המכללה האקדמית להנדסה בראודה בכרמיאל

המחלקה להנדסת חשמל ואלקטרוניקה

מערכות לומדות ולמידה עמוקה (31245)

Lab 5 report

פרנסיס עבוד

Prepared for: Dr. Amer Adler

Date: 04/05/2025

Ex.A: Multinomial Logistic Regression (+Softmax)

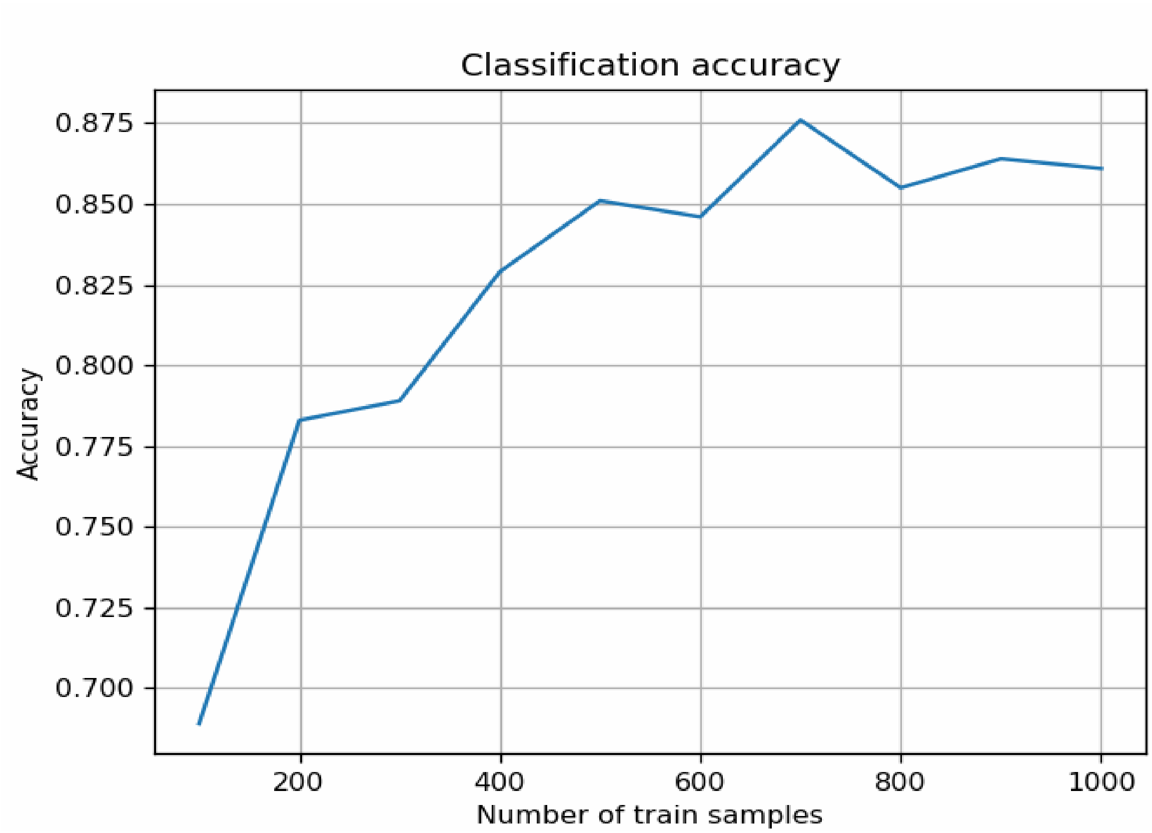
1) Plot the graph of classification accuracy versus trainset size: for S=100 to 1000 in steps of 100. Compute the accuracy only on the test set.

Code:

```python
# -*- coding: utf-8 -*-
"""
MNIST Digit Classification
"""

import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils import check_random_state

print(__doc__)

# Timer for performance measurement
start_time = time.time()

# Initialize variables
softmax_test_scores = np.zeros((10, 1))
softmax_cv_scores = np.zeros((10, 1))

# Load MNIST dataset
if 'X_data' not in globals():
    mnist_data = np.load('mnist.npz', allow_pickle=True)
    X_data = mnist_data['data']
    y_data = mnist_data['label']

# Shuffle the dataset
random_state = check_random_state(0)
shuffled_indices = random_state.permutation(X_data.shape[0])
X_data = X_data[shuffled_indices]
y_data = y_data[shuffled_indices]
X_data = X_data.reshape((X_data.shape[0], -1))

# A_1: Classification accuracy vs training set size
train_sizes = np.arange(100, 1100, 100)
test_accuracies = np.zeros(len(train_sizes))

for idx, size in enumerate(train_sizes):
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, train_size=size, test_size=1000, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train Logistic Regression (Softmax)
    softmax_model = LogisticRegression(multi_class='multinomial', penalty='l1', solver='saga', tol=0.01)
    softmax_model.fit(X_train, y_train)
    test_accuracies[idx] = softmax_model.score(X_test, y_test)

print("Accuracy on number of Test data: %\n", train_sizes, "\n", (test_accuracies * 100))
plt.figure(1)
plt.plot(train_sizes, test_accuracies, marker='o', label="Softmax Test Accuracy")
plt.grid()
plt.title("Classification Accuracy vs Training Set Size")
plt.xlabel("Number of Train Samples")
plt.ylabel("Accuracy")
plt.legend()
```

Output:

```
Accuracy on number of Test data: %
 [ 100  200  300  400  500  600  700  800  900 1000]
 [68.9 78.3 78.9 82.9 85.1 84.6 87.6 85.5 86.4 86.1]
```

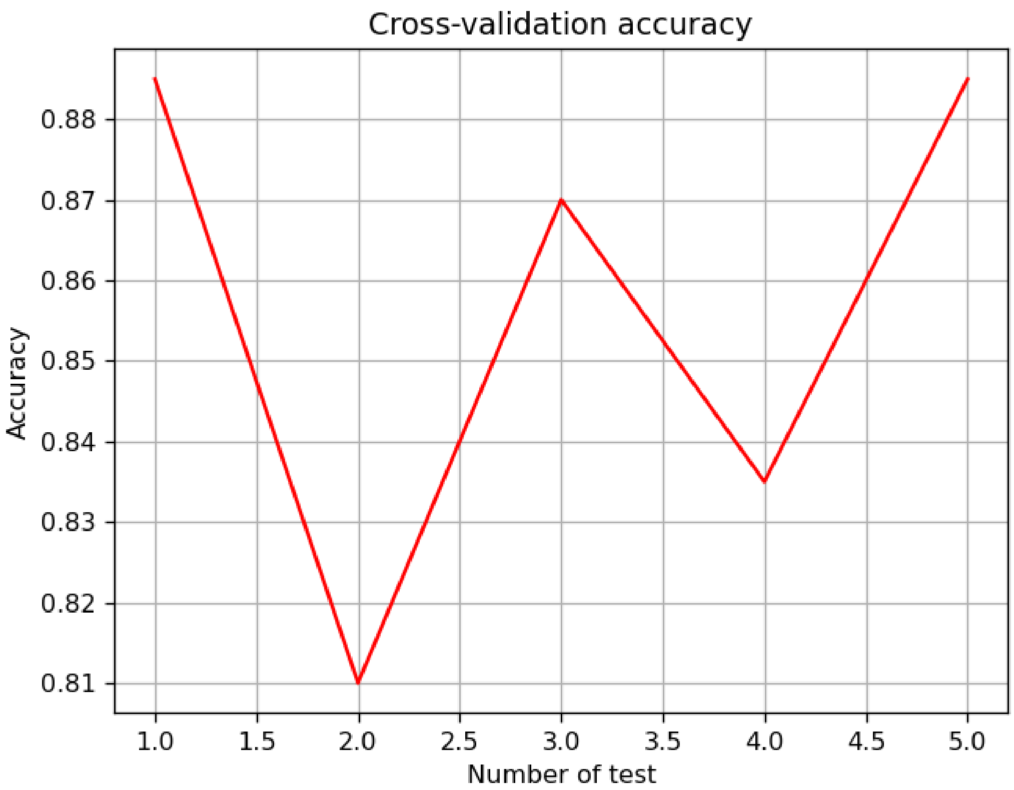


2) Compare to the same graph, but with accuracy computed with K-fold cross-validation (without testing set), and K=5.

Code:

```python
# A_2: K-fold cross-validation
cv_scores = cross_val_score(softmax_model, X_train, y_train, cv=5)
softmax_test_scores = test_accuracies
softmax_cv_scores = np.mean(cv_scores)
plt.figure(2)
plt.plot(np.arange(1, 6), cv_scores, 'r', marker='x', label="Cross-Validation Scores")
plt.grid()
plt.title("Cross-Validation Accuracy")
plt.xlabel("Fold Number")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
print("Accuracy of Cross-validation Test data: %\n", (cv_scores * 100), "\nAverage: % ", (softmax_cv_scores * 100))
```

Output:




```
Accuracy of Cross-validation Test data: %
 [88.5 81.   87.   83.5 88.5]
Average: %  85.7
```

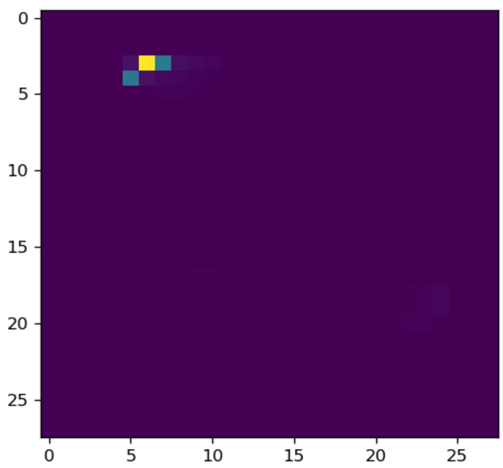
3) Display 5 wrong classified digits from the Softmax classifier, and their Softmax output probability vector.

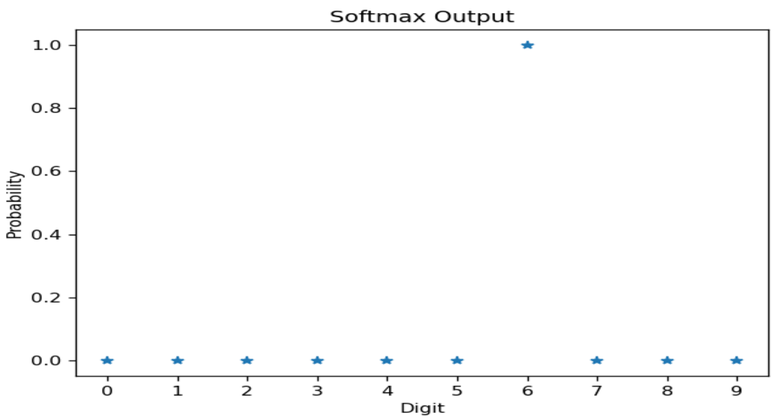
Code:

```python
# A_3: Display misclassified samples
predictions = softmax_model.predict(X_test)
misclassified = np.where(predictions != y_test)[0]
error_count = 0

for i in misclassified[:5]:
    print('Multinomial Regression Example:')
    plt.figure(figsize=(6, 6))

    # Display the misclassified digit
    plt.subplot(2, 1, 1)
    plt.imshow(X_test[i, :].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test[i]}, Predicted: {predictions[i]}")
    plt.axis('off')

    # Display the probability distribution
    softmax_output = softmax_model.predict_proba(X_test[i, :].reshape(1, -1)).flatten()
    plt.subplot(2, 1, 2)
    bars = plt.bar(np.arange(10), softmax_output, color='blue', alpha=0.7)
    plt.xticks(np.arange(10), ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9'))
    plt.title('Softmax Output Probabilities')
    plt.xlabel('Digit')
    plt.ylabel('Probability')

    # Annotate the bars with probability values
    for bar, prob in zip(bars, softmax_output):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f"{prob:.2f}",
                 ha='center', va='bottom', fontsize=8, color='black')

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

Output:

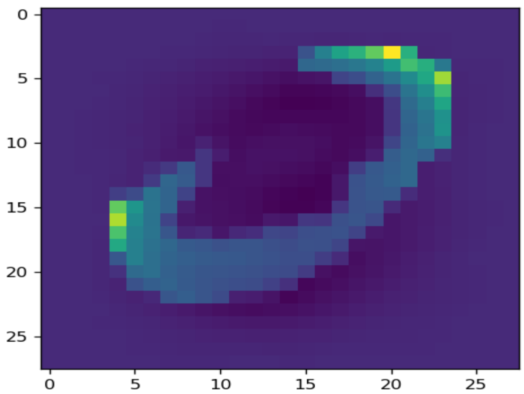
Error 1:

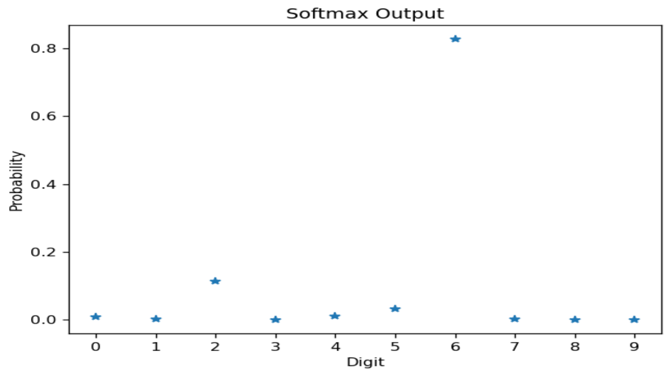





```
Multinomial Regression Example:
Correct Label: 2
Predicted Label: 6
```

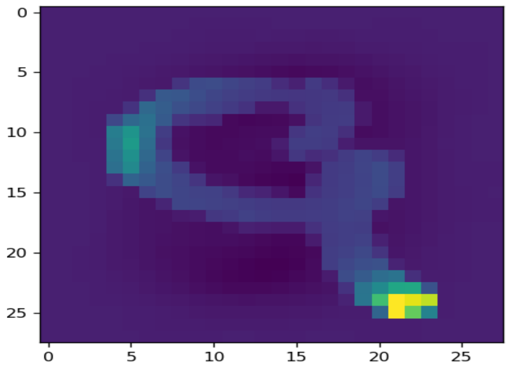
Error 2:

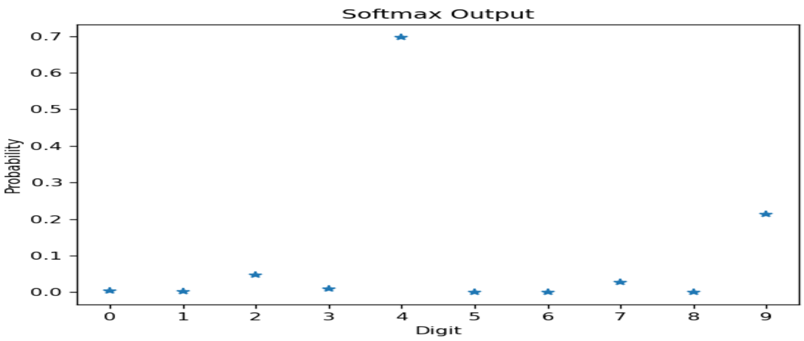





```
Multinomial Regression Example:
Correct Label: 0
Predicted Label: 6
```

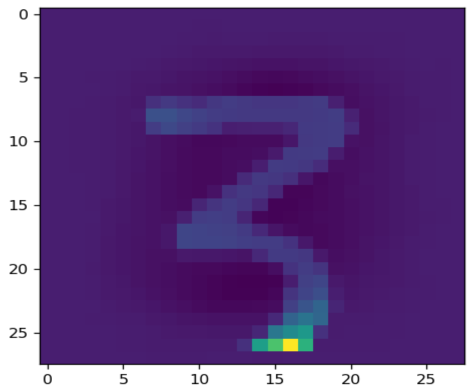
Error 3:

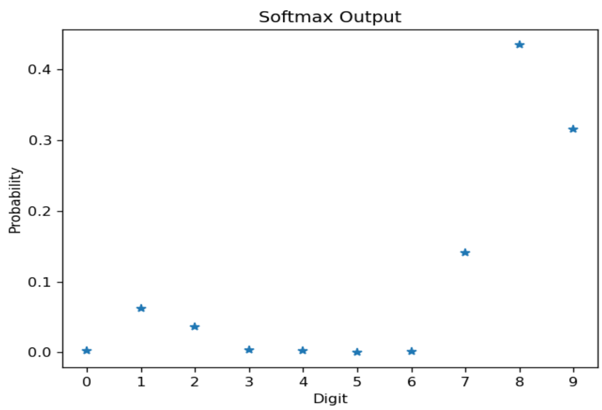





```
Multinomial Regression Example:
Correct Label: 9
Predicted Label: 4
```

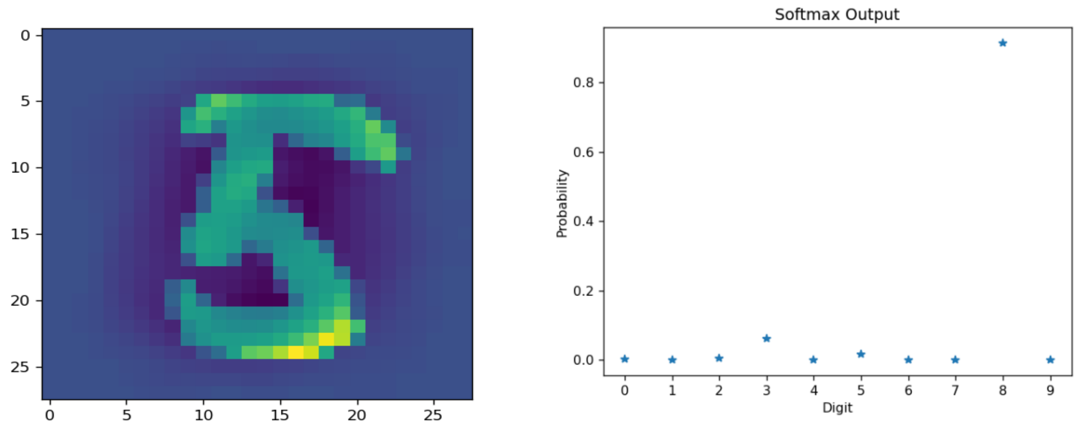
Error 4:






```
Multinomial Regression Example:
Correct Label: 3
Predicted Label: 8
```

Error 5:




```
Multinomial Regression Example:
Correct Label:  5
Predicted Label:  8
```

<u>Ex.B:</u> <u>K-NN</u>

4) Train a K-NN classifier, using 3 different values of K between 3 to 10 and choose the K which gives the best accuracy for S=500.

<u>Code:</u>

```python
# B_4: K-NN classifier with different K values
X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X_data, y_data, train_size=500, test_size=1000, random_state=42)
knn_scores = np.zeros(3)
k_values = np.random.randint(3, 11, 3)

for idx, k in enumerate(k_values):
    knn_model = KNeighborsClassifier(n_neighbors=k, weights='uniform')
    knn_model.fit(X_train_knn, y_train_knn)
    knn_scores[idx] = knn_model.score(X_train_knn, y_train_knn)

best_k = k_values[np.argmax(knn_scores)]
print("The KNN results: \n", k_values, "\n", (knn_scores * 100), "\n Maximum for k = ", best_k)
```

<u>Output:</u>

```
The KNN results:
 [9 3 7]
 [84.8 91.  86.2]
 Maximum for k =  3
```

5) Plot the graph of K-NN classification accuracy (with K as chosen in task 4) versus trainset size: for S=100 to 1000 in steps of 100. Compute the accuracy only on the test set. Draw also the graph of task (1) on the same plot.

<u>Code:</u>

```python
# B_5: K-NN accuracy vs training set size
knn_train_sizes = np.arange(100, 1100, 100)
knn_accuracies = np.zeros((3, len(knn_train_sizes)))

for idx1, k in enumerate(k_values):
    for idx2, size in enumerate(knn_train_sizes):
        X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X_data, y_data, train_size=size, test_size=1000, random_state=42)
        knn_model = KNeighborsClassifier(n_neighbors=k, weights='uniform')
        knn_model.fit(X_train_knn, y_train_knn)
        knn_accuracies[idx1, idx2] = knn_model.score(X_train_knn, y_train_knn)
    plt.plot(knn_train_sizes, knn_accuracies[idx1, :], label=f"K={k}", marker='s')

plt.plot(train_sizes, test_accuracies, label="Softmax", linestyle='--', color='black')
plt.grid()
plt.title("Classification Accuracy vs Training Set Size")
plt.xlabel("Number of Train Samples")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

<u>Output:</u>