Classifying Digits with ML

Isha Singh

MSDS 422 – Practical Machine Learning

Irene Tsapara

26 February 2025

I. <u>OVERVIEW AND METHODOLGY</u>

The following research is based on the Kaggle Competition: Digit Recognizer. The main goal of this study is to classify handwritten numbers (0-9) using machine learning models, specifically Random Forest classifiers and K-Means. Different model factors will be evaluated by modifying hyperparameters, applying PCA for feature selection, and refining the model. The study will analyze performance metrics and validation scores to determine which model provides the most accurate and precise results. This will be done by reviewing validation accuracy, generating confusion matrices, and evaluating clustering results to better understand the model's performance.

The dataset used has 42,000 training images and 28,000 test images, where each specific image is 28 x 28 pixels in grayscale with values that range from 0 to 255 (Digit Recognizer). The value 0 represents white, and 255 represents black. The training set includes a label column that indicates the digits 0-9; however, the test dataset incorporates only test set image data. Some of the observations that were noticed during the procedure were that no missing values were found within the dataset, each of the digits from 0-9 is explicitly there, and there were various different samples of handwriting styles.

To have the data become ready, data preprocessing and feature engineering were important. MinMaxScaler was used in order to normalize pixel values between 0 and 1. PCA was used to reduce the dimensionality (by removing 5 percent of the variance but keeping the 95 percent), which resulted in optimizing efficiency without worrying about how much was gone. Furthermore, a few data augmentation techniques that were used are rotation, shifting, and zooming to enhance and create some sort of generalization across different and unique handwriting styles.

Afterwards, EDA or exploratory data analysis was conducted in order to understand the samples visually and get a clear representation of different types of handwriting. We examined the average and variance, samples, as well as pixel intensity distributions.

The digit recognizer employed both supervised and unsupervised learning models to be able to understand their perspective on classification strategies. Random Forest classifiers had the strongest performance compared to the other models: KMeans, Random Forest with PCA, and modified KMeans (an approach done with KMeans and RandomForest).

To optimize the performance of the models, hyperparameter tuning was performed with the help of GridSearchCV for the RandomForest. The hyperparameters that were chosen were the number of trees, depth, and split criteria. For KMeans, it was focused upon the number of clusters or K, and with the help of silhouette scores and inertia analysis, the efficacy was understood.

Overall, answering the management question is that the goal of this machine learning research was to be able to develop an efficient model that will recognize different and unique handwritten digits. This is useful in the outside world to be able to understand when in work scenario one is not able to understand a specific numeric. Improving accuracy with the help of feature selection and clustering will help create models that will be able to assist onto the next step: generalizing handwritten digits, which is crucial for real world and practical settings.

II. DETAILED STEPS AND CODE EXPLANATION

After the preprocessing part of the research is completed, the next part is model training. The four models that were focused upon were Random Forest with the entire dataset, Random Forest with 95 percent focus (PCA removed 5 percent for dimensionality), K-Means clusters, and K-Means with the help of Random Forest. Each of these models had separate training, testing, and validation splits for organization purposes.

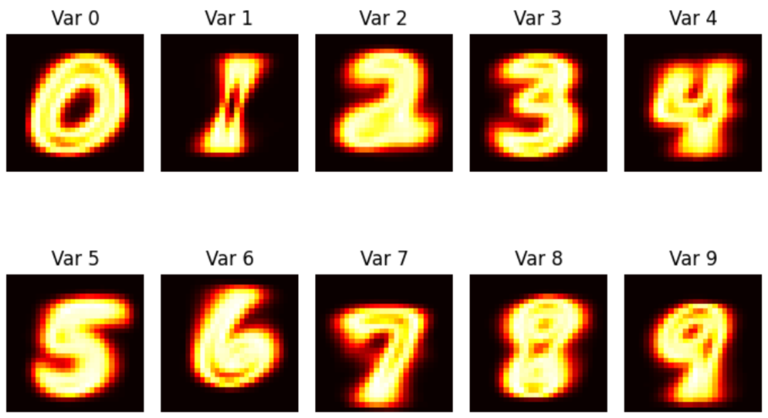
The dataset was split for Random Forest in 80-20, or 80 for training, 10 for testing, and 10 for validating using the popular method Stratified Train-Test Splitting (Scikit-learn). PCA-based Random Forest had the same approach as both Model 1 and Model 2 used Random Forest, which is a supervised learner. However, since Model 3 is unsupervised, which is not dependent on labeled data, splitting was not needed. For Model 4, the K-Means with Random Forest required no splitting, but an extra step was made for the process to become easier. The cluster labels that were found from K-Means were mapped to actual digits before training with Random Forest, which is why splitting was not required.

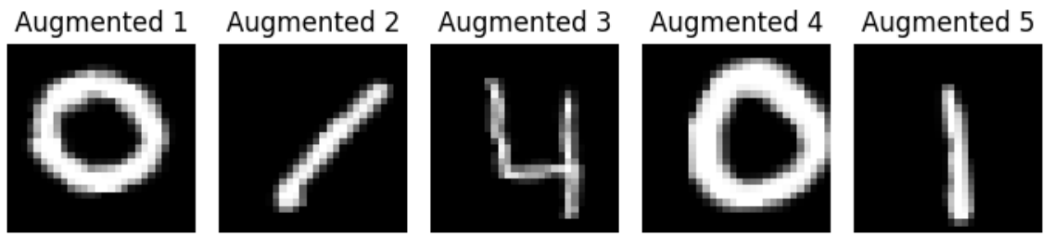
During the training process, different types of approaches were given. The Random Forest had the default parameters (ChatGPT) to understand the entire model procedure. Cross-validation using KFold was performed with a focus on 5 splits, and that was important as it was required to make sure there was no bias for data splits. After the initial training was done, hyperparameters were tuned with the help of GridSearchCV, which took a long time, but the time was cut short with the help of n_jobs=-1, as it allowed parallel processing. Moreover, the best parameters then improved the model accuracy, which improved the model accuracy.

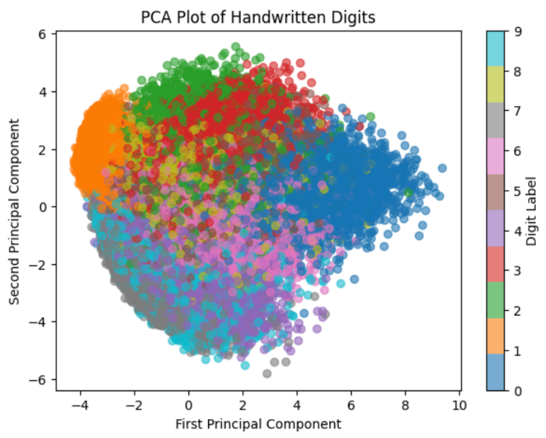
For K-Means, the most important part was understanding the number of clusters or K. Instead of choosing a fixed value, different values of K were tested by measuring Silhouette Scores and Inertia. The Elbow Method was used later to find the point or location where there is no significance if more clusters are found. Model 3 did not perform well as it was getting confused in classifying digits, and the same went with Model 4.

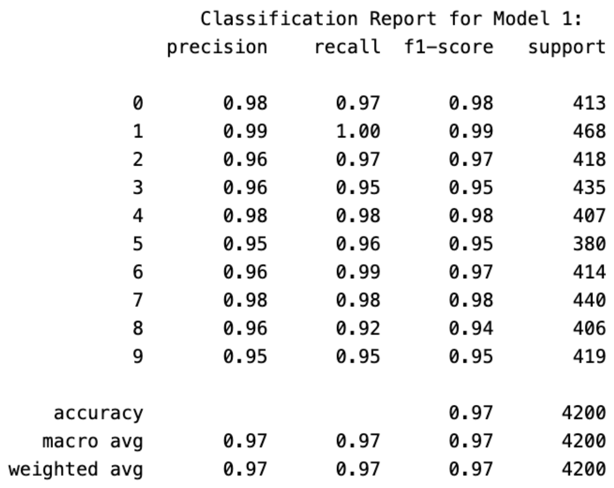
Once training was complete, model performance was evaluated by seeing different metrics, and the results were sent to Kaggle. These were the scores: Random Forest using the full dataset achieved the highest accuracy (96.46%), PCA-based Random Forest (94.26%), and K-Means clustering performed the worst (23.64%). The fourth model had the same result as K-Means.

III. VISUAL AND TABULAR INFORMATION









```
                Classification Report for Model 1:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       413
           1       0.99      1.00      0.99       468
           2       0.96      0.97      0.97       418
           3       0.96      0.95      0.95       435
           4       0.98      0.98      0.98       407
           5       0.95      0.96      0.95       380
           6       0.96      0.99      0.97       414
           7       0.98      0.98      0.98       440
           8       0.96      0.92      0.94       406
           9       0.95      0.95      0.95       419

    accuracy                           0.97      4200
   macro avg       0.97      0.97      0.97      4200
weighted avg       0.97      0.97      0.97      4200
```

Since there were a lot of EDA visualizations created for the research, I selected the four most important that I believed showcase the most important key insights and effectively highlight the structure, impact, and importance. The visualizations involve pixel intensity, data augmentation, the PCA scatterplot, and lastly, the classification report for Random Forest, which was the strongest model.

The first visualization shows the variance of the different types of handwritten digits, which helps in understanding the variations in pixel intensity across a variety of digits. The second visualization focuses on data augmentation, where it used the following factors: rotation, width shift, and height shift (Scikit-learn). This produced different variations for us to be able to understand how a digit looks when it is rotated.

The third visualization is a PCA scatterplot, where it shows the comparison of high-dimensional pixel data transformed into a lower-dimensional space while still keeping the key variance. The colors represent the different numbers (0-9), and it demonstrates how well separated they are within a 2D space. The fourth visualization shows the classification report for the strongest model, which was the first model: Random Forest. The classification report presents precision, recall, and F1-scores for each unique digit class. The model got an accuracy score of 97 percent, which was concluded to show it is very high with recall and high precision. The classification report shows that digits 3 and 8 have slightly lower values, which means sometimes they can be misclassified.

To summarize, these four visualizations have separate benefits and provide an important and comprehensive overview, as they support the findings of the study.

IV. **CONCLUSION**

Overall, this research was a great way to learn how each model works with handwriting recognition. The Random Forest model, which was the first model, gave the best accuracy overall with a score of 0.96460, meaning the hyperparameters worked well. Secondly, the Random Forest with PCA had a lower score of 0.62010 as we had to reduce dimensions, which possibly removed important information. K-means clustering was the third model with a score of 0.23646 since it did not use labeled data and instead clustered similar data, making it difficult to match the clusters to the digits.The last model was improved through a K Means approach and then afterward using the random forest approach once the clusters were matched to digit labels. However, unfortunately, after testing this experiment out, the model's score was the same. That overall meant that the model was still not able to capture the classification of the digits accurately. Therefore, in the future, focusing on hyperparameters and using alternative methods will be better.

V. REFERENCES

OpenAI. *ChatGPT.* Accessed February 26, 2025. https://chatgpt.com/.

Scikit-learn. "Confusion Matrix." *Scikit-learn Documentation.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

Kaggle. "Digit Recognizer." *Kaggle Competitions.* Accessed February 26, 2025. https://www.kaggle.com/competitions/digit-recognizer.

GeeksforGeeks. "Python Most_common () Function." *GeeksforGeeks*, March 20, 2024. https://www.geeksforgeeks.org/python-most_common-function/.

GeeksforGeeks. "Recovering Feature Names of Explained_variance_ratio_ in PCA with Sklearn." *GeeksforGeeks*, June 27, 2024. https://www.geeksforgeeks.org/recovering-feature-names-of-explainedvarianceratio-in-pca-with-sklearn/.

Scikit-learn. "KMeans." *Scikit-learn Documentation.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

Scikit-learn. "MinMaxScaler." *Scikit-learn Documentation.* Accessed February 26, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

Scikit-learn. "PCA." *Scikit-learn Documentation.* Accessed February 22, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

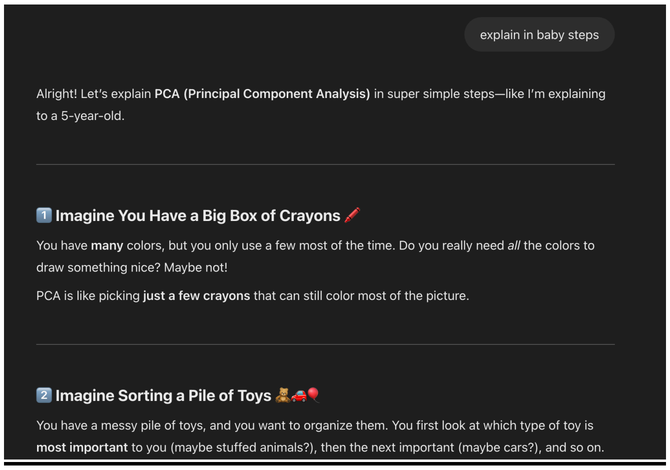
Scikit-learn. “RandomForestClassifier.” *Scikit-learn Documentation.* Accessed February 25, 2025. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

VI. APPENDIX

LINK TO PANAPTO

- https://northwestern.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=1a6f7658-baae-4667-b013-b2910042aa6b&start=0

CHATGBT SCREENSHOTS



explain in baby steps

Alright! Let's explain **PCA (Principal Component Analysis)** in super simple steps—like I'm explaining to a 5-year-old.

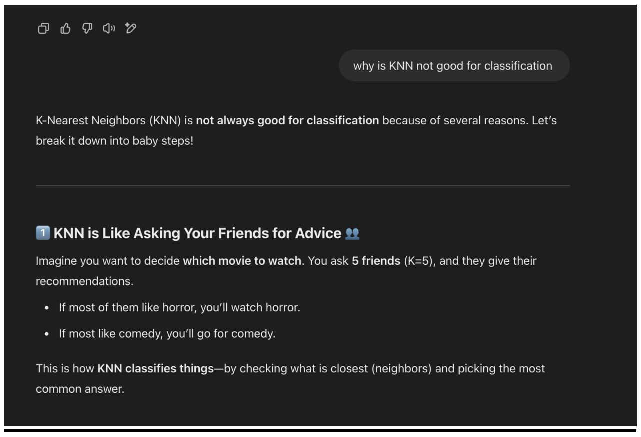
1️⃣ Imagine You Have a Big Box of Crayons 🖍️

You have **many** colors, but you only use a few most of the time. Do you really need *all* the colors to draw something nice? Maybe not!

PCA is like picking **just a few crayons** that can still color most of the picture.

2️⃣ Imagine Sorting a Pile of Toys 🧸🚗🎈

You have a messy pile of toys, and you want to organize them. You first look at which type of toy is **most important** to you (maybe stuffed animals?), then the next important (maybe cars?), and so on.



why is KNN not good for classification

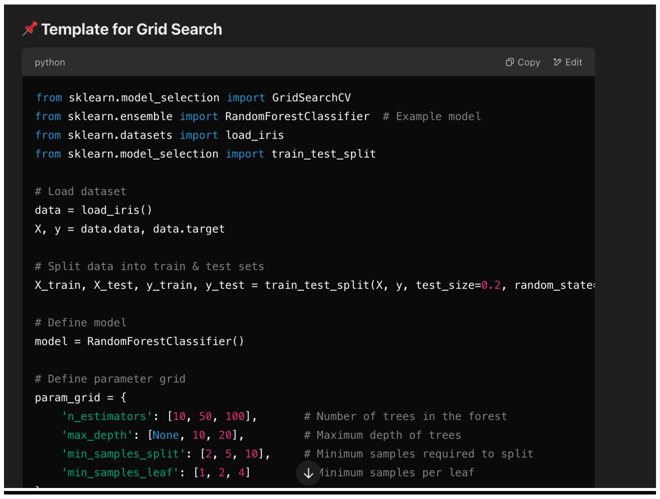
K-Nearest Neighbors (KNN) is **not always good for classification** because of several reasons. Let's break it down into baby steps!

1️⃣ KNN is Like Asking Your Friends for Advice 👥

Imagine you want to decide **which movie to watch**. You ask **5 friends** (K=5), and they give their recommendations.

- If most of them like horror, you'll watch horror.
- If most like comedy, you'll go for comedy.

This is how **KNN classifies things**—by checking what is closest (neighbors) and picking the most common answer.



📌 Template for Grid Search

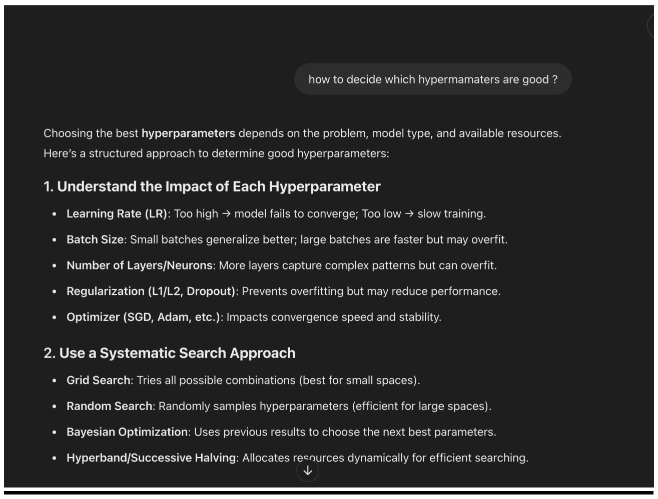
```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier  # Example model
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Define model
model = RandomForestClassifier()

# Define parameter grid
param_grid = {
    'n_estimators': [10, 50, 100],       # Number of trees in the forest
    'max_depth': [None, 10, 20],         # Maximum depth of trees
    'min_samples_split': [2, 5, 10],     # Minimum samples required to split
    'min_samples_leaf': [1, 2, 4]        # Minimum samples per leaf
```



how to decide which hypermamaters are good ?

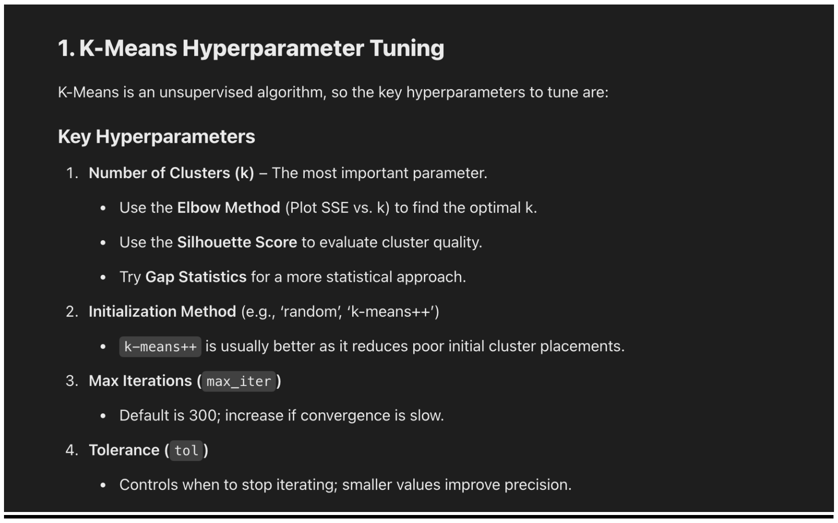
Choosing the best **hyperparameters** depends on the problem, model type, and available resources. Here's a structured approach to determine good hyperparameters:

1. Understand the Impact of Each Hyperparameter

- **Learning Rate (LR)**: Too high → model fails to converge; Too low → slow training.
- **Batch Size**: Small batches generalize better; large batches are faster but may overfit.
- **Number of Layers/Neurons**: More layers capture complex patterns but can overfit.
- **Regularization (L1/L2, Dropout)**: Prevents overfitting but may reduce performance.
- **Optimizer (SGD, Adam, etc.)**: Impacts convergence speed and stability.

2. Use a Systematic Search Approach

- **Grid Search**: Tries all possible combinations (best for small spaces).
- **Random Search**: Randomly samples hyperparameters (efficient for large spaces).
- **Bayesian Optimization**: Uses previous results to choose the next best parameters.
- **Hyperband/Successive Halving**: Allocates resources dynamically for efficient searching.

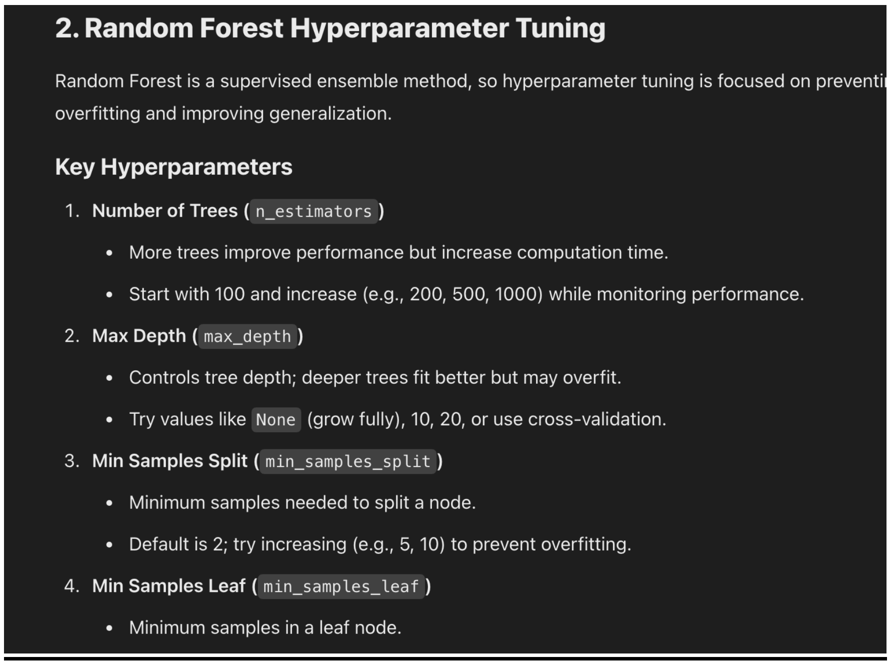


1. K-Means Hyperparameter Tuning

K-Means is an unsupervised algorithm, so the key hyperparameters to tune are:

Key Hyperparameters

1. **Number of Clusters (k)** – The most important parameter.
   - Use the **Elbow Method** (Plot SSE vs. k) to find the optimal k.
   - Use the **Silhouette Score** to evaluate cluster quality.
   - Try **Gap Statistics** for a more statistical approach.
2. **Initialization Method** (e.g., 'random', 'k-means++')
   - `k-means++` is usually better as it reduces poor initial cluster placements.
3. **Max Iterations (`max_iter`)**
   - Default is 300; increase if convergence is slow.
4. **Tolerance (`tol`)**
   - Controls when to stop iterating; smaller values improve precision.



2. Random Forest Hyperparameter Tuning

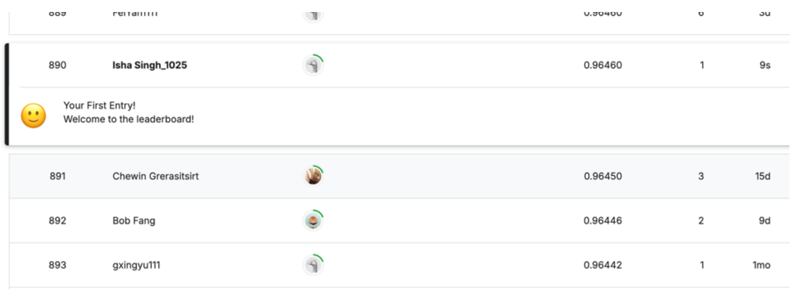
Random Forest is a supervised ensemble method, so hyperparameter tuning is focused on preventi
overfitting and improving generalization.

Key Hyperparameters

1. **Number of Trees (`n_estimators`)**
   - More trees improve performance but increase computation time.
   - Start with 100 and increase (e.g., 200, 500, 1000) while monitoring performance.
2. **Max Depth (`max_depth`)**
   - Controls tree depth; deeper trees fit better but may overfit.
   - Try values like `None` (grow fully), 10, 20, or use cross-validation.
3. **Min Samples Split (`min_samples_split`)**
   - Minimum samples needed to split a node.
   - Default is 2; try increasing (e.g., 5, 10) to prevent overfitting.
4. **Min Samples Leaf (`min_samples_leaf`)**
   - Minimum samples in a leaf node.

<u>KAGGLE SCORES</u>

<u>MODEL 01 – RANDOM FOREST</u>

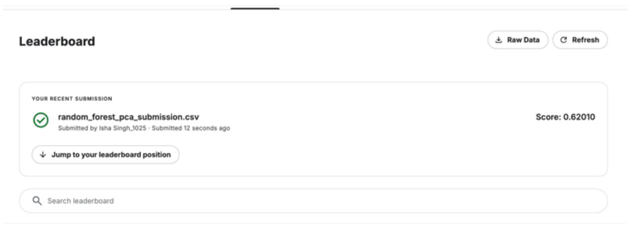


889	FeiYan1111	0.96460	6	3d
890	**Isha Singh_1025**	0.96460	1	9s

Your First Entry!
Welcome to the leaderboard!

891	Chewin Grerasitsirt	0.96450	3	15d
892	Bob Fang	0.96446	2	9d
893	gxingyu111	0.96442	1	1mo

<u>MODEL 02 – RANDOM FOREST WITH PCA</u>

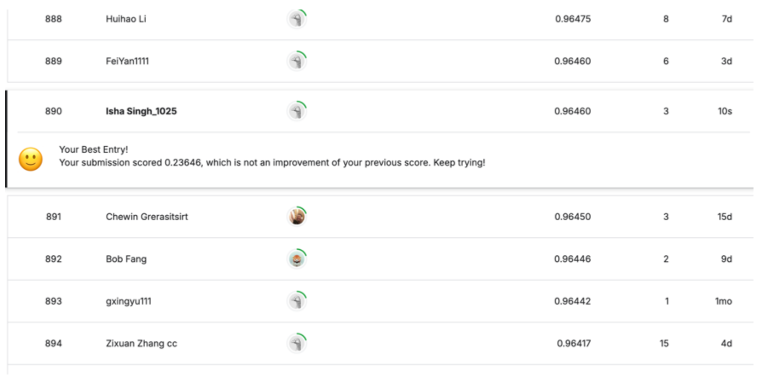


YOUR RECENT SUBMISSION

random_forest_pca_submission.csv
Submitted by Isha Singh_1025 · Submitted 12 seconds ago

Score: 0.62010

<u>MODEL 03 – KMEANS</u>

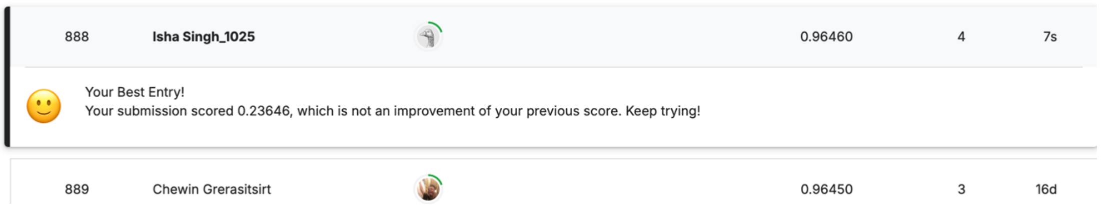


888	Huihao Li	0.96475	8	7d
889	FeiYan1111	0.96460	6	3d
890	**Isha Singh_1025**	0.96460	3	10s

Your Best Entry!
Your submission scored 0.23646, which is not an improvement of your previous score. Keep trying!

891	Chewin Grerasitsirt	0.96450	3	15d
892	Bob Fang	0.96446	2	9d
893	gxingyu111	0.96442	1	1mo
894	Zixuan Zhang cc	0.96417	15	4d

MODEL 04 – KMEANS MODIFIED



888	**Isha Singh_1025**	0.96460	4	7s

Your Best Entry!
Your submission scored 0.23646, which is not an improvement of your previous score. Keep trying!

889	Chewin Grerasitsirt	0.96450	3	16d