CS589: Homework 6

Vishnu Vardhan Reddy Bheem Reddy

November 20, 2025

Question 1: Active Learning

a. Random Acquisition

Below is the implementation of the `random_acquisition` function. It selects indices uniformly at random from the available pool.

```
def random_acquisition(Xpool, classifier, batch_size=5):
    """ Selects batch_size data cases from Xpool to acquire at random. """
    N_pool = Xpool.shape[0]
    indices = np.random.choice(N_pool, size=batch_size, replace=False)
    return indices
```

b. Entropy Acquisition

Below is the implementation of the `entropy_acquisition` function. It calculates the entropy of the predictive distribution for each instance in the pool and selects those with the highest uncertainty.

```
def entropy_acquisition(Xpool, classifier, batch_size=5):
    """ Selects the batch_size data cases from Xpool that have the highest entropy
    . """
    probas = classifier.predict_proba(Xpool)

    # Calculate entropy: H(x) = -sum(p * log(p))
    plogp = np.where(probas > 0, probas * np.log(probas), 0)

    entropies = -np.sum(plogp, axis=1)

    # Sort indices by entropy (ascending) and take the last batch_size (highest
    entropy)
    indices = np.argsort(entropies)[-batch_size:]
    return indices.tolist()
```

c. Labeling Oracle

The `labeling_oracle` moves data from the pool to the training set.

```
def labeling_oracle(Xtr, Ytr, Xpool, Ypool, inds):
    """ Simulates labeling by moving points from pool to training set. """
    X_acquire = Xpool[inds]
    Y_acquire = Ypool[inds]

    # Add to training set
    Xtr_updated = np.vstack((Xtr, X_acquire))
```

```
    Ytr_updated = np.concatenate((Ytr, Y_acquire))

    # Remove from pool
    Xpool_updated = np.delete(Xpool, inds, axis=0)
    Ypool_updated = np.delete(Ypool, inds, axis=0)

    return Xtr_updated, Ytr_updated, Xpool_updated, Ypool_updated
```

d. Active Learning Loop

The loop iterates through the active learning process, training the model, evaluating error, and acquiring new points.

```
def active_learning_loop(Xtr, Ytr, Xpool, Ypool, Xtest, Ytest,
    acquisition_function, num_iter=100, batch_size=5):
    test_errors = []

    Xtr_curr = Xtr.copy()
    Ytr_curr = Ytr.copy()
    Xpool_curr = Xpool.copy()
    Ypool_curr = Ypool.copy()

    for i in range(num_iter):
        # 1. Initialize and fit classifier
        classifier = LogisticRegression(max_iter=1000)
        classifier.fit(Xtr_curr, Ytr_curr)

        # 4. Compute test error
        score = classifier.score(Xtest, Ytest)
        test_errors.append(1.0 - score)

        # 2. Call acquisition function
        inds_to_acquire = acquisition_function(Xpool_curr, classifier, batch_size)

        # 3. Call labeling oracle
        Xtr_curr, Ytr_curr, Xpool_curr, Ypool_curr = labeling_oracle(
            Xtr_curr, Ytr_curr, Xpool_curr, Ypool_curr, inds_to_acquire
        )

    return test_errors
```

e. Experimental Results

The plot below compares the test error rate of random acquisition versus entropy-based acquisition as the size of the training set increases.

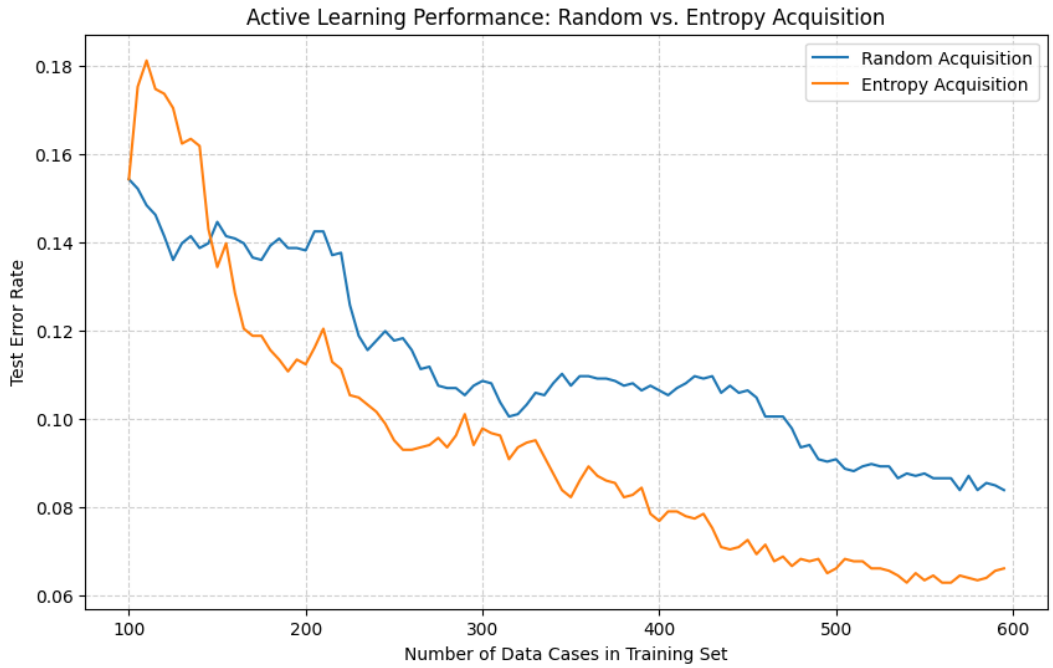

Figure 1: Test Error Rate vs. Training Set Size for Random and Entropy Acquisition.

Analysis:
Yes, entropy-based active learning outperforms the random baseline for this data set. As shown in Figure 1, the test error for the entropy strategy (orange line) decreases more rapidly than the random strategy (blue line).

This result demonstrates higher sample efficiency: the entropy method reaches a test error of roughly 0.08 with only ∼400 samples, whereas random acquisition requires nearly 600 samples to reach similar performance. This confirms that querying instances near the decision boundary (those with high entropy/uncertainty) provides significantly more information gain per labeled instance than selecting samples blindly.

Question 2: Clustering

a. Cluster Center

The function `cluster_center` computes the mean of the data points assigned to a specific cluster.

```
def cluster_center(X, ind):
    if len(ind) == 0:
        return np.zeros(X.shape[1])
    return X[ind].mean(axis=0)
```

Below is the image of the global centroid (assuming all data is in one cluster):

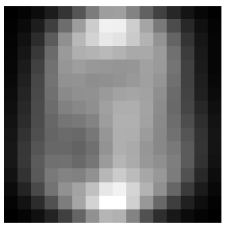

Figure 2: Centroid of the entire training set.

b. Cluster Objective

The function `cluster_objective` calculates the total within-cluster sum of squared distances.

```
def cluster_objective(centers, X, z):
    assigned_centers = centers[z]
    total_ssd = np.sum((X - assigned_centers)**2)
    return total_ssd
```

Objective Function Value (K=1): 631,481.5625

c. K-Means Clustering

I ran K-Means with $K = 10$ and 20 restarts.

- **Cluster Objective Value:** 338,838.125

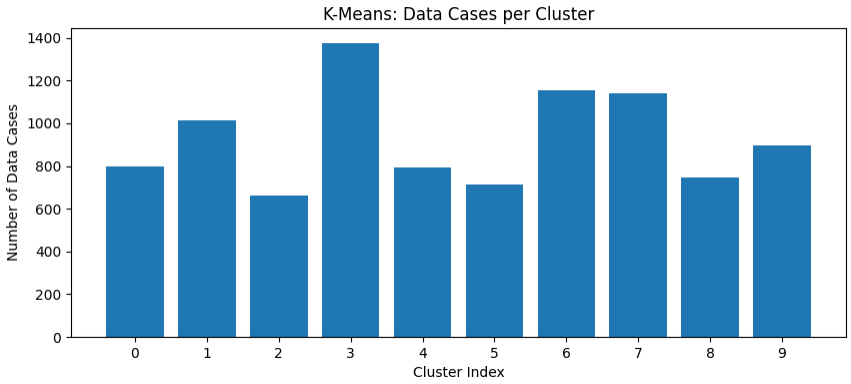

Figure 3: K-Means: Number of data cases per cluster.

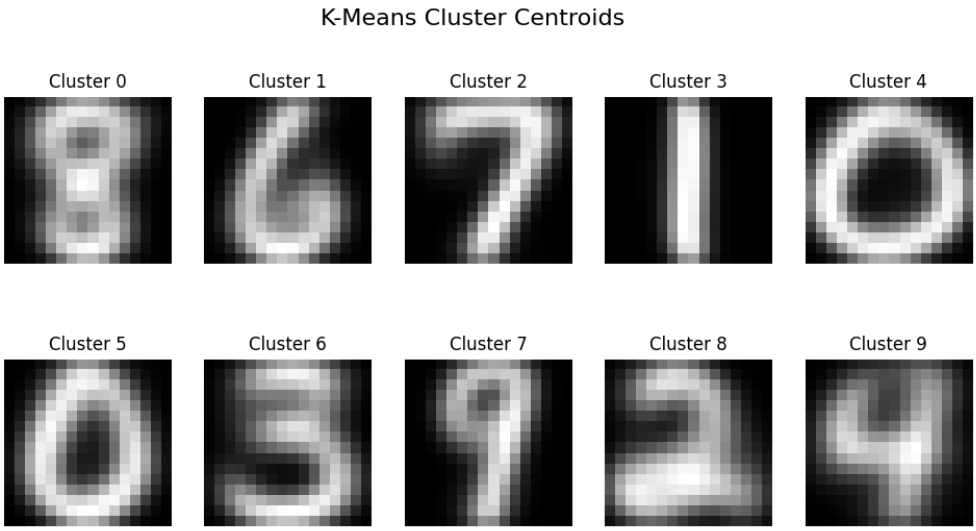

Figure 4: K-Means: Cluster Centroids.

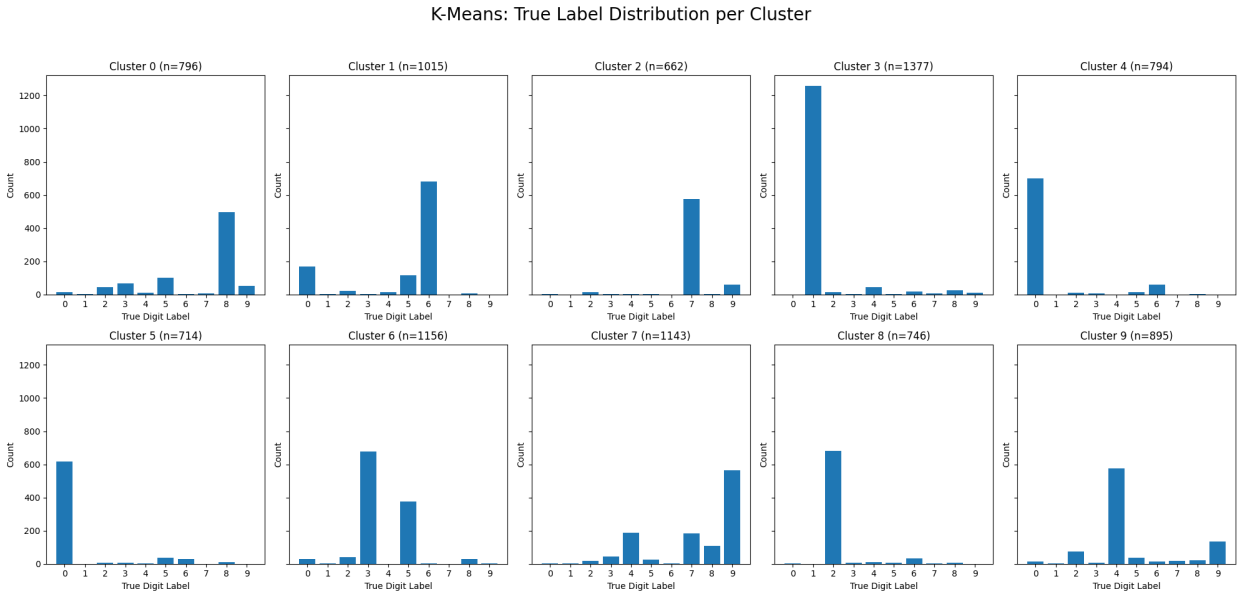

Figure 5: K-Means: True label distribution within each cluster.

d. Hierarchical Agglomerative Clustering

I ran Hierarchical Clustering with $K = 10$ using average linkage.

- **Cluster Objective Value:** 561,044.1875

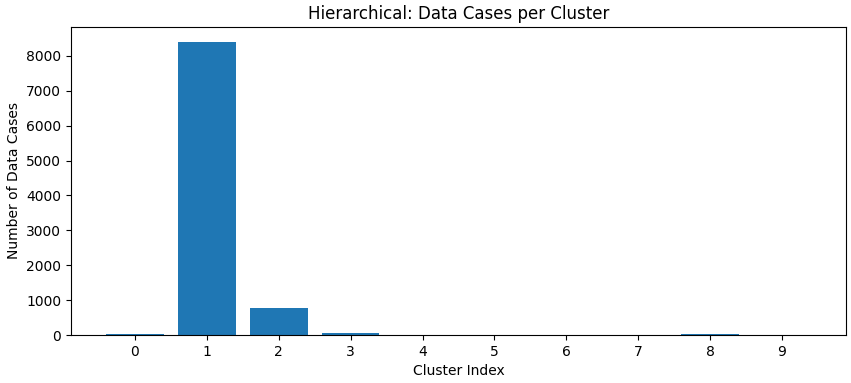

Figure 6: Hierarchical: Number of data cases per cluster.

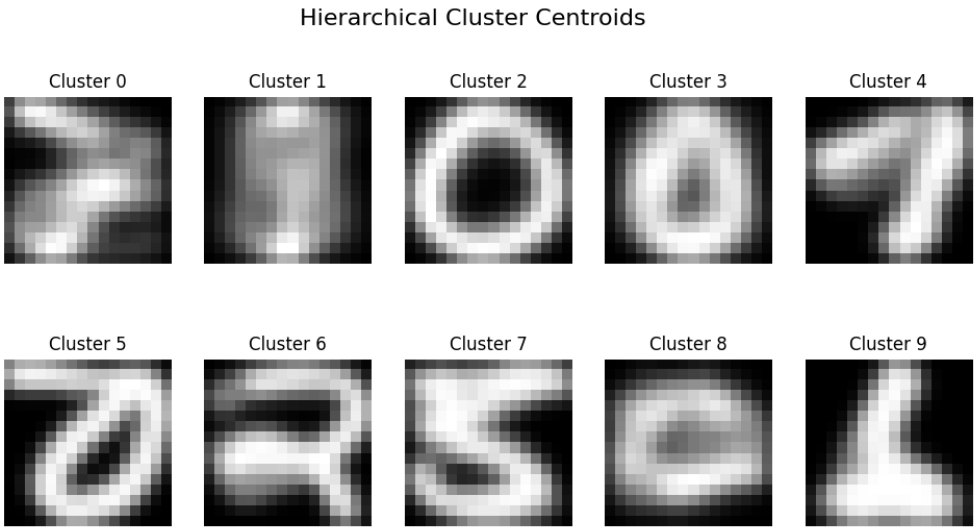

Figure 7: Hierarchical: Cluster Centroids.

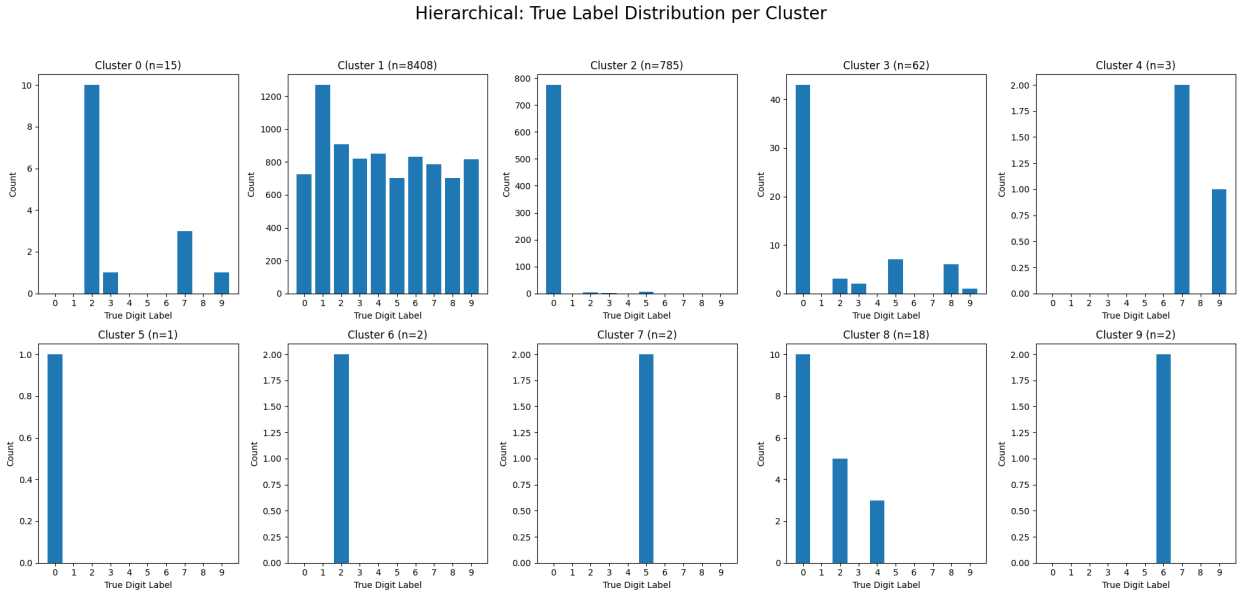

Figure 8: Hierarchical: True label distribution within each cluster.

e. Comparison and Analysis

Comparison of Methods: Comparing the two methods, **K-Means appears to be doing a much better job** on this data set.

1. **Objective Function:** K-Means achieved a significantly lower objective function value (338,838) compared to Hierarchical clustering (561,044). Since the objective measures the within-cluster sum of squared errors, a lower value indicates tighter, more compact clusters.

2. **Cluster Balance:** The bar charts showing data cases per cluster reveal weakness in the Hierarchical approach using average linkage on this high-dimensional data. Hierarchical clustering resulted in highly unbalanced clusters, one massive cluster (Cluster 1) containing nearly all the data, and several smaller clusters acting as outliers. This is due to the chaining effect and the difficulty of Euclidean distance discrimination in high-dimensional space. Meanwhile, K-Means produced a relatively even distribution of data points across the 10 clusters.

3. **Correspondence to Digits:** Checking centroids in K-Means shows clear digit-like shapes (e.g., distinct 0s, 1s, etc.), indicating that the clusters correspond well to specific digit types. The label distribution plots further confirm that K-Means clusters are relatively pure. The Hierarchical centroids, show high impurity (blurriness) in the large cluster, failing to separate the digit types effectively.

Question 3: Generative AI Statement

I used AI tools to assist with coding certain parts, like for method syntax (such as ‘np.where‘ for entropy stability) and plotting code with Matplotlib.