CSE 575: Statistical Machine Learning Portfolio

Shachi Shah
School of Computing and Augmented Intelligence
Arizona State University Online
Azusa, California, United States of America
spshah22@asu.edu

I. Introduction

The two main projects covered in this portfolio include the K-Means-Strategy Project and Classification Using Neural Networks and Deep Learning Project. Each project consisted of two parts, each with unique objectives and implementations. This portfolio outlines the project descriptions, solution approaches, results, and skills acquired during the course of these projects. Through these projects, we explored fundamental clustering techniques, convolutional neural networks (CNNs), and optimization strategies that form the backbone of modern machine learning applications. By analyzing performance metrics and adjusting hyperparameters, we gained insights into improving model efficiency and effectiveness.

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

K-Means is a widely used clustering algorithm in the field of data mining across different disciplines in the past fifty years [Qi, 1]. The objective of this project was to implement the K-Means clustering algorithm and apply it to a dataset of 2-D points. In this part, the cluster centers were chosen randomly from the dataset, and the implementation was tested for cluster sizes ranging from 2 to 10. The centroids and the loss function values were computed for each cluster count.

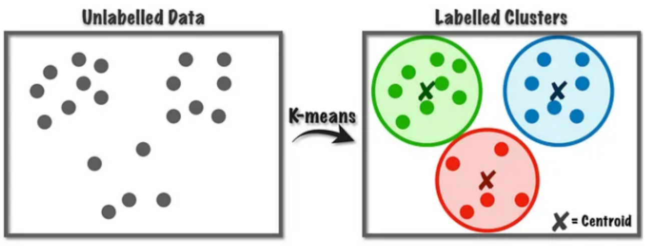

Fig. 1 - K-Means Clustering

The algorithm aimed to minimize intra-cluster variance by iteratively updating the cluster centers (see Fig. 1). Since the initial selection of cluster centers significantly affects the final result, this part explored the impact of random initialization on the clustering outcome.

The K-Means clustering algorithm works as follows from article:

1) Choose the number of clusters (K).
2) Initialize cluster centers - randomly select *K* points (centroids) from the dataset as the initial cluster centers.
3) Assign data points to the nearest cluster - each data point is assigned to the cluster whose centroid is the closest (using distance metrics like Euclidean distance).
4) Recalculate centroids - compute the new centroid (average position) for each cluster based on the points assigned to it.
5) Repeat until convergence. Steps 3 and 4 are repeated until the centroids no longer change significantly, meaning the clusters are stable.

2) Part 2 - Distance-Based Initialization

One of the most popular heuristics for solving the k-means problem is based on a simple iterative scheme for finding a locally minimal solution [Tapas, 100]. The second part improved upon the first by implementing a more informed centroid selection strategy. The first centroid was chosen randomly, while subsequent centroids were selected based on the maximum average distance from previously chosen centroids. The evaluation was done for cluster sizes ranging from 2 to 10.

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Baseline CNN Model

Adaptive non-parametric neural-net classifiers work well for many real-world problems (Lippman, 47]. The goal was to analyze and modify a Convolutional Neural Network (CNN) for classifying handwritten digits from the MNIST dataset. The baseline model was tested, and experiments were conducted by varying kernel sizes and the number of feature maps in convolutional layers.

This experiment aimed to understand the impact of network architecture modifications on accuracy and loss.

2) Part 2 - Evaluation of CNN on Subset of MNIST

The MNIST dataset is well known in introducing machine learning for several reasons (i.e. to classify irregularities) [Palvanov, 128]. This part focused on training and evaluating a CNN on a subset of MNIST, selecting four random categories with 2000 training samples and 400 test samples. The goal was to optimize model performance while understanding CNN principles.
algorithm.

II. Explanations of Solutions

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

The implementation involved initializing cluster centers by randomly selecting *k* points from the dataset. The algorithm iteratively assigned points to the nearest centroid, recalculated the centroid positions, and minimized the loss function until convergence.

The following key functions were implemented:

- `initial_point_idx(id, k N)`: Generates a set of *k* random indices to select cluster centers.
- `init_point(data, idx)`: Extracts data points corresponding to the indices.
- `initial_S1(id, k)`: Uses a seeded random selection method to initialize the centroids.

```
# Compute final centroids for k in range 2
to 10
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k,
init=initial_centers[k], n_init=1,
random_state=0)
    kmeans.fit(data)
    final_centroids[k] =
kmeans.cluster_center
# Compute loss (inertia) for k in range 2 to
10
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k,
init=initial_centers[k], n_init=1,
random_state=0)
    kmeans.fit(data)
    loss_values[k] = kmeans.inertia_
```

This method ensures each run has a unique et reproducible initialization, which allowed me to assess how the random selection influenced the final clusters.

2) Part 2 - Distance Based Initialization

A modified initialization function was introduced to ensure better centroid separationL

- `initial_point_idx2(id, k, N)`: Selects the first centroid randomly and subsequent centroids using a distance-based heuristic.
- `initial_S2(idx, k)`: Implemented the distance-maximizing strategy to spread initial centroids more effectively.

The clustering process followed the same iterative refinement as in *Part 1*, but with an improved starting condition that helped mitigate local optima issues.

```
def initial_point_idx2(id,k, N):
    random.seed((id+k))
    return random.randint(0,N-1)

def initial_S2(idx,k):
    # print("Strategy 2: k and initial
points")
    i = int(idx)%150
    random.seed(i+800)
    init_idx2 = initial_point_idx2(i,
k,data.shape[0])
    init_s2 = data[init_idx2,:]
    return init_s2
```

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Random Initialization

A CNN was implemented using Keras, consisting of convolutional layers followed by max pooling, fully connected layers, and a softmax output layer. The optimizer used was Adadelta, and the model was trained for 12 epochs.

Modifications included:

- Increasing the kernel size from 3x3 to 5x5.
- Increasing the number of feature maps in convolutional layers.

Each experiment was analyzed by tracking training loss, validation loss, and accuracy metrics.

```
Final Results Summary:
Baseline Model - Test Accuracy: 0.9782
Kernel Size 5x5 - Test Accuracy: 0.9806
Increased Feature Maps - Test Accuracy:
0.9876
```

2) Part 2 - Distance-Based Initialization

The CNN architecture was similar to Part 1, with modifications to accommodate the smaller dataset. The model was trained for 10 epochs, and accuracy and loss metrics were tracked.

The evaluation function `evaluate(net, images, labels)` was implemented to compute loss and accuracy over both training and test datasets. The results were visualized to analyze convergence trends.

```
def evaluate(net, images, labels):
    acc = 0
    loss = 0
    batch_size = 1  # Evaluating one sample
at a time
    for batch_index in range(0,
images.shape[0], batch_size):
        # Extract single image and label
        x = images[batch_index]
        y = labels[batch_index]
        # Forward pass through the network
        for layer in net.layers:
            x = layer.forward(x)
        # Compute loss using cross-entropy
        loss += cross_entropy(x, y)
        # Compute accuracy by comparing
predicted label with true label
        if np.argmax(x) == np.argmax(y):
            acc += 1
    # Normalize loss by the number of
samples
    loss /= images.shape[0]
    # Compute accuracy as a percentage
    acc /= images.shape[0]
    return acc, loss
evaluate
```

III. Description of Results

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

The results showed that as the number of clusters increased, the loss function decreased. However, the diminishing returns indicated that beyond a certain k, adding more clusters did not significantly reduce loss.

Using different random seeds resulted in slightly different clustering structures, reinforcing the sensitivity of the method to initial conditions. This provided motivation for exploring more robust initialization strategies.

2) Part 2 - Distance-Based Initialization

This method produced better clustering performance, as evidenced by lower loss values compared to Part 1. The improved initialization reduced the likelihood of poor local minima and resulted in more stable clustering patterns across different runs.

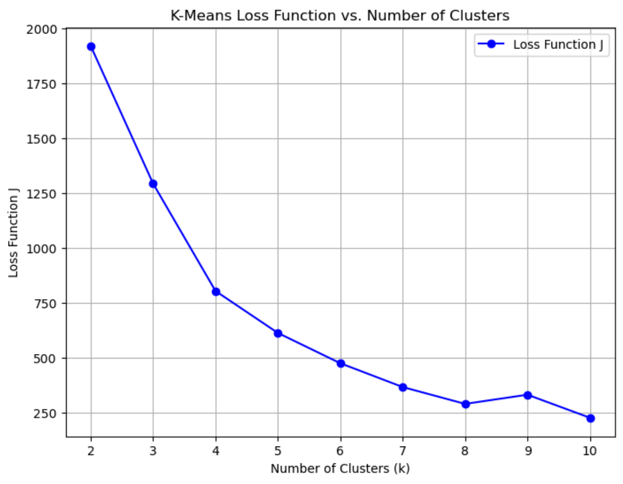

Fig. 2 - K-Means Loss Function vs. Number of Clusters

By analyzing the loss function trends (see Fig. 2), we confirmed that this strategy consistently led to better-separated clusters, highlighting the importance of intelligent initialization in K-Means clustering.

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Random Initialization

The baseline model achieved an accuracy of 97.82%. Increasing the kernel size improved accuracy slightly, while increasing feature maps provided the best performance improvement, reaching 98.76% accuracy.

Graphs of accuracy and loss trends confirmed that adding feature maps led to better feature extraction and improved generalization.

2) Part 2 - Distance-Based Initialization

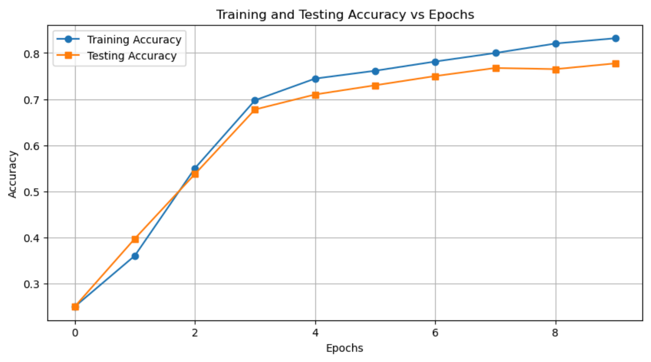

Fig. 3 - Training and Testing Accuracy vs. Epochs

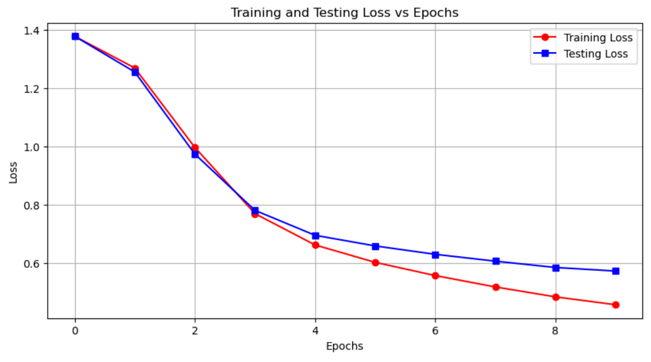

Fig. 4 - Training and Testing Loss vs. Epochs

The results were as follows:

- Initial Accuracy: 25% (random prediction level).
- Final Test Accuracy: 77.7% after 10 epochs.
- Final Training accuracy: 83.2%.
- THe model exhibited stable improvement across epochs, demonstrating effective learning.

Graphs of accuracy and loss trends further illustrated the model's learning progression (see Fig. 3 and Fig. 4), showing steady optimization over time.

IV. Skills and Knowledge Acquired

Throughout this learning experience, I have gained practical knowledge and hands-on expertise in various aspects of machine learning and deep learning. One of the key areas of focus was machine learning algorithms, particularly K-Means clustering. I explored its real-world applications and different initialization strategies, improving my understanding of how data can be effectively grouped and analyzed.

In the domain of deep learning, I developed experience in designing and training convolutional neural networks (CNNs) for image classification tasks. This involved working with complex architectures, understanding feature extraction, and fine-tuning models to achieve better accuracy. The process provided deeper insight into the functioning of neural networks and their applications in computer vision.

Additionally, I explored optimization techniques, focusing on how hyperparameter tuning effects model performance. By experimenting with various parameters such as learning rates, activation functions, and batch sizes, I learned how small adjustments can significantly impact the efficiency and accuracy of a model. This knowledge is crucial for building robust machine learning models.

My data analysis skills improved as I worked with clustering results and visualized neural network training progress. I became more proficient in interpreting loss curves, accuracy trends, and cluster distributions, helping me make informed decisions about model adjustments and refinements.

Finally, I enhanced my programming skills, particularly in Python, and deepened my expertise in various packages for implementing machine learning models. Working extensively with these tools strengthened my ability to write efficient code, debug issues, and optimize machine learning workflows. These technical skills are essential for tackling real-world problems in AI and data science.

References

[1] Palvanov, Akmaljon. "Comparisons of Deep Learning Algorithms for MNIST in Real-Time Environment." *International Journal of Fuzzy Logic and Intelligent Systems,* 2018. pp.126-134. [Online]. Available: https://www.ijfis.org/journal/view.html?volume=18&number=2&spage=126.

[2] Qi, Jainpeng. "K*-Means: An Effective and Efficient k-Means Clustering Algorithm," *IEEE Xplore,* 2016. pp.xii-xiv. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7723700.

[3] R.P. Lippmann. "Pattern classification using neural networks." *IEEE Xplore*, 1989. pp. 47-50. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/41401.

[4] Tapas, Kanungo. "The analysis of a simple k-means clustering algorithm," *ACM Digital Library*, 2000. pp. 100. [Online]. Available: http://dl.acm.org/doi/abs/10.1145/336154.336189