DA2401 PROJECT REPORT

DA24B046

algorithms.py

This file contains following models

Linear Regression Ovm

Linear Regression model trains one linear regression classifier per class, treating the target class as 1 and all others as 0. Each classifier learns weights that minimize squared error between predictions and one-hot encoded labels. Training uses mini-batch gradient descent, L2-regularization, and a bias term. At inference time, the model computes class scores and predicts the class with the highest score.

Hyperparameters used:

- lr (learning rate)
- epochs
- batch_size
- lambda_l2 (L2 regularization strength)

HP Tuning :

lr	epochs	batch_size	lambda_l2	accuracy	f1 score
0.01	40	56	0.01	0.8455	0.8431
0.03	30	112	0.01	0.8451	0.8423
0.01	30	56	0.01	0.8459	0.8434

Table 1: Lr Hp tuning

Softmax Regression

Softmax Regression models the probability of each class using linear scores + softmax, and trains via cross-entropy loss, also performs mini-batch gradient descent on the weight matrix , updates scores and uses one-hot labels. The class with maximum probability is selected during prediction.

Hyperparameters used:

- mini_batch_size
- learning_rate
- n_epochs

HP Tuning :

mini_batch_size	learning_rate	n_epochs	accuracy	f1 score
56	0.01	40	0.8375	0.8379
56	0.01	30	0.8603	0.8576
112	0.01	30	0.8499	0.8468

Table 2: Sr Hp tuning

XGBoost

XGBoost model builds an ensemble of decision stumps/trees, where each tree fits the gradients and Hessians of the softmax loss for each class. Each round adds K trees (one per class), updating the logits with a learning rate. Splits are chosen by maximizing gain, taking into account lambda regularization and gamma for penalizing splits. It uses softmax to compute probabilities after all boosting rounds.

Hyperparameters used:

- n_estimators (number of boosting rounds)
- learning_rate
- max_depth (tree depth)
- subsample_features (column subsampling)

n_estimators	learning_rate	max_depth	subsample_features	accuracy	f1 score
40	0.5	3	0.1	0.9147	0.9139
30	0.4	3	0.1	0.9091	0.9083
30	0.5	3	0.1	9052	9047

Table 3: XGB Hp tuning

kNN

KNN classifier stores all training samples and classifies a new point by computing Euclidean distance to all training points. It selects the k nearest neighbors using efficient vectorized distance computation, then predicts the class by majority vote among those neighbors.

Hyperparameters used:

- k (number of neighbors)
- p (principle components)

HP Tuning :

k	p	accuracy	f1 score
112	6	0.9519	0.9518
112	5	0.9523	0.9523
56	5	0.9555	0.9554

Table 4: kNN Hp tuning

main.py

This file trains and evaluates an ensemble classifier by combining predictions from three different models: softmax regression, XGBoost, and a PCA-based KNN classifier. It begins by loading data then trains each model separately and obtains their predictions. A stacking function selects the final prediction for each sample by taking the last mode (most frequent label, choosing the last if tied) across the three model predictions. After forming the ensemble prediction, we compute evaluation metrics such as F1-score, accuracy, precision, recall, and also reports total training time.

Observations

Here I have used optimal values as hyper parameters considering both accuracy and traing time (since we have constraint on it).

- Softmax Regression : mini_batch_size=56,learning_rate=0.01, n_epochs=30
- XGBoost : n_estimators=40, learning_rate=0.5, max_depth=3,subsample_features=0.1
- kNN : k=6,p=112

We got an accuracy of 94.23%, the ensemble model performs very well on the MNIST validation set, showing that combining softmax regression, XGBoost, and PCA-KNN improves reliability.

The macro F1-score of 0.942 indicates strong and balanced performance across all digits. No major class appears to be highly misclassified.

Precision (0.943) and recall (0.942) are closely matched, suggesting the model is not biased toward over-predicting or under-predicting specific classes. It shows a stable trade-off between false positives and false negatives.

Training time is also around 4∼5 min (which is in our time constraint) obtained from tuning hyper parameters and PCA.

Out of all models I have used kNN(with PCA) gave the best validation accuracy and fastest , Softmax and XGBoost gives good accuracy and Linear Regression gave comparative low (not worst, around 80%) and XGBoost have taken more time to train.

For this dataset tuning correct HP of XGBoost can give more accuracy but it will increase training time significantly (violates our constraint). Hence considering our terms we have kNN as best classifier for given dataset

The final predictions come from taking the most frequent label across the three models. from the obtained metrics we can say that stacking helps correct mistakes that individual models might make alone.

Confusion Matrix

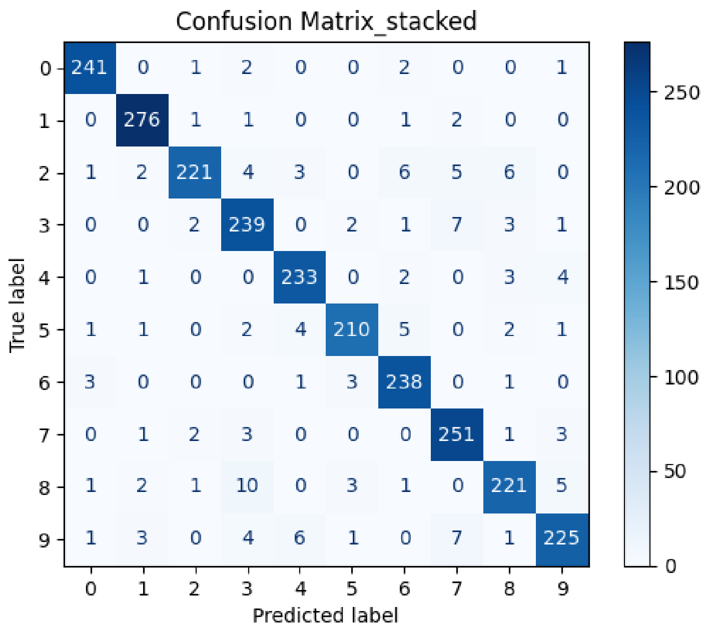

Figure 1: Stacked Results (SR, XGB, kNN)