DA2401 ENDSEM

DA24B024 Sahithi Macharapu

System Architecture

- The best model was the one where I used a combination of PCA and KNN clustering to predict the labels.
- KNN is a form of supervised learning model that learns labels according to the k nearest data points around it.
- Usually K-nearest neighbor clustering is prone to the curse of dimensionality because distance between points in a higher dimensional space is more complicated than that in the 3 dimensional space and the results get progressively worse as the dimensions increase.
- But in this case the data is very sparse, with most of the values in each set just being 0, so using KNNs seemed to be a good approach.
- PCA is a form of dimensionality reduction that ensures that the variance of the data isn't lost.
- When we don't want to process the entire data, it is usually a good idea to first reduce the dimensions without losing out on any information, which can be done by PCA.
- So, I first used PCA for dimensionality reduction to reduce the computational complexity while ensuring loss of the least amount of information possible.
- This helps reduce the negative effect of having a large dimension in the data.

Hyperparameter Tuning

- From this table we can see that if n_neighbors is equal to 1, the model has a tendency to overfit giving rise to an F1 score of 1.
- And having a larger number for n_neighbors didn't really change the performance by much, but it increased the computational complexity.
- So, I decided to use 3 as the value for n_neighbors.
- Increasing the number of components in the PCA part doesn't improve performance after a while, so there's no point in choosing a large value.
- We don't want to take the cases where the training results are way better than the validation ones because the model is overfitting in these cases.
- So, the best case would be when n_neighbors is equal to 3 and n_components is equal to 30.

S.No.	n_components	n_neighbors	F1 score(Train)	F1 score(Val)
1.	20	1	1	0.950
2.	20	3	0.972	0.948
3.	20	5	0.965	0.951
4.	30	1	1	0.958
5.	**30**	**3**	**0.976**	**0.956**
6.	30	5	0.970	0.956
7.	50	3	0.978	0.956
8.	50	5	0.970	0.956
9.	100	3	0.975	0.950
10.	100	5	0.967	0.951
11.	200	3	0.974	0.945
12.	250	3	0.973	0.945

Other Methods Tried

- I tried multiple other methods as well before I decided that the model with PCA and KNN gave the best results.
- Starting from the least complicated models, I tried working my way up to the more complicated ones.
- I tried training the data on multinomial logistic regression models, tree based models and I even trained it on an ensemble of these models as well.
- I have added all the results of these models in the table below.

S.No.	**Model Architecture**	**F1 score (Train)**	**F1 score (Val)**	**Accuracy (Train)**	**Accuracy (Val)**
1.	Multinomial Regression	0.940	0.893	94.151	89.515
2.	XGBoost	0.907	0.877	90.711	87.634
3.	Ensemble of XGBoost and Multinomial Regression	0.939	0.893	94.041	89.435
4.	**PCA + KNN**	**0.975**	**0.958**	**97.520**	**95.838**

- Multinomial Regression is a very simple model with a linear layer of weights and biases that help predict which number the image has.
- The reason XGBoost and the other Tree based models weren't able to perform as well as KNNs is because of the way the data is structured.
- KNN models work the best when the data is easily separable into clusters.
- So, this means that after we performed PCA on the data, the data was converted into a form that could be easily clustered, hence giving rise to such good results.
- Multinomial Regression most likely was unable to learn the best relationships between the data and the labels, so its F1 score wasn't as good as the others.
- I tried using an ensemble of XGBoost with Multinomial Regression.
- It used a version of soft voting between the two different models and gave the final probabilities based on that.
- I tried fine-tuning the hyperparameters like the weights of the probabilities used for each model and so on, but this model's training f1 score was considerably higher than that of the validation f1 score. This implies that the model is probably overfitting on the training data.
- Also, the ensemble model was taking way too long to run and I couldn't train it under 5 minutes, so I decided not to use this ensemble model.

Steps Taken to Optimize System Performance and Limit Run Time

- Using KNN models on the MNIST data would be very time consuming because of the number of columns that exist
- For KNN models, the more the number of feature columns, the more compute power is required because it has to find the distance between every feature and compare.
- So, in this MNIST dataset, if we were to use all the features as given and perform KNN on it, the compute power required would be a lot.
- But, if we were to perform PCA on the data to reduce the dimensions, the compute time would also reduce and hence run faster without losing out on any important data.
- Initially I used a for loop for the prediction part in the KNN model, but this took a very long time to run(~9 minutes).
- So, I vectorised the data and ran the predictions on that. This helped speed up the code and the run time reduced to about 0.1 minutes long.

Conclusion

- Overall, the MNIST dataset is a very sparse one, with many entries in a row just being 0.
- So, simple methods work the best for this dataset.
- Keeping in mind the compute and memory constraints, PCA would be a great way to reduce the dimension of the data before performing KNN.
- This exercise shows that we don't need very complicated models for simple datasets.
- The more complicated the model is, the more it has a tendency to overfit which is something we don't want because we want the model to be able to generalise better on unseen test data.
- That is why XGBoost and the ensemble method performed worse than the simple KNN one.