IML LAB 3 PROJECT REPORT

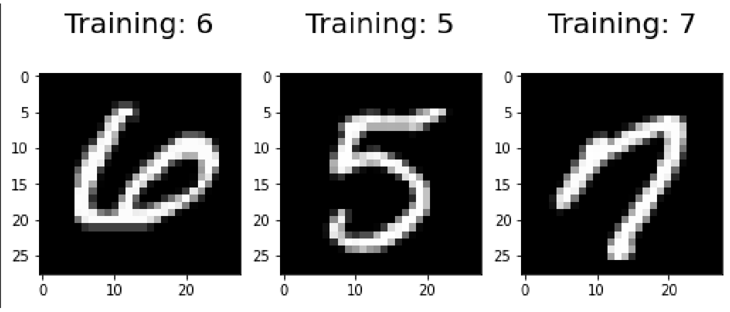
Handwritten Digit Analysis

Abhimanyu Timbadia, Medini Chopra, Arundhati Balasubramaniam, Anna Upreti

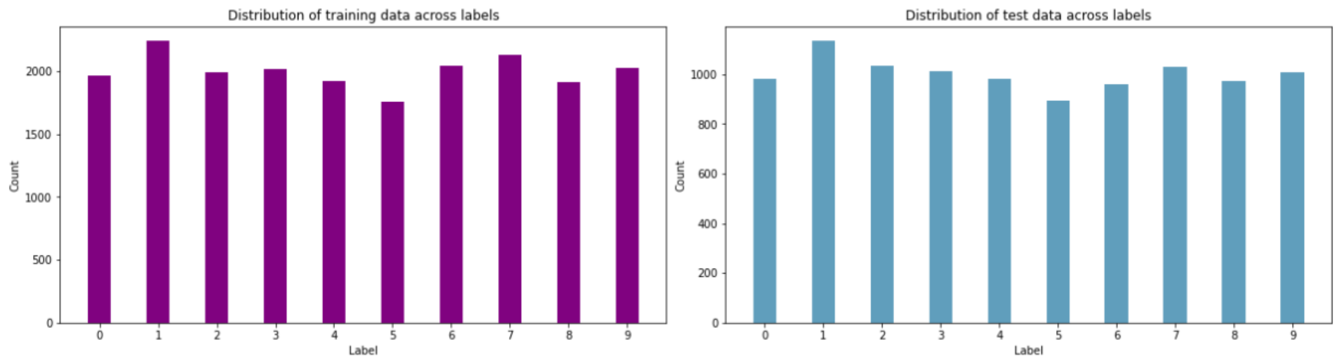
Part 1: Exploring the dataset

The data files train and test contain black and white images of hand-drawn digits, from 0 through 9 which are the labels of the images in the training data. The first column of the training dataset has the labels of the images and the rest of the columns are the individual pixel values of each image (from pixel 0 to pixel 783). We have 20,000 training examples and 10,000 examples we can test. Each image is 28 pixels by 28 pixels in width. This pixel-value is an integer between 0 and 255, inclusive.

Firstly, we do not need to clean the datasets as we have no NULL values in both train and test (This was checked using a simple ISBLANK function in excel). Secondly, normalizing this dataset would not make sense as our pixel values are not evenly distributed from 0 to 255. There is a heavy skew towards these extreme values. Finally, this dataset is already standardized as there are no varying scales for our data (all pixel values are between 0 and 255).



Visual representation of some examples.



X-axis: label, Y-axis: count.

The above figure shows that we have a fairly even spread of the numbers in the training dataset. This is paramount to ensure good predictive power of our ML algorithms.

This dataset is compiled by the National Institute of Standards and Technology (NIST). It was created by rehashing the samples from NIST's original datasets. The reason it was modified was because the test and training datasets were taken from two completely different sources and hence would not be appropriate for machine learning applications. This dataset was optimized for ML applications as all the images were converted into 28 pixels by 28 pixels and the contrast was increased to ensure sharp boundaries between what is an image and what is not. This really reduces the burden on people using this dataset for ML purposes.

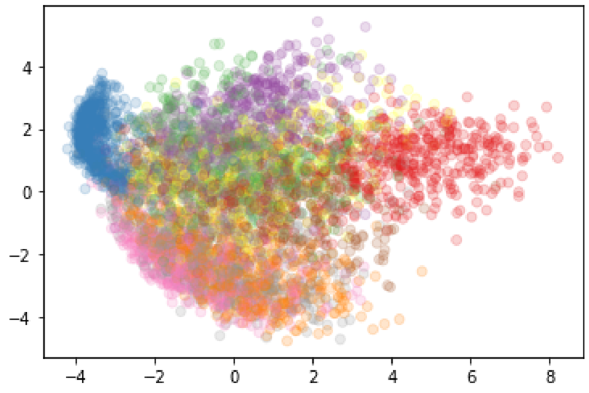
Part 2: Explaining the Methods used

1. **Logistic Regression (LR):** In Logical Regression, we estimate the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. The dependent variable will be bound by 0 and 1 as it returns a probability.
2. **K-nearest neighbors (KNN):** This is a supervised learning classification technique that at every step classifies data points based on proximity and predicts the grouping of these individual data points.
3. **Decision Trees (DT):** This is a classification algorithm which uses a tree-like structure where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each terminal node holds a class label.
4. **Random Forest (RF):** Random Forest is a classifier that contains several decision trees on various subsets of a given dataset and takes the average to enhance the

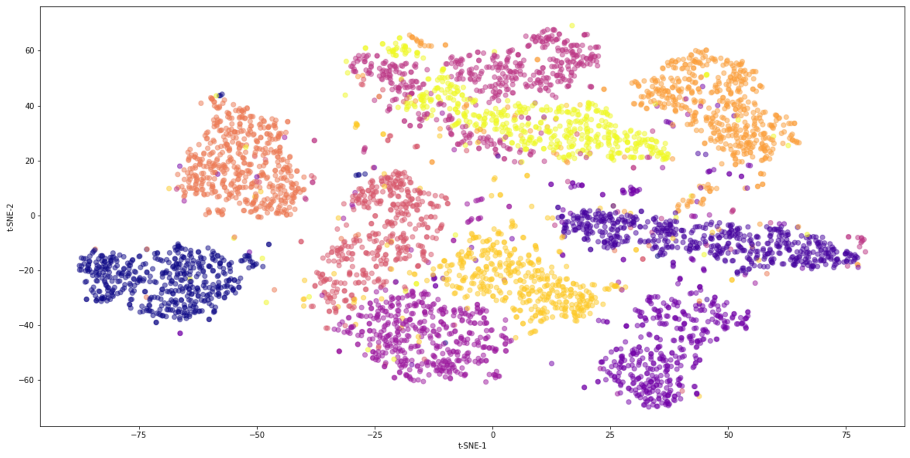
predicted accuracy of that dataset. Instead of relying on a single decision tree, the random forest collects the result from each tree and expects the final output based on the majority votes of predictions. Overfitting is less as compared to DTs since RFs are formed from subsets of data, and the final output is based on average or majority rating.

Part 3: Observations

Principal component analysis is a dimensionality reduction method allowing us for easier access to the data for visualization and computation while retaining crucial information. We reduced the dimensions from 784 to 2, and run t-SNE on the MNIST datasets to project from the original 784D space to 2D.



The 10 categories corresponding to the digits are well separated, but two classes (pink and yellow) remain entangled at the top: 4 and 9. This is consistent with the confusion matrices presented later, which tells us that 4 and 9 are commonly misclassified as each other.

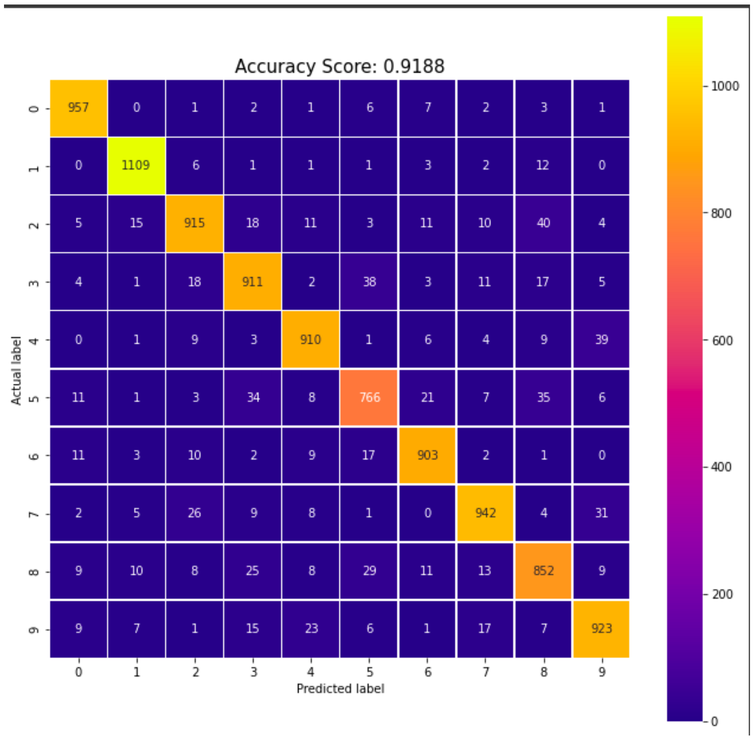


To perform the following algorithms, we first split both the training and testing data into their x and y components where y is essentially the class (i.e. the dependent variable) and and x is the pixel values (i.e. the independent variables). We used the sklearn and mlextend libraries to do the main ML aspect of this assignment.

1. **Multiclass Logistic Regression (MLR):** The hyperparameters used were Stochastic Average Gradient descent (sag) as the solver for the loss function and 25 as the maximum number of iterations this solver ran for. Sag essentially finds the best local minima for the loss function.

   We got an accuracy score of 91.88% with these parameters which is good enough for applications of digit recognition from pixelated images. The reason for our lower accuracy is because of the hyperparameters we chose. The sag solver prioritizes speed of execution over the most precise and accurate predictions. A solver like newton-cg would increase the accuracy but would be computationally more expensive. Furthermore, the L-BFGS solver works best on smaller datasets but with a dataset this large it risks not converging to anything. Hence, sag was the best option. We could also increase the accuracy by increasing the number of iterations we run the solver for but within 25 iterations, we manage to get a decently high accuracy and massive changes to this were only marginally increasing our accuracy so it did not make sense to increase the maximum iterations.

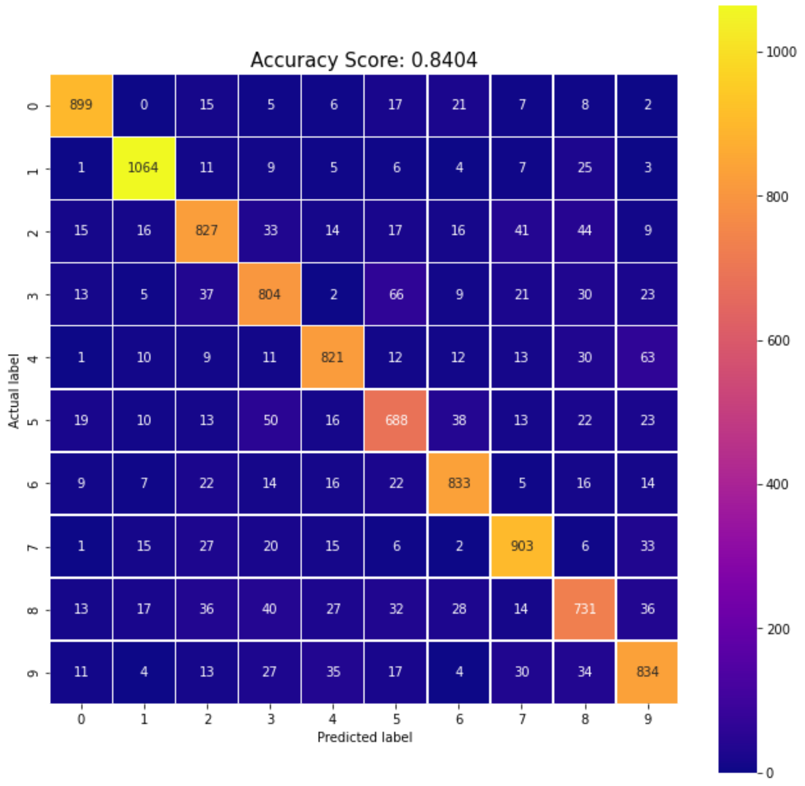
   We hypothesized that in simple models like our logistic regression model, there tends to be a higher bias and smaller variance. This hypothesis was proven true as we had a MSE of 1.528 with a bias (squared) of 1.106 and variance of 0.422. More complex models tend to trade off more variance for a lower bias.



Confusion Matrix for Logistic Regression.

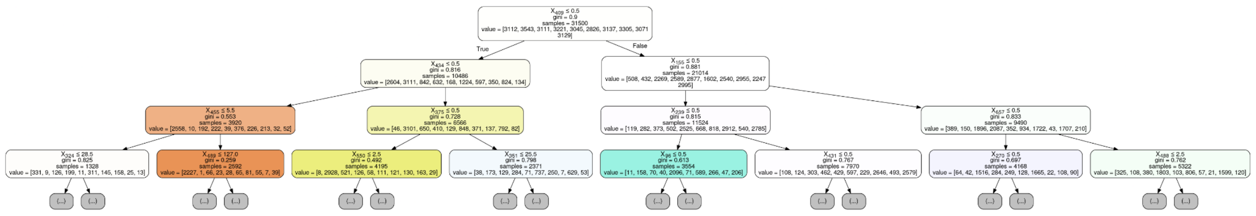
2. **Decision Tree:** We got an accuracy score of 84.04% with these parameters which is good enough for applications of digit recognition from pixelated images. We could increase the accuracy by increasing the number of iterations we run the solver for but within 10 iterations, we manage to get a decently high accuracy and massive changes to this were only marginally increasing our accuracy so it did not make sense to increase the maximum iterations. We hypothesized that in complex models like our decision tree model, there tends to be a lower bias and higher variance. This hypothesis was proven true as we had a MSE of 2.966 with a bias (squared) of 1.075 and variance of 1.891. More complex models tend to trade off more variance for a lower bias.

   Below is the confusion matrix for decision trees:

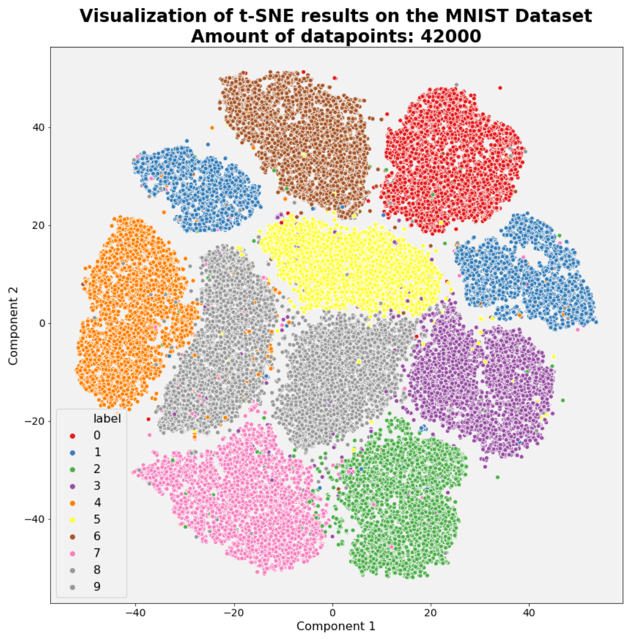


Decision Tree is not equipped to handle a lot of features. Since our data consists of a tabular format where every pixel is a feature, our initial decision tree algorithm gives us very low accuracy of 84.04%. Thus, we are trying to compress the data by first using Truncated Singular Value Decomposition (TSVD) and then using t-distributed Stochastic Neighbor Embedding (t-SNE) on the it to reduce the 784 features to 2 t-SNE features. We are using the scikit-learn library for the same and visualizing using the pydotplus python library.

Our baseline accuracy for the decision tree is 84.04% and the visualized decision tree looks as follows:



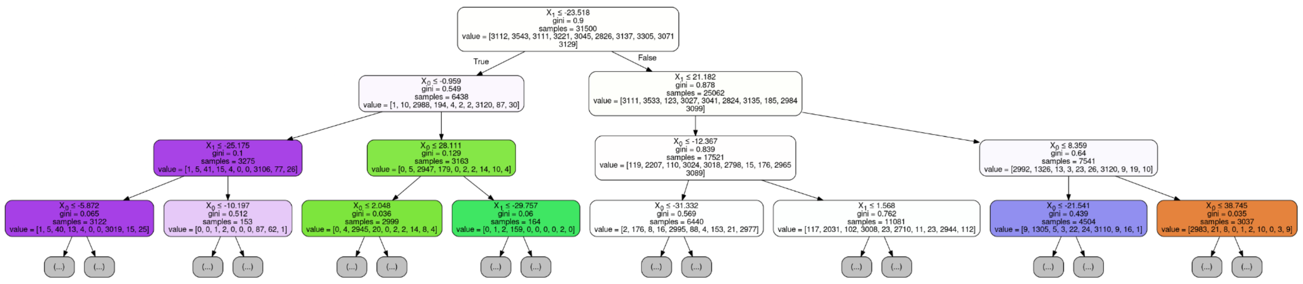
Then we compress the data using TSVD and get an accuracy of 79.41%. However, this is not our final accuracy. We did not apply t-SNE on our uncompressed data to preserve stability and computation. Since the data is sparse, we are using TSVD to compress the data. Otherwise, Principal Component Analysis (PCA) is often a better dimensionality reduction technique. The TSVD compression helps us reduce the 784 pixel features to 50 features. Now, we train the t-SNE algorithm on the training and testing data as t-SNE is unsupervised. We can visualize the classes after compression as follows:



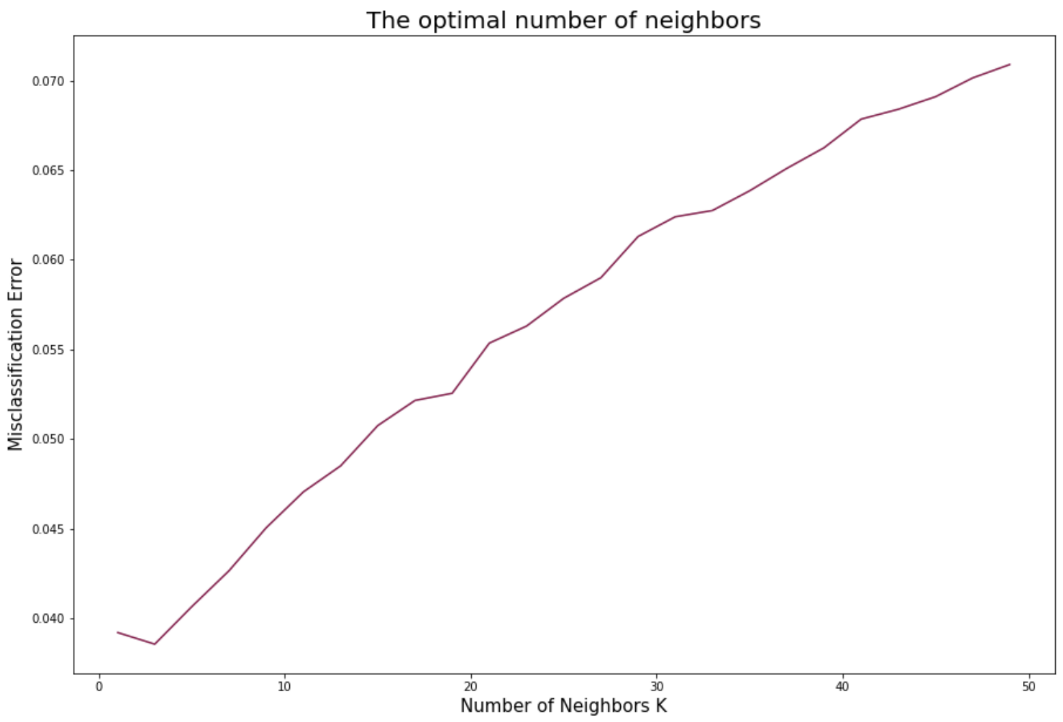
We can see that the classes are distinctly represented in the 2-D space. Moreover, we know that 7 and 1 in the MNIST dataset are pretty similar and that is represented in the t-SNE classes visualization.

Limitations: Making classes using compressed data for a Decision Tree model can make the model less interpretable as you don't know which feature is made up of what compressed features. However, this can be easily countered by compressing similar and highly correlated groups of features.

Training our Decision Tree on this compressed data, we get an accuracy of 98.71% which is an improvement of 12.44% over the baseline.


X₁ ≤ -23.518
gini = 0.9
samples = 31500
value = [3112, 3543, 3111, 3221, 3045, 2826, 3137, 3305, 3071 3129]
True
False
X₀ ≤ -0.959
gini = 0.549
samples = 6438
value = [1, 10, 2988, 194, 4, 2, 2, 3120, 87, 30]
X₁ ≤ 21.182
gini = 0.878
samples = 25062
value = [3111, 3533, 123, 3027, 3041, 2824, 3135, 185, 2984 3099]
X₁ ≤ -25.175
gini = 0.1
samples = 3275
value = [1, 5, 41, 15, 4, 0, 0, 3106, 77, 26]
X₀ ≤ 28.111
gini = 0.129
samples = 3163
value = [0, 5, 2947, 179, 0, 2, 2, 14, 10, 4]
X₀ ≤ -12.367
gini = 0.839
samples = 17521
value = [119, 2207, 110, 3024, 3018, 2798, 15, 176, 2965 3089]
X₀ ≤ 8.359
gini = 0.64
samples = 7541
value = [2992, 1326, 13, 3, 23, 26, 3120, 9, 19, 10]
X₀ ≤ -5.872
gini = 0.065
samples = 3122
value = [1, 5, 40, 13, 4, 0, 0, 3019, 15, 25]
X₀ ≤ -10.197
gini = 0.512
samples = 153
value = [0, 0, 1, 2, 0, 0, 0, 87, 62, 1]
X₀ ≤ 2.048
gini = 0.036
samples = 2999
value = [0, 4, 2945, 20, 0, 2, 2, 14, 8, 4]
X₁ ≤ -29.757
gini = 0.06
samples = 164
value = [0, 1, 2, 159, 0, 0, 0, 0, 2, 0]
X₀ ≤ -31.332
gini = 0.569
samples = 6440
value = [2, 176, 8, 16, 2995, 88, 4, 153, 21, 2977]
X₁ ≤ 1.568
gini = 0.762
samples = 11081
value = [117, 2031, 102, 3008, 23, 2710, 11, 23, 2944, 112]
X₀ ≤ -21.541
gini = 0.439
samples = 4504
value = [9, 1305, 5, 3, 22, 24, 3110, 9, 16, 1]
X₀ ≤ 38.745
gini = 0.035
samples = 3037
value = [2983, 21, 8, 0, 1, 2, 10, 0, 3, 9]
(...)

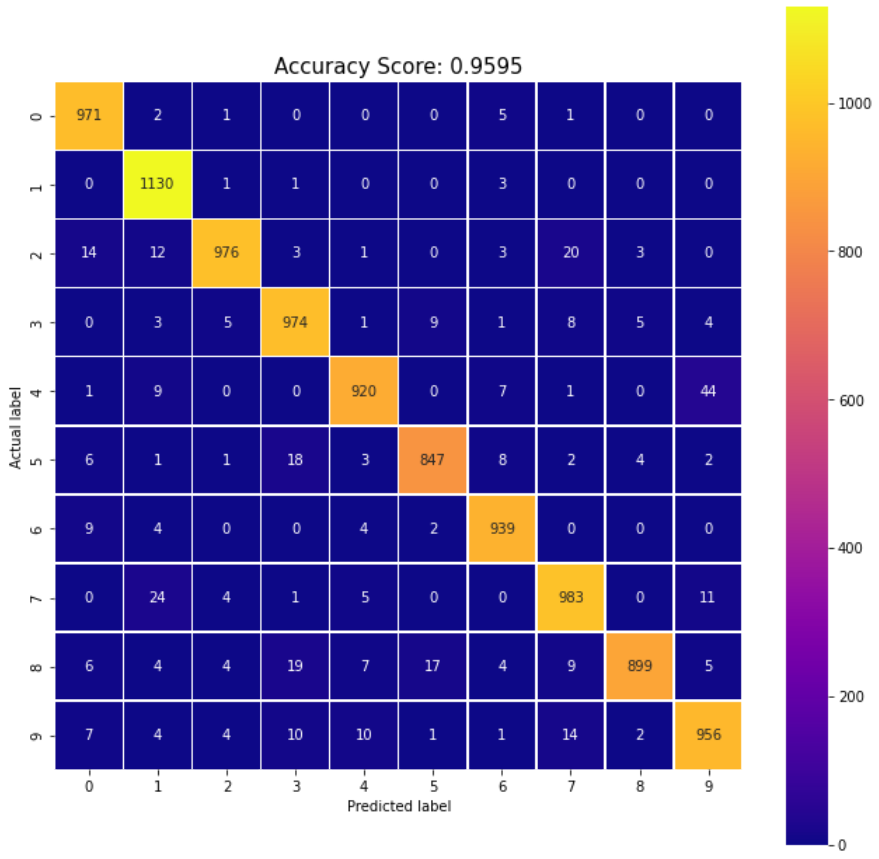
3. **K-nearest neighbors:** The hyperparameters included number of neighbors, power parameters, weights, etc. The n_neighbours used was 3, after running the optimal number of neighbors weighed against the miscalculation errors.



The power parameter was set to the default of p= 2, which was equivalent to the Euclidean distance.

KNN is a simple model so we expect high bias and low variance as it tends to generalize over the data and is not too sensitive to tiny fluctuations in training data. We got an MSE of 0.942, Bias (squared) of 0.620 and Variance of 0.322. This confirms our hypothesis.

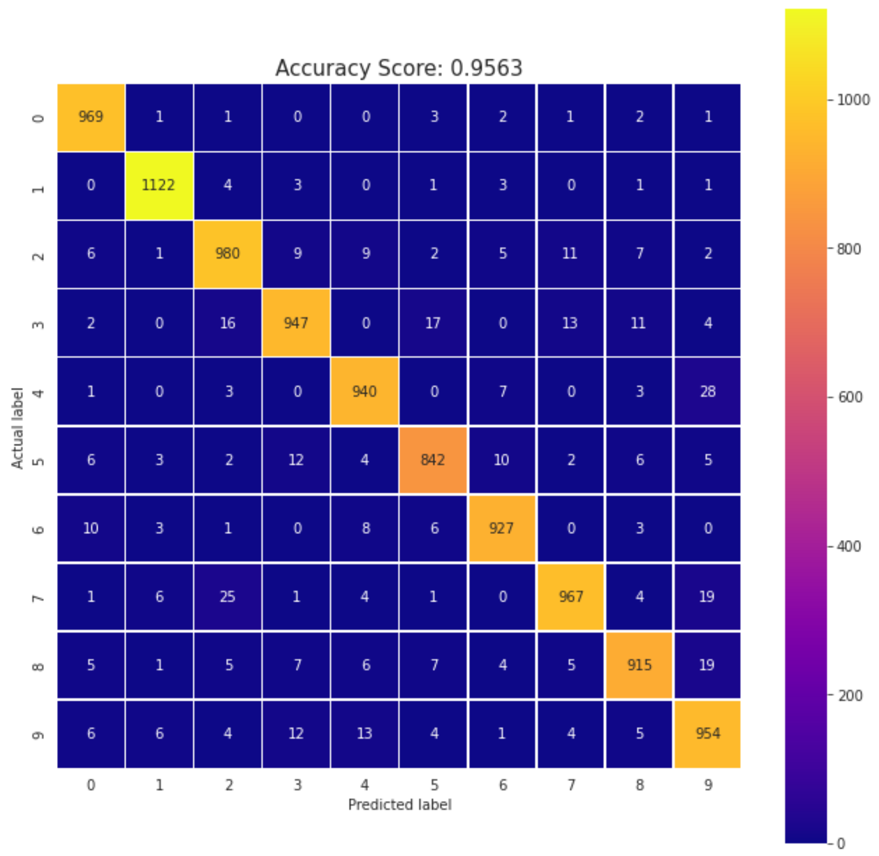
The confusion matrix is as follows:



4. **Random Forest Classification:** The *hyperparameters* include number of trees, depth of tree, maximum features and so on. By increasing the number of trees (estimators) we can create a more robust aggregate model. In this case, the tuning led us to pick 100 such estimators. We can also restrict the maximum depth of each tree which allows us to prevent overfitting. This is so because without restriction the tree will grow on forever until it fully classifies each point. In this case, the optimum depth that gave us this accuracy while preserving generalisability was 14 levels. One may also restrict the maximum features and maximum samples of training data for increasing efficiency. Taking `n_estimators as 100 and max_depth as 14` we get an accuracy of approximately **96%**.

   Advantages and Limitations: A Random forest creates multiple trees with random features, and the trees are not very deep, preventing overfitting as compared to decision trees. It also maximizes the efficiency as it averages the trees, providing generalized results. Therefore, Random Forest has better accuracy and efficiency than a decision tree, but this comes at a cost of time, storage and computational power.

The confusion matrix is as follows:



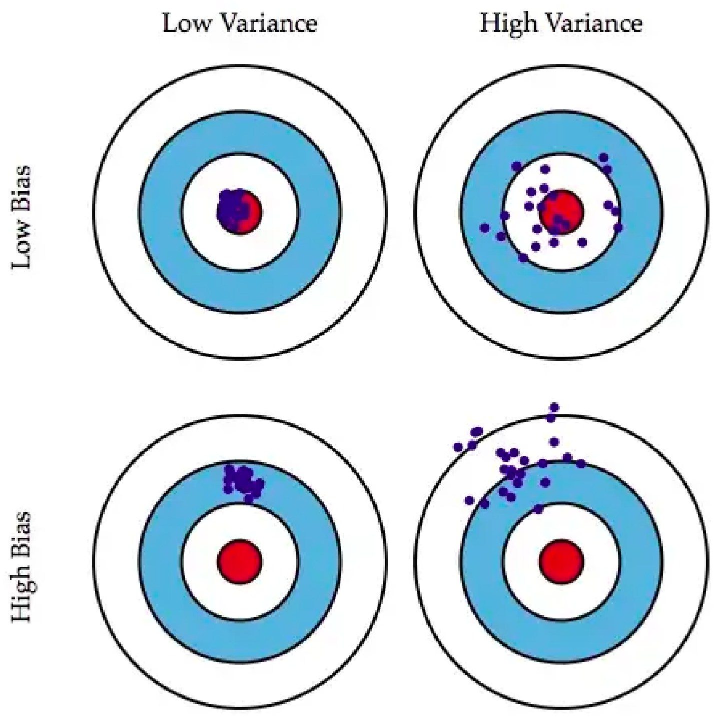
Actual label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	969	1	1	0	0	3	2	1	2	1
1	0	1122	4	3	0	1	3	0	1	1
2	6	1	980	9	9	2	5	11	7	2
3	2	0	16	947	0	17	0	13	11	4
4	1	0	3	0	940	0	7	0	3	28
5	6	3	2	12	4	842	10	2	6	5
6	10	3	1	0	8	6	927	0	3	0
7	1	6	25	1	4	1	0	967	4	19
8	5	1	5	7	6	7	4	5	915	19
9	6	6	4	12	13	4	1	4	5	954

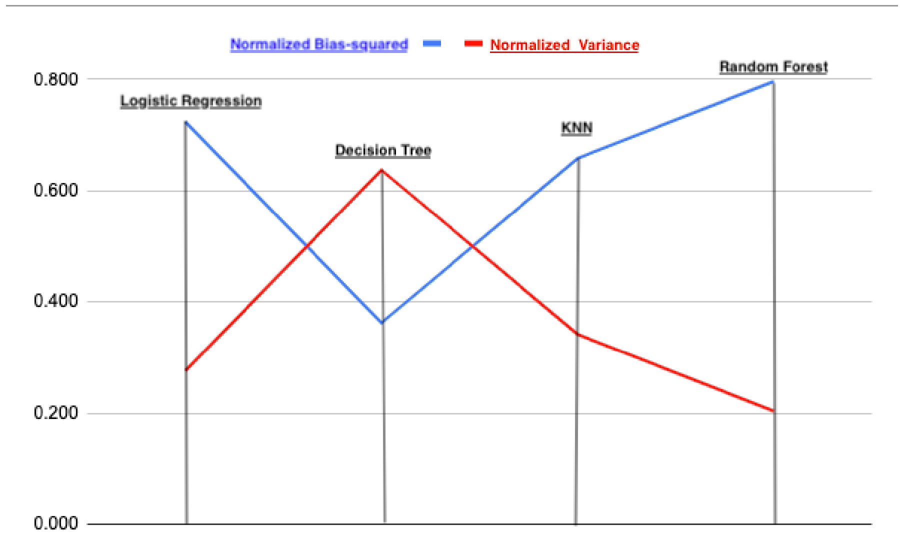
According to the confusion matrix we see that some possible *mis-classification* is 9 and 4, 9 and 7, 7 and 2, 5 and 3. The highest of them is 4 and 9, which is due to the similar structure of the digits, where there is an upper loop and a line. The same goes for 7 and 2 in the way they are written, leading to a consistency in mis-classification.

Bias-Variance tradeoff: Random forest algorithms follow low bias and high variance, but both values are still lesser for random forest than decision tree. Random forest will convert a given low bias high variance model (decision trees) into a low bias low variance model, i.e. increases generalisability. For the given model, the error metrics are as follows – `MSE: 0.871, Bias: 0.695, Variance: 0.177`

This is consistently less than the metrics of the aforementioned *decision tree algorithm*, supporting the argument that random forest algorithms have lower bias and variance than decision trees, and are thus more efficient.

Comparing The Bias and Variances of the Algorithms





The blue line represents the normalized bias-squared of our four algorithms while the red line represents the normalized variance of the same. From this graph we can see that there is a clear tradeoff between bias and variance. No model can have a decrease in one without an increase in the other. It also shows that simple models like Logistic regression and KNN have a higher bias and lower variance while for a more complex model like Decision Tree, the converse is true. Random Forest is a special case where it aims at minimizing the variance through a generalization of a variety of decision trees.

Lessons on hands-on Machine Learning

- We require diverse data to make accurate models for the real-world, resulting in fewer false positives and false negatives.
- Explainability and transparency of hyperparameters, and the fairness of our models is important when it comes to fairly distributed decision making.
- Safety and privacy have to be maintained while forming these models. One must avoid transferring bias, which comes with the assumption that past data speaks for the future.

Google Collab Link

https://colab.research.google.com/drive/1VVlSrNe_FLXYnRnrq9aToMyWZ2Yjmuv5?usp=sharing

REFERENCES

1. https://www.kaggle.com/competitions/digit-recognizer/data?select=test.csv
2. https://en.wikipedia.org/wiki/MNIST_database
3. https://www.turing.com/kb/random-forest-algorithm
4. https://www.ibm.com/in-en/topics/knn
5. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e805
6. https://www.mygreatlearning.com/blog/random-forest-algorithm
7. https://thedatafrog.com/en/articles/visualizing-datasets/
8. https://www.kaggle.com/code/devchauhan1/digit-recognition-with-logistic-regression/notebook
9. https://medium.com/analytics-vidhya/handwritten-digit-recognition-using-logistic-regression-8d3b3f7e31c0#:~:text=Logistic%20Regression%20is%20the%20Supervis

ed,digits%20(0%20to%209)
10. https://realpython.com/logistic-regression-python/
11. https://towardsdatascience.com/logistic-regression-in-python-f66aeb15e83e
12. https://stackoverflow.com/questions/62658215/convergencewarning-lbfgs-failed-to-converge-status-1-stop-total-no-of-iter
13. https://machinelearningmastery.com/calculate-the-bias-variance-trade-off/