Midterm Project Presentation

Abhi Desai
Vinish Kandadi

Common instructions for all models

- Input: 70,000 handwritten digit images , flattened to 784 features
- Data Preprocessing:
  - Pixel values scaled from [0-255] to [0-1]
  - Data split: 80% training, 20% testing
- Performance Metrics used :
  - accuracy
  - Precision
  - Recall
  - F1-score
  - Confusion matrix

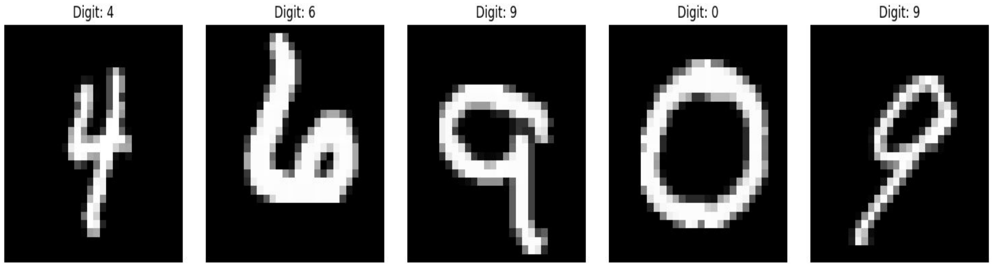
Logistic Regression

- Overall:
  Data Preparation→ Data Split (80% train/20% test) → Multinomial Logistic regression (softmax regression), 1000 iterations) → Training & Prediction (Fit & Predict) → Evaluation (92% accuracy, confusion matrix, per-digit performance) → Visualization (Sample predictions & accuracy plots)
- Logistic Regression :
  Model → Probability Output → Hard Classification (select highest probability class, since image must belong to exactly one digit class)

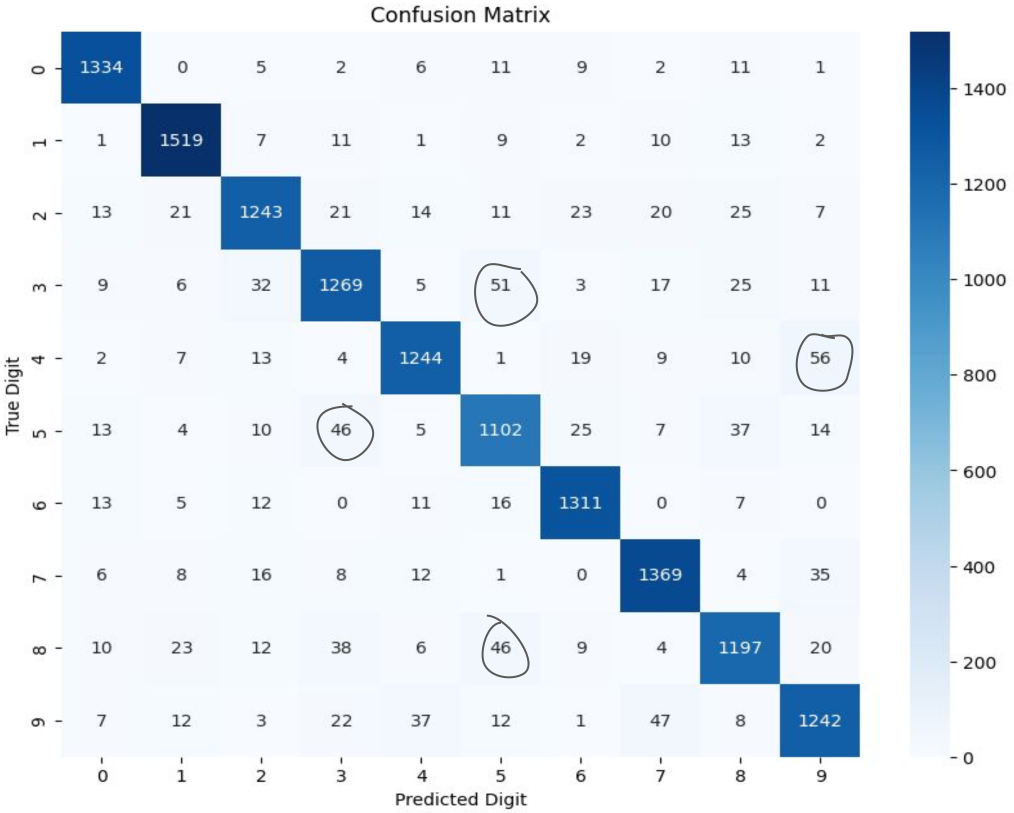
Logistic Regression - Performance

```
Model Performance Metrics:
Accuracy: 0.9164
Precision: 0.9162
Recall: 0.9164
F1-score: 0.9162

Detailed Performance Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1381
           1       0.95      0.96      0.96      1575
           2       0.92      0.89      0.90      1398
           3       0.89      0.89      0.89      1428
           4       0.93      0.91      0.92      1365
           5       0.87      0.87      0.87      1263
           6       0.94      0.95      0.94      1375
           7       0.92      0.94      0.93      1459
           8       0.90      0.88      0.89      1365
           9       0.89      0.89      0.89      1391

    accuracy                           0.92     14000
   macro avg       0.92      0.92      0.92     14000
weighted avg       0.92      0.92      0.92     14000
```

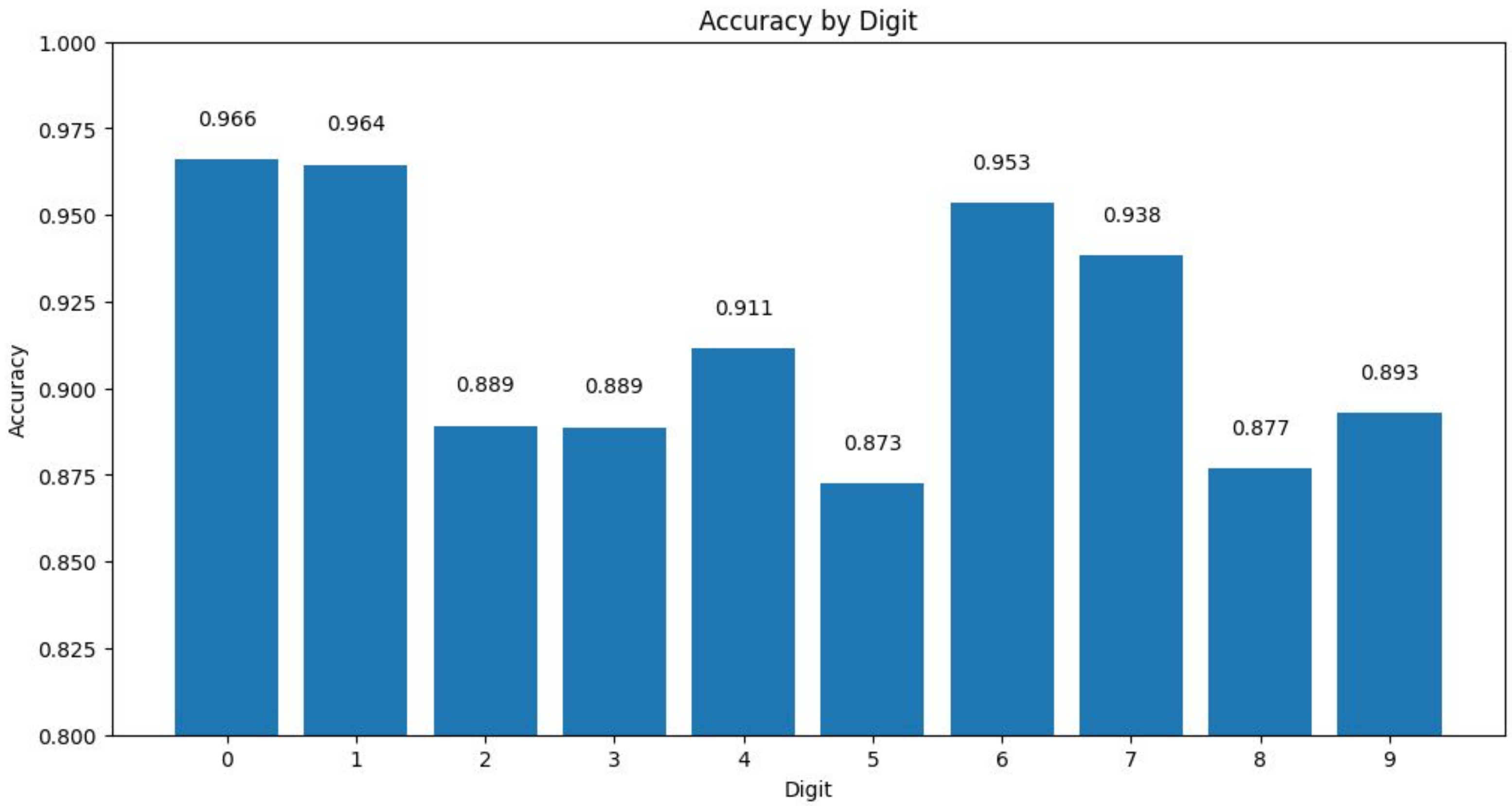
Sample predictions:



Logistic Regression - Confusion Matrix



Logistic Regression - Per Digit Performance



Logistic Regression - Strengths and Weaknesses

Strengths:

- good performance - hitting ~92% accuracy with minimal fuss
- fast training compared to deep learning approaches
- Super lightweight Easy to interpret what's happening under the hood
- Great baseline model for benchmarking more complex solutions

Weaknesses:

- Struggles with similar-looking digits (especially 3/8 and 4/9 pairs) for complex cases.
- Performance is lower than other complex approaches (CNNs easily hit 99%+)
- Might fail with real-world, messy handwriting unlike MNIST's clean dataset

SVM

- Tested with 3 kernels: linear, poly, sigmoid and 2 regularization values: 1, 10
- Evaluated on accuracy, precision, recall, and f1-score

SVM - Performance

```
Kernel: linear, C: 1
 Mean CV Accuracy: 0.9050
 Accuracy: 0.8990
 Precision: 0.8993
 Recall: 0.8990
 F1-Score: 0.8985
```

```
Kernel: poly, C: 1
 Mean CV Accuracy: 0.8870
 Accuracy: 0.8925
 Precision: 0.9067
 Recall: 0.8925
 F1-Score: 0.8952
```

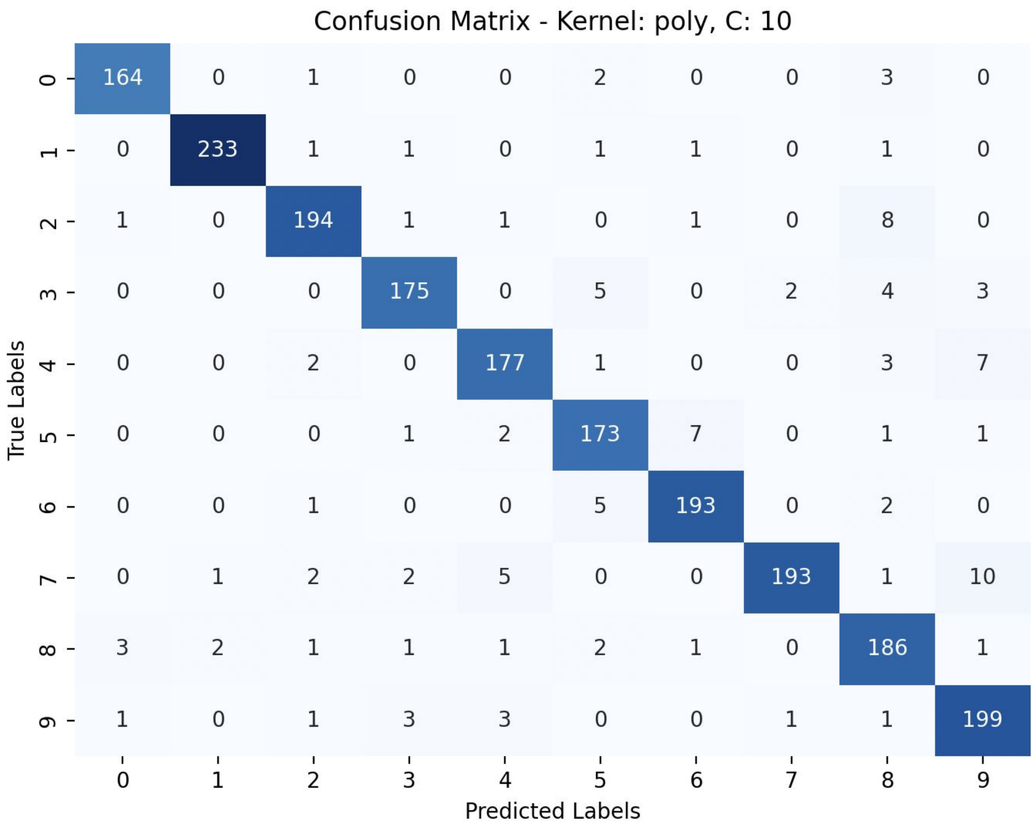
```
Kernel: sigmoid, C: 1
 Mean CV Accuracy: 0.9040
 Accuracy: 0.8930
 Precision: 0.8932
 Recall: 0.8930
 F1-Score: 0.8927
```

```
Kernel: linear, C: 10
 Mean CV Accuracy: 0.9050
 Accuracy: 0.8990
 Precision: 0.8993
 Recall: 0.8990
 F1-Score: 0.8985
```

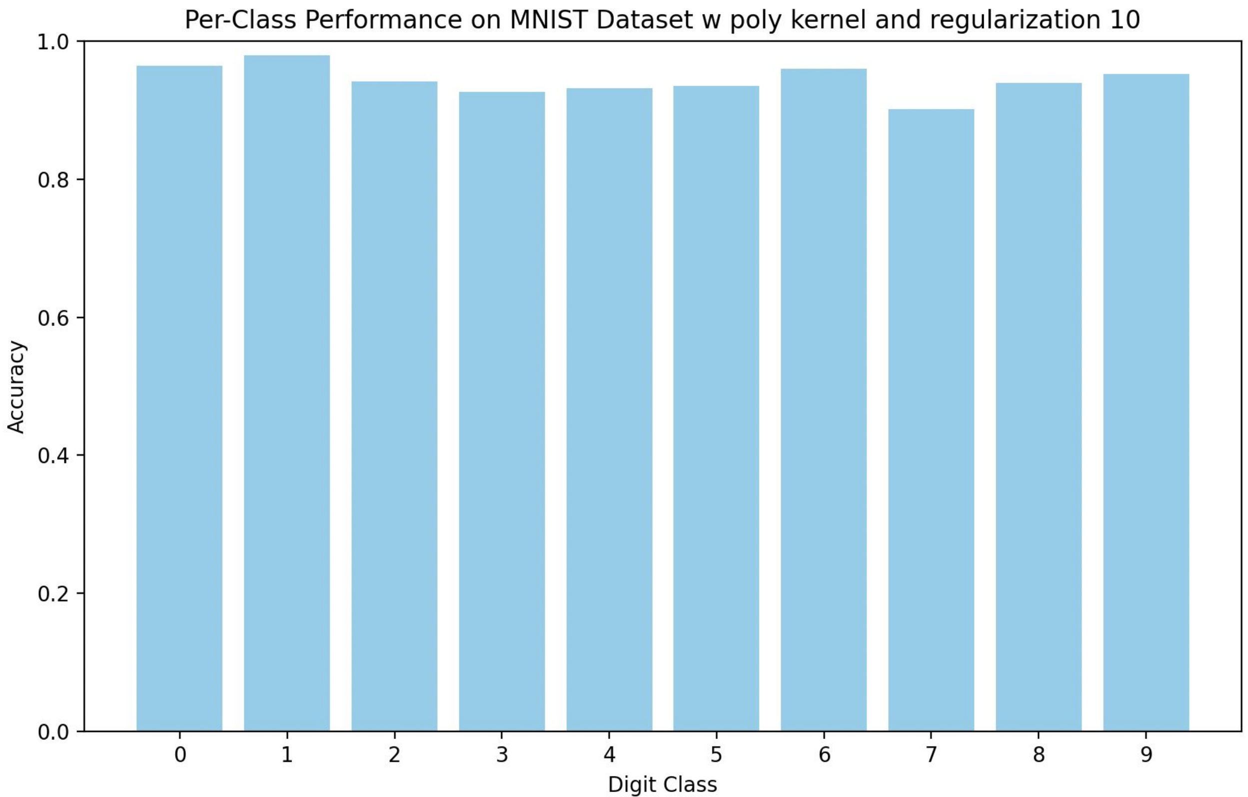
```
Kernel: poly, C: 10
 Mean CV Accuracy: 0.9468
 Accuracy: 0.9435
 Precision: 0.9446
 Recall: 0.9435
 F1-Score: 0.9437
```

```
Kernel: sigmoid, C: 10
 Mean CV Accuracy: 0.8686
 Accuracy: 0.8615
 Precision: 0.8622
 Recall: 0.8615
 F1-Score: 0.8611
```

SVM - Confusion Matrix



SVM - Per Digit Performance



SVM - Strengths and Weaknesses

Strengths:

- Good with high-dimensional data (like MNIST, 784)
- Different kernels for flexibility

Weaknesses:

- Slow training speed
- Complex parameter tuning needed
- Not naturally multi-class

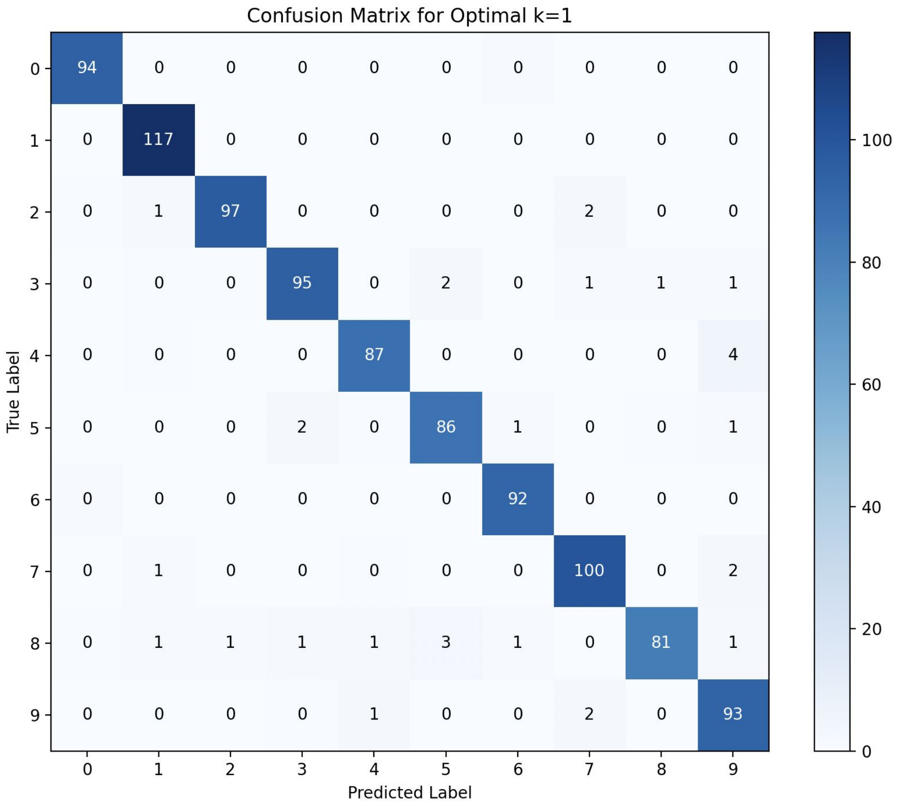
KNN

- Perform 10-fold cross validation to find best k value from a preset range from 1-10
- KFold from sklearn to shuffle and split data into 10 folds
- For each data partition (one of the 10 folds), the model is evaluated using every k value
    - Accuracy scores obtained for each k are averaged across all partitions (folds)
- We repeat this process 10 times each time with a different seed which affects how the data is partitioned
    - Calculate the average of each k across the different seeds and find the k value with the highest score

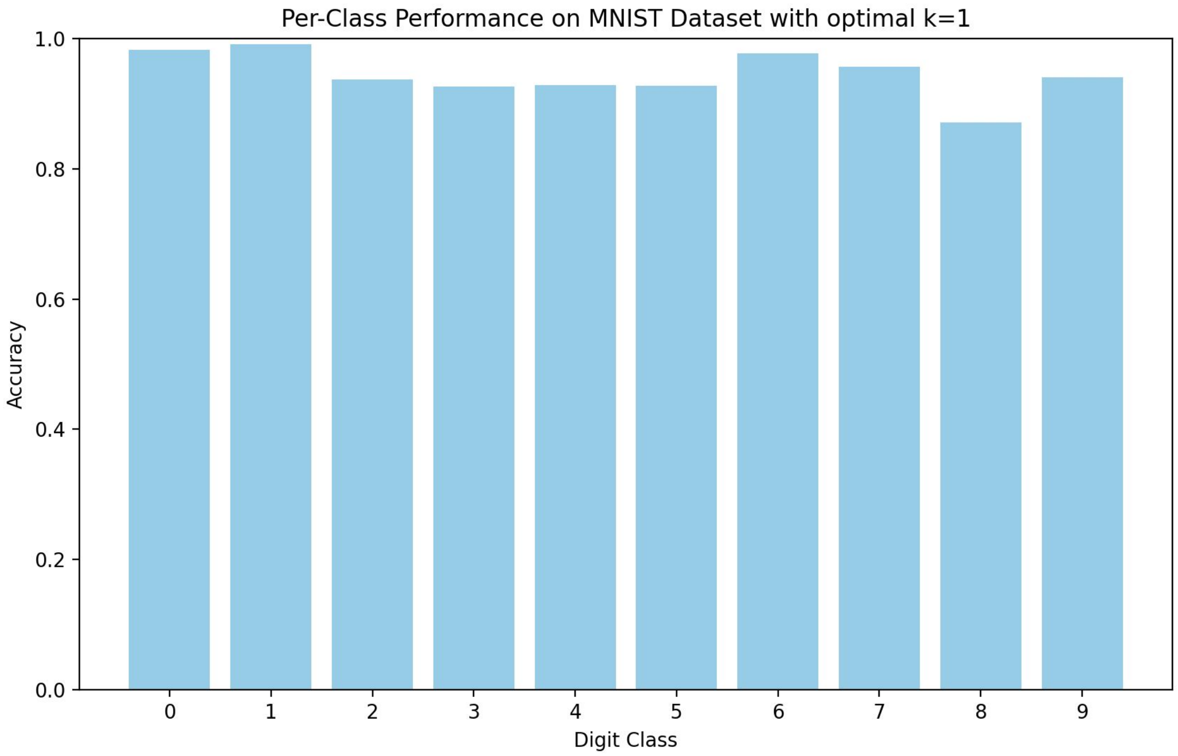
Performance

```
Overall average accuracy for k=1: 0.9446299999999999
Precision: 0.9452591178695622, Recall: 0.9435253235388925, F1 Score: 0.9436872645660201
Overall average accuracy for k=2: 0.9337799999999999
Precision: 0.9363279845984757, Recall: 0.9319876790556461, F1 Score: 0.9323037018713587
Overall average accuracy for k=3: 0.9441199999999998
Precision: 0.9456260438314821, Recall: 0.9428403119111625, F1 Score: 0.943240117027436
Overall average accuracy for k=4: 0.94331
Precision: 0.9453807930824262, Recall: 0.9420512976499419, F1 Score: 0.9426325733798849
Overall average accuracy for k=5: 0.9426200000000001
Precision: 0.9447239967984075, Recall: 0.9414358095104381, F1 Score: 0.9420058873588953
Overall average accuracy for k=6: 0.94141
Precision: 0.943811020361094, Recall: 0.9401930787505762, F1 Score: 0.9408527612020358
Overall average accuracy for k=7: 0.9394
Precision: 0.9418116629942419, Recall: 0.9382012952055359, F1 Score: 0.9387767395141526
Overall average accuracy for k=8: 0.9388000000000002
Precision: 0.9414181752975764, Recall: 0.9376079310699735, F1 Score: 0.9382526643522879
Overall average accuracy for k=9: 0.93814
Precision: 0.9409868902660732, Recall: 0.9369519339990571, F1 Score: 0.9376195658561464
Optimal k=1 metrics
Accuracy:0.9446299999999999
Precision: 0.9452591178695622
Recall: 0.9435253235388925
F1 Score: 0.9436872645660201
```

Confusion Matrix



Per Digit Performance



Strengths and Weaknesses

Strengths:

- Doesn’t rely on obscure hyperparameters like weights and biases
  - Can get intuition on why a classification was made by looking at its nearest neighbors
- High accuracy on MNIST

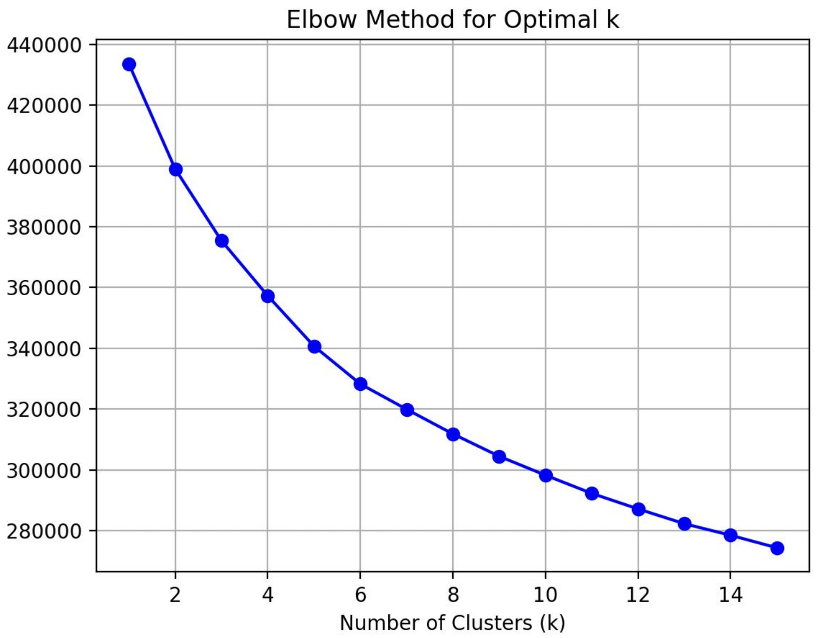
Weaknesses:

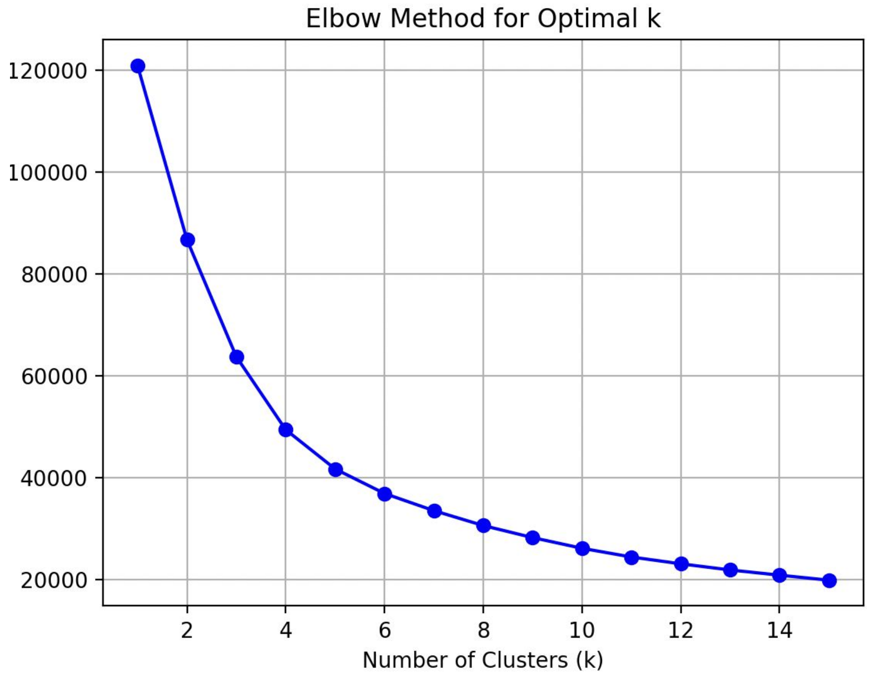
- Computationally expensive to find the nearest neighbor
- On a high dimensional dataset like MNIST, could suffer from curse of dimensionality as distances get less meaningful with increase in more dimensions
  - However, this could be mitigated with dimensionality reduction such as PCA
- High memory usage as it has to store the entire training dataset

Kmeans Clustering

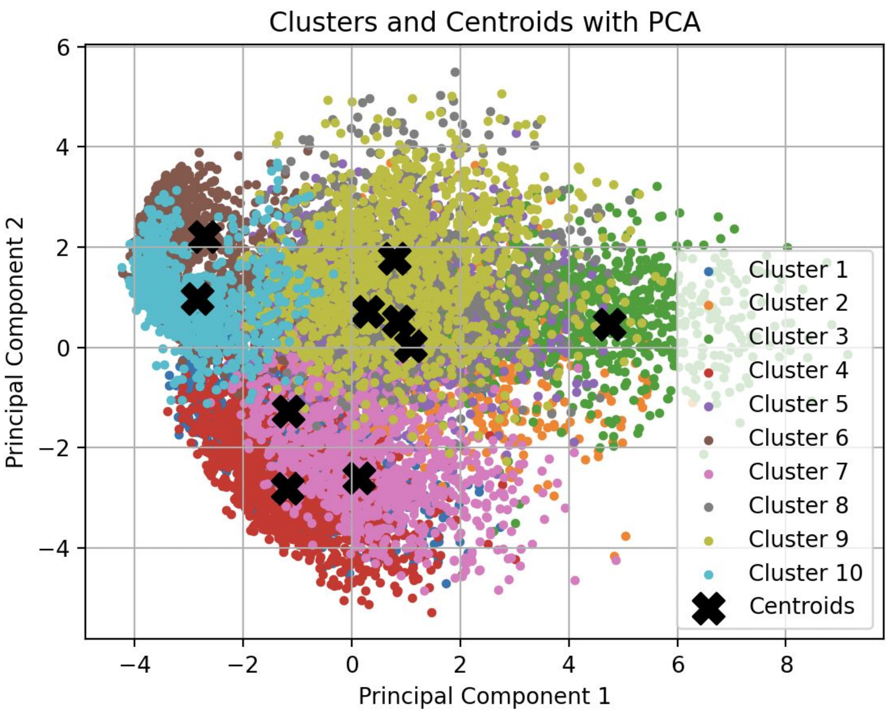
- Use elbow method to find number of clusters
- Initialize centroids using kmeans++
- Use KMeans from sklearn to cluster samples (784 dimensions)
- Use PCA to reduce samples and centroids to 2 dimensions and plot them
- Reconstruct flattened samples into original 28x28 shape and display the cluster means and some random samples in that cluster

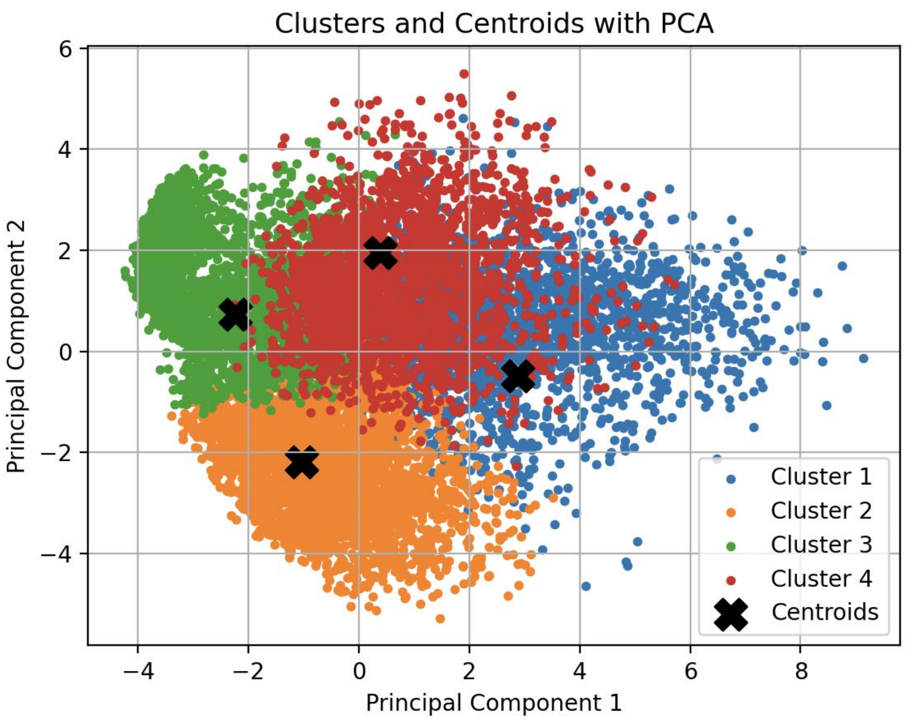
Elbow Chart


Elbow Method for Optimal k
440000
420000
400000
380000
360000
340000
320000
300000
280000
2
4
6
8
10
12
14
Number of Clusters (k)

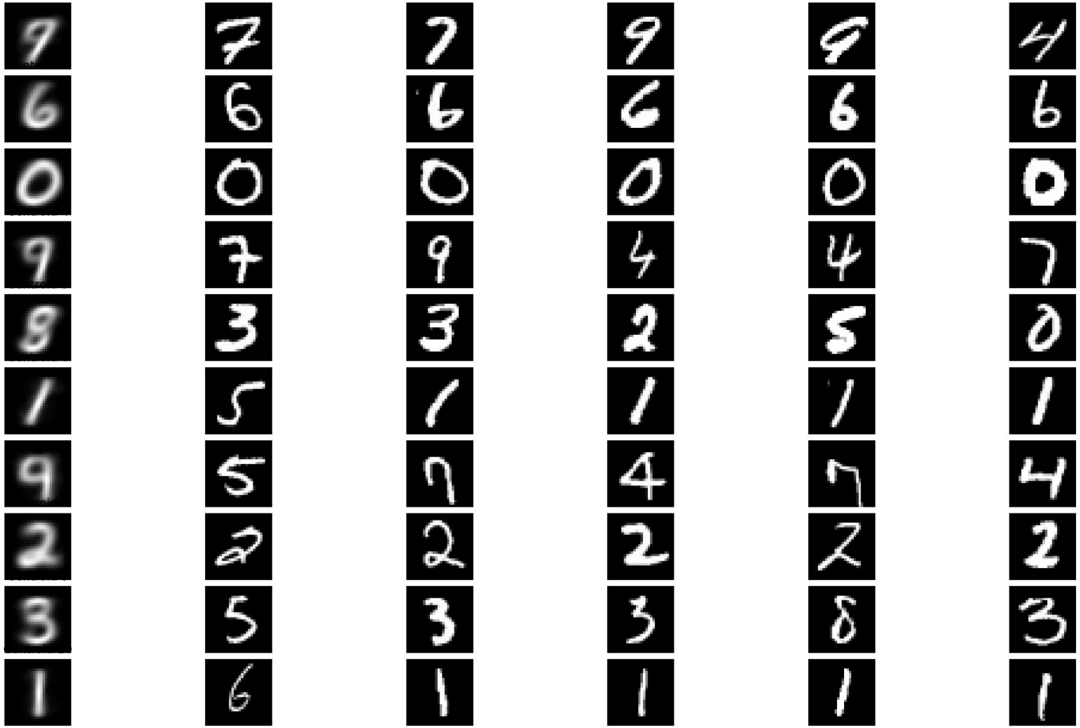

Elbow Method for Optimal k
120000
100000
80000
60000
40000
20000
2
4
6
8
10
12
14
Number of Clusters (k)

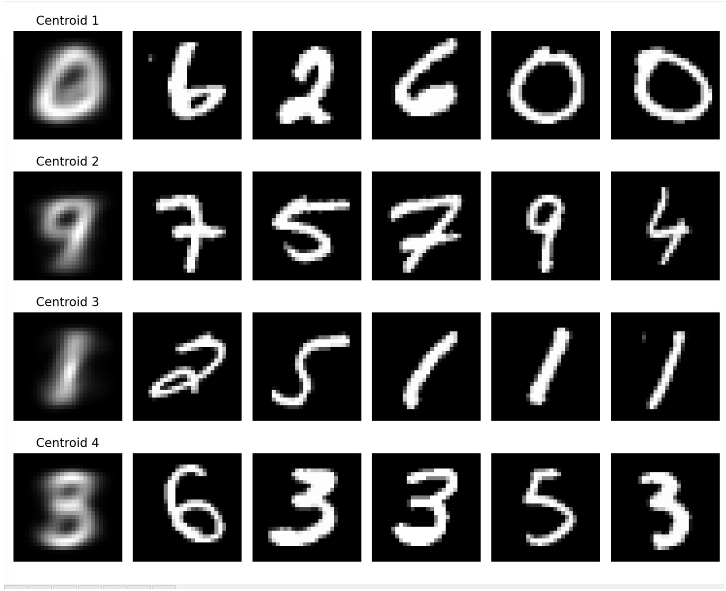
Clusters


Clusters and Centroids with PCA
Principal Component 1
Principal Component 2
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Centroids


Clusters and Centroids with PCA
Principal Component 1
Principal Component 2
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids

Visualized Results





Strengths and Weaknesses

Strengths:

- No labels required
- Low computation cost, especially with PCA

Weaknesses:

- Elbow method doesn’t yield expected number of clusters
- Even in cases of MNIST where underlying structures of the data are known (number of classes), clustering is still inaccurate

Thank You

Questions?