CS412: Machine Learning Homework 1 Report

Name: Ozan Kaçmaz ID: 32123

Jupyter Notebook Link:

https://github.com/kacmazozan/Fashion_mnist_Dataset_ML/blob/main/CS412-HW1-ozankacmaz.ipynb

1. A Clear Overview of Methodology

This report details the process of training, evaluating, and analyzing a k-Nearest Neighbors (k-NN) classifier on the Fashion-MNIST dataset. The methodology follows a structured approach including data analysis, preprocessing, hyperparameter tuning, final model evaluation, and error analysis to understand the model's behavior and performance.

1.1. Data Analysis

The Fashion-MNIST dataset consists of 60,000 training and 10,000 testing images of 10 different clothing categories. Each image is a 28x28 grayscale image.

An initial exploratory analysis was performed to understand the dataset's characteristics:

- **Class Distribution:** The dataset is well-balanced, with each of the 10 classes having an equal number of samples (6,000 in the first training set, then it will be 4800 after the train-validation split.)
- **Pixel Statistics:** The global mean pixel value was calculated to be approximately **73.0**, with a global standard deviation of **90.1**. This indicates a wide range of pixel intensities. Per-class mean pixel intensities were also calculated, revealing that classes like 'Trouser' and 'Sandal' tend to be darker (lower mean intensity) on average than classes like 'Pullover' and 'Coat'.
- **Data Visualization:** To visually inspect the data, one random sample from each of the 10 classes was displayed.

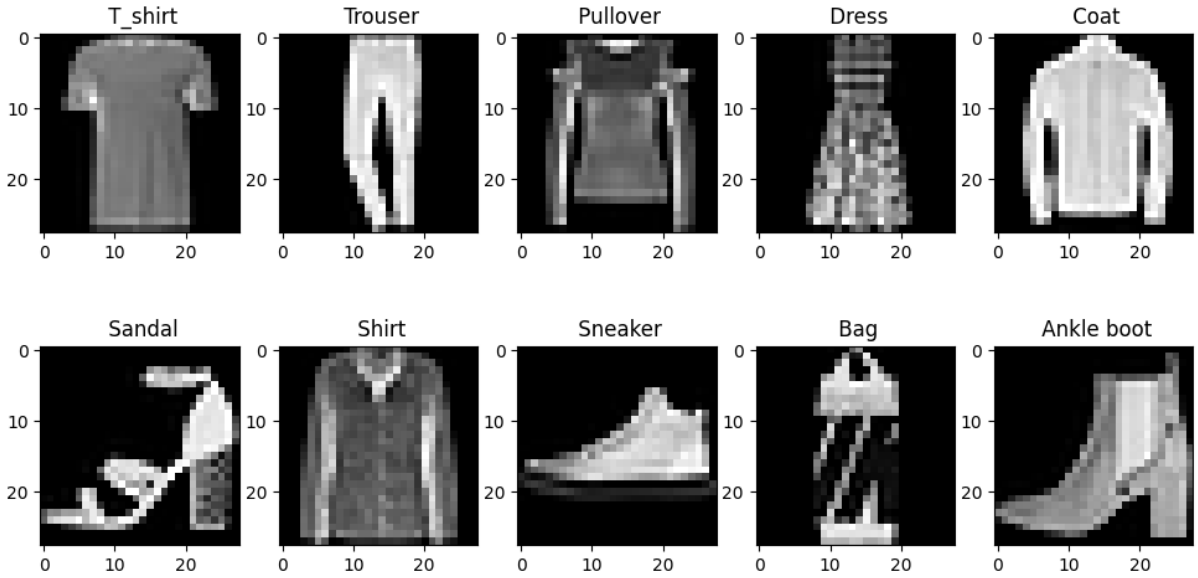

Figure 1: A sample image from each of the 10 classes in the Fashion-MNIST dataset.

1.2. Data Preprocessing

Before training the model, the data underwent two crucial preprocessing steps:

1. **Train-Validation-Test Split:** The initial 60,000 training samples were split into a training set (80%, 48,000 samples) and a validation set (20%, 12,000 samples). This split was performed with shuffling and stratification to ensure both sets were representative of the overall data distribution. A `random_state` of 42 was used for reproducibility. The 10,000-sample test set was kept separate and untouched until the final model evaluation.
2. **Data Normalization (Scaling):** Distance-based algorithms like k-NN are sensitive to the scale of features. To ensure each pixel contributes equally to the distance calculations, `StandardScaler` was used. The scaler was **fit only on the training data** to learn its mean and standard deviation. This same fitted scaler was then used to **transform the training, validation, and test sets**. This critical step prevents **data leakage**, where information from the validation or test sets could influence the training process, leading to artificially inflated performance metrics. After scaling, the training data's mean was confirmed to be approximately 0 and its standard deviation was approximately 1.
3. **Data Reshaping:** The 28x28 image matrices were flattened into 784-element vectors to be compatible with the scikit-learn k-NN classifier.

2. Justification for Hyperparameter and Distance Metric Choices

The goal of hyperparameter tuning was to find the optimal combination of `k` (the number of neighbors) and the distance metric for the k-NN model. The following were tested:

- **Number of Neighbors (`k`):** {1, 3, 5, 7}
- **Distance Metrics:** {'euclidean', 'manhattan'}

Each combination was trained on the scaled training data and evaluated on the scaled validation data. The validation accuracy for each combination was recorded to produce the following performance curve.

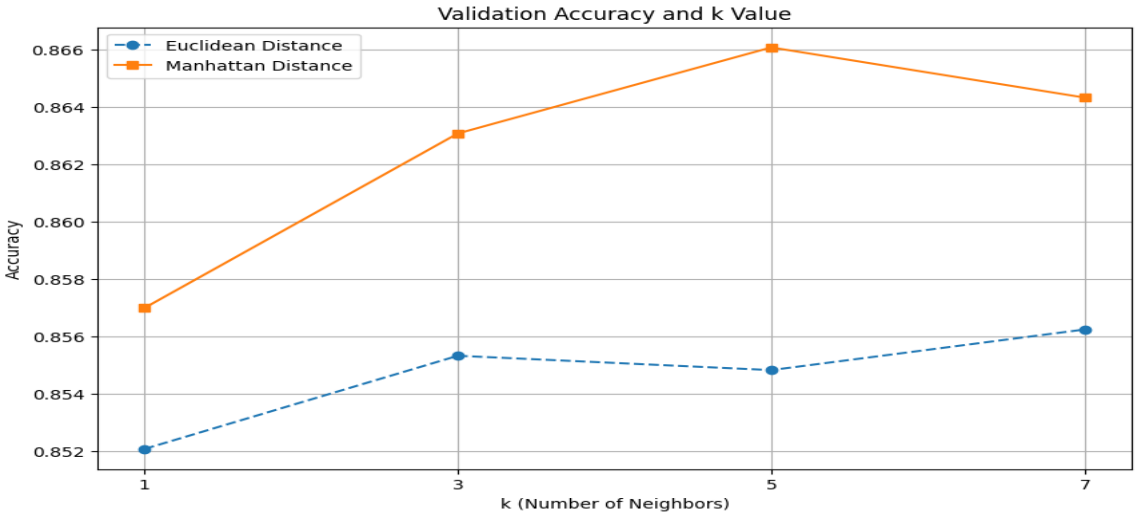

Figure 2: Validation accuracy as a function of k for both Euclidean and Manhattan distance metrics.

Justification: As seen in Figure 2, the model's performance varied with both `k` and the distance metric. The highest validation accuracy of 0.8660 is achieved with k set to 5 and Manhattan used as distance metric.

```
k_values =[1,3,5,7]
best_k = None
best_accuracy = 0
best_metric = ""
accuracies={}

metrics=["euclidean","manhattan"]
for m in metrics:
    accuracies[m] =[]
    for k in k_values:
        knn= KNeighborsClassifier(n_neighbors=k,metric=m)
        knn.fit(x_train_scaled,y_train_full)

        y_val_predict = knn.predict(x_val_scaled)
        accuracy_val = accuracy_score(y_val_full,y_val_predict)
        accuracies[m].append(accuracy_val)
        print(f"Validation accuracy using the metric as \"{m}\" and k={k} is {accuracy_val}")
        if accuracy_val > best_accuracy:
            best_accuracy = accuracy_val
            best_k = k
            best_metric = m
```

```
[23]  ✓ 2m 52.4s                                                    Python

Validation accuracy using the metric as "euclidean" and k=1 is 0.8520833333333333
Validation accuracy using the metric as "euclidean" and k=3 is 0.8553333333333333
Validation accuracy using the metric as "euclidean" and k=5 is 0.8548333333333333
Validation accuracy using the metric as "euclidean" and k=7 is 0.85625
Validation accuracy using the metric as "manhattan" and k=1 is 0.857
Validation accuracy using the metric as "manhattan" and k=3 is 0.8630833333333333
Validation accuracy using the metric as "manhattan" and k=5 is 0.8660833333333333
Validation accuracy using the metric as "manhattan" and k=7 is 0.8643333333333333
```

This combination was therefore selected for the final model, as it demonstrated the best ability to generalize to unseen data during the validation phase.

3. Comprehensive Test Results

Using the best hyperparameters found (`k=5`, `metric='manhattan'`), the final model was retrained on the concatenated training and validation sets (a total of 60,000 samples). This allows the model to learn from the maximum amount of available data before final evaluation. The model was then evaluated on the unseen test set.

- **Overall Accuracy:** The final model achieved an overall accuracy of 0.8613 In other words 86.13% on the test set.
- **Precision, Recall, and F1-score:** The detailed performance per class, including the required macro-averaged scores, is shown in the classification report below. The macro-averaged F1-score was 0.86.

```
...   Overall Test Accuracy: 0.8613

      Classification Report:
                    precision    recall  f1-score   support

          T_shirt       0.77      0.86      0.81      1000
          Trouser       0.99      0.97      0.98      1000
         Pullover       0.73      0.80      0.76      1000
            Dress       0.90      0.88      0.89      1000
             Coat       0.77      0.77      0.77      1000
           Sandal       0.99      0.90      0.94      1000
            Shirt       0.66      0.58      0.61      1000
          Sneaker       0.91      0.96      0.94      1000
              Bag       0.98      0.94      0.96      1000
       Ankle boot       0.93      0.97      0.95      1000

         accuracy                           0.86     10000
        macro avg       0.86      0.86      0.86     10000
     weighted avg       0.86      0.86      0.86     10000
```

- **Confusion Matrix and Discussion:**

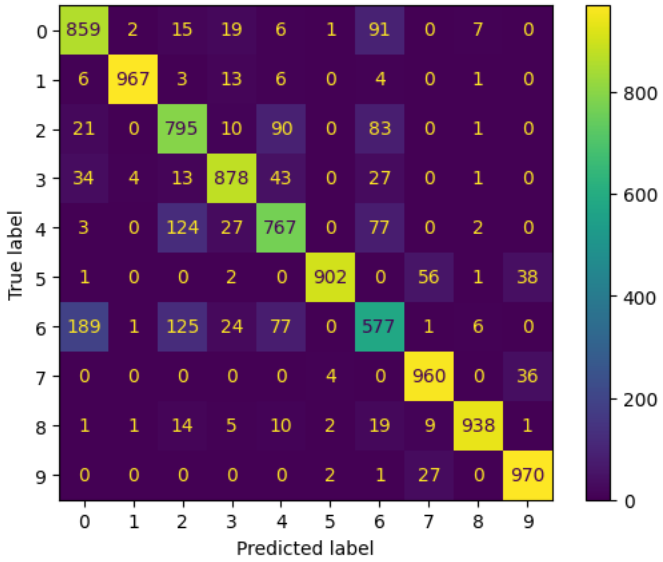

Figure 3: Confusion matrix on the test set for the final k-NN model.

The confusion matrix in Figure 3 reveals the model's strengths and weaknesses. The high values along the diagonal indicate a high number of correct predictions for most classes.

- **Well-Classified Classes:** The model performs exceptionally well on classes with distinct silhouettes, such as 'Trouser', 'Bag', and 'Sandal', achieving high precision and recall.
- **Poorly-Classified Classes:** The most significant confusion occurs between classes that are visually similar. The matrix shows a high number of misclassifications

between 'Shirt' and 'T-shirt', as well as between 'Pullover' and 'Coat'. This suggests the model struggles to differentiate between these upper-body garments based on the low-resolution images.

4. Error Analysis

To further investigate the model's failures, the top 3 most confused class pairs were identified from the confusion matrix.

1. True: 'Shirt' vs. Predicted: 'T-shirt'

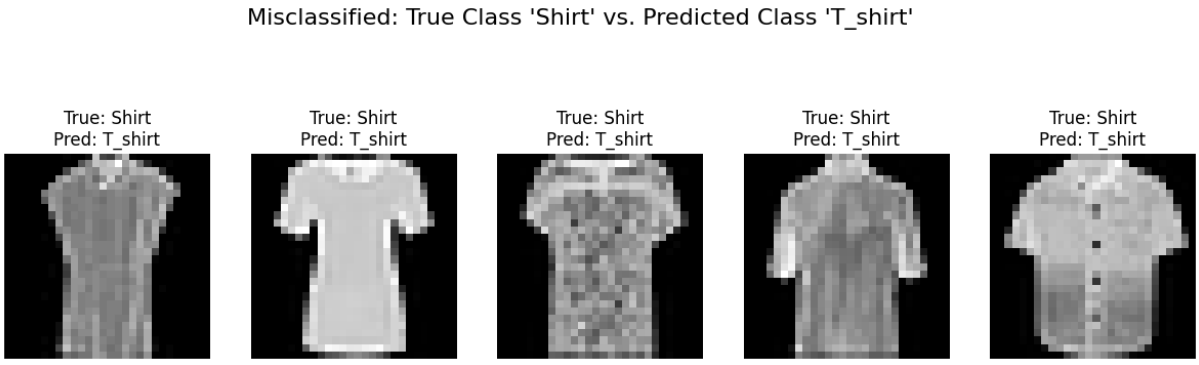

Figure 4: Examples of 'Shirt' class items misclassified as 'T-shirt'.

Possible reason: The visual similarity between these classes is the primary cause of confusion. In 28x28 grayscale images, defining features like collars or buttons on a 'Shirt' are often blurred or lost, making its silhouette nearly identical to that of a 'T-shirt'.

2. True: 'Coat' vs. Predicted: 'Pullover'

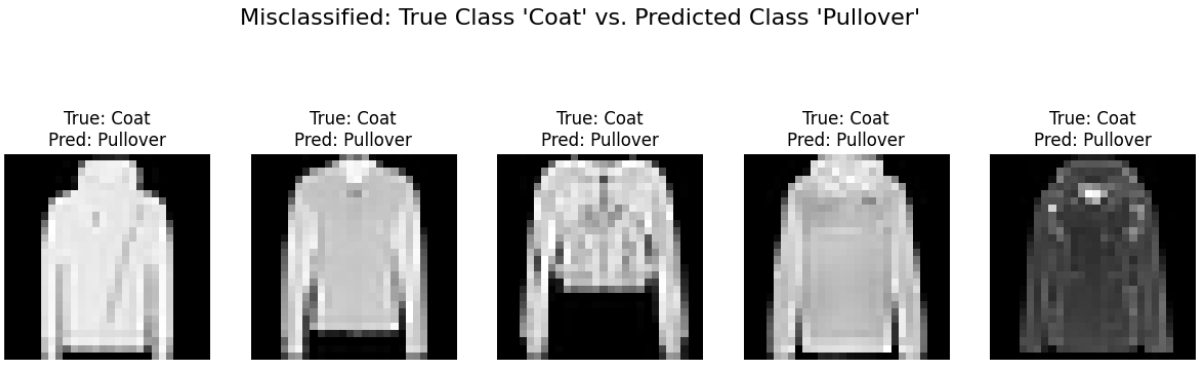

Figure 5: Examples of 'Pullover' class items misclassified as 'Coat'.

Possible reason: Both pullovers and coats are long-sleeved outer garments. The model struggles to distinguish them, likely because the key differences (material thickness, presence of a lapel) are not easily discernible from the shape alone. A dark, thick pullover can appear very similar to a coat.

3. True: 'Shirt' vs. Predicted: 'Pullover'

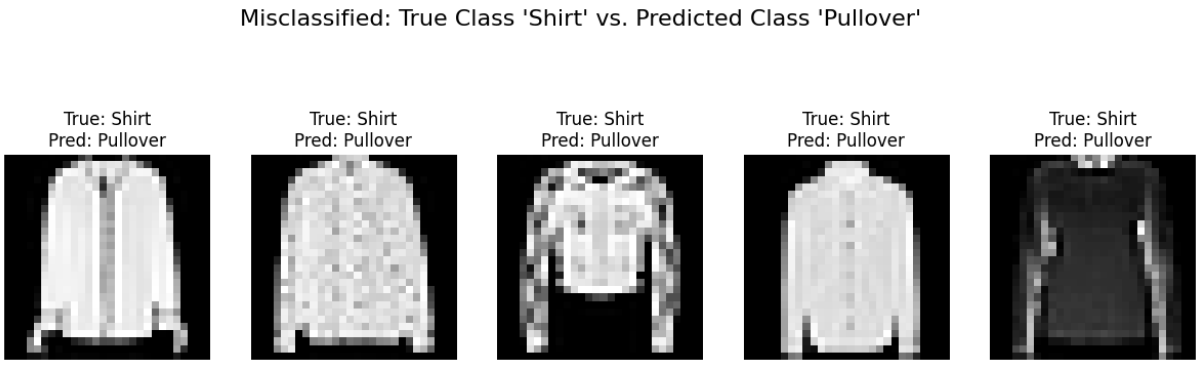

Figure 6: Examples of 'Shirt' class items misclassified as 'Pullover'.

Possible reason: The 'Shirt' class is arguably the most ambiguous in the dataset and is frequently confused with other upper-body garments. In this case, it is misclassified as 'Pullover'. Both are long-sleeved tops, and the model fails to capture the fine-grained details like a collar or button placket that would define a 'Shirt'. These features are often distorted or lost in the low-resolution 28x28 images, causing the model to default to a simpler classification.

5. Computational Analysis and Trade-offs

The training and prediction times were measured to analyze the computational characteristics of the k-NN model.

- **Training (Fit) Time: 0.0118 seconds**
- **Prediction (Predict) Time on Test Set: 47.5246 seconds**

Possible reason: The results clearly demonstrate the nature of k-NN as a **"lazy learner"**. The training time is extremely short because the `.fit()` method does not build a complex model; it simply stores the entire training dataset in memory. It does not produce coefficients unlike in regression models. This is a fast and simple process.

On the contrary, the prediction time is significantly longer. During the `.predict()` phase, the model must perform the computationally expensive task of calculating the distance from each test point to every single point in the stored training data (60,000 samples in this case) to find the nearest neighbors.

This presents a clear trade-off: k-NN has a virtually non-existent training cost, but it pays for this "laziness" with a high computational cost at prediction time. This makes it unsuitable for real-time applications with large datasets, but effective for smaller problems where training time is a major constraint.