DA2401 EndsemProject

DA24B041 K Sree Hasini

November 2025

Preprocessing of Dataset:

- The given dataset has a column called 'even'(a binary variable indicating whether the digit is even or odd).
- But as that label implies the data leakage(as it is to be inferred from label), i removed it.
- As each pixel value can take any integer value from 0 to 255, the dataset becomes so sparse, which is problematic.
- So, I standardised the pixel values(features), by dividing the features by 255, i.e squished to be b/w 0 and 1.
- **PCA:** As we have 784 features, and anyway as the features are correlated, I took only principle components capturing most of the variance.Because keeping all the features anyway takes more time to run when algorithms like random forest, XGBoost are used.

Algorithms Implemented:

1.MultiClass Logistic Regression:

- Implemented softmax multiclass logistic regression, where the backward pass is done using basic gradient descent.
- Then I tried tuning the parameters learning rate and no.of epochs.
- I trained both on the actual train set and also after PCA with 500 comp.
-

Epochs	Validation Accuracy	F1 Score	Time (s)	Bias	Variance	Final Loss
200	0.8759	0.8742	12.84	0.2143	0.0527	0.4937
400	0.8864	0.8847	23.41	0.1818	0.0603	0.4109
500	0.8904	0.8889	30.76	0.1742	0.0623	0.3903
700	0.8968	0.8953	41.13	0.1649	0.0648	0.3628
900	0.8992	0.8978	55.32	0.1586	0.0665	0.3442
1200	0.9000	0.8986	72.55	0.1528	0.0681	0.3253

Table 1: Performance comparison for increasing training epochs on MNIST with PCA.

- **Observations from Table-2**
- **Effect of Learning Rate:** The learning rate has a strong influence on convergence:
  - LR = 0.01 converges slowly and achieves significantly lower accuracy.
  - LR = 0.1 consistently achieves much higher accuracy and F1 score.

  This indicates that a higher learning rate is better suited for MNIST PCA features.
- **Effect of Number of Epochs:** Increasing epochs leads to incremental improvements:

- For LR = 0.01, accuracy improves from 0.8475 to 0.8647.
- For LR = 0.1, accuracy improves from 0.8952 to 0.8992.

However, the improvement diminishes beyond 1000 epochs, showing convergence saturation.

- **Bias–Variance Trade-off:** The patterns follow expected theory:
  - High learning rate reduces **bias** but increases **variance**.
  - Low learning rate produces higher bias and lower variance.

Epochs	LR	Accuracy	F1 Score	Bias	Variance	Time (s)
900	0.01	0.8475	0.8455	0.2776	0.0414	36.67
900	0.1	0.8952	0.8938	0.1628	0.0667	34.94
1000	0.01	0.8523	0.8503	0.2676	0.0432	40.23
1000	0.1	0.8959	0.8947	0.1605	0.0673	40.04
1500	0.01	0.8647	0.8628	0.2345	0.0495	59.32
1500	0.1	0.8992	0.8978	0.1527	0.0693	59.13

Table 2: Softmax Regression performance for different learning rates and epochs on MNIST PCA data.

- **Training Time Increases Linearly with Epochs:** The time nearly doubles when moving from 900 to 1500 epochs. This aligns with linear time complexity of gradient descent training.
- **Best Parameters:** The combination LR = 0.1, Epochs = 1500 yields greater f1 score making it the best performing Softmax Regression setting in the experiment.
- **General Observation:** Softmax Regression performs surprisingly well on PCA-reduced MNIST, achieving close to 90% accuracy despite being a linear classifier.

Raw MNIST

Epochs	LR	Accuracy	F1 Score	Bias	Variance	Time (s)
900	0.01	0.8475	0.8455	0.2776	0.0422	36.67
900	0.1	0.9008	0.8994	0.1564	0.0671	60.54
1000	0.01	0.8523	0.8503	0.2676	0.0432	40.23
1000	0.1	0.8959	0.8947	0.1605	0.0673	40.04
1500	0.01	0.8707	0.8687	0.2330	0.0487	89.90
1500	0.1	0.9016	0.9002	0.1490	0.0692	90.77

Table 3: Softmax Regression performance on raw MNIST features for different learning rates and epochs.

- **Bias–Variance Patterns:** Results align with theoretical expectations:
  - Lower LR produces higher **bias** and lower **variance**.
  - Higher LR reduces **bias** but increases **variance**.

  For example, at 1500 epochs bias drops from 0.2330 to 0.1490 when LR increases.
- **Training Time:** Training time scales almost linearly with the number of epochs, and models trained with raw MNIST features require significantly more time than PCA-based models. This indicates that dimensionality reduction is crucial for computational efficiency.
- **Best Parameters:** The best performance is achieved at LR = 0.1, Epochs=1500

2. K-Nearest Neighbours:

- Implemented KNN, considered euclidean distance itself.
- Tuned over different values of K and used f1 score on validation set as a metric to decide upon the K value.
- **PCA:**

k	Validation Accuracy	F1 Score	Time (s)
3	0.9508	0.9503	7.36
5	0.9520	0.9519	8.21
7	0.9540	0.9539	7.16
11	0.9496	0.9493	8.27
13	0.9472	0.9469	7.33

-

Table 4: KNN performance on MNIST for different values of k after PCA(500 components.

- **Best Value of k:** The highest accuracy and F1 score are obtained at $k = 7$, achieving an accuracy of 0.9540 and f1 score of 0.9539. This shows that k = 7 gives the best trade-off b/w bias and variance.
- As it can be seen from the above table, as k increases after a certain value(11), accuracy and f1 score starts to decrease, due to high variance and lower bias.This is expected because a larger neighbourhood begins to "over-smooth" class boundaries,
- Smaller k (3,5) → Lower bias, higher variance.
- Larger k (11,13) → Higher bias, lower variance.
- **Computational Cost:** The run-time for each configuration remains between 7–8.3 seconds.
- **Conclusion:** The value **k = 7** provides the optimal balance between accuracy, robustness, and run-time efficiency for this MNIST PCA setup.

k	Validation Accuracy	F1 Score	Time (s)
3	0.9456	0.9450	89.55
5	0.9504	0.9503	88.39
7	0.9448	0.9448	88.04
9	0.9440	0.9440	87.94
11	0.9404	0.9401	87.86
15	0.9340	0.9338	88.24

Table 5: KNN performance on MNIST for different values of k.

- **Best Value of k:** The highest accuracy and F1 score are achieved at $k = 5$.
- After $k = 5$, both accuracy and F1 score are decreasing This occurs because large k over-smooths decision boundaries, leading to higher bias.
- **Bias–Variance Trade-off:**
  - Small k (3) → Lower bias, higher variance.
  - Medium k (5) → Optimal balance.
  - Large k (11, 15) → Higher bias, lower variance.
- **Computational Time:** The relatively long times show the computational limitations of KNN on large datasets. This can be
- **Conclusion:** The model performs best at $k = 5$, offering the highest accuracy and F1 score while maintaining computational consistency.

3. XGBoost Classifier:

- Boosting algorithms are known for their sequential learning techniques.
- I implemented XGBoost classifier, but so as to lower the time taken, I considered only a random subset of features each time.
- I tuned the two hyperparameters - no.of estimators and max depth.

n_estimators	max_depth	Accuracy	F1 Score	Time (s)
50	3	0.9288	0.9281	161.45
100	3	0.9532	0.9529	324.71
50	5	0.9368	0.9362	323.60
100	5	0.9572	0.9570	646.33

Table 6: XGBoost Multi-Class Performance on MNIST for different hyperparameters.

- Increasing the no.of estimators(boosting rounds) improves accuracy and f1 score.
- For depth = 3, accuracy increases from 0.9288 to 0.9532.
- For depth = 5, accuracy increases from 0.9368 to 0.9572.
- Deeper trees lead to higher accuracy but significantly slower training.
- Depth = 3 gives strong results but underfits slightly.
- Depth = 5 improves accuracy but nearly doubles computation time. Thus, deeper trees offer more expressive power at the cost of run time.
- **Bias–Variance Behaviour:**
  - Low depth (3) → Higher bias, lower variance.
  - Higher depth (5) → Lower bias, higher variance.

  The results illustrate this clearly: models with depth 5 outperform those with depth 3 at the cost of greater complexity and risk of overfitting.
- **Impact on Computational Time:** Training time scales almost linearly with the number of estimators and increases steeply with depth:
  - Depth 3: 161 s → 324 s
  - Depth 5: 324 s → 646 s
- **Best Parameters:** The combination $_estimators = 100, max\_depth = 5getsthehighestperformance, makingitthemost$
- **The above results confirm that boosting enhances digit classification accuracy.(¿0.92)**

Conclusion:

Model	Best Hyperparameters	Accuracy	F1 Score	Time (s)
KNN	$k = 5$	0.9504	0.9503	88.39
Softmax Regression	LR = 0.1, Epochs = 1500	0.8992	0.8978	59.13
XGBoost Multi-Class	$n = 100$, depth = 5	0.9572	0.9570	646.33

Table 7: Comparison of best-performing classical ML models on MNIST.

- **Linear vs Non-Linear Models:** Softmax Regression performed well given it's linear scope by giving nearly 90% accuracy on PCA-reduced MNIST. But it's performance is lesser than non-linear models such as KNN and XGBoost, which can capture more complex decision boundaries.

- **XGBoost as the Best Classical Model:** XGBoost gave the highest overall results, reaching **95.7% accuracy** with $n = 100$ and depth 5. Its ability to use both first- and second-order gradient information enables fast and stable convergence.

- **KNN as the best model:** KNN achieved strong results with $k = 5$ and reached an accuracy of 95.4%. This highlights that instance-based learning remains a powerful technique for MNIST.But KNN suffers from relatively high prediction-time cost due to euclidean distance computations. But it is **not any less better** than **XGBoost** by the **accuracy and f1 score metrics** achieved.

- **Bias–Variance Observations:** Softmax Regression showed the expected behaviour: increasing epochs decreased bias while slightly increasing variance. KNN showed the opposite pattern, with mid-range values of k achieving the best trade-off. Boosting-based models achieved low bias but higher computational cost.

- **Effect of PCA:** PCA significantly reduced model training time (especially for KNN and boosting). This confirms PCA's importance as a preprocessing step for high dimensional datasets.

- I also tried **bagging** but it is taking a lot of time.

- **Overall Summary:**
  - XGBoost achieved the **best accuracy**.
  - KNN offered the **best performance-to-implementation simplicity ratio**.
  - Softmax Regression provided the **fastest training** but lowest accuracy.