DMML Assignment 3

Arka Roy

April 2024

1 Introduction

This report aims to explain the various details of the models and methodology used for the tasks assigned. The task required us to do clustering for semi-supervised learning on the **fashion MNIST** dataset. We have to use K-Means clustering to identify a small subset of labelled images to seed the classification process.We have also carried out a similar experiment for overhead MNIST dataset.The MNIST example started with 50 clusters. Here we will be experimenting with different (relatively small) values of K for these two datasets.

2 The Data Set

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST serves as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits. There are 8 classes namely **T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.**

3 Data Pre-processing

The Fashion MNIST dataset is loaded and then split into training, validation, and test sets. The main training set (X-train, y-train) will be used for model training. The validation set (X-valid, y-valid) helps monitor the model's performance during training to adjust hyperparameters or prevent overfitting. The test set (X-test, y-test) serves as a final evaluation to measure the model's accuracy and generalization ability. We have selected the first 55000 elements for the training set and the last 5000 elements for the validation set. We then checked for unique counts for each dataset and found out that every unique label has 6000 values in the training set and 1000 values in the test set.

4 Functions for Seed classification process using k-means clustering

4.1 Full Cluster Labeling

The function `seed_classification_full_clusters` assigns a label to each data point based on the closest cluster center. The labels for each cluster are determined by finding the point closest to its center, then assigning that point's original label to the entire cluster.

4.2 Partial Data Labeling

The function `seed_classification_part` returns a subset of the training set. The subset comprises a certain percentage of the closest points in each cluster, where the percentage is specified by the `per` argument. The label for each cluster is determined similarly to the previous function.

4.3 Cluster Centers Labeling

The function `seed_classification_centers` returns the cluster centers and assigns a label to each center based on the point closest to it. This approach provides a reduced dataset, useful for simplified visualization or other tasks.

We have taken K=20

5 Model creation using sequential API

Here we create a new Keras Sequential model. This type of model is a linear stack of layers, meaning each layer has a single input and a single output, arranged in a sequential manner. It's suitable for simpler models or when there's a clear feed-forward structure.

5.1 Input Layer

The input layer is defined with a shape of [28, 28], indicating that the model expects 28x28 grayscale images. This shape aligns with the structure of the Fashion MNIST dataset, commonly used for testing and training image classification models.

5.2 Flattening Layer

Following the input layer, the data is flattened using a `Flatten()` layer, converting the 2D input into a 1D vector of 784 elements. This transformation is a common step when transitioning from image data to fully connected layers. This layer has 0 parameters

5.3 First Dense Layer

The first dense (fully connected) layer contains 300 neurons with a ReLU (Rectified Linear Unit) activation function. ReLU is popular for hidden layers due to its simplicity and efficiency. This layer adds complexity and enables the model to learn non-linear relationships. This layer has Has 300 neurons and 784 input units, resulting in 784×300+300=235,500 parameters. Here biases are equal to the output units.

5.4 Second Dense Layer

The second dense layer contains 100 neurons, also using the ReLU activation function. This layer builds on the previous layer's outputs, further processing the data to create a meaningful representation for classification.This layer has 100 neurons and 300 input units, leading to 300×100+100=30,100 parameters. Here also biases are equal to output units.

5.5 Output Layer

The final layer is a dense layer with 10 output neurons and a softmax activation function. This layer outputs a probability distribution over 10 classes, making it suitable for multi-class classification tasks. The softmax function ensures that the sum of all outputs is equal to 1, providing a clear probability-based interpretation of the model's predictions. This layer has 10 neurons and 100 input units, resulting in 100×10+10=1,010 parameters. Here there are 266,610 parameters among which all are trainable indicating all layers are trainable. 1.02 MB is the estimated memory size required to store the model's parameters.

5.6 Model Compilation

Then we went for model compilation to set up its learning configurations. This step is required before training the model with data. We have used **sparse categorical loss function** for calculating loss function. This loss function is used when the target labels are integers representing class indices (e.g., [0, 1, 2,..., 9]). It's useful for multi-class classification where the classes are mutually exclusive. It is equivalent to categorical cross-entropy but works with sparse labels, avoiding the need to one-hot encode them. We have used **ADAM** that is Adaptive Moment Estimation as our optimizer. This is a popular optimization algorithm that combines the benefits of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam maintains adaptive learning rates for each parameter and uses moving averages of gradients to update the weights. It often converges faster and performs well across a wide range of problems.

6 Performance by training using the whole training data (with given labels)

We trained the sequential model with the whole training data with 20 epochs and 2000 values as batch size. Then we used a validation dataset to monitor performance after each epoch, allowing to track the model's learning progress and identify potential overfitting or underfitting.

- **Training Progression**
  - Training accuracy increases from 55.65% (Epoch 1) to 91.28% (Epoch 20), indicating significant improvement in the model's learning over time.
  - Training loss decreases from 1.3808 to 0.2438, reflecting a notable reduction in the model's error.
- **Validation Progression**
  - Validation accuracy starts at 80.08% (Epoch 1) and reaches 88.70% (Epoch 19), suggesting improved generalization on unseen validation data.
  - Validation loss starts at 0.5676 and decreases to 0.3098, indicating a reduction in the model's error on the validation set.
- **Overfitting or Underfitting**
  - Generally, if training accuracy increases while validation accuracy decreases, it can be a sign of overfitting. This does not seem to be the case here, as validation accuracy improves with training accuracy, indicating good generalization.
  - If training accuracy plateaus early and validation accuracy remains low, it could indicate underfitting, suggesting the model hasn't learned enough. However, this does not seem to apply in this instance.

6.1 Model Performance on Test Data

The results indicate that the model performs well on the test data:

- An accuracy of 88.53% suggests that the model correctly predicts a high proportion of the test set's labels.
- A loss of 0.3281 suggests the model has a relatively low error when compared to the actual outputs.

When we trained the model on 50 samples, we can clearly see that for training, validation and test dataset, accuracy was poor and loss was high.

6.2 Performance with labels from K-means clustering

Here we tried to understand how accuracy varies with cluster size. For the second strategy, we use only the data points closest to the cluster centroids, resulting in a smaller training dataset with k data points. The first strategy uses all data points in the original dataset, with labeling derived from the cluster structure, resulting in a large training dataset with all 55,000 points. In the second strategy,labels are derived from the data points closest to the cluster centroids. The resulting training dataset represents the "central" points of each cluster. In the first strategy, labels are derived from assigning each data point to its nearest cluster centroid, and then assigning the label of the closest training point to the centroid.

We can clearly observe that with increment in the number of clusters, the accuracy is higher in the second strategy of assigning each data point to its nearest cluster centroid. The accuracy is 0.72 for k = 100 whereas for the first strategy it is 0.67 for 100 clusters.

Finally we also explored the impact of different K-means cluster sizes and a 25% subset of data for training on the model's performance and we observe that the model gives approx accuracy of 0.69 for 100 clusters.