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}")

```

```python
  # 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]:
```python
# ========== 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
```

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

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)
```

```python
# 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]:
```python
# ==========================================================
# 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 ---
```

```python
    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%|          | 0/1 [00:00<?, ?it/s]INFO - --- Softmax
LR=0.1/Iter=1000 ---
Train Regressions:   0%|          | 0/1 [00:00<?, ?it/s]INFO - --- Softmax
LR=0.1/Iter=1000 ---
INFO - Iter 1/1000, Loss: 2.3315, Avg Adaptive LR: 14.249133
INFO - Iter 11/1000, Loss: 0.4386, Avg Adaptive LR: 2.859228
INFO - Iter 21/1000, Loss: 0.3750, Avg Adaptive LR: 2.854575
INFO - Iter 31/1000, Loss: 0.3516, Avg Adaptive LR: 2.853408
INFO - Iter 41/1000, Loss: 0.3366, Avg Adaptive LR: 2.852652
INFO - Iter 51/1000, Loss: 0.3258, Avg Adaptive LR: 2.852100
INFO - Iter 61/1000, Loss: 0.3174, Avg Adaptive LR: 2.851672
INFO - Iter 71/1000, Loss: 0.3106, Avg Adaptive LR: 2.851327
INFO - Iter 81/1000, Loss: 0.3050, Avg Adaptive LR: 2.851042
INFO - Iter 91/1000, Loss: 0.3003, Avg Adaptive LR: 2.850800
INFO - Iter 101/1000, Loss: 0.2962, Avg Adaptive LR: 2.850592
```

```
INFO - Iter 111/1000, Loss: 0.2927, Avg Adaptive LR: 2.850410
INFO - Iter 121/1000, Loss: 0.2895, Avg Adaptive LR: 2.850250
INFO - Iter 131/1000, Loss: 0.2868, Avg Adaptive LR: 2.850108
INFO - Iter 141/1000, Loss: 0.2843, Avg Adaptive LR: 2.849979
INFO - Iter 151/1000, Loss: 0.2820, Avg Adaptive LR: 2.849863
INFO - Iter 161/1000, Loss: 0.2799, Avg Adaptive LR: 2.849757
INFO - Iter 171/1000, Loss: 0.2780, Avg Adaptive LR: 2.849659
INFO - Iter 181/1000, Loss: 0.2763, Avg Adaptive LR: 2.849570
INFO - Iter 191/1000, Loss: 0.2747, Avg Adaptive LR: 2.849487
INFO - Iter 201/1000, Loss: 0.2732, Avg Adaptive LR: 2.849409
INFO - Iter 211/1000, Loss: 0.2718, Avg Adaptive LR: 2.849337
INFO - Iter 221/1000, Loss: 0.2705, Avg Adaptive LR: 2.849269
INFO - Iter 231/1000, Loss: 0.2692, Avg Adaptive LR: 2.849206
INFO - Iter 241/1000, Loss: 0.2681, Avg Adaptive LR: 2.849146
INFO - Iter 251/1000, Loss: 0.2670, Avg Adaptive LR: 2.849089
INFO - Iter 261/1000, Loss: 0.2659, Avg Adaptive LR: 2.849035
INFO - Iter 271/1000, Loss: 0.2649, Avg Adaptive LR: 2.848985
INFO - Iter 281/1000, Loss: 0.2640, Avg Adaptive LR: 2.848936
INFO - Iter 291/1000, Loss: 0.2631, Avg Adaptive LR: 2.848890
INFO - Iter 301/1000, Loss: 0.2622, Avg Adaptive LR: 2.848846
INFO - Iter 311/1000, Loss: 0.2614, Avg Adaptive LR: 2.848804
INFO - Iter 321/1000, Loss: 0.2606, Avg Adaptive LR: 2.848763
INFO - Iter 331/1000, Loss: 0.2599, Avg Adaptive LR: 2.848725
INFO - Iter 341/1000, Loss: 0.2592, Avg Adaptive LR: 2.848688
INFO - Iter 351/1000, Loss: 0.2585, Avg Adaptive LR: 2.848652
INFO - Iter 361/1000, Loss: 0.2578, Avg Adaptive LR: 2.848618
INFO - Iter 371/1000, Loss: 0.2572, Avg Adaptive LR: 2.848585
INFO - Iter 381/1000, Loss: 0.2565, Avg Adaptive LR: 2.848553
INFO - Iter 391/1000, Loss: 0.2559, Avg Adaptive LR: 2.848522
INFO - Iter 401/1000, Loss: 0.2554, Avg Adaptive LR: 2.848492
INFO - Iter 411/1000, Loss: 0.2548, Avg Adaptive LR: 2.848463
INFO - Iter 421/1000, Loss: 0.2543, Avg Adaptive LR: 2.848435
INFO - Iter 431/1000, Loss: 0.2537, Avg Adaptive LR: 2.848408
INFO - Iter 441/1000, Loss: 0.2532, Avg Adaptive LR: 2.848382
INFO - Iter 451/1000, Loss: 0.2527, Avg Adaptive LR: 2.848357
INFO - Iter 461/1000, Loss: 0.2523, Avg Adaptive LR: 2.848332
INFO - Iter 471/1000, Loss: 0.2518, Avg Adaptive LR: 2.848308
INFO - Iter 481/1000, Loss: 0.2513, Avg Adaptive LR: 2.848285
INFO - Iter 491/1000, Loss: 0.2509, Avg Adaptive LR: 2.848262
INFO - Iter 501/1000, Loss: 0.2505, Avg Adaptive LR: 2.848240
INFO - Iter 511/1000, Loss: 0.2501, Avg Adaptive LR: 2.848218
INFO - Iter 521/1000, Loss: 0.2497, Avg Adaptive LR: 2.848198
INFO - Iter 531/1000, Loss: 0.2493, Avg Adaptive LR: 2.848177
INFO - Iter 541/1000, Loss: 0.2489, Avg Adaptive LR: 2.848157
INFO - Iter 551/1000, Loss: 0.2485, Avg Adaptive LR: 2.848138
INFO - Iter 561/1000, Loss: 0.2481, Avg Adaptive LR: 2.848119
INFO - Iter 571/1000, Loss: 0.2478, Avg Adaptive LR: 2.848100
INFO - Iter 581/1000, Loss: 0.2474, Avg Adaptive LR: 2.848082
```

```
INFO - Iter 591/1000, Loss: 0.2471, Avg Adaptive LR: 2.848064
INFO - Iter 601/1000, Loss: 0.2467, Avg Adaptive LR: 2.848047
INFO - Iter 611/1000, Loss: 0.2464, Avg Adaptive LR: 2.848030
INFO - Iter 621/1000, Loss: 0.2461, Avg Adaptive LR: 2.848013
INFO - Iter 631/1000, Loss: 0.2458, Avg Adaptive LR: 2.847997
INFO - Iter 641/1000, Loss: 0.2455, Avg Adaptive LR: 2.847981
INFO - Iter 651/1000, Loss: 0.2452, Avg Adaptive LR: 2.847965
INFO - Iter 661/1000, Loss: 0.2449, Avg Adaptive LR: 2.847950
INFO - Iter 671/1000, Loss: 0.2446, Avg Adaptive LR: 2.847935
INFO - Iter 681/1000, Loss: 0.2443, Avg Adaptive LR: 2.847920
INFO - Iter 691/1000, Loss: 0.2440, Avg Adaptive LR: 2.847906
INFO - Iter 701/1000, Loss: 0.2437, Avg Adaptive LR: 2.847892
INFO - Iter 711/1000, Loss: 0.2435, Avg Adaptive LR: 2.847878
INFO - Iter 721/1000, Loss: 0.2432, Avg Adaptive LR: 2.847864
INFO - Iter 731/1000, Loss: 0.2429, Avg Adaptive LR: 2.847851
INFO - Iter 741/1000, Loss: 0.2427, Avg Adaptive LR: 2.847838
INFO - Iter 751/1000, Loss: 0.2424, Avg Adaptive LR: 2.847825
INFO - Iter 761/1000, Loss: 0.2422, Avg Adaptive LR: 2.847812
INFO - Iter 771/1000, Loss: 0.2419, Avg Adaptive LR: 2.847799
INFO - Iter 781/1000, Loss: 0.2417, Avg Adaptive LR: 2.847787
INFO - Iter 791/1000, Loss: 0.2415, Avg Adaptive LR: 2.847775
INFO - Iter 801/1000, Loss: 0.2412, Avg Adaptive LR: 2.847763
INFO - Iter 811/1000, Loss: 0.2410, Avg Adaptive LR: 2.847751
INFO - Iter 821/1000, Loss: 0.2408, Avg Adaptive LR: 2.847740
INFO - Iter 831/1000, Loss: 0.2406, Avg Adaptive LR: 2.847729
INFO - Iter 841/1000, Loss: 0.2403, Avg Adaptive LR: 2.847717
INFO - Iter 851/1000, Loss: 0.2401, Avg Adaptive LR: 2.847706
INFO - Iter 861/1000, Loss: 0.2399, Avg Adaptive LR: 2.847696
INFO - Iter 871/1000, Loss: 0.2397, Avg Adaptive LR: 2.847685
INFO - Iter 881/1000, Loss: 0.2395, Avg Adaptive LR: 2.847674
INFO - Iter 891/1000, Loss: 0.2393, Avg Adaptive LR: 2.847664
INFO - Iter 901/1000, Loss: 0.2391, Avg Adaptive LR: 2.847654
INFO - Iter 911/1000, Loss: 0.2389, Avg Adaptive LR: 2.847644
INFO - Iter 921/1000, Loss: 0.2387, Avg Adaptive LR: 2.847634
INFO - Iter 931/1000, Loss: 0.2385, Avg Adaptive LR: 2.847624
INFO - Iter 941/1000, Loss: 0.2383, Avg Adaptive LR: 2.847614
INFO - Iter 951/1000, Loss: 0.2382, Avg Adaptive LR: 2.847605
INFO - Iter 961/1000, Loss: 0.2380, Avg Adaptive LR: 2.847595
INFO - Iter 971/1000, Loss: 0.2378, Avg Adaptive LR: 2.847586
INFO - Iter 981/1000, Loss: 0.2376, Avg Adaptive LR: 2.847577
INFO - Iter 991/1000, Loss: 0.2374, Avg Adaptive LR: 2.847568
INFO - SoftmaxRegression training completed in 39.08 seconds.
INFO - --- Linear Regression LR=0.1/Iter=1000 ---
INFO - Iter 100/1000, Loss: 0.5989, Gradient Norm: 14.6290, Avg Adaptive LR:
1.3980995305524941
INFO - Iter 200/1000, Loss: 0.3252, Gradient Norm: 10.4496, Avg Adaptive LR:
0.9920557558799945
INFO - Iter 300/1000, Loss: 0.2276, Gradient Norm: 8.4748, Avg Adaptive LR:
```

```
0.8113521729798768
INFO - Iter 400/1000, Loss: 0.1759, Gradient Norm: 7.2141, Avg Adaptive LR:
0.7033050218502679
INFO - Iter 500/1000, Loss: 0.1456, Gradient Norm: 6.3602, Avg Adaptive LR:
0.6294784290522984
INFO - Iter 600/1000, Loss: 0.1260, Gradient Norm: 5.7409, Avg Adaptive LR:
0.5749269428938112
INFO - Iter 700/1000, Loss: 0.1117, Gradient Norm: 5.2465, Avg Adaptive LR:
0.5325050884203198
INFO - Iter 800/1000, Loss: 0.1013, Gradient Norm: 4.8515, Avg Adaptive LR:
0.49825569272356784
INFO - Iter 900/1000, Loss: 0.0929, Gradient Norm: 4.5122, Avg Adaptive LR:
0.46988245783589533
INFO - Iter 1000/1000, Loss: 0.0870, Gradient Norm: 4.2530, Avg Adaptive LR:
0.44587542272655645
INFO - LinearRegressionClassifier training completed in 31.59 seconds.
Train Regressions: 100%|     | 1/1 [01:10<00:00, 70.67s/it]
INFO - Training complete for Softmax and Linear.
INFO - === TRAINING PERCEPTRON MODELS (Clean & Pocket) ===
Train Clean & Pocket:   0%|          | 0/1 [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: 100%|     | 1/1 [01:23<00:00, 83.48s/it]
INFO - Training complete for Clean PLA and Pocket PLA.
INFO - === ALL TRAINING COMPLETE ===
```

6 Evaluate

[7]:
```
##################################################
# 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)
```

```python
    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),
```

```python
        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)
```

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

```
Evaluate Clean & Pocket:   0%|          | 0/1 [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 - 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 - 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, 3270.92it/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, 3646.90it/s]
Evaluate Clean & Pocket: 100%|      | 1/1 [00:00<00:00, 56.18it/s]
Aggregating train losses across Perceptron models: 100%|      | 1/1
[00:00<00:00, 4837.72it/s]
Aggregating train losses across Perceptron models: 100%|      | 1/1
[00:00<00:00, 6743.25it/s]
Evaluate Regressions:   0%|          | 0/1 [00:00<?, ?it/s]INFO - Built-in
Confusion Matrix:
[[ 961    0    0    2    0    5    7    3    2    0]
 [   0 1113    5    3    0    1    3    2    8    0]
 [   6    9  924   19    8    3   11   10   39    3]
 [   3    0   17  926    1   21    3   10   23    6]
 [   1    1    3    2  916    0   12    5   10   32]
 [   7    3    1   35    8  775   15   11   31    6]
 [  13    3    9    1    7   15  907    1    2    0]
 [   1    6   24    6    5    0    0  952    4   30]
 [   6    9    5   21    9   22    7   14  876    5]
 [  10    7    1    8   25    7    0   23    7  921]]
INFO - Overall Accuracy: 92.71%
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=0.92, 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, 3087.00it/s]
INFO - Built-in Confusion Matrix:
[[ 770    1   20    9   15    2  154    6    2    1]
 [   0 1117    5    1    3    0    7    1    1    0]
 [   2   74  855   14   17    0   39   22    9    0]
 [   1   35   40  852    8    1   34   35    2    2]
 [   0   28    4    0  917    0   19    3    1   10]
 [   5   39   26  157   80  390  123   49   11   12]
 [   1   10    5    0   18    4  920    0    0    0]
 [   1   52   22    5   22    0    4  909    0   13]
 [   2  147   48   70   87    8  145   40  423    4]
 [   0   26   11   17  185    0   27  133    1  609]]
INFO - Overall Accuracy: 77.62%
INFO - Class '0': TPR=0.79, TNR=1.00
INFO - Class '1': TPR=0.98, TNR=0.95
INFO - Class '2': TPR=0.83, TNR=0.98
INFO - Class '3': TPR=0.84, TNR=0.97
INFO - Class '4': TPR=0.93, TNR=0.95
INFO - Class '5': TPR=0.44, TNR=1.00
INFO - Class '6': TPR=0.96, TNR=0.94
INFO - Class '7': TPR=0.88, TNR=0.97
INFO - Class '8': TPR=0.43, TNR=1.00
INFO - Class '9': TPR=0.60, TNR=1.00
Evaluating class metrics: 100%|      | 10/10 [00:00<00:00, 2605.64it/s]
Evaluate Regressions: 100%|      | 1/1 [00:00<00:00, 32.58it/s]
INFO - Evaluation complete for Perceptrons & Regressions.
```

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

[8]:
```python
##################################################
# 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,
```

```python
    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,
```

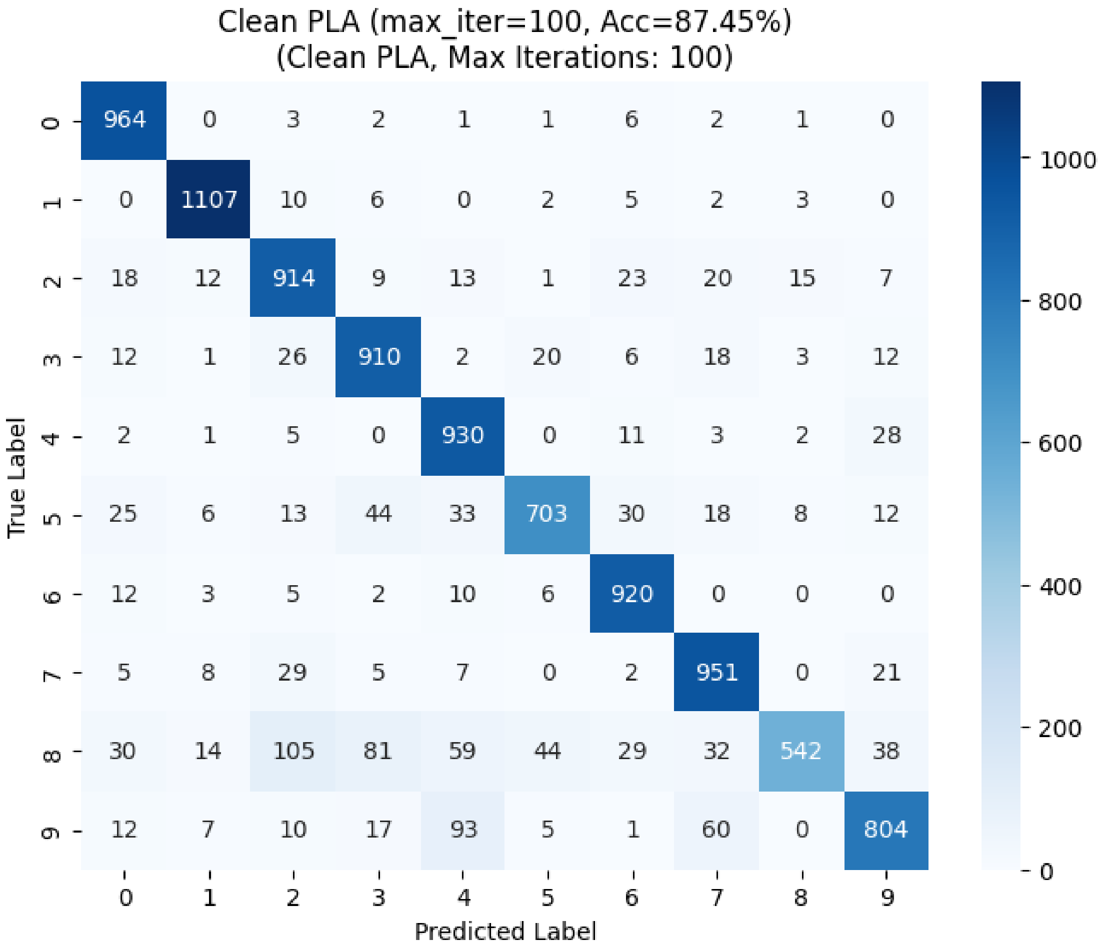
```python
    acc_clean=accuracies_clean,
    rt_pocket=runtimes_pocket,
    acc_pocket=accuracies_pocket,
    rt_softmax=runtimes_softmax,
    acc_softmax=accuracies_softmax,
    rt_linear=runtimes_linear,
    acc_linear=accuracies_linear,
    title="Accuracy vs. Runtime (4 Models)"
)

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

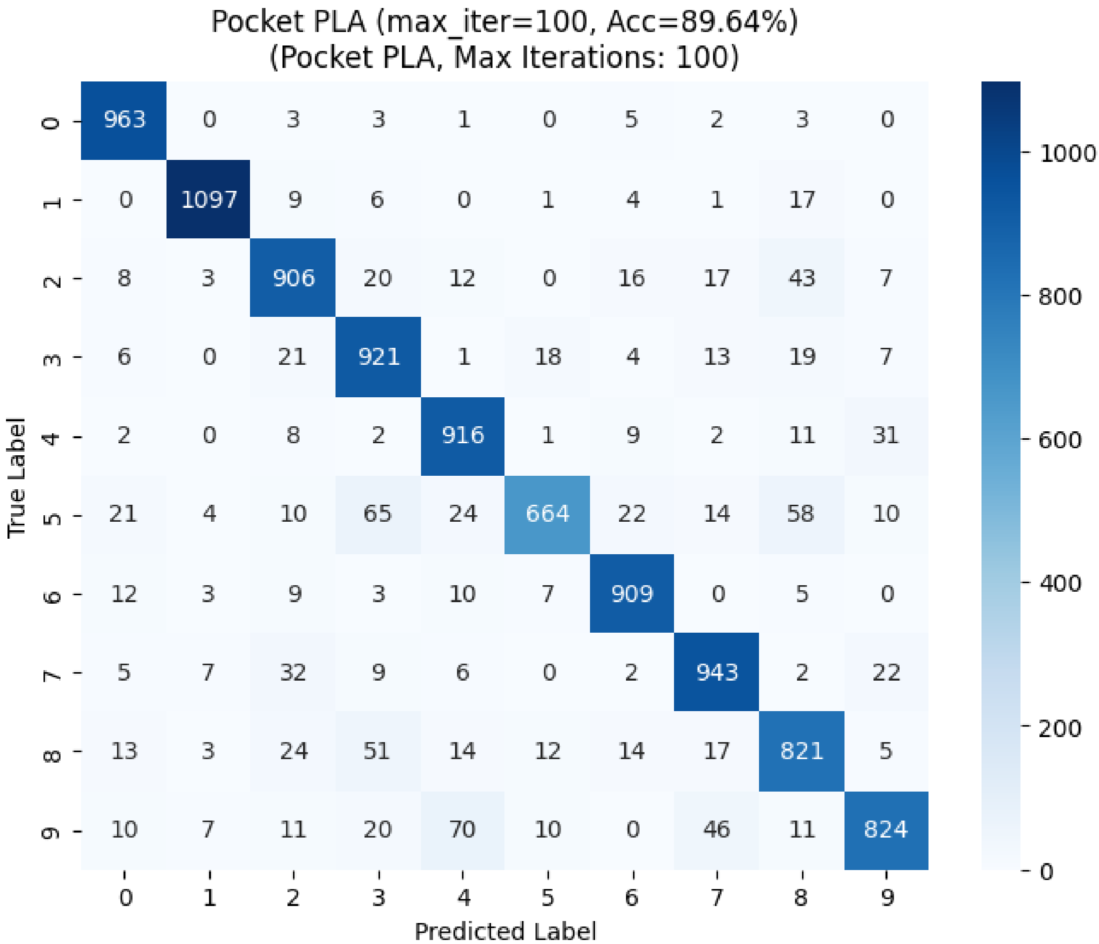
```
Collecting Clean PLA: 100%|          | 1/1 [00:00<00:00, 7854.50it/s]
Collecting Pocket PLA: 100%|          | 1/1 [00:00<00:00, 10082.46it/s]
Collecting Softmax: 100%|          | 1/1 [00:00<00:00, 31775.03it/s]
Collecting Linear: 100%|          | 1/1 [00:00<00:00, 20360.70it/s]
INFO - Combined Results DataFrame:
        model  max_iter    runtime  accuracy  sensitivity  selectivity
0   Clean PLA       100  41.746791    0.8745     0.871954     0.986055
1  Pocket PLA       100  41.736155    0.8964     0.894188     0.988493
2     Softmax      1000  39.079236    0.9271     0.926004     0.991905
3      Linear      1000  31.588556    0.7762     0.769537     0.975084
```

	model	max_iter	runtime	accuracy	sensitivity	selectivity
0	Clean PLA	100	41.746791	0.8745	0.871954	0.986055
1	Pocket PLA	100	41.736155	0.8964	0.894188	0.988493
2	Softmax	1000	39.079236	0.9271	0.926004	0.991905
3	Linear	1000	31.588556	0.7762	0.769537	0.975084

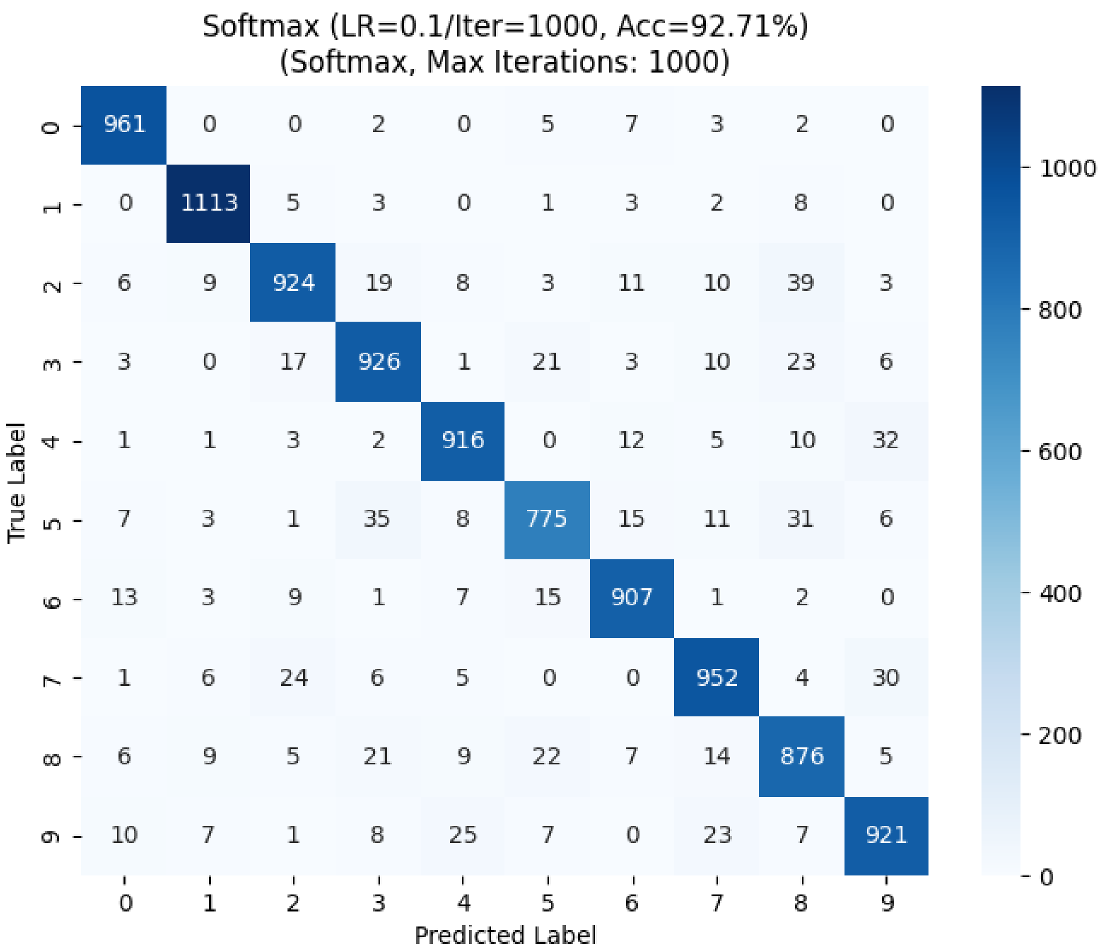
```
INFO - === Plotting ALL Confusion Matrices ===
Confusions: Clean PLA:   0%|          | 0/1 [00:00<?, ?it/s]
```



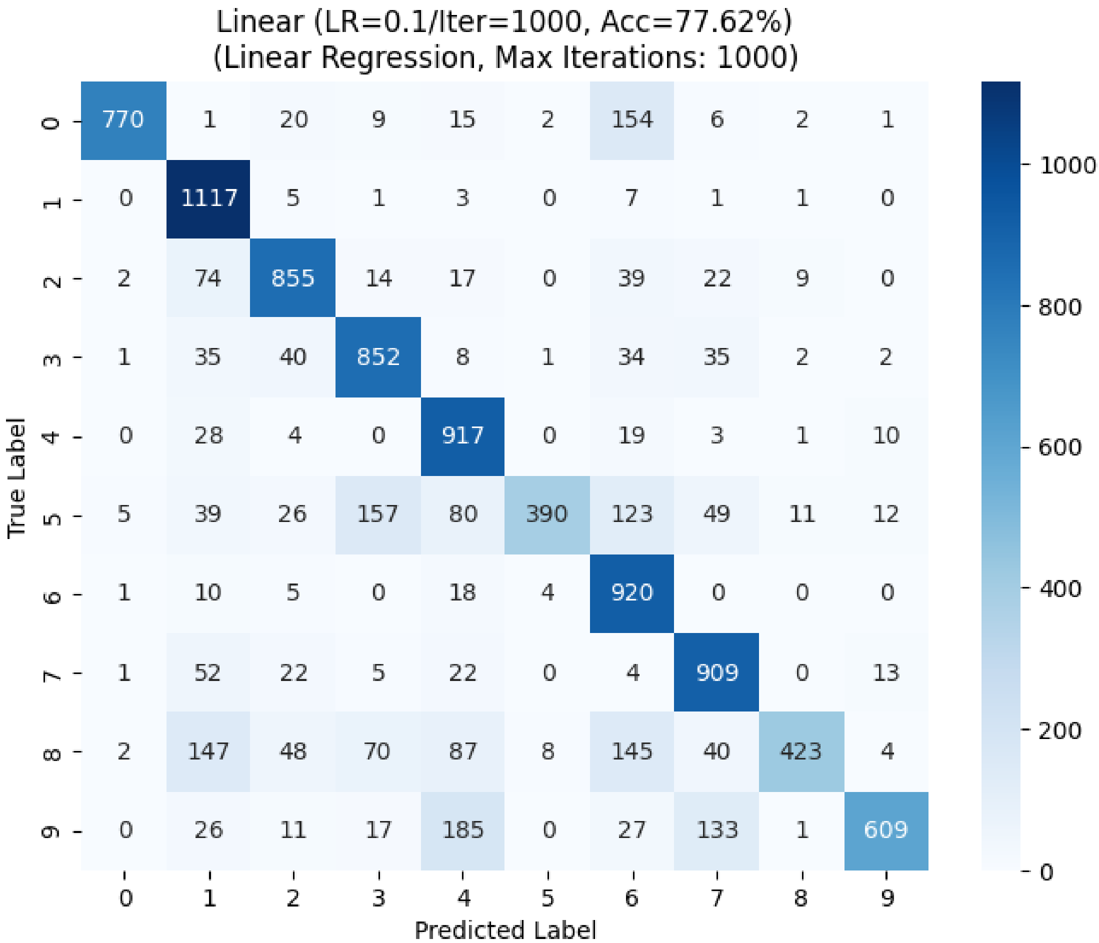

```
Confusions: Clean PLA: 100%|      | 1/1 [00:00<00:00,  7.58it/s]
Confusions: Pocket PLA:   0%|           | 0/1 [00:00<?, ?it/s]
```



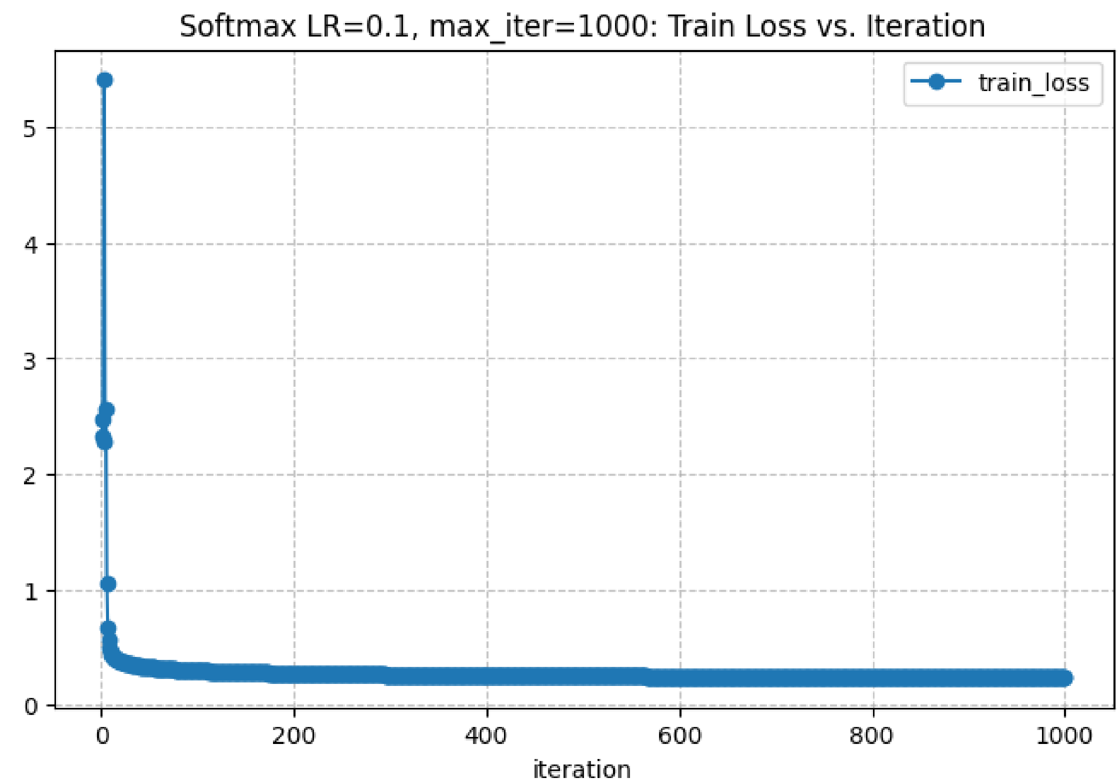

```
Confusions: Pocket PLA: 100%|          | 1/1 [00:00<00:00,  9.17it/s]
Confusions: Softmax:   0%|          | 0/1 [00:00<?, ?it/s]
```



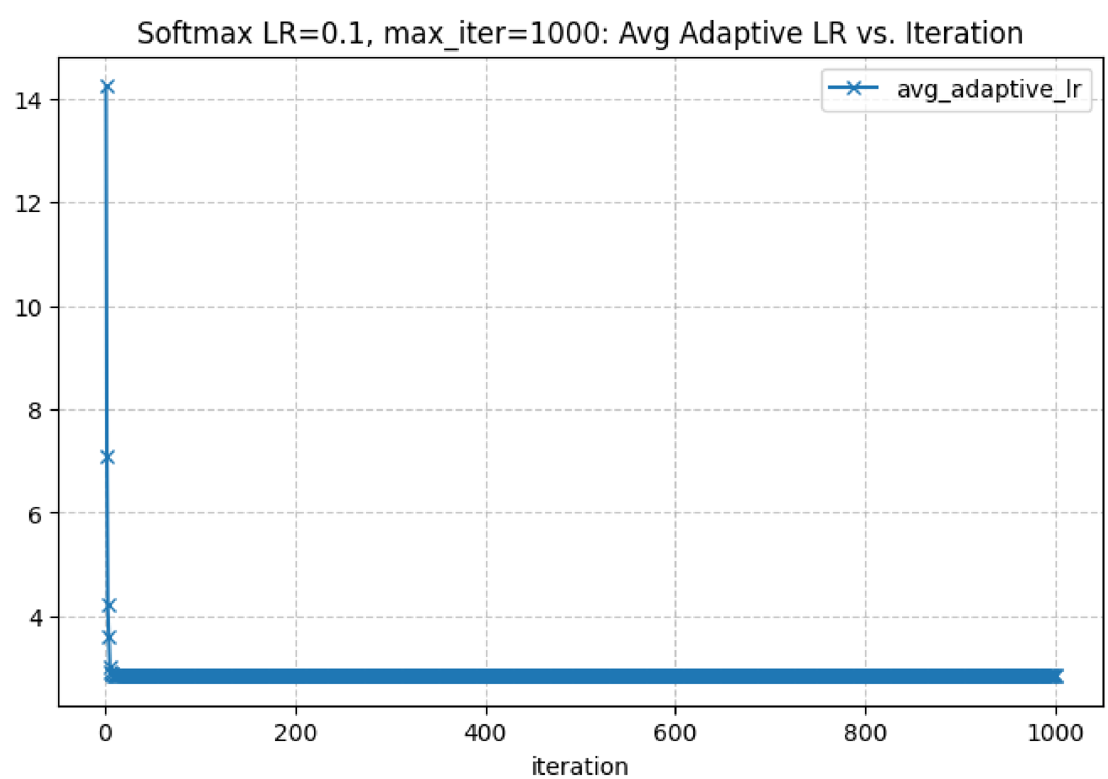

```
Confusions: Softmax: 100%|      | 1/1 [00:00<00:00,  9.48it/s]
Confusions: Linear:    0%|          | 0/1 [00:00<?, ?it/s]
```



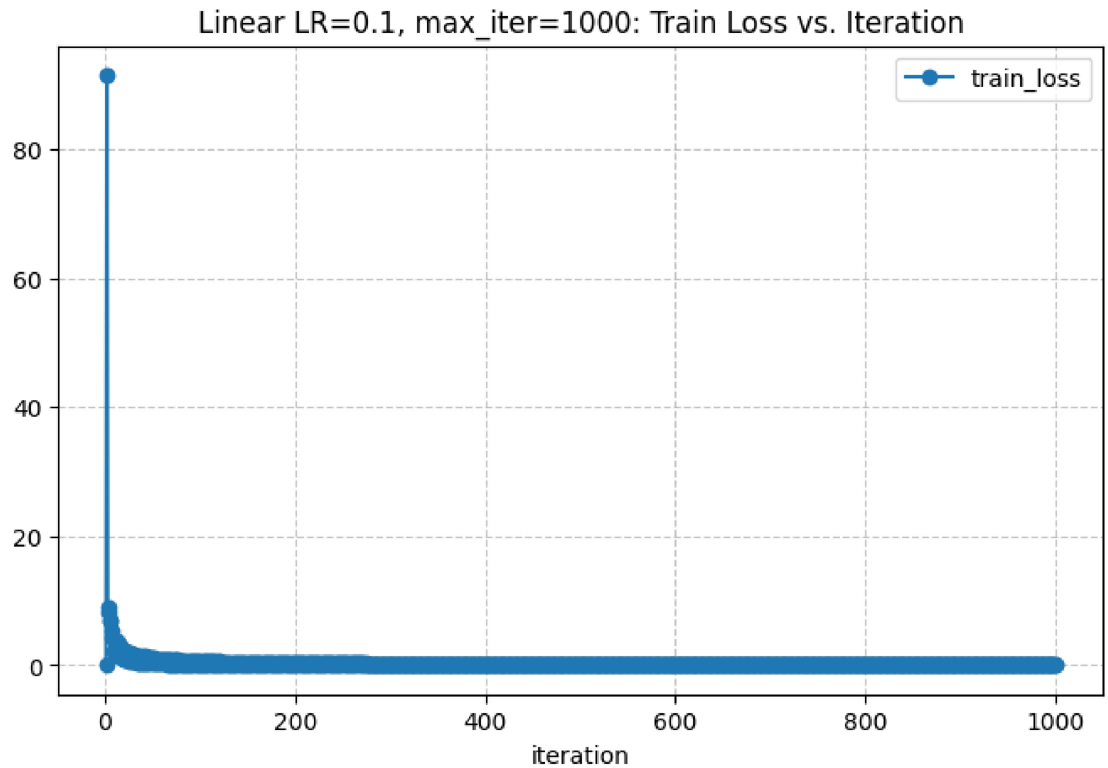

```
Confusions: Linear: 100%|      | 1/1 [00:00<00:00,  9.04it/s]
INFO - === Iteration-Level Visualization (All Models) ===
```



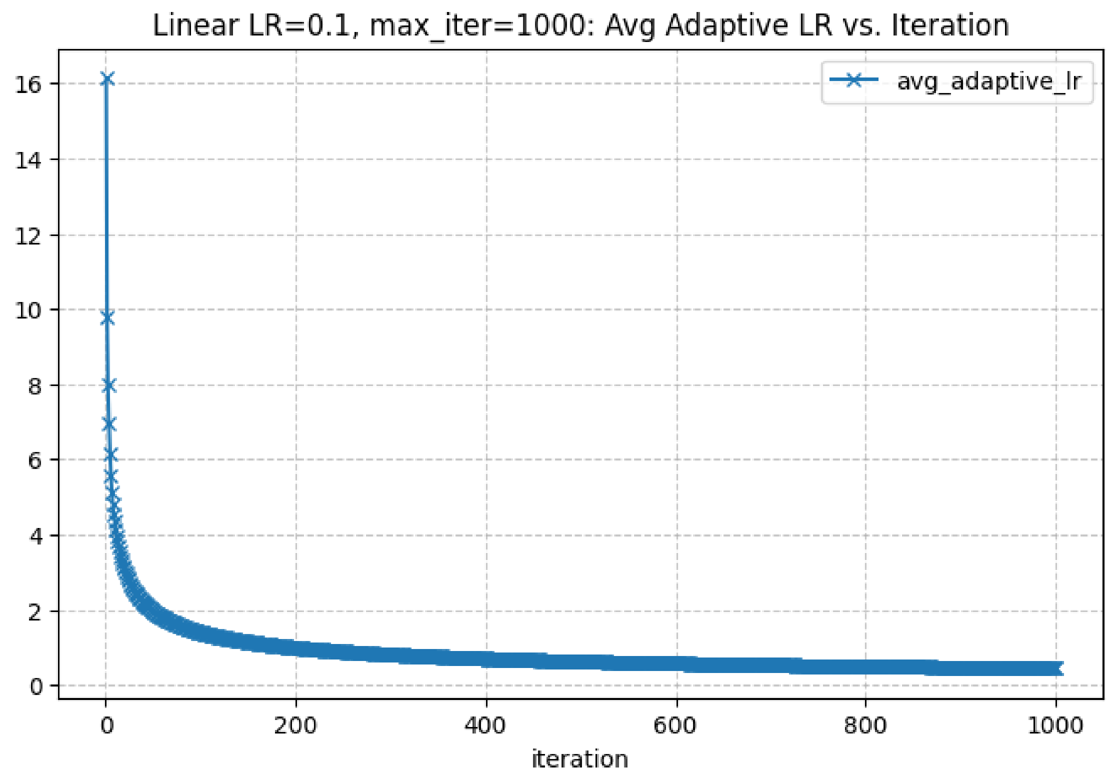
Softmax LR=0.1, max_iter=1000: Train Loss vs. Iteration
train_loss
0
1
2
3
4
5
0
200
400
600
800
1000
iteration



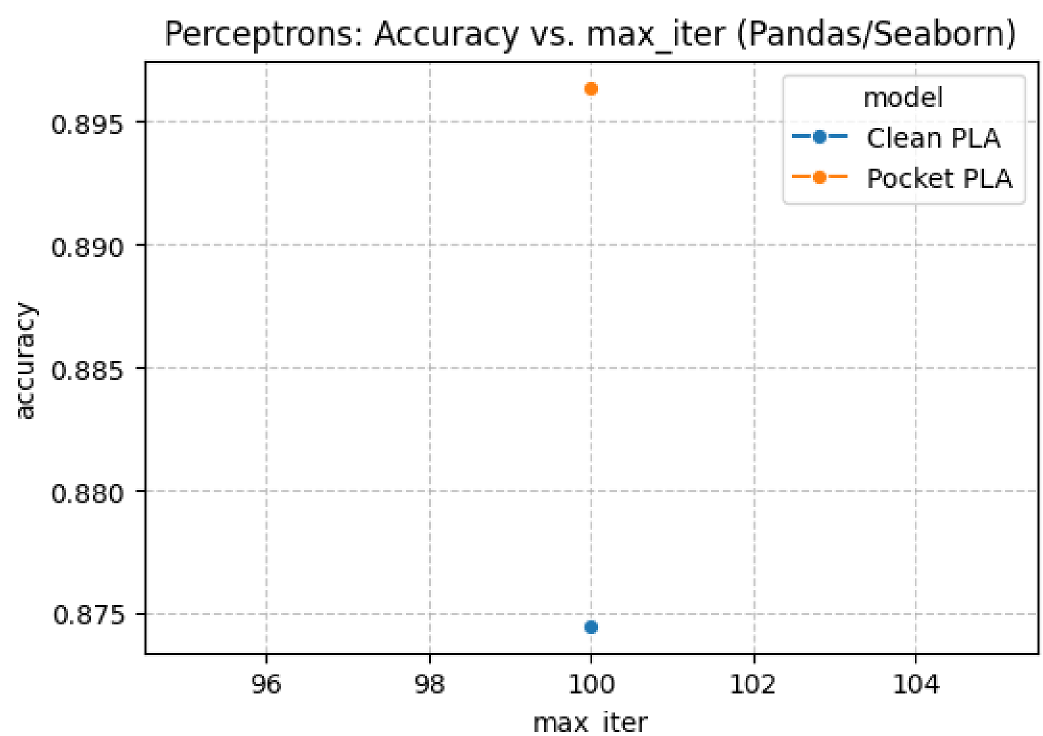
Softmax LR=0.1, max_iter=1000: Avg Adaptive LR vs. Iteration
avg_adaptive_lr
4
6
8
10
12
14
0
200
400
600
800
1000
iteration

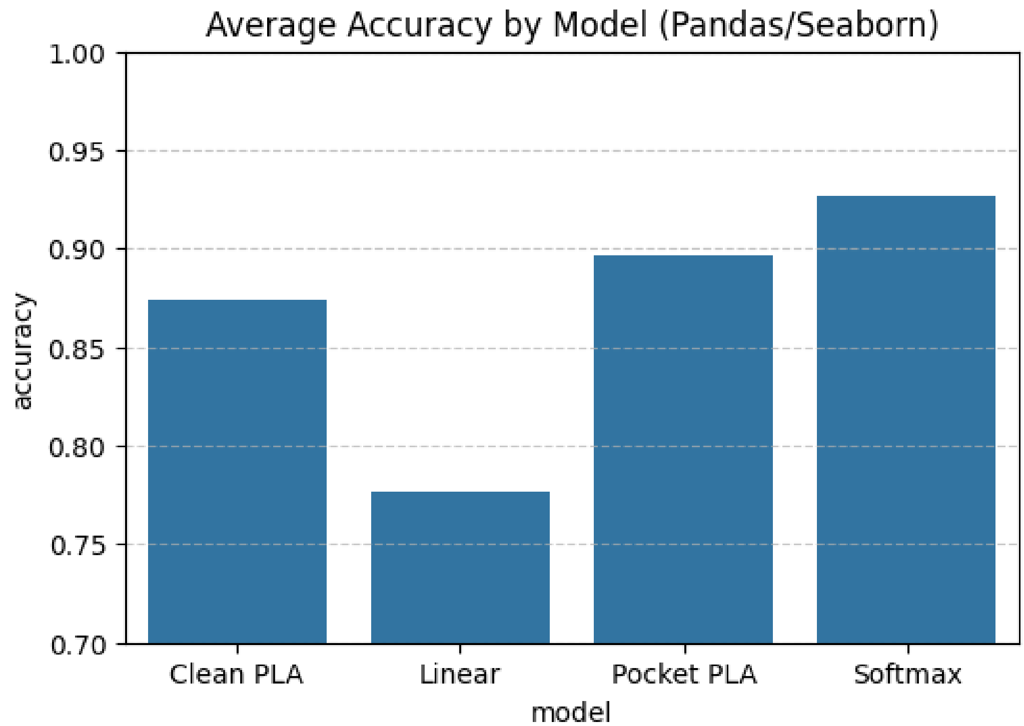


Linear LR=0.1, max_iter=1000: Train Loss vs. Iteration
train_loss
80
60
40
20
0
0
200
400
600
800
1000
iteration

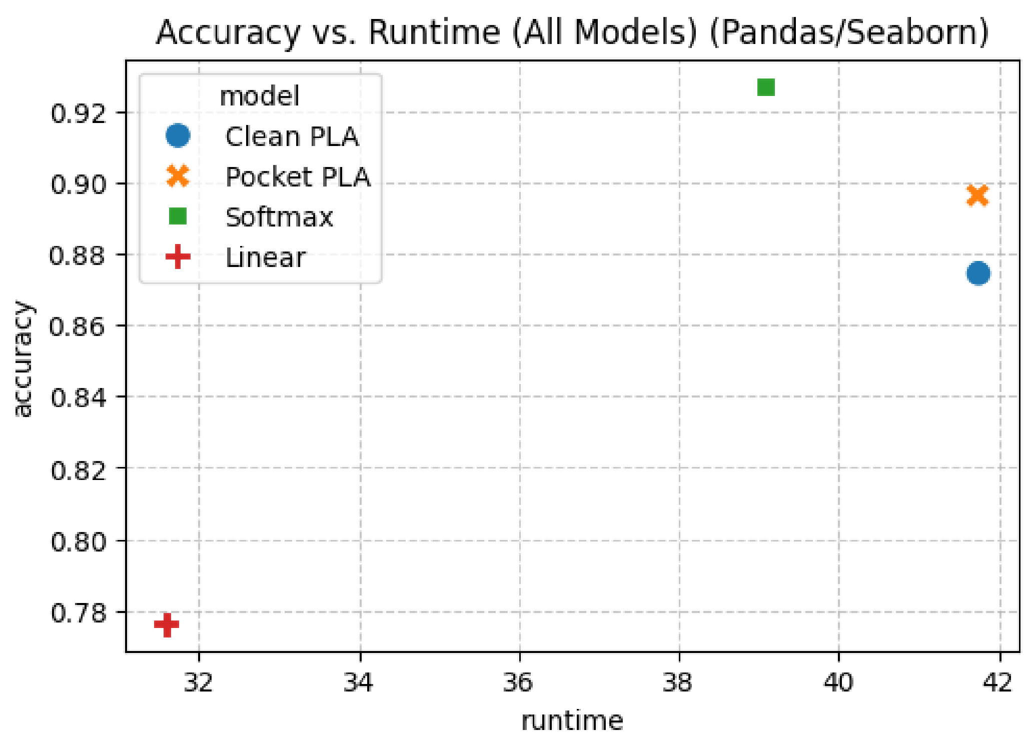



```
INFO - === Pandas + Seaborn Plots ===
```



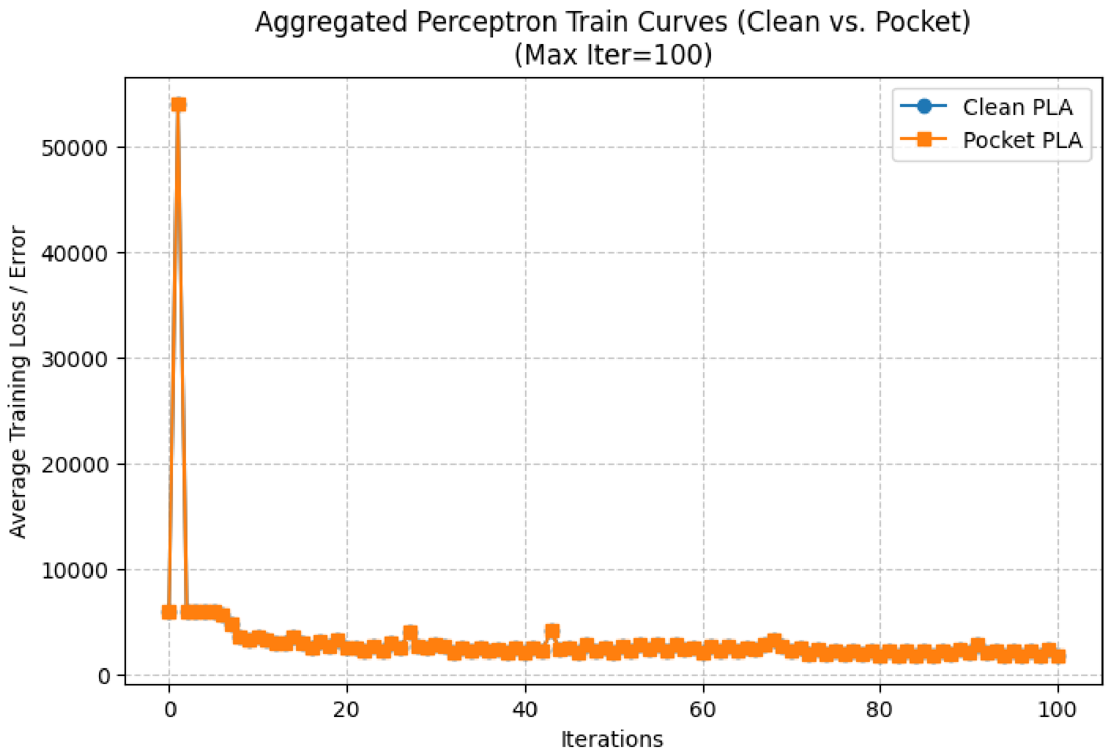


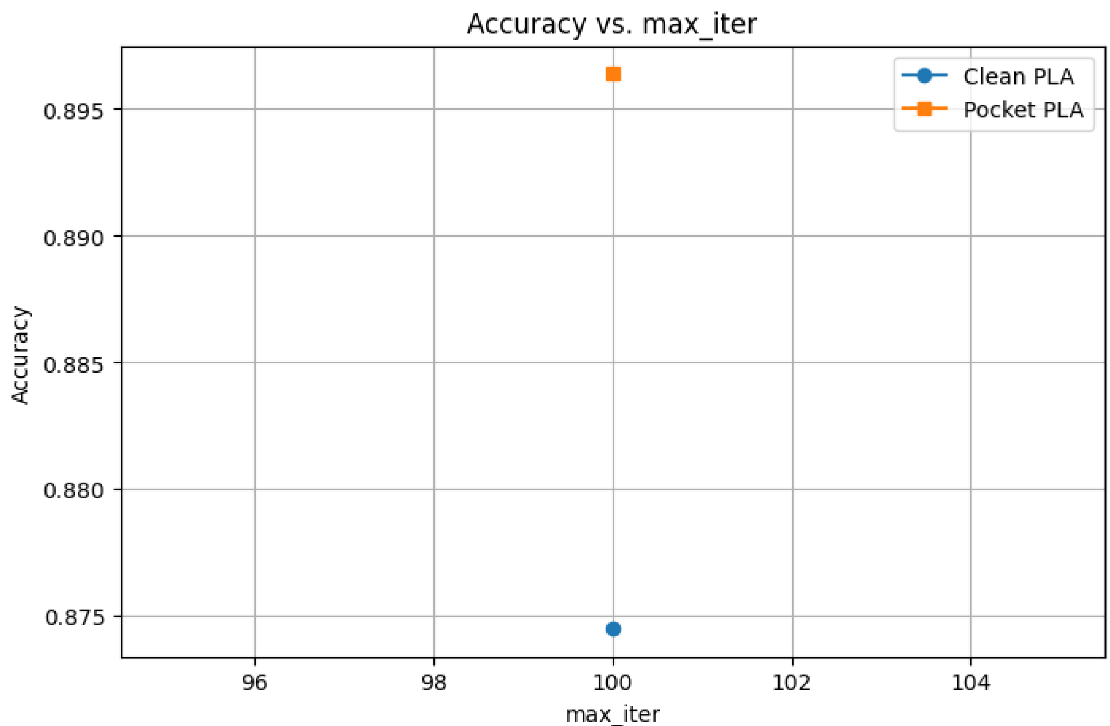
Average Accuracy by Model (Pandas/Seaborn)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
accuracy
Clean PLA
Linear
Pocket PLA
Softmax
model

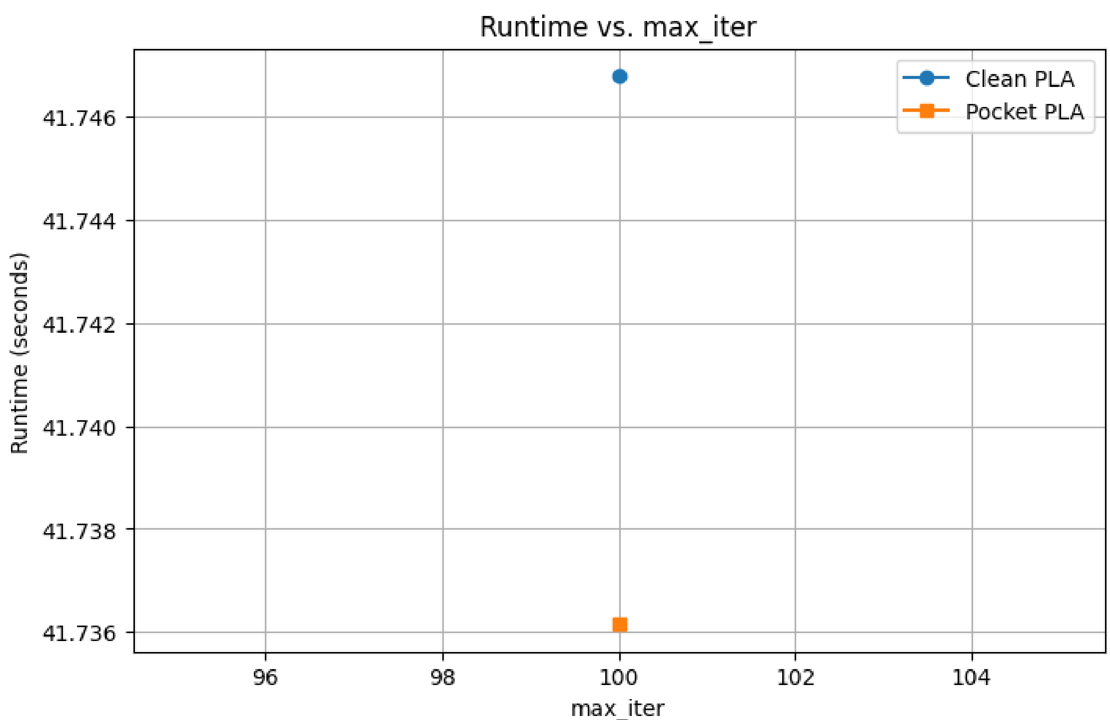


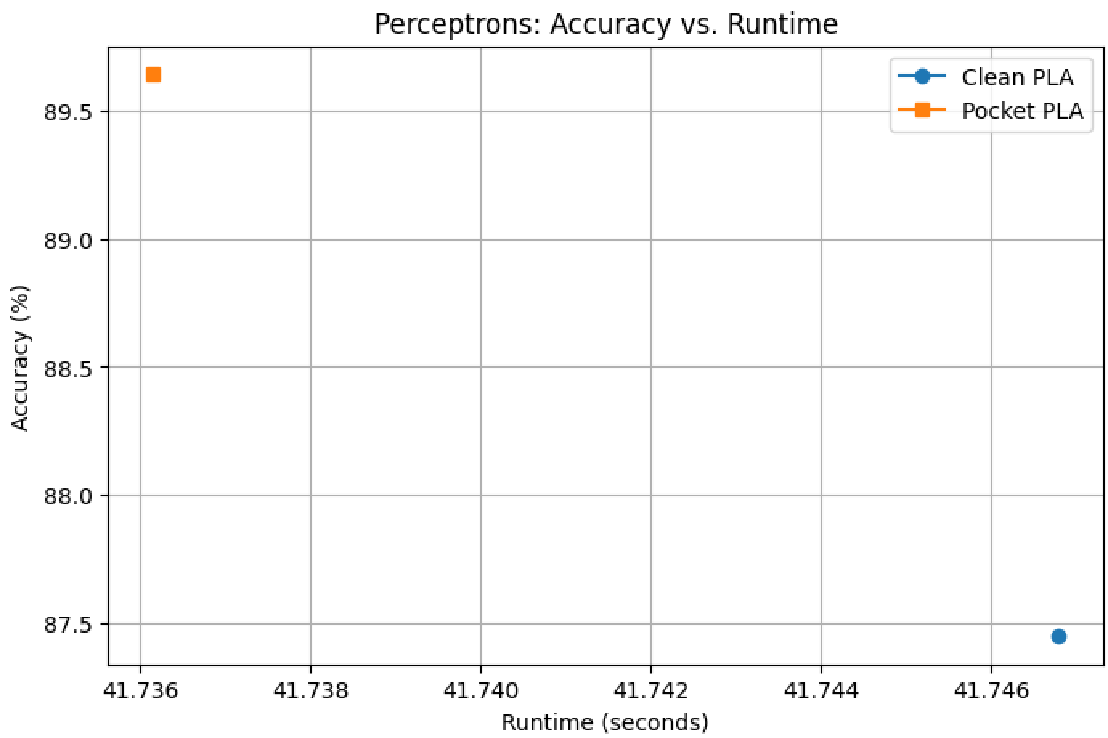
Accuracy vs. Runtime (All Models) (Pandas/Seaborn)
model
Clean PLA
Pocket PLA
Softmax
Linear
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
accuracy
32
34
36
38
40
42
runtime

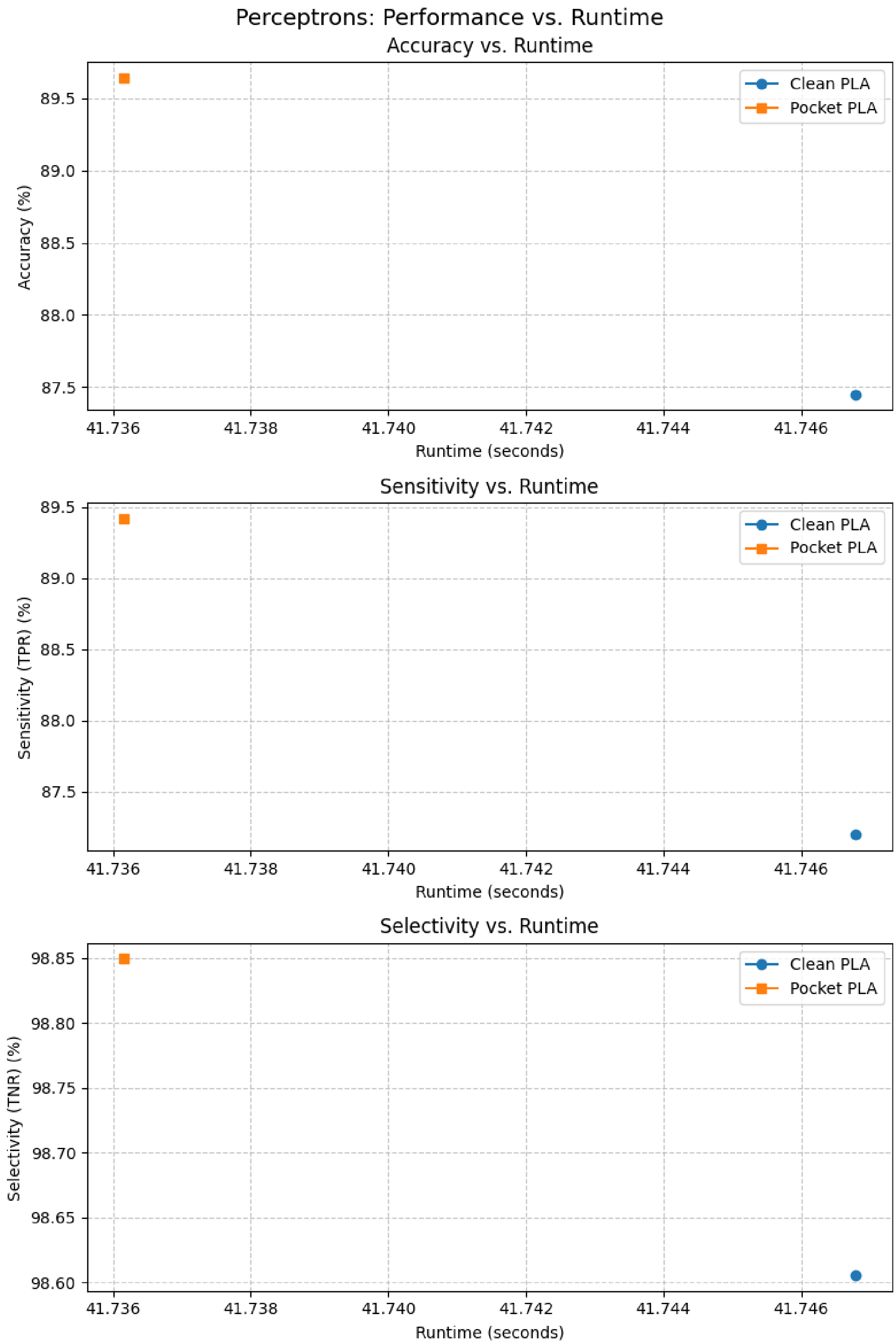
```
INFO - === Custom Summaries (Aggregated Curves, etc.) ===
```

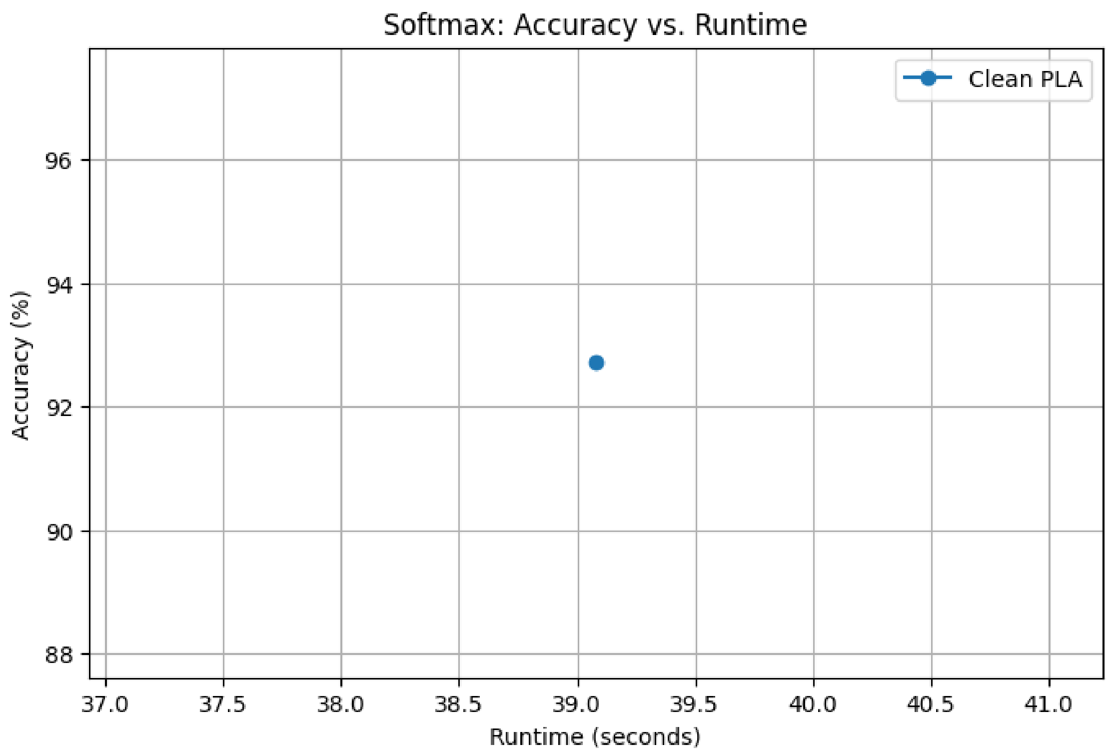


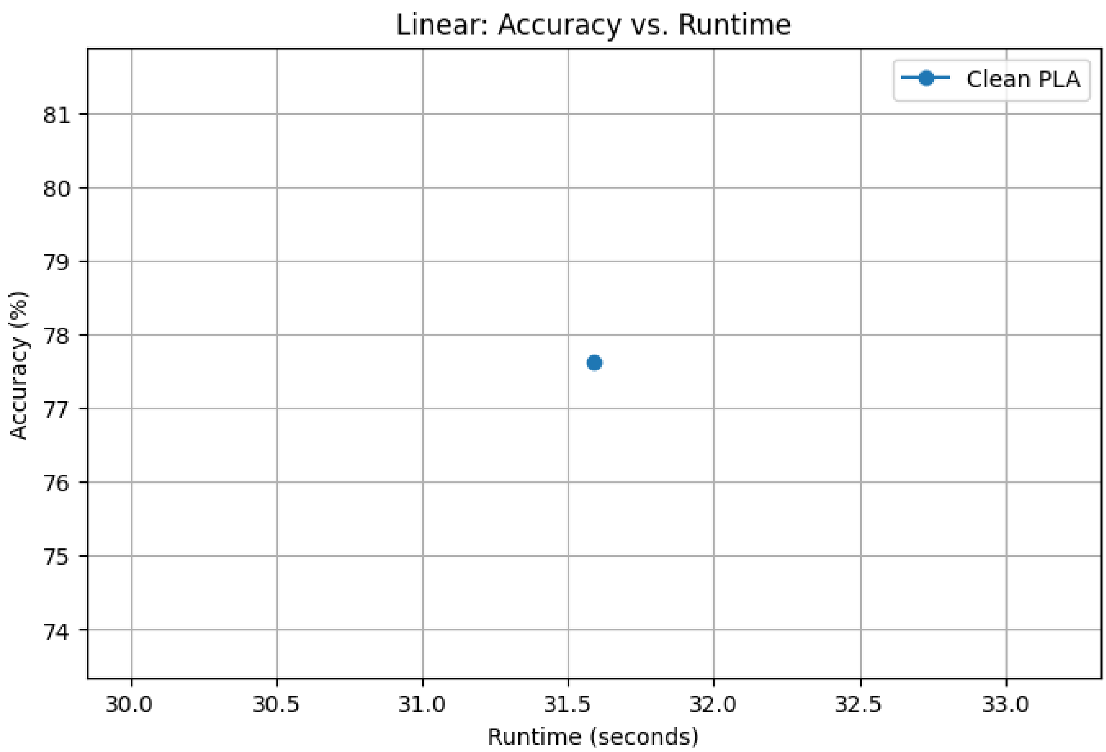

Accuracy vs. max_iter
Clean PLA
Pocket PLA
0.895
0.890
0.885
0.880
0.875
Accuracy
96
98
100
102
104
max_iter

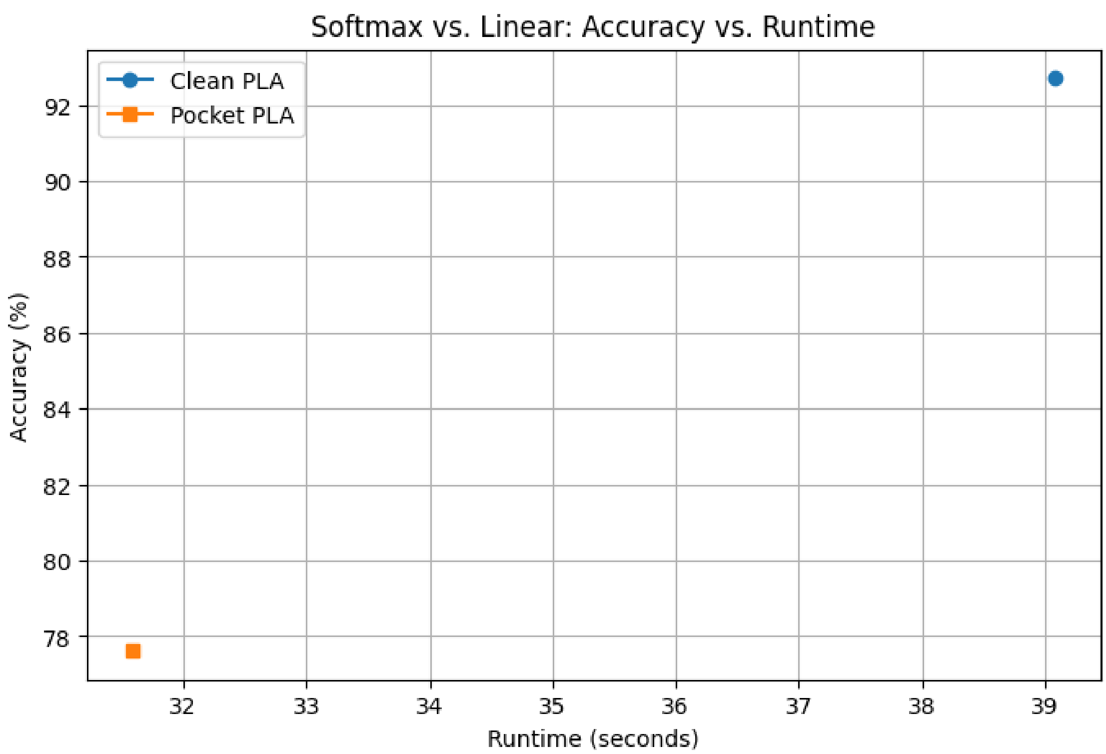

Runtime vs. max_iter
Clean PLA
Pocket PLA
41.746
41.744
41.742
41.740
41.738
41.736
Runtime (seconds)
96
98
100
102
104
max_iter

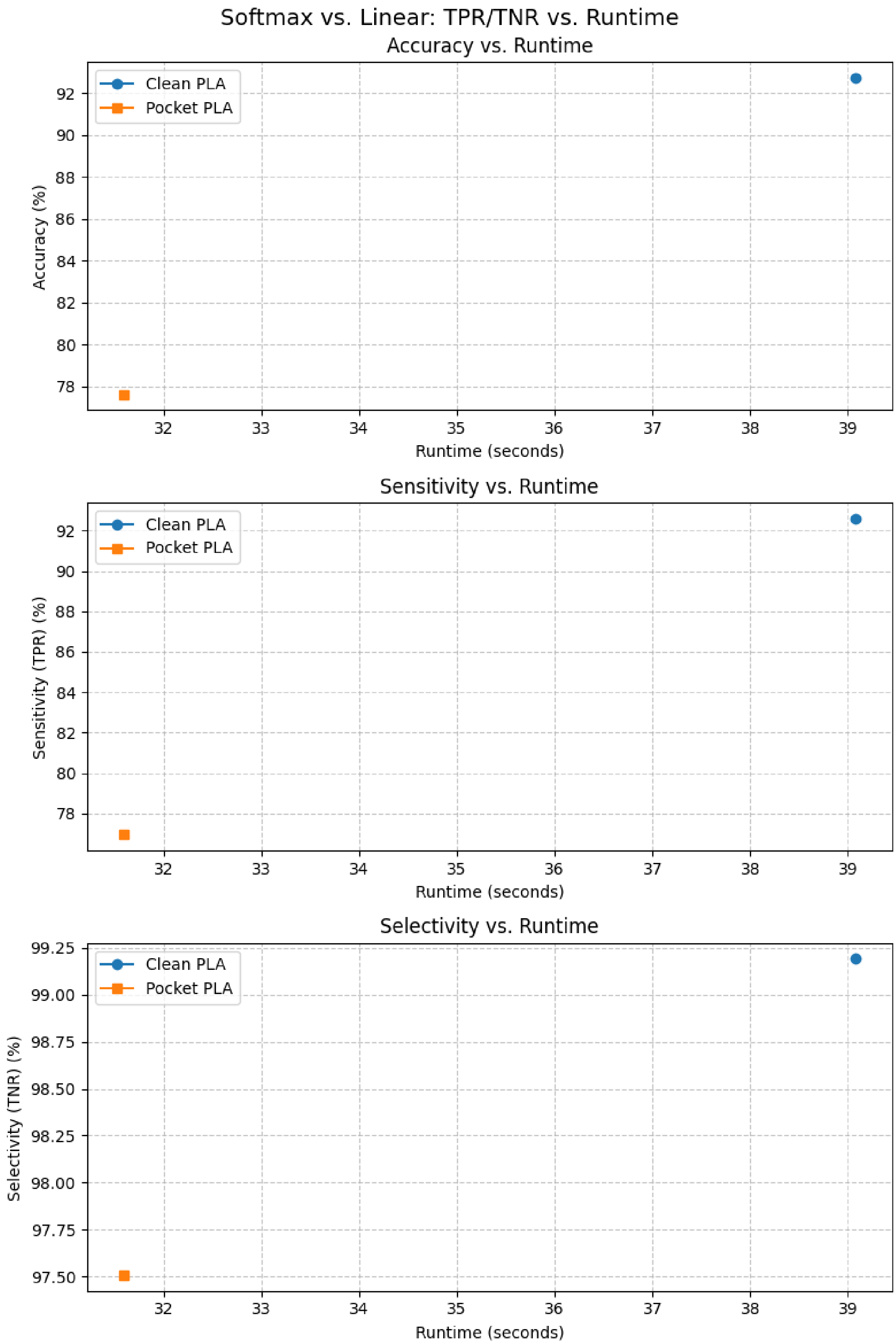

Perceptrons: Accuracy vs. Runtime
Clean PLA
Pocket PLA
89.5
89.0
88.5
88.0
87.5
Accuracy (%)
41.736
41.738
41.740
41.742
41.744
41.746
Runtime (seconds)

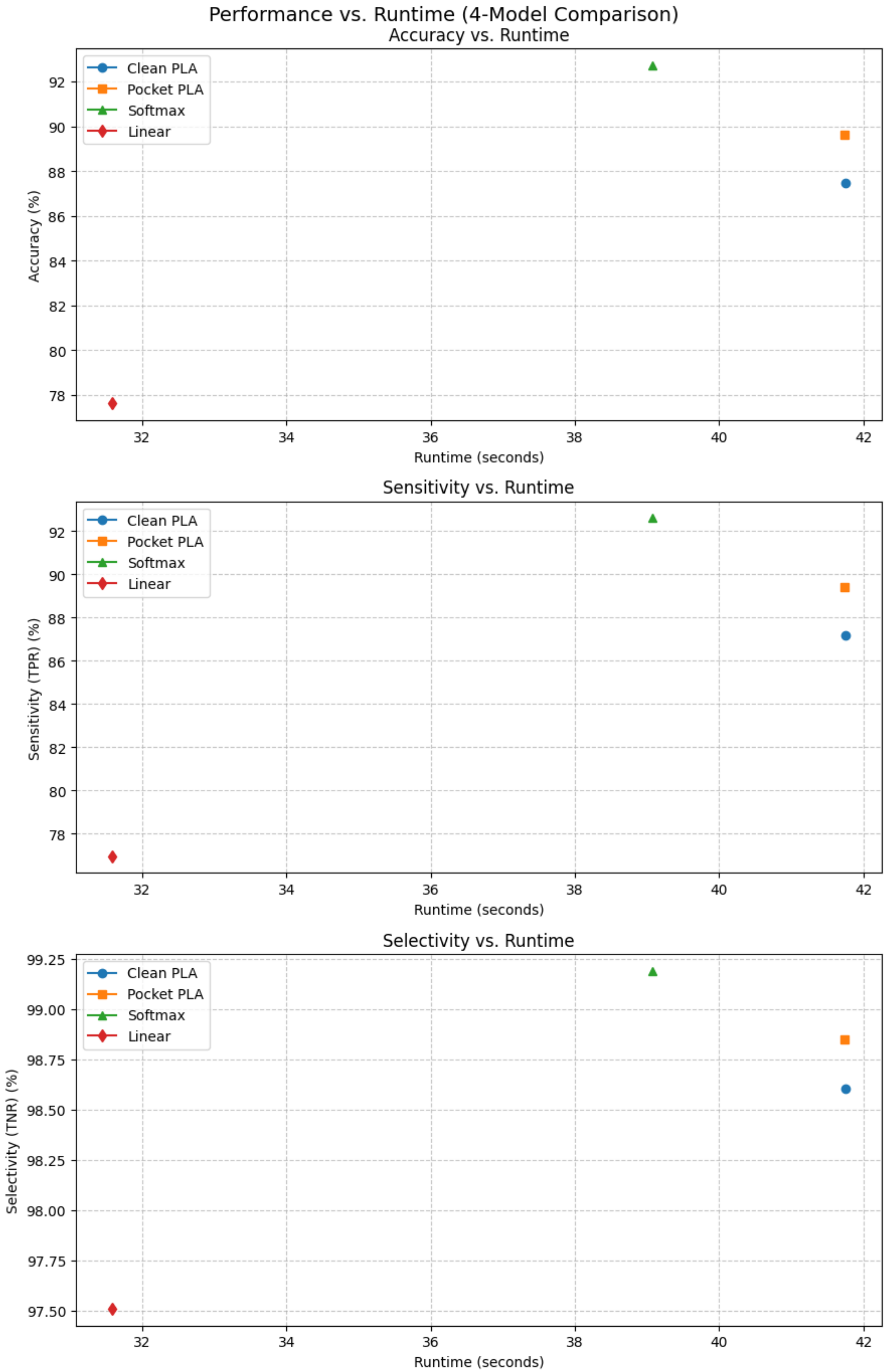

Perceptrons: Performance vs. Runtime
Accuracy vs. Runtime
Accuracy (%)
Runtime (seconds)
Sensitivity vs. Runtime
Sensitivity (TPR) (%)
Runtime (seconds)
Selectivity vs. Runtime
Selectivity (TNR) (%)
Runtime (seconds)
Clean PLA
Pocket PLA
41.736
41.738
41.740
41.742
41.744
41.746

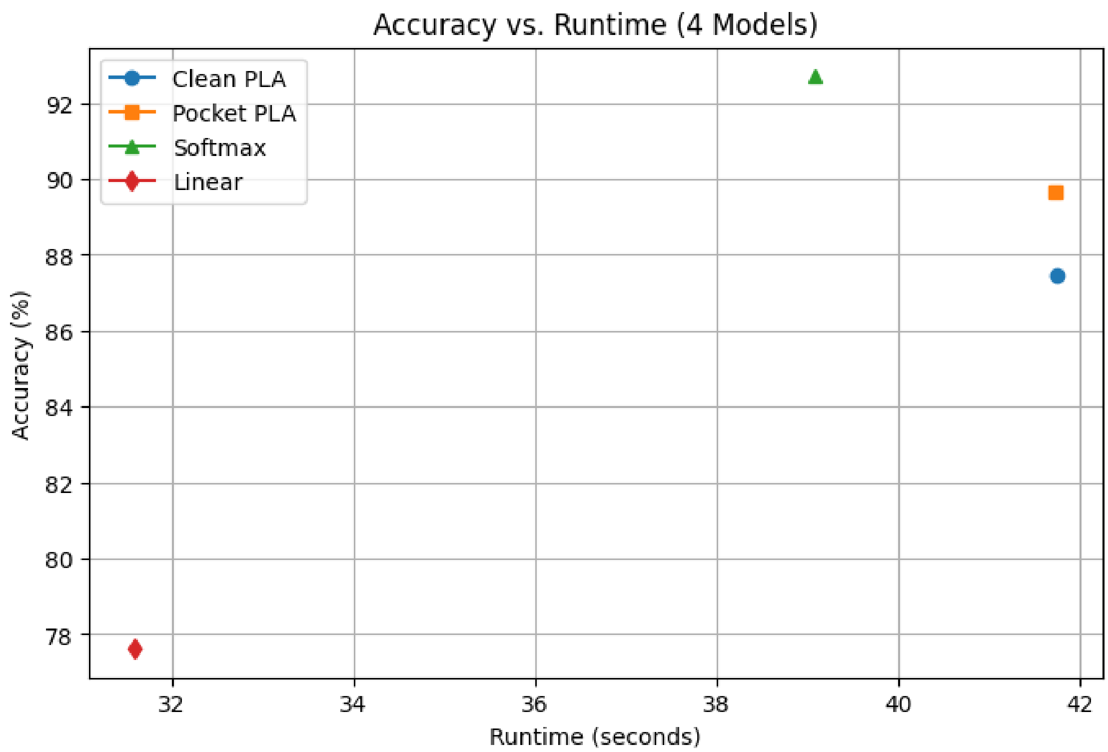

Softmax: Accuracy vs. Runtime
Clean PLA
Accuracy (%)
96
94
92
90
88
37.0
37.5
38.0
38.5
39.0
39.5
40.0
40.5
41.0
Runtime (seconds)


Linear: Accuracy vs. Runtime
Clean PLA
Accuracy (%)
81
80
79
78
77
76
75
74
30.0
30.5
31.0
31.5
32.0
32.5
33.0
Runtime (seconds)


Softmax vs. Linear: Accuracy vs. Runtime
Clean PLA
Pocket PLA
Accuracy (%)
92
90
88
86
84
82
80
78
32
33
34
35
36
37
38
39
Runtime (seconds)


Softmax vs. Linear: TPR/TNR vs. Runtime
Accuracy vs. Runtime
Clean PLA
Pocket PLA
Accuracy (%)
92
90
88
86
84
82
80
78
32
33
34
35
36
37
38
39
Runtime (seconds)
Sensitivity vs. Runtime
Clean PLA
Pocket PLA
Sensitivity (TPR) (%)
92
90
88
86
84
82
80
78
32
33
34
35
36
37
38
39
Runtime (seconds)
Selectivity vs. Runtime
Clean PLA
Pocket PLA
Selectivity (TNR) (%)
99.25
99.00
98.75
98.50
98.25
98.00
97.75
97.50
32
33
34
35
36
37
38
39
Runtime (seconds)


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




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