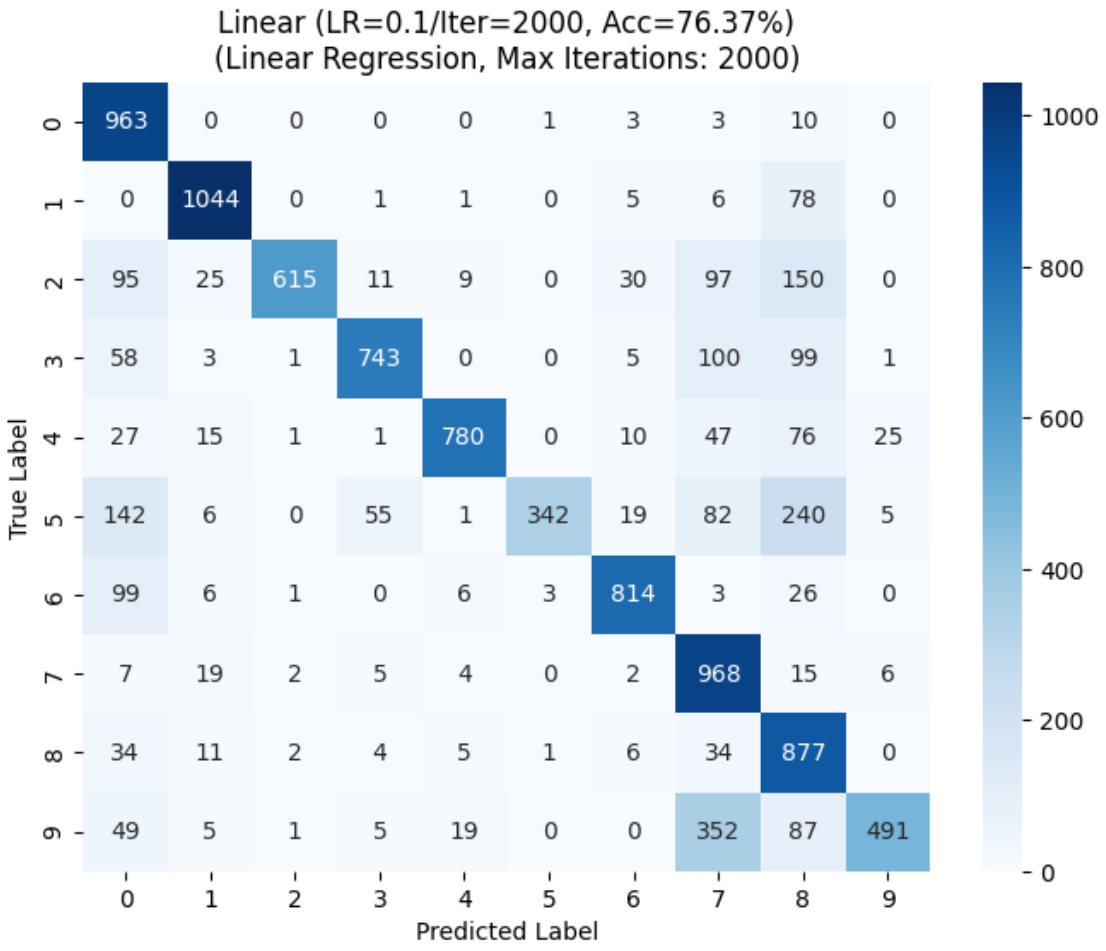

```
Confusions: Softmax:  50%|          | 1/2 [00:00<00:00,  9.26it/s]
```



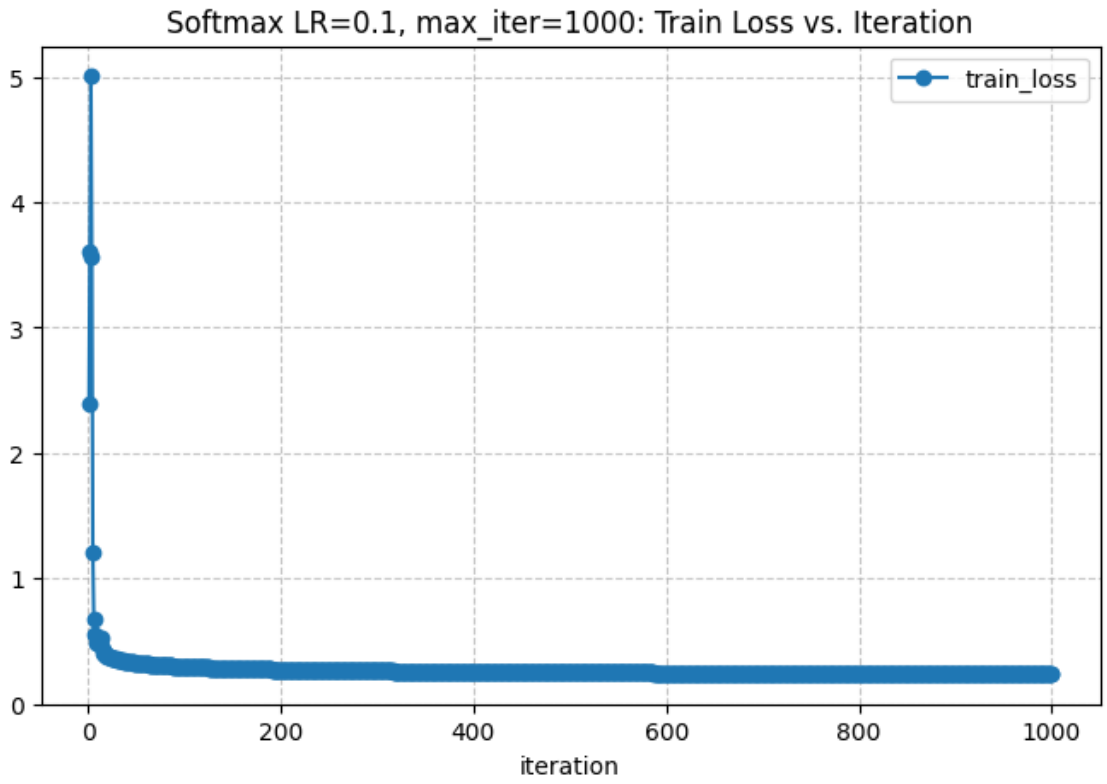

```
Confusions: Softmax: 100%|      | 2/2 [00:00<00:00,  9.33it/s]
Confusions: Linear:   0%|          | 0/2 [00:00<?, ?it/s]
```

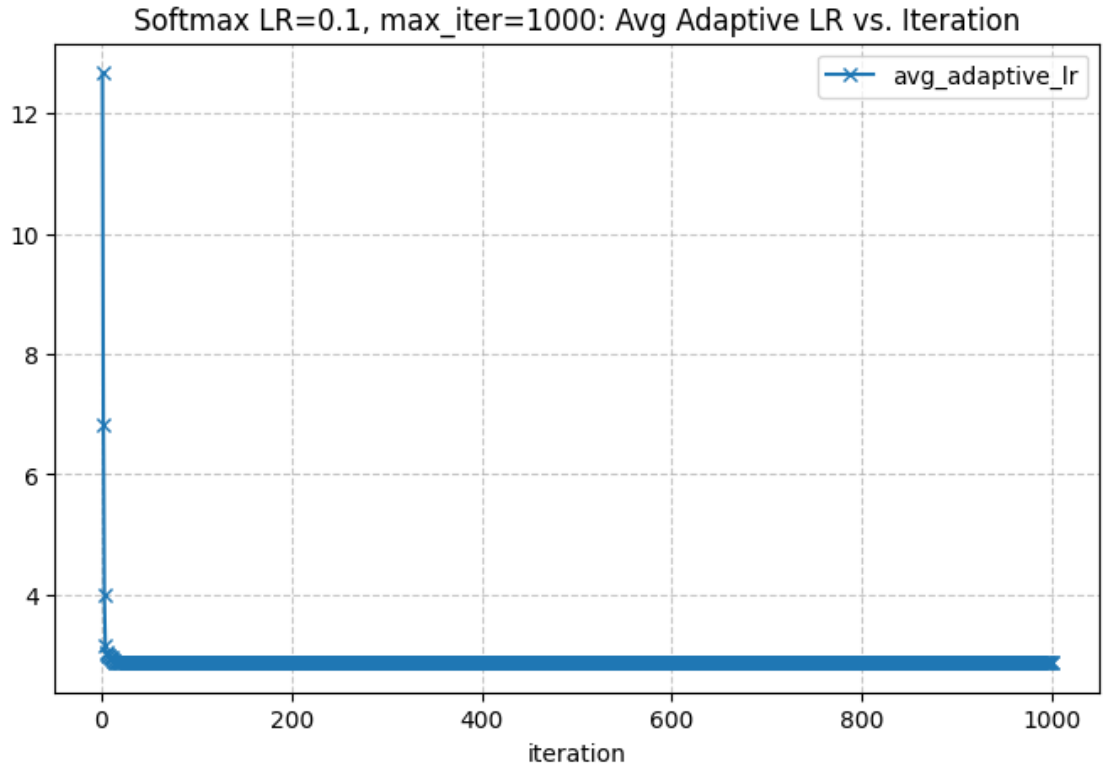


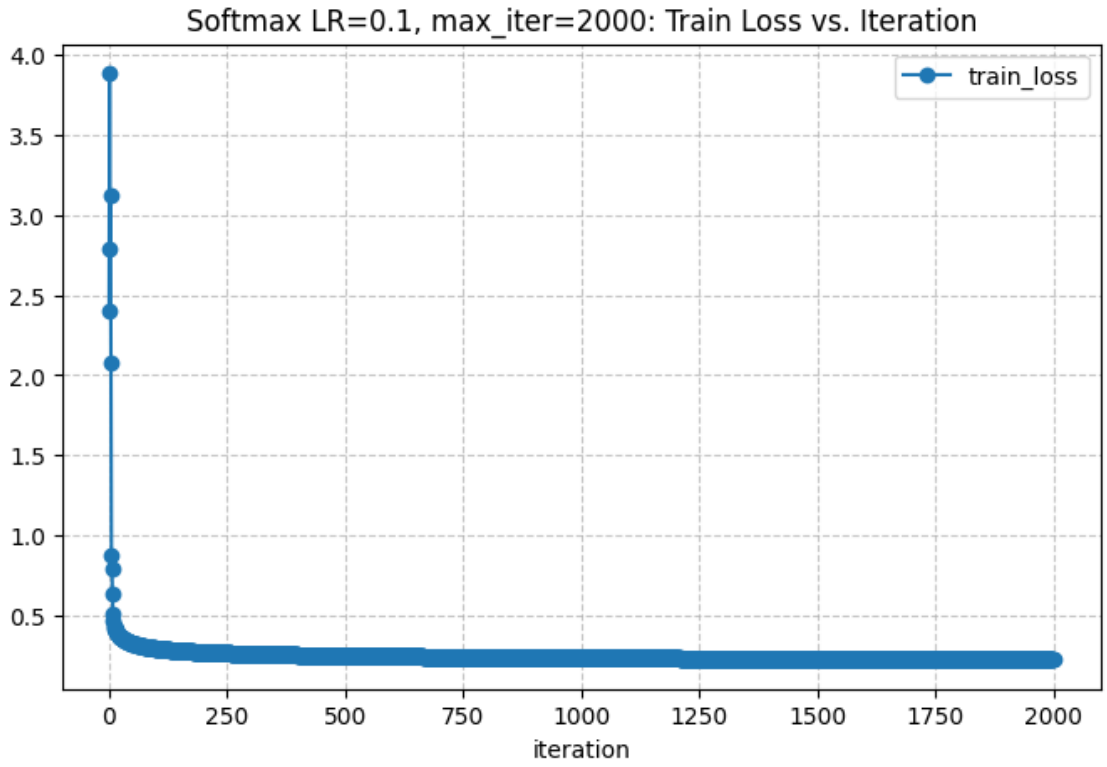

```
Confusions: Linear:  50%|          | 1/2 [00:00<00:00,  9.21it/s]
```

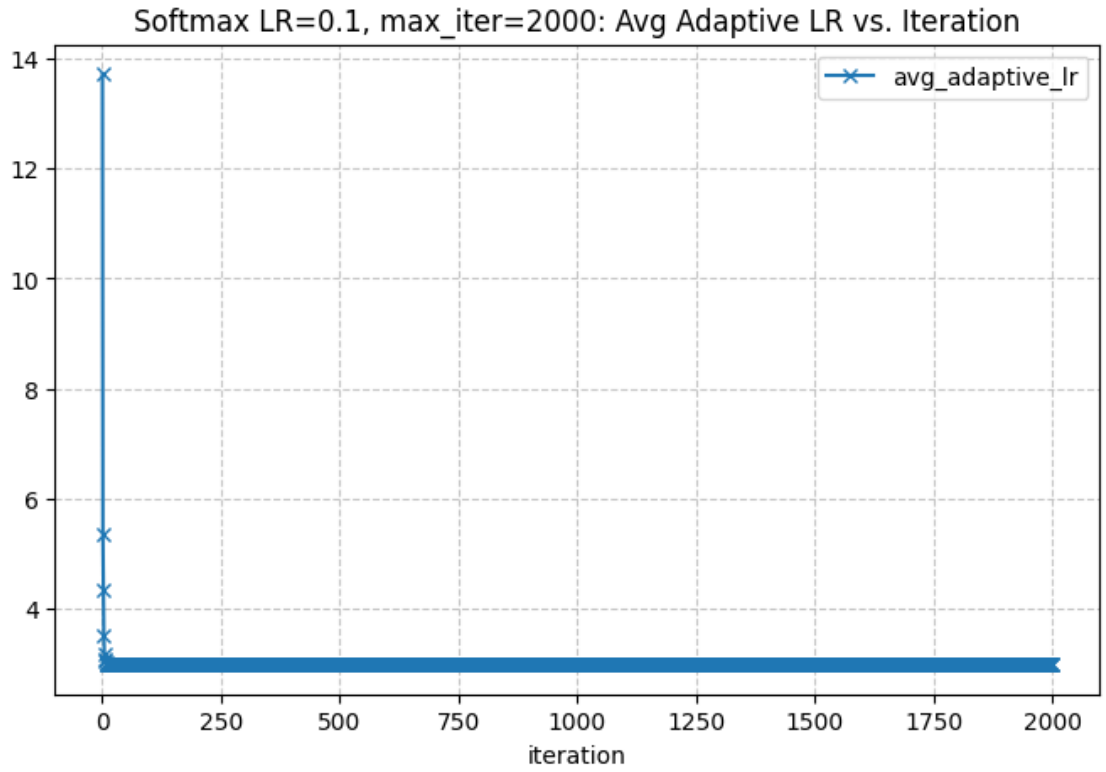


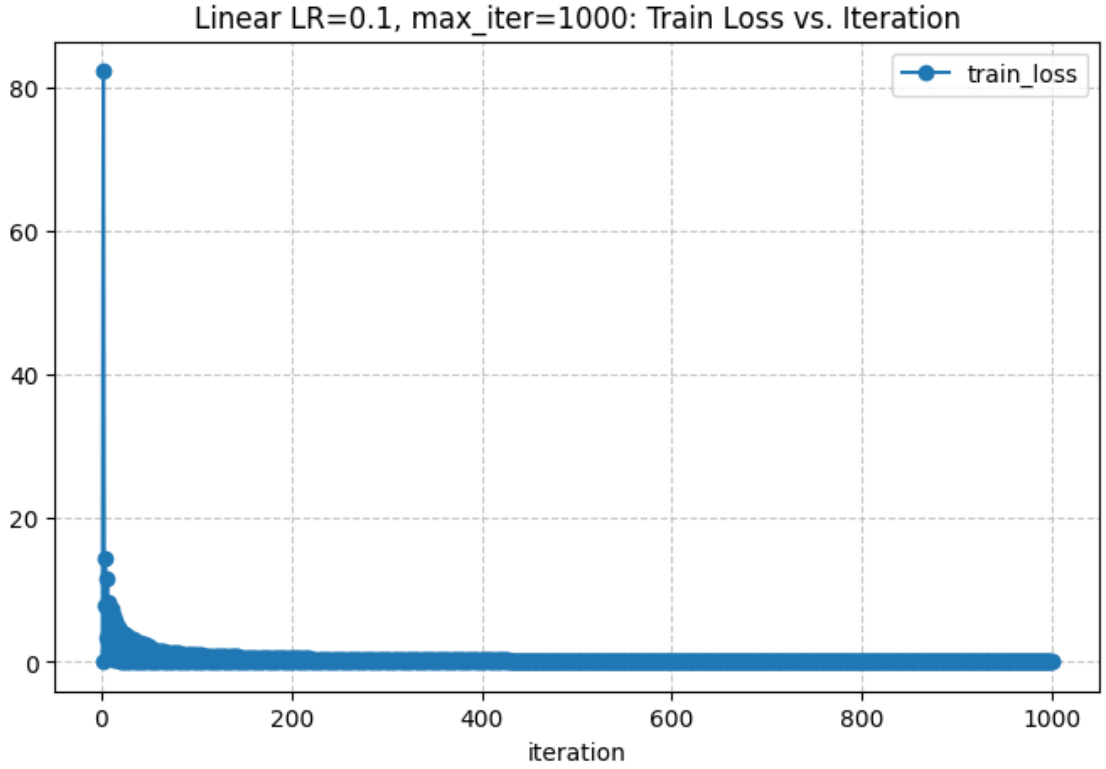

```
Confusions: Linear: 100%|      | 2/2 [00:00<00:00,  9.19it/s]
INFO - === Iteration-Level Visualization (All Models) ===
```

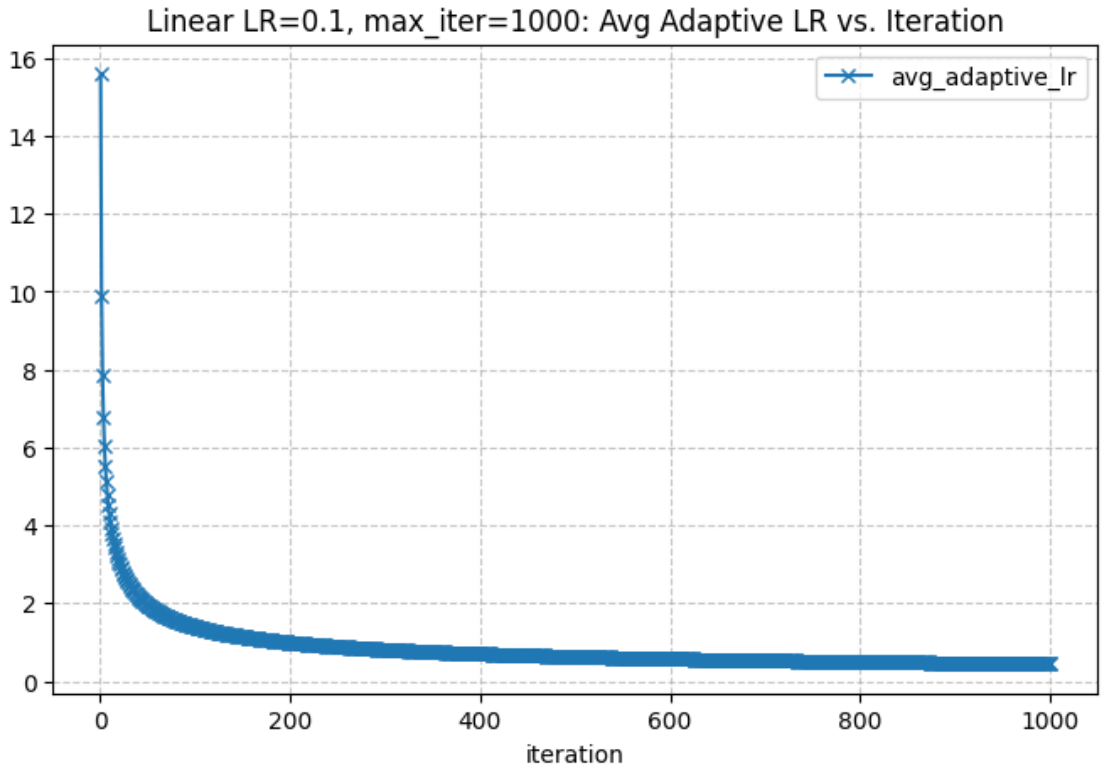

Softmax LR=0.1, max_iter=1000: Train Loss vs. Iteration
train_loss
5
4
3
2
1
0
0
200
400
600
800
1000
iteration

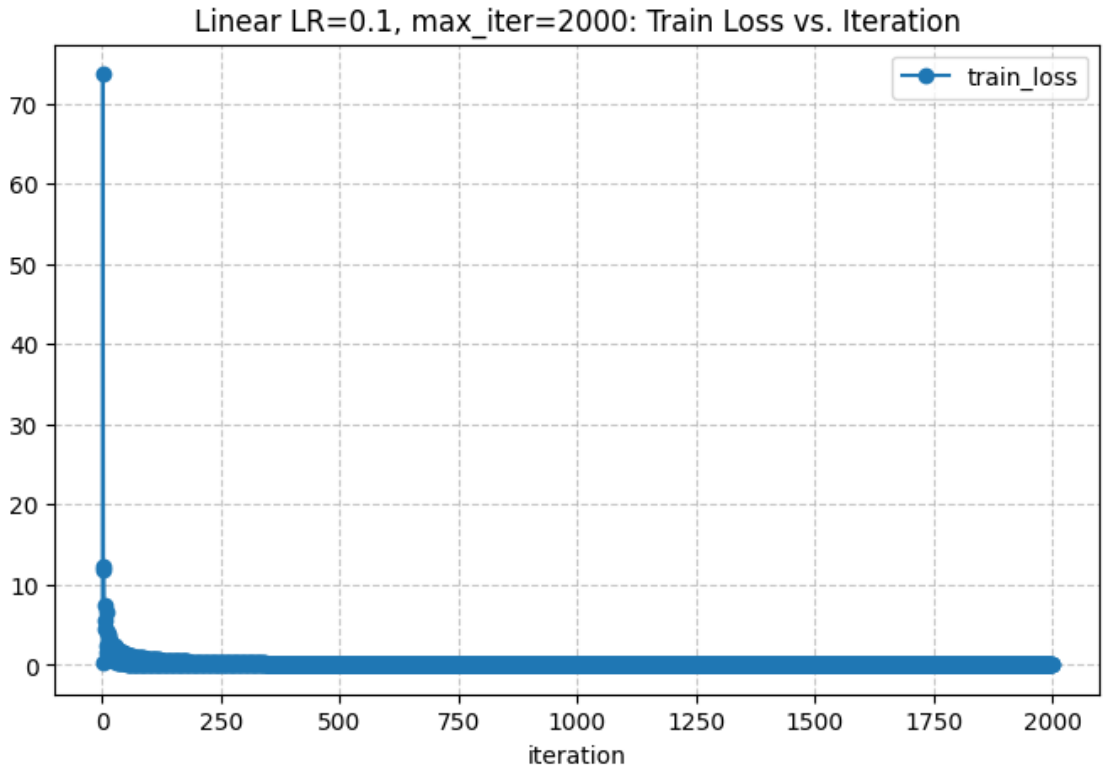

Softmax LR=0.1, max_iter=1000: Avg Adaptive LR vs. Iteration
avg_adaptive_lr
12
10
8
6
4
0
200
400
600
800
1000
iteration

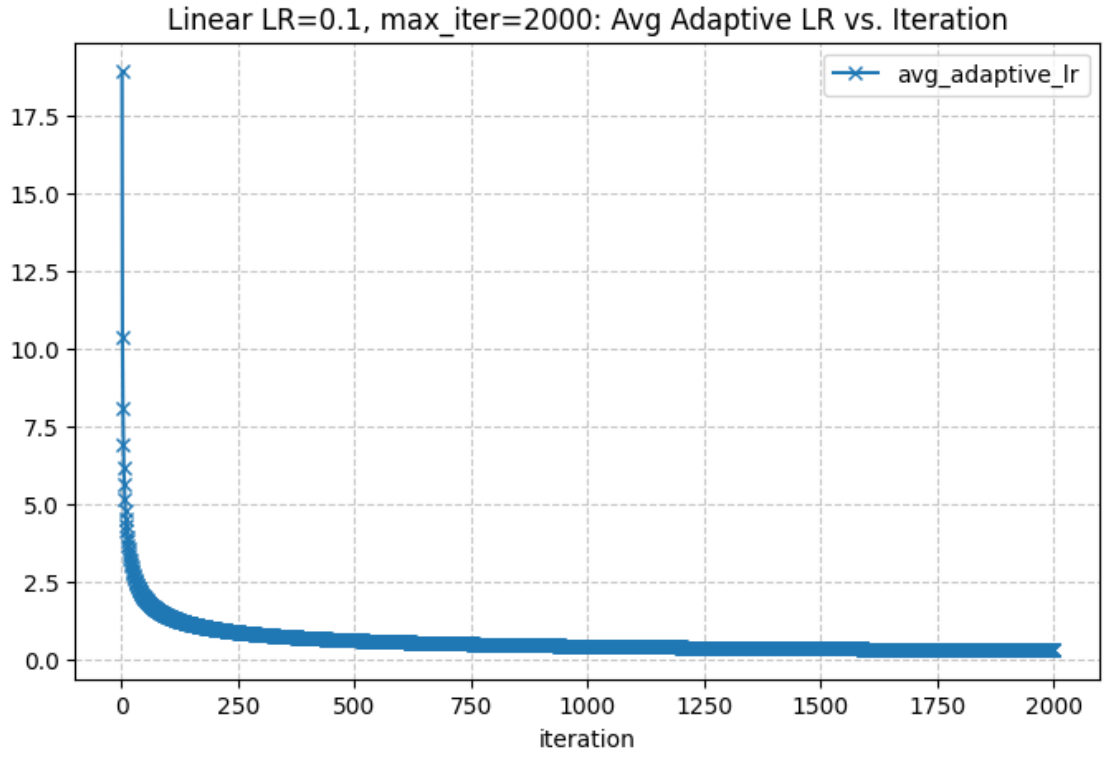

Softmax LR=0.1, max_iter=2000: Train Loss vs. Iteration
train_loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
250
500
750
1000
1250
1500
1750
2000
iteration


Softmax LR=0.1, max_iter=2000: Avg Adaptive LR vs. Iteration
avg_adaptive_lr
14
12
10
8
6
4
0
250
500
750
1000
1250
1500
1750
2000
iteration

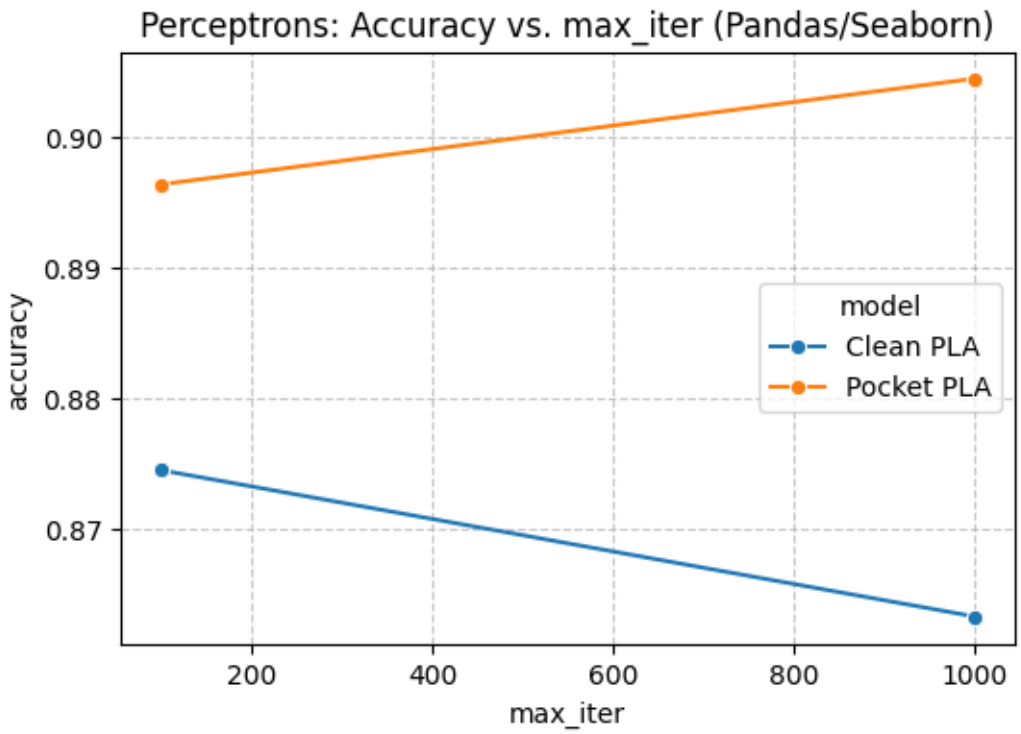

Linear LR=0.1, max_iter=1000: Train Loss vs. Iteration
train_loss
80
60
40
20
0
0
200
400
600
800
1000
iteration

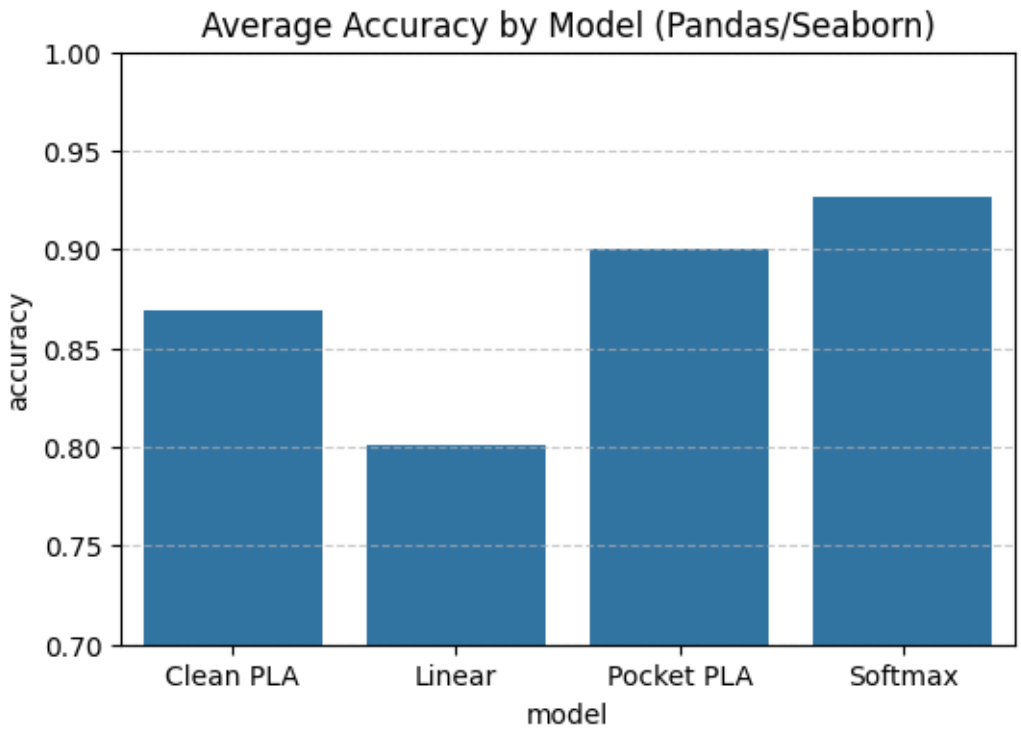

Linear LR=0.1, max_iter=1000: Avg Adaptive LR vs. Iteration
avg_adaptive_lr
16
14
12
10
8
6
4
2
0
0
200
400
600
800
1000
iteration

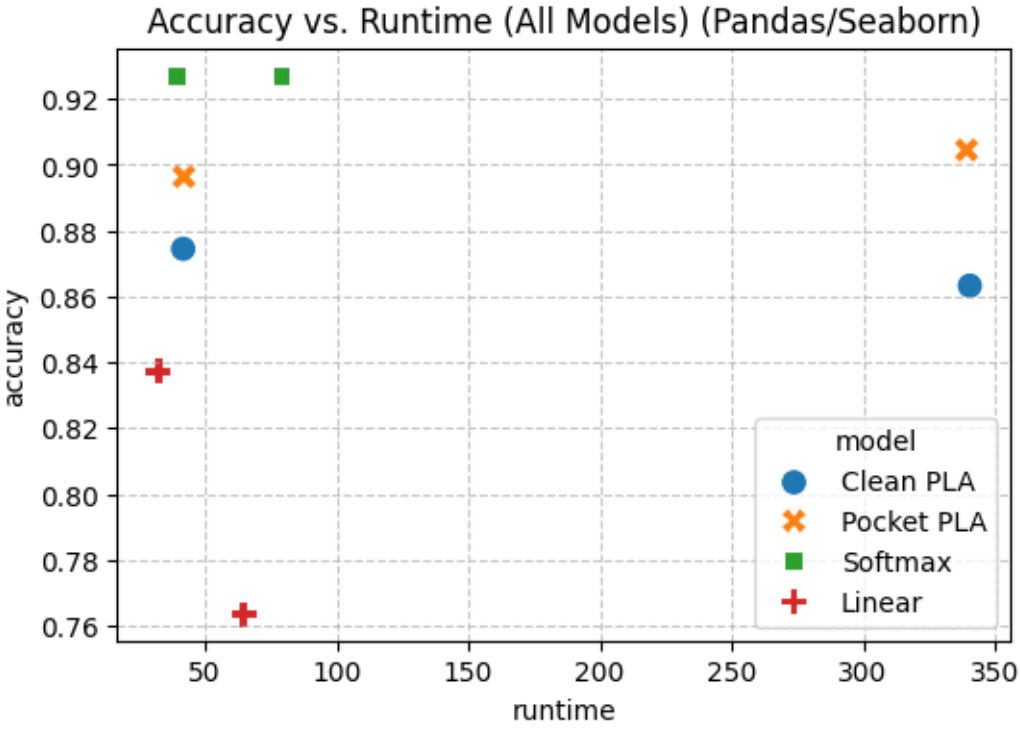

Linear LR=0.1, max_iter=2000: Train Loss vs. Iteration
train_loss
70
60
50
40
30
20
10
0
0
250
500
750
1000
1250
1500
1750
2000
iteration


Linear LR=0.1, max_iter=2000: Avg Adaptive LR vs. Iteration
avg_adaptive_lr
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
250
500
750
1000
1250
1500
1750
2000
iteration

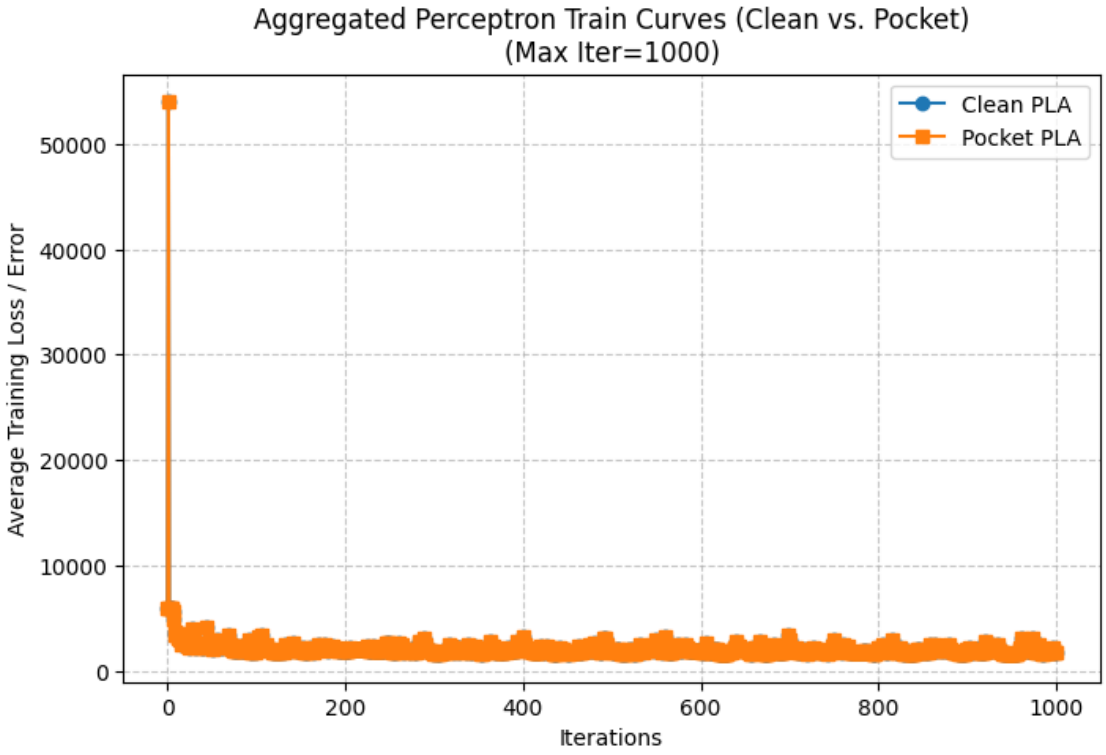
```
INFO - === Pandas + Seaborn Plots ===
```

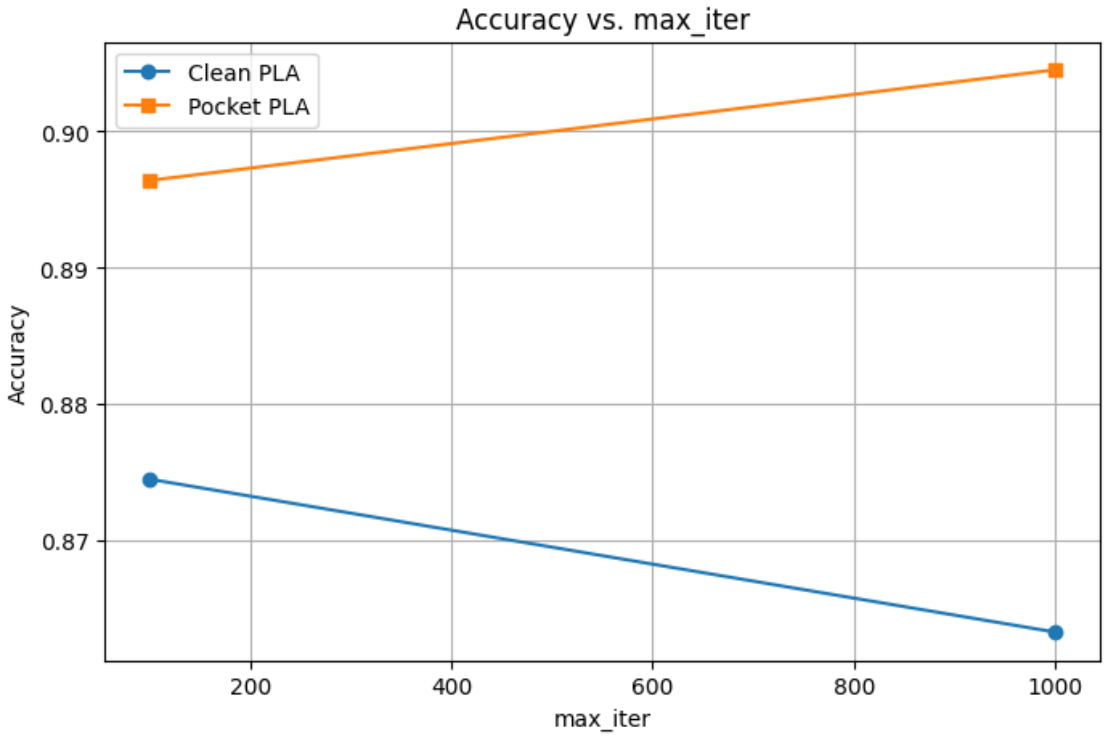


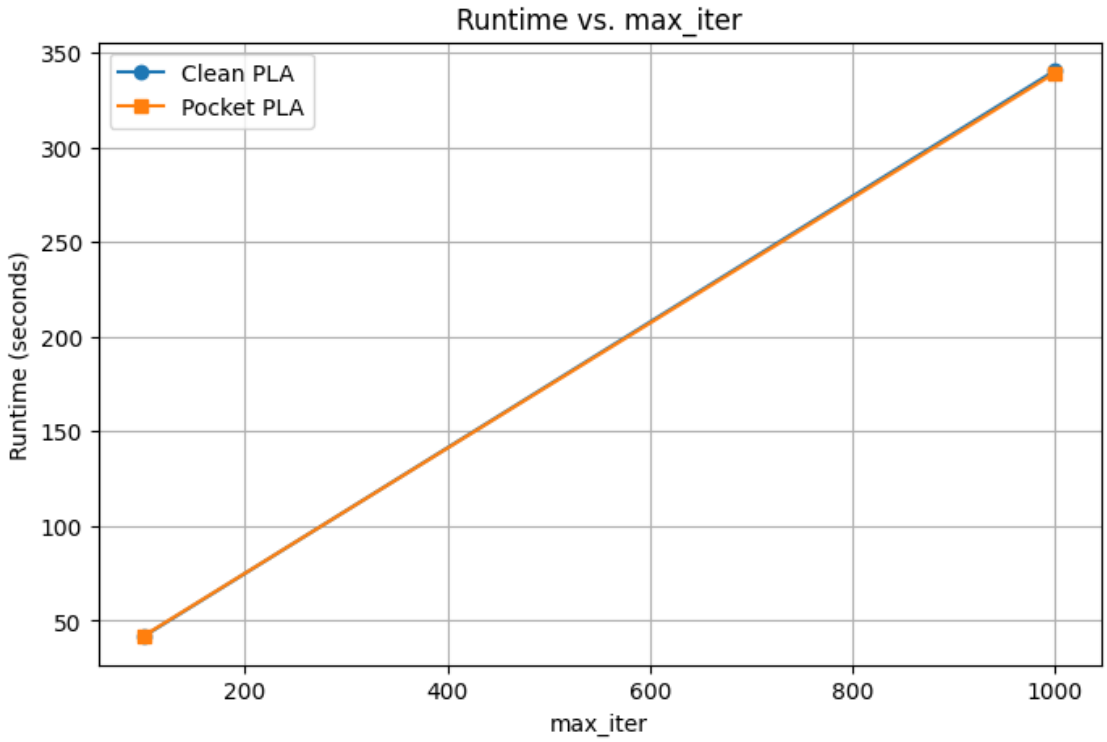

Average Accuracy by Model (Pandas/Seaborn)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
accuracy
Clean PLA
Linear
Pocket PLA
Softmax
model

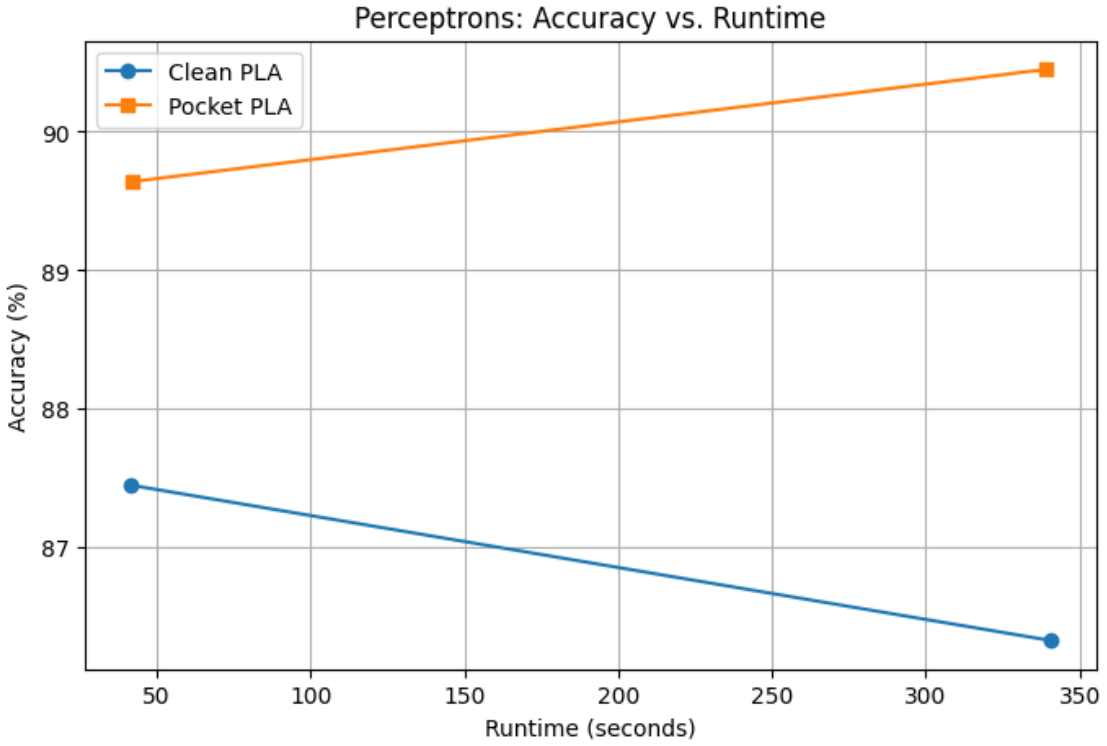

Accuracy vs. Runtime (All Models) (Pandas/Seaborn)
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
accuracy
50
100
150
200
250
300
350
runtime
model
Clean PLA
Pocket PLA
Softmax
Linear

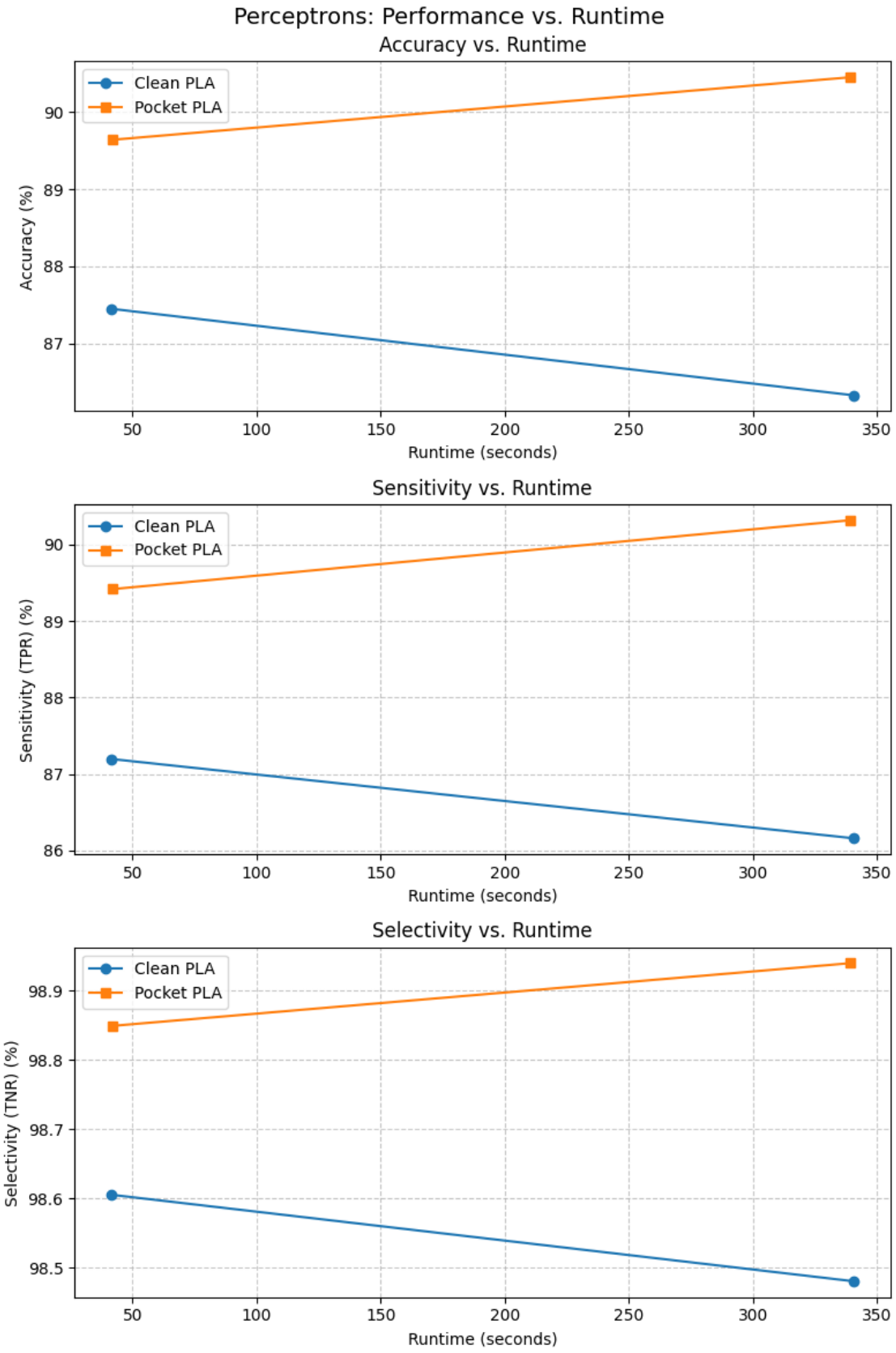
```
INFO - === Custom Summaries (Aggregated Curves, etc.) ===
```

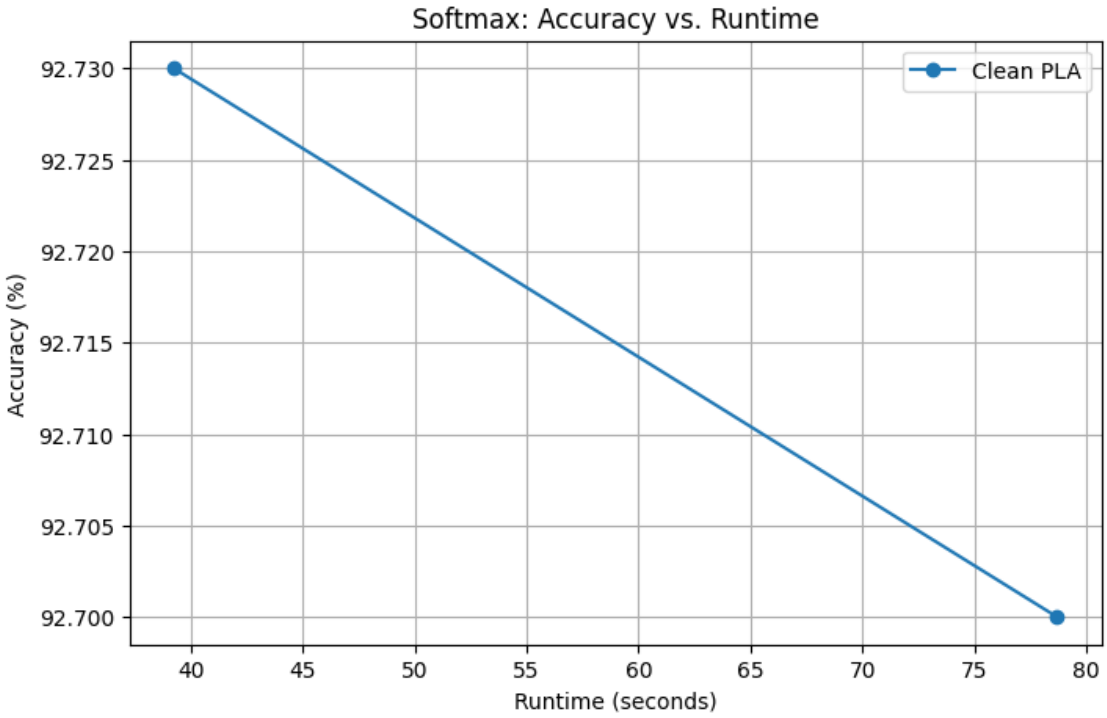


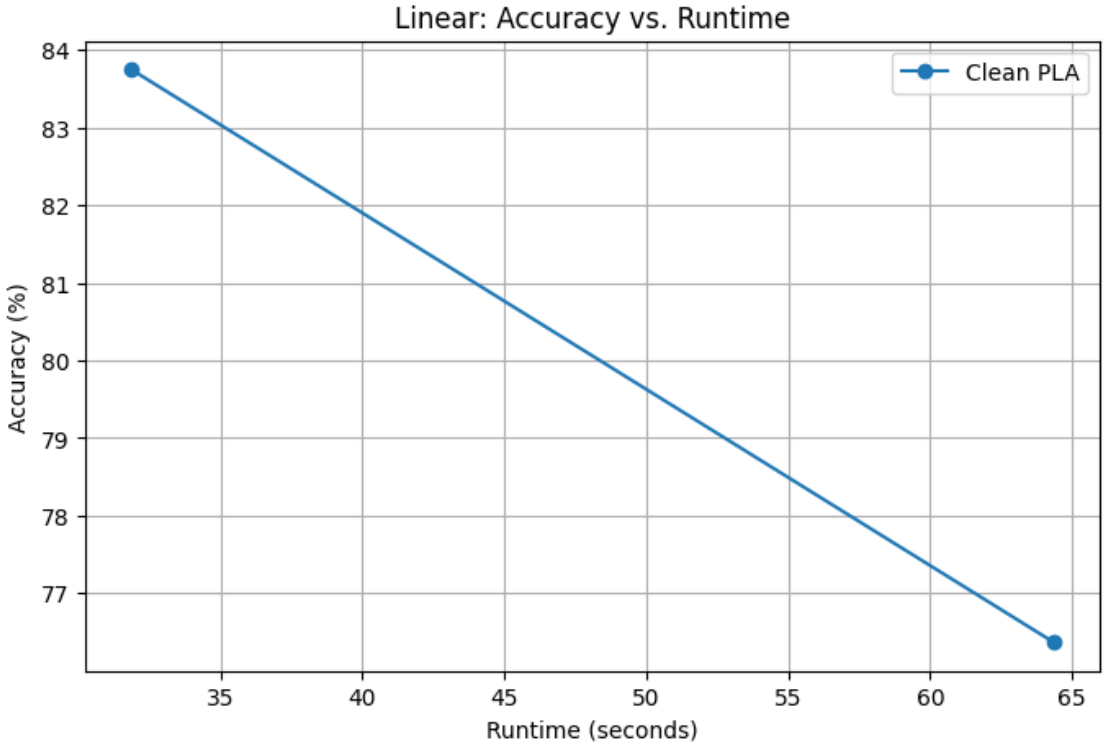


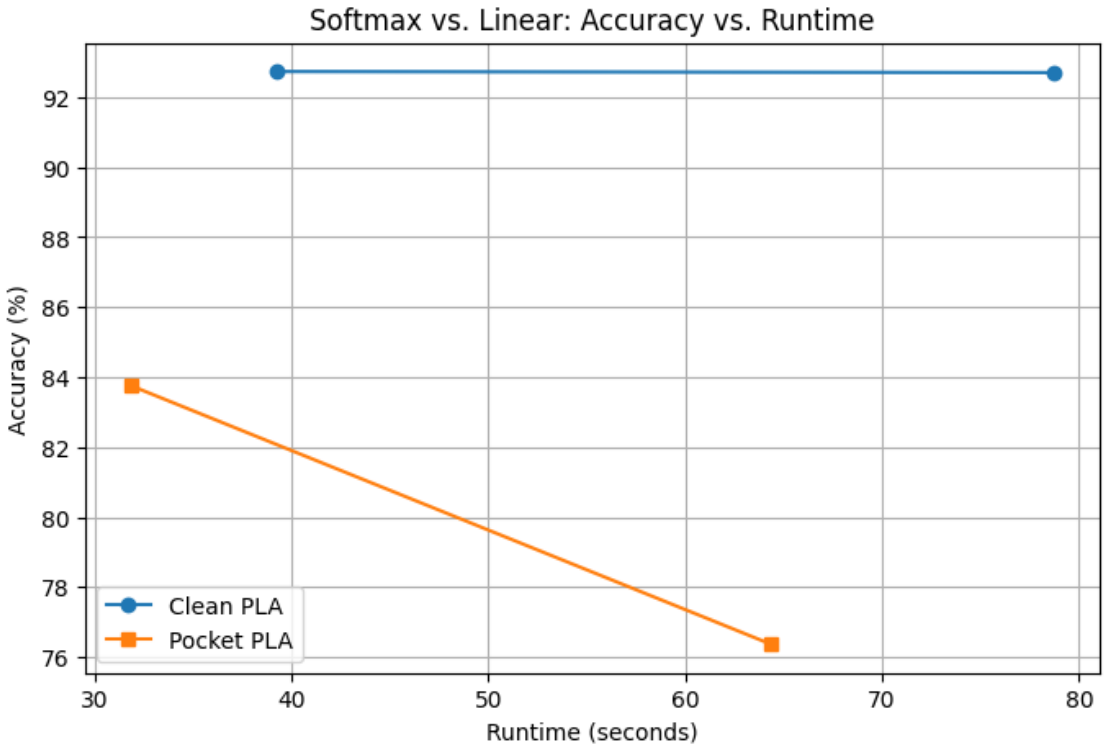

Runtime vs. max_iter
Clean PLA
Pocket PLA
Runtime (seconds)
max_iter
350
300
250
200
150
100
50
200
400
600
800
1000

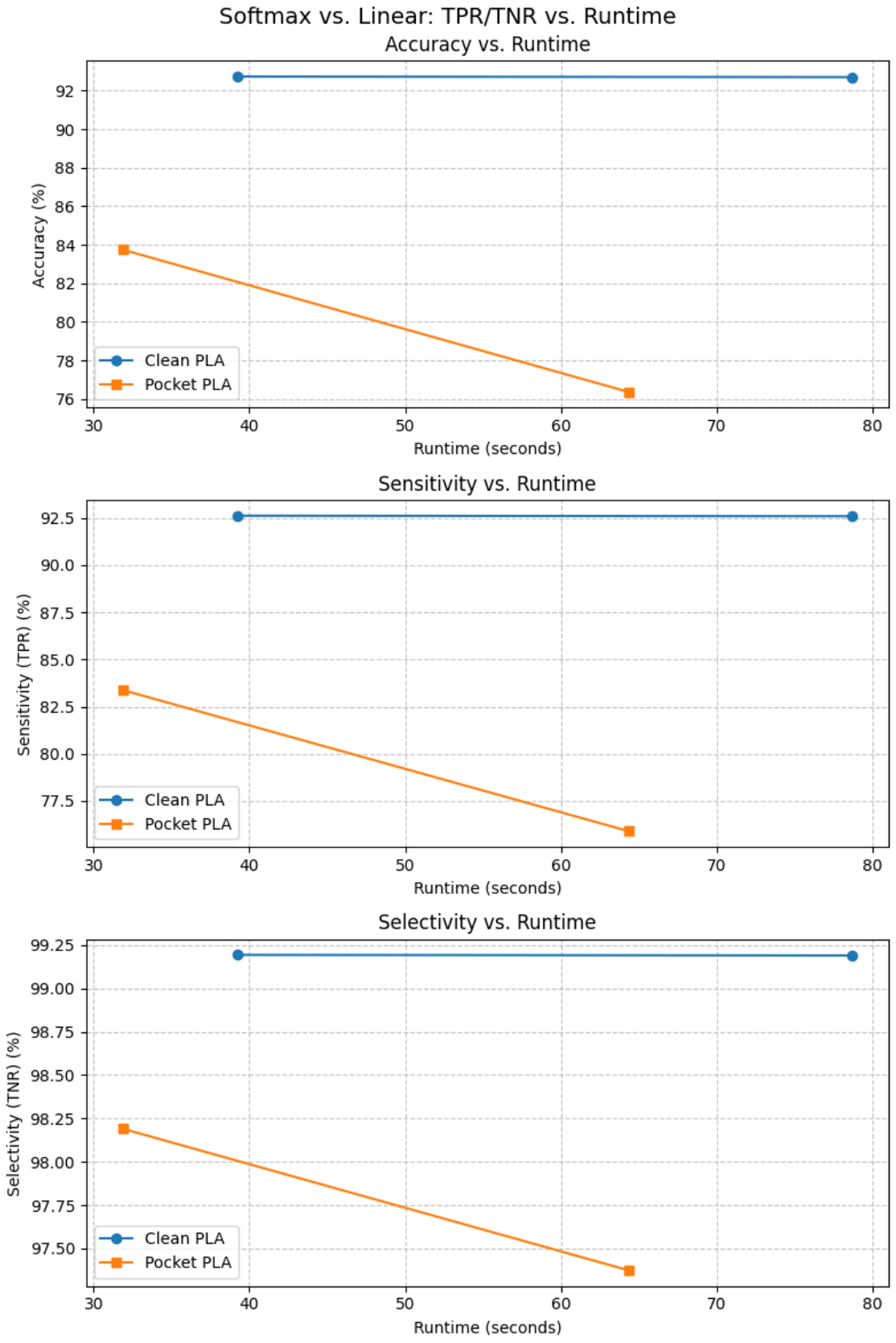

Perceptrons: Accuracy vs. Runtime
Clean PLA
Pocket PLA
Accuracy (%)
Runtime (seconds)
90
89
88
87
50
100
150
200
250
300
350

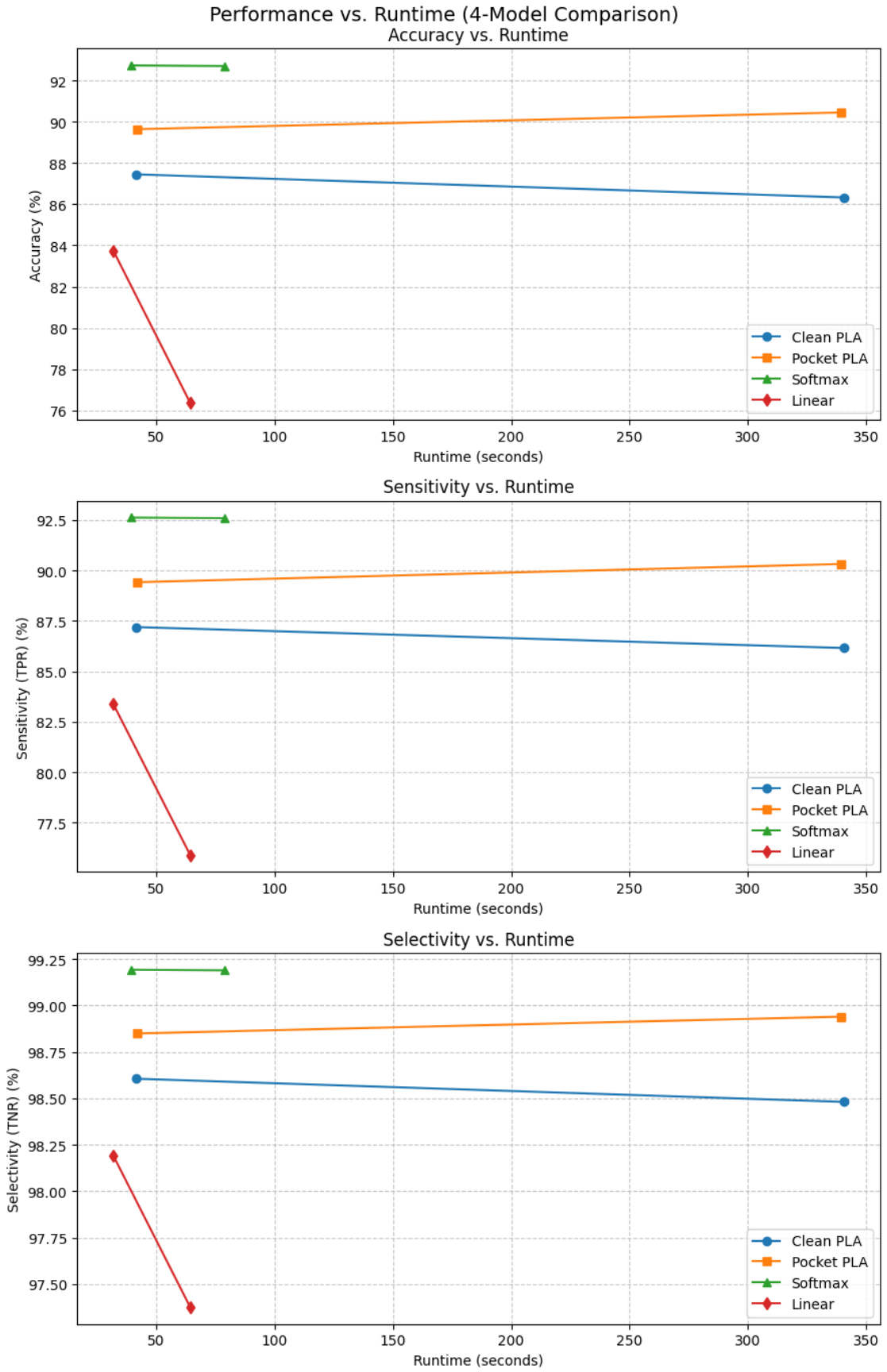

Perceptrons: Performance vs. Runtime
Accuracy vs. Runtime
Clean PLA
Pocket PLA
Accuracy (%)
90
89
88
87
50
100
150
200
250
300
350
Runtime (seconds)
Sensitivity vs. Runtime
Clean PLA
Pocket PLA
Sensitivity (TPR) (%)
90
89
88
87
86
50
100
150
200
250
300
350
Runtime (seconds)
Selectivity vs. Runtime
Clean PLA
Pocket PLA
Selectivity (TNR) (%)
98.9
98.8
98.7
98.6
98.5
50
100
150
200
250
300
350
Runtime (seconds)

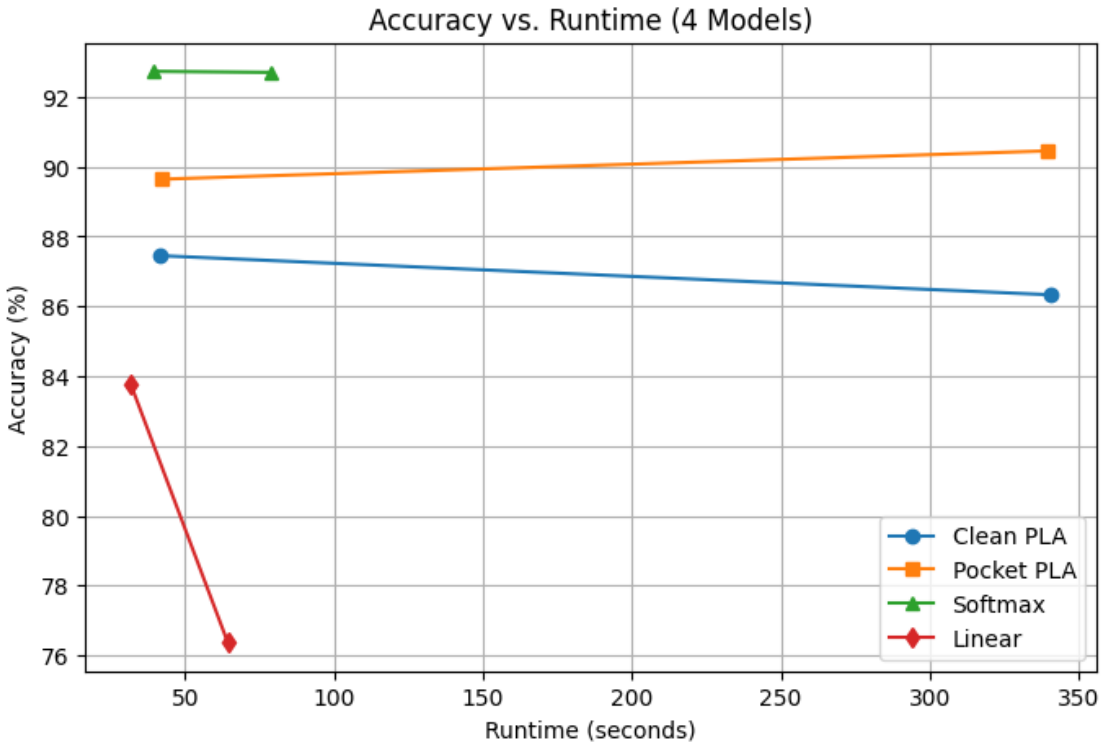

Softmax: Accuracy vs. Runtime
Clean PLA
92.730
92.725
92.720
92.715
92.710
92.705
92.700
Accuracy (%)
40
45
50
55
60
65
70
75
80
Runtime (seconds)


Linear: Accuracy vs. Runtime
Clean PLA
84
83
82
81
80
79
78
77
Accuracy (%)
35
40
45
50
55
60
65
Runtime (seconds)


Softmax vs. Linear: Accuracy vs. Runtime
Accuracy (%)
Runtime (seconds)
92
90
88
86
84
82
80
78
76
30
40
50
60
70
80
Clean PLA
Pocket PLA


Softmax vs. Linear: TPR/TNR vs. Runtime
Accuracy vs. Runtime
Accuracy (%)
92
90
88
86
84
82
80
78
76
Clean PLA
Pocket PLA
30
40
50
60
70
80
Runtime (seconds)
Sensitivity vs. Runtime
Sensitivity (TPR) (%)
92.5
90.0
87.5
85.0
82.5
80.0
77.5
Clean PLA
Pocket PLA
30
40
50
60
70
80
Runtime (seconds)
Selectivity vs. Runtime
Selectivity (TNR) (%)
99.25
99.00
98.75
98.50
98.25
98.00
97.75
97.50
Clean PLA
Pocket PLA
30
40
50
60
70
80
Runtime (seconds)


Performance vs. Runtime (4-Model Comparison)
Accuracy vs. Runtime
Accuracy (%)
Runtime (seconds)
Clean PLA
Pocket PLA
Softmax
Linear
Sensitivity vs. Runtime
Sensitivity (TPR) (%)
Runtime (seconds)
Clean PLA
Pocket PLA
Softmax
Linear
Selectivity vs. Runtime
Selectivity (TNR) (%)
Runtime (seconds)
Clean PLA
Pocket PLA
Softmax
Linear




```
INFO - === All Visualizations Complete ===
```