

BUDAPEST UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONLINE SIGNATURE VERIFICATION USING MACHINE
LEARNING

Submitted by: Basel Al-Raoush

Student ID: SDR1AO

Supervisor: Dr. Mohammed Saleem

Date: May, 2025

Abstract.....	3
Introduction.....	4
Methodology.....	5
Models Used and Rationale.....	7
Logistic Regression.....	7
Support Vector Machine (SVM).....	7
k-Nearest Neighbors (KNN).....	7
Random Forest.....	7
XGBoost.....	7
MLP (Multi-Layer Perceptron).....	8
Voting Classifier (Ensemble).....	8
Stacking Classifier.....	8
Evaluation Metrics.....	9
Accuracy.....	9
Precision and Recall.....	9
F1 Score.....	9
Equal Error Rate (EER).....	9
Experiments.....	10
Phase 1: Exploratory Data Analysis (EDA).....	10
Dataset Overview.....	10
Pressure Analysis.....	10
Trajectory and Pressure Visualization.....	11
Pen Orientation – Pressure, Altitude and Azimuth.....	11
Phase 2: Baseline Models (Logistic Regression, SVM, Random Forest).....	12
With and Without X/Y Features.....	12
Adding X and Y Features.....	13
Phase 3: Improved Pipeline with KNN and Random Forest.....	14
Phase 4 : Advanced Models: XGBoost, MLP, and a Voting Ensemble.....	16
XGBoost Classifier.....	16
MLP Classifier.....	16
Voting Classifier (XGBoost + MLP).....	16
Confusion Matrix and EER.....	17
Phase 5 – Hyperparameter Tuning and Model Stacking.....	18
Grid Search: XGBoost and MLP.....	18
Weighted Voting Ensemble.....	18
Stacking Classifier.....	19
Phase 6 – Benchmarking Models Across Multiple Datasets.....	20
MCYT.....	20
Chinese Dataset.....	20
Dutch Dataset.....	20
German Dataset.....	20
SVC Dataset.....	21
Cross-Dataset Trends and Takeaways.....	21
Results & Discussion.....	23
Conclusion.....	23
References.....	24

Abstract

This project explores the use of machine learning to verify online handwritten signatures, distinguishing genuine signatures from forgeries. We tackle the problem by extracting features from the dynamic signing process (e.g. pen coordinates, pressure, angles) and training various classifiers to decide if a given signature is authentic or fake. The goal is to find a model that achieves high accuracy in spotting forgeries while minimizing false rejections of genuine signatures. We evaluated multiple algorithms from logistic regression and SVM to ensemble methods and neural networks on several datasets (MCYT and four other signature datasets in Chinese, Dutch, German, and SVC). Our approach involved systematic experiments: initial data analysis, baseline modeling, incremental improvements with advanced models and tuning, and finally benchmarking across datasets. The best-performing model achieved an F1 score in the mid-80% range with a low Equal Error Rate around 0.16 on the primary dataset, and the cross-dataset tests revealed that model performance can vary with the nature of the signatures. Overall, the project demonstrates that machine learning methods can effectively verify online signatures, and it highlights which features and models contribute most to accurate verification.

Introduction

Online signature verification is the process of using dynamic handwriting data to confirm a writer's identity [1]. Unlike a static image of a signature (used in offline verification), online signature data is captured in real time via devices like tablets or digital pens. This means we can record not only the shape of the signature but also how it was made, for example, the sequence of pen movements, the pressure applied, and the pen's orientation. These biometric traits are difficult for an impostor to mimic exactly, making online signature verification a valuable security tool [2]. In real-world scenarios, it can be used for identity authentication in banking (e.g. verifying signatories of digital transactions), secure document signing, and as a forensic aid to detect signature fraud [1]. It provides a convenient yet reliable alternative to passwords or PINs for confirming a person's identity.

In this project, we focus on verifying genuine vs. forged signatures using machine learning models trained on online signature data. A genuine signature is one produced by the claimed person, while a forged signature is an attempt to imitate someone else's signature. The goal is to train a model that can learn subtle patterns in pressure, movement, and pen behavior that tend to differ between authentic signatures and forgeries. We use multiple datasets to test the robustness of our approach. Our primary dataset is MCYT [3], a public online signature dataset that includes dynamic signing data such as pen pressure (P), pen altitude angle (al), azimuth angle (az), and statistical summaries of pen movement in the X and Y directions. The dataset consists of samples from 100 individuals, each providing several genuine signatures and multiple forgeries written by others. In addition to MCYT, we evaluate our models on four other datasets: a Chinese, Dutch, German, and SVC dataset (from the Signature Verification Competition). These datasets vary in size, language background, and balance between genuine and forged signatures. By testing across them, we can better understand how well our models generalize to different writing styles and forgery challenges. Some datasets are balanced, while others have uneven distributions between classes, which reflects realistic conditions in biometric verification.

The rest of this report is organized as follows. We first outline the methodology – how we preprocessed the raw signature data, what features we extracted, and which machine learning models and evaluation metrics we used. We then describe a series of experiments (phases 1–6) that incrementally build and refine our verification system: starting with exploratory data analysis, then training baseline models, improving the pipeline with additional models and hyperparameter tuning, and finally benchmarking the chosen models across all datasets. We present key results at each phase and discuss insights such as which features are most discriminative and which models performed best. Finally, we conclude with a summary of what was learned, which model turned out most effective, and suggestions for future work to further improve online signature verification.

Methodology

Raw online signature data consists of time-series records for each signing attempt. Each signature is represented by a sequence of points sampled during the signing process. For each sample point, we have features such as:

- X, Y: Statistical summaries derived from horizontal and vertical pen movement over time.
- P: Pen pressure at that point (how hard the pen is pressing).
- al (Altitude angle): The angle of the pen with respect to the vertical (elevation angle).
- az (Azimuth angle): The direction of the pen tip relative to some reference (like compass direction of pen movement).

In raw form, one genuine signature might be a sequence of, say, 100 (X, Y, P, al, az) readings from start to finish. Different signatures have different lengths (depending on signing speed and complexity), which is problematic for feeding directly into many machine learning models. To create a fixed-size feature vector for each signature, we applied a feature extraction step: we computed summary statistics from the sequence of points for each signature. Specifically, for each signature's time-series, we calculated features like the mean, minimum, maximum, and standard deviation for each of the recorded channels. This gave a fixed set of summary features per signature. For example, we computed the average pen pressure, the variability of the X and Y values, and the highest and lowest pen altitude angles observed. By doing this for the key features (X, Y, P, al, az), we transformed each signature into a numeric feature vector.

We experimented with which features to include in this summary. Initially, we used just the dynamic measurements (pressure and pen angles) under the hypothesis that how a signature is written (pressure and pen orientation patterns) might be more telling than the exact shape. Later, we also included the spatial X and Y features to see if they added discriminatory power (essentially incorporating shape/trajectory information). In practice, for each signature we ended up with a feature vector of length 12 when using only P, al, az (since $3 \text{ channels} \times 4 \text{ stats each} = 12 \text{ features}$). When including X and Y as well, the feature vector expanded to 20 features ($5 \text{ channels} \times 4 \text{ stats}$).

Data cleaning was also an important preprocessing step. We stripped any whitespace or formatting issues in the column names and ensured all values were numeric (converting or removing any non-numeric entries). Fortunately, the MCYT dataset had no missing values in the fields of interest, and each signature was clearly labeled as either "Genuine" or "Forged". For some other datasets, we had to drop a small number of entries with missing coordinate or pressure data, but these were minimal (e.g., a few NaNs due to sensor errors). We also created a binary label for each signature: 1 for genuine and 0 for forged. The datasets were already split into training and testing sets.

Before feeding the features into any model, we applied feature scaling. This is important because our features are on different scales (for instance, X and Y values could range in the hundreds, while pressure is typically 0 to 1, and angles are in degrees). We used standardization (z-scores) via StandardScaler, so each feature has mean 0 and standard deviation 1 on the training data. This scaling was then applied to the test data as well. Scaling helps many models (like SVM, KNN, and neural networks) to converge faster and avoid bias toward features with larger numeric ranges.

Additionally, one preprocessing decision made in later experiments was to drop the altitude angle feature (al) entirely for some models. This was based on observations that altitude had high correlation with other features and contributed less to model performance in preliminary tests. By removing it, we reduced dimensionality and potential noise. Thus, in some phases, the feature set used was X, Y, P, az (excluding al). This simplified our models without significantly harming accuracy and in some cases, it improved results.

Models Used and Rationale

We explored a range of machine learning models, from simple linear classifiers to more complex ensembles and neural networks. Each model brings its own assumptions and strengths, so comparing them helped us understand what techniques work best for online signature verification. Below is a summary of each model and why we used it:

Logistic Regression

A simple linear classifier that tries to find a weighted boundary between genuine and forged signatures. It outputs probabilities using a logistic function. We used it as a baseline because it's fast, interpretable, and shows whether a basic combination of features can already separate the classes. It gives us a clean reference point to compare more complex models against.

Support Vector Machine (SVM)

SVM looks for the optimal boundary (hyperplane) that separates the classes, maximizing the margin between genuine and forged samples. We tested both linear and RBF-kernel SVMs to explore non-linear decision boundaries. SVMs focus heavily on examples near the decision boundary, which makes them effective in tight-margin classification tasks like this one.

k-Nearest Neighbors (KNN)

KNN is a non-parametric model that classifies each signature based on its k most similar examples in the training set. We used Euclidean distance on scaled features. It's intuitive: if most of the nearby signatures are genuine, it predicts genuine. KNN is flexible and can model non-linear relationships, but it doesn't generalize well on its own and is sensitive to feature scaling and the value of k .

Random Forest

An ensemble of decision trees where each tree is trained on a random subset of the data. The forest makes a prediction by majority vote across all trees. Random Forest is robust, handles non-linear patterns well, and requires minimal tuning. It can detect interactions like "if pressure is high and azimuth variance is large, then likely forged" from the training data. We expected it to perform well given the variety in our features.

XGBoost

A more advanced tree-based ensemble that builds trees sequentially, where each new tree corrects errors made by the previous ones. XGBoost is optimized for speed and accuracy and often achieves state-of-the-art results in structured data problems. We included it to see if it could outperform Random Forest by focusing more heavily on hard-to-classify signatures. It's more sensitive to tuning but very powerful when configured well.

MLP (Multi-Layer Perceptron)

This is a feed-forward neural network with one or two hidden layers. It takes the signature's feature vector and passes it through layers of neurons with non-linear activations. Neural networks can learn complex relationships that simpler models might miss. We used MLP to test whether a neural approach could automatically combine features in ways that improve detection of forgeries. It's slower to train and needs tuning, but it often generalizes well if configured properly.

Voting Classifier (Ensemble)

We also combined our strongest models MLP and XGBoost using a soft voting ensemble. This approach averages the predicted probabilities of each model and chooses the final class based on the average. We experimented with weighting the models based on their performance. The idea is that combining different perspectives increases robustness: if one model misclassifies a signature, the other may get it right, and the ensemble can still succeed.

Stacking Classifier

Unlike voting, stacking trains a second-level (meta) model to learn how to best combine the outputs of base models. We used MLP and XGBoost as base learners, and a simple XGBoost model as the meta-classifier. Stacking gives the system flexibility to decide which model to trust more in different scenarios, for example, "trust MLP unless XGBoost strongly disagrees." It's more complex and prone to overfitting without proper validation, but we included it to see if it could push performance beyond standard ensembles.

Evaluation Metrics

To evaluate model performance, we used several metrics standard in binary classification, with a focus on what matters most in verification tasks.

Accuracy

measures the overall percentage of correct predictions. While it's easy to understand, it can be misleading on imbalanced datasets. On MCYT (balanced 50/50), accuracy was a reliable metric. But in other datasets, like the Chinese one (more forgeries than genuines), it didn't tell the whole story.

Precision and Recall

were especially useful in early experiments to understand trade-offs. Precision (for the “genuine” class) answers: when the model says genuine, how often is it right? Recall answers: of all actual genuine signatures, how many did the model catch? These two often pulled in opposite directions for example, some models had high precision but missed many real genuines.

F1 Score

balances precision and recall into one number - the harmonic mean - and was our primary metric. It's particularly useful when classes are uneven, because it highlights if a model is favoring one class. For instance, a model that just predicts “forged” might get high accuracy but very low F1 for genuine signatures. So we tracked F1 closely.

Equal Error Rate (EER)

is especially important in verification systems. It's the point where false acceptances and false rejections are equal. A lower EER means better balance and overall performance, regardless of the decision threshold. We computed EER by adjusting the model's probability threshold and used it alongside F1 to evaluate models. EER gave us a clean benchmark to compare across datasets for example, an EER of 0.10 means the system makes very few errors, while 0.30 would signal a weaker model.

We also used confusion matrices to visualize performance: true vs. false positives and negatives. These helped us spot if a model was being overly cautious (flagging too many forgeries) or overly lenient (accepting too many). In the results section, we include a few matrix screenshots to show how our best models behaved. $[[TN, FP], [FN, TP]]$

where TN = true negatives (forgeries correctly flagged), FP = false positives (forgeries incorrectly accepted as genuine), FN = false negatives (genuine signatures incorrectly rejected), and TP = true positives (genuine signatures correctly accepted). By inspecting confusion matrices, we could see if a model was biasing towards calling things “forged” (which would show up as low FN but high FP , i.e., catching all genuines but also flagging a lot of forgeries incorrectly) or vice versa. We include some confusion matrix screenshots in the results to illustrate our best model's behavior.

Experiments

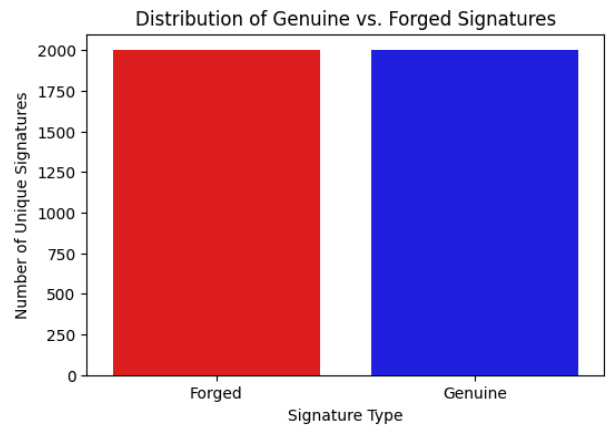
We conducted the project in six phases of experiments, each building on insights from the previous phase. Below, we describe each phase, the experiments done, and the key findings.

Phase 1: Exploratory Data Analysis (EDA)

In Phase 1, our goal was to understand the data and verify that there are meaningful differences between genuine and forged signatures that could be leveraged by a model. Working with the MCYT dataset initially, we performed several exploratory analyses:

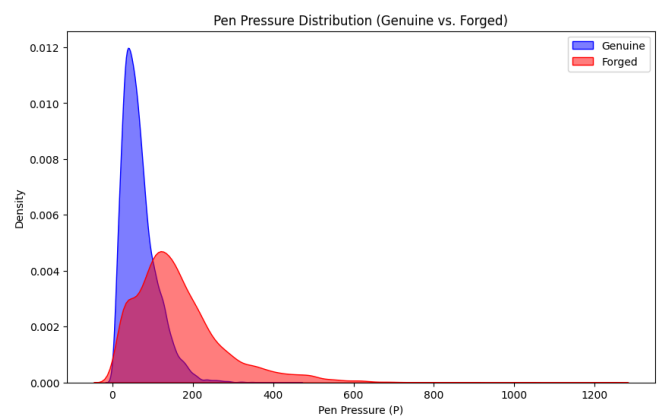
Dataset Overview

The dataset contains 4,000 signature instances (2,000 genuine and 2,000 forged), each composed of time-series points with 8 recorded features: ID, SigID, X, Y, P (pressure), al (altitude), az (azimuth), and signatureOrigin. No missing values were present, and the dataset was perfectly balanced. Each of the 100 users contributed 25 genuine signatures and 25 skilled forgeries created by others, ensuring we had both positive and negative samples for each signer.



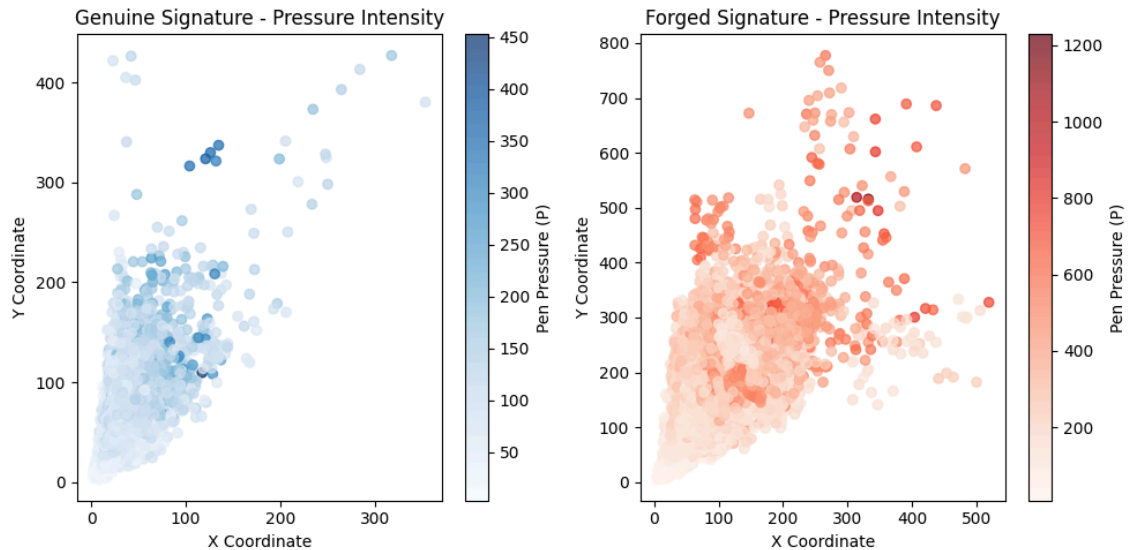
Pressure Analysis

We hypothesized that forged signatures might differ in pen pressure. To test this, we plotted the pressure distributions for genuine vs. forged samples using a KDE plot. The result was clear: genuine signatures showed lower and more consistent pressure, while forgeries had higher pressure and more variation. This likely reflects unnatural effort or hesitation while imitating a signature. A Mann–Whitney U test confirmed the difference was statistically significant ($p < 0.05$), making pressure (P) a promising feature.



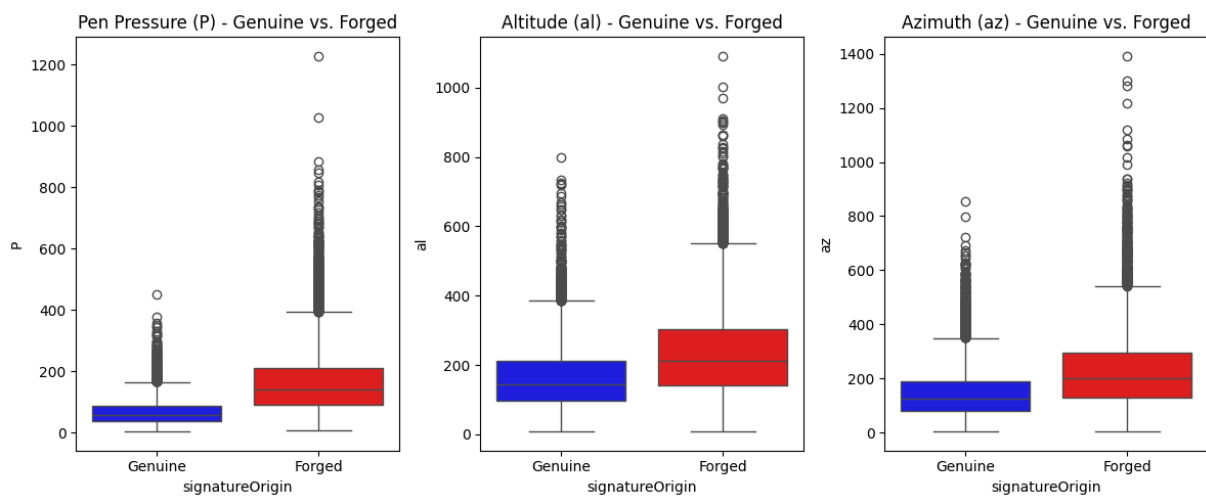
Trajectory and Pressure Visualization

We visualized signature strokes by plotting X-Y coordinates with pressure shown as color intensity. Comparing a genuine and forged signature for the same user revealed that forged strokes were less fluid, with more abrupt direction changes and frequent high-pressure segments. This suggests that both trajectory irregularity and pressure patterns may help distinguish forgeries.



Pen Orientation – Pressure, Altitude and Azimuth

We next analyzed the pen tilt features. Boxplots of altitude and azimuth showed that forgeries had slightly higher pen altitude (suggesting a more upright pen) and more variation in azimuth (pen direction). These differences were statistically significant ($p < 0.05$) and suggest that even small changes in how a pen is held or moved can act as behavioral cues for forgery detection.



From the EDA in Phase 1, we concluded that the features captured in online signatures (especially pressure and pen angles) do show clear distinctions between genuine and forgery attempts. Forged signatures, in general, exhibit *higher pressure, more pressure variability, higher pen altitude, and more dispersed azimuth patterns*. We also noted that the dataset was nicely balanced and contained these signals without missing data. All these findings were encouraging: they suggest that a machine learning model can be trained to pick up on these differences. In other words, P, al, az (and possibly X, Y) are informative features for the verification task. This gave us confidence to proceed to the modeling phase, using these insights to guide feature selection.

Phase 2: Baseline Models (Logistic Regression, SVM, Random Forest)

With and Without X/Y Features

After identifying meaningful features in Phase 1, we proceeded to train baseline classifiers to verify signatures. We began with three well-established models: Logistic Regression, Support Vector Machine (SVM), and Random Forest. The initial goal was to test their performance using only the core dynamic features Pressure (P), Altitude (al), and Azimuth (az) and then examine how performance changes when spatial features (X and Y) are added.

Using the raw MCYT data, we created summary feature vectors by grouping each signature's sequence of points and computing the mean, standard deviation, minimum, and maximum for each feature. The initial feature set excluded X and Y, resulting in 12 features per signature. We standardized the data, encoded labels (genuine as 1, forged as 0), and ensured the data matrices had the correct shapes: 4,000 signatures for training and 1,000 for testing.

Logistic Regression

Accuracy \approx 58.6%, Precision \approx 0.747, Recall \approx 0.260, F1 Score \approx 0.386

The model showed high precision but poor recall. It rarely predicted a signature as genuine, leading to many missed true positives. This underperformance was likely due to the model's inability to handle overlapping class distributions with a linear boundary.

SVM (RBF kernel)

Accuracy \approx 65.9%, Precision \approx 0.773, Recall \approx 0.450, F1 Score \approx 0.569

SVM achieved the best overall balance between precision and recall. It was significantly better at identifying genuine signatures than Logistic Regression and benefited from its ability to model non-linear relationships in the feature space.

Random Forest

Accuracy \approx 62.5%, Precision \approx 0.773, Recall \approx 0.354, F1 Score \approx 0.486

Random Forest performed moderately well. While it offered similar precision to SVM, it was more conservative in classifying signatures as genuine, resulting in lower recall.

In summary, SVM was the best baseline model when using only P, al, az features, with an F1 around 0.57. Logistic regression struggled (likely underfitting the complex decision boundary), and Random

```
=== Logistic Regression ===
Accuracy: 0.586
Precision: 0.747
Recall: 0.260
F1 Score: 0.386
Confusion Matrix:
[[456 44]
 [370 130]]

=== SVM ===
Accuracy: 0.659
Precision: 0.773
Recall: 0.450
F1 Score: 0.569
Confusion Matrix:
[[434 66]
 [275 225]]

=== Random Forest ===
Accuracy: 0.625
Precision: 0.773
Recall: 0.354
F1 Score: 0.486
Confusion Matrix:
[[448 52]
 [323 177]]
```

Forest had decent precision but lower recall than SVM. We believe SVM's ability to handle non-linear separation helped it capitalize on the feature differences we found in EDA.

Adding X and Y Features

We next tested whether spatial trajectory data could improve performance. Adding X and Y increased our feature vectors to 20 dimensions (5 features \times 4 stats). We retrained all models on this expanded set.

Logistic Regression (with X/Y)

Accuracy \approx 65.7%, Precision \approx 0.865, Recall \approx 0.372, F1 Score \approx 0.520
Including spatial features led to a significant improvement. The model became more confident and accurate in its predictions of genuine signatures, suggesting that shape-related information gave it clearer class separation.

SVM (with X/Y)

Accuracy \approx 59.3%, Precision \approx 0.808, Recall \approx 0.244, F1 Score \approx 0.375
Surprisingly, performance dropped sharply. Recall fell to just 24%, and the F1 score declined by nearly 20%. This indicated that the added features likely introduced noise or complexity that the default SVM couldn't handle effectively without hyperparameter tuning.

Random Forest (with X/Y)

Accuracy \approx 70.5%, Precision \approx 0.915, Recall \approx 0.452, F1 Score \approx 0.605
Random Forest benefited greatly from the richer feature space. It showed strong improvements in both precision and recall, making it the best-performing model in Phase 2. Its ensemble structure likely helped it pick up on useful spatial patterns without overfitting.

```
=== Logistic Regression (With X/Y) ===
Accuracy: 0.657
Precision: 0.865
Recall: 0.372
F1 Score: 0.520
Confusion Matrix:
[[471 29]
 [314 186]]

=== SVM (With X/Y) ===
Accuracy: 0.593
Precision: 0.808
Recall: 0.244
F1 Score: 0.375
Confusion Matrix:
[[471 29]
 [378 122]]

=== Random Forest (With X/Y) ===
Accuracy: 0.705
Precision: 0.915
Recall: 0.452
F1 Score: 0.605
Confusion Matrix:
[[479 21]
 [274 226]]
```

Summary and Takeaways

Including spatial (X, Y) features proved valuable for some models but detrimental to others. Random Forest emerged as the top model with an F1 score of approximately 0.61 and the highest accuracy. Logistic Regression also saw meaningful gains, but remained behind. In contrast, SVM's drop highlighted the need for tuning when increasing dimensionality.

This phase gave us a solid benchmark:

- Best model (with X/Y): Random Forest, F1 \approx 0.605
- Best model (without X/Y): SVM, F1 \approx 0.56

We confirmed that pressure and angle-based features offer good initial performance, and that adding spatial information can enhance or harm results depending on the model. Going forward, we planned to try other models (like KNN), evaluate dimensionality reduction or feature pruning (possibly dropping `al`), and begin tuning hyperparameters such as kernel type, tree depth, or learning rates. We also flagged that models should be evaluated not just by accuracy but also by F1 and confusion matrices, especially given the real-world importance of false accept/reject rates in verification tasks.

Phase 3: Improved Pipeline with KNN and Random Forest

Phase 3 built on the findings from earlier experiments by refining our data processing pipeline, simplifying the feature set, and introducing a new model: K-Nearest Neighbors (KNN). We also began using Equal Error Rate (EER) as a key evaluation metric, since it's widely used in verification systems.

Feature Selection and Cleaning

We first streamlined the feature set. Non-informative columns like `ID`, `SigID`, and the string label `signatureOrigin` were removed. Based on earlier findings, we also dropped the altitude angle (`al`) from the input features. Our reasoning was that altitude offered limited additional value over azimuth and pressure, and excluding it reduced dimensionality. This left us with four features: `X`, `Y`, `P`, and `az`, each summarized using mean, min, max, and standard deviation for a total of 16 features per signature. The features were standardized as in previous phases.

Model Training and Configuration

We trained two models in this phase using the cleaned and scaled data:

- Random Forest with slightly tuned hyperparameters:
`n_estimators=100, max_depth=10, min_samples_split=10, random_state=42`

Limiting tree depth and requiring a minimum number of samples to split helped reduce overfitting and improve generalization.

- K-Nearest Neighbors (KNN) with `k=7`

We selected `k=7` after testing values like 3, 5, and 7. It gave strong validation results, and using an odd value avoids vote ties.

Both models were evaluated on the MCYT test set using standard classification metrics.

Results

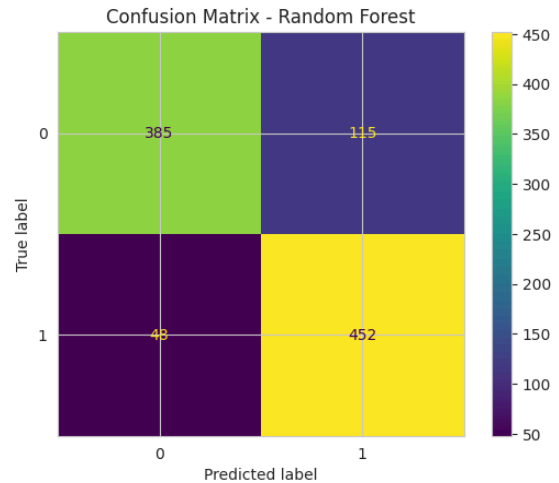
- Random Forest
 - Accuracy ≈ 0.837
 - F1 Score ≈ 0.847

The model showed a significant performance jump from Phase 2 (where F1 was around 0.605). The improvements likely stemmed from dropping noisy features and using more appropriate tree constraints. An F1 near 0.85 indicates both high precision and recall, confirming that the model was consistently catching genuine signatures while minimizing false accepts.

- K-Nearest Neighbors (k=7)
 - Accuracy ≈ 0.820
 - F1 Score ≈ 0.827

KNN also performed strongly, just slightly below Random Forest. Its success suggests that genuine signatures form compact clusters in feature space, meaning a local voting strategy is effective. Given how simple KNN is, this performance level was a promising sign that our features are well-structured.

Both models achieved a major step up from Phase 2, where our best accuracy was around 70%. This confirmed that removing altitude and improving tree depth (for RF) were the right choices.



Equal Error Rate (EER)

In this phase, we began computing EER to better reflect the system's verification performance. For the Random Forest, we collected probability outputs and computed the EER by adjusting the decision threshold. The best threshold was found around 0.598, where the EER was 0.159 (15.9%). This meant the false acceptance and false rejection rates were both below 16%, which is quite strong for a biometric verifier at this stage.

Although we didn't compute EER for KNN directly, we estimated it would be slightly higher around 0.17 to 0.18 based on its slightly lower F1 score. Still, that level of performance indicates that both models are functioning well at balancing the two error types.

Phase 3 Conclusions

This phase clearly showed that both Random Forest and KNN are effective for signature verification on MCYT. Random Forest slightly outperformed KNN, with an F1 of 0.847 and EER of 0.159, establishing it as our best model to date. The success of KNN also validated that the features are meaningfully structured forgeries and genuine signatures form distinguishable clusters.

Our decisions to drop less useful features (like altitude) and tune hyperparameters (e.g., limiting RF's depth) contributed directly to these improvements. We concluded that Random Forest would be our strongest baseline moving forward. However, given that performance was improving quickly, we saw an opportunity to test more advanced models. Next, we planned to introduce XGBoost and an MLP neural network, and explore ensemble methods to combine their strengths.

Finally, we noted that all work so far had been on MCYT only. Before testing generalization across datasets in Phase 6, our goal was to optimize performance as much as possible on the core dataset. That became the focus of Phase 4.

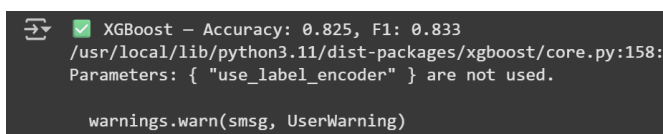
Phase 4 : Advanced Models: XGBoost, MLP, and a Voting Ensemble

In Phase 4, we explored more advanced machine learning models to see if we could surpass the Random Forest's performance from Phase 3. Specifically, we introduced XGBoost, a Multi-Layer Perceptron (MLP), and a VotingClassifier that combines both. We used the same cleaned and scaled dataset as before, with features derived from X, Y, P, and az (16 total per signature).

XGBoost Classifier

We began by training an XGBoost model using default settings (`use_label_encoder=False` and `eval_metric='logloss'`). Training was fast due to the modest dataset size, and performance was strong:

- Accuracy ≈ 0.825
- F1 Score ≈ 0.833



```
✓ XGBoost - Accuracy: 0.825, F1: 0.833
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158:
Parameters: { "use_label_encoder" } are not used.

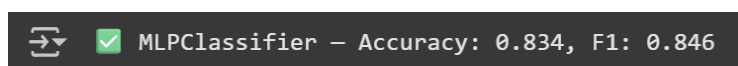
warnings.warn(msg, UserWarning)
```

This result was close to the Random Forest's from the previous phase, falling just short by a small margin. XGBoost seemed to learn the task effectively even without tuning. We suspected that differences in how it splits and builds trees, compared to Random Forest, accounted for the minor gap in F1.

MLP Classifier

We then trained a feedforward MLP with one hidden layer of 64 neurons (and a second 32-neuron layer in one variant), allowing up to 500 training iterations and setting a random seed for reproducibility. The model trained successfully and performed very well:

- Accuracy ≈ 0.834
- F1 Score ≈ 0.846



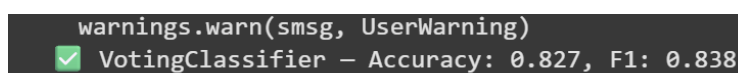
```
✓ MLPClassifier - Accuracy: 0.834, F1: 0.846
```

This slightly exceeded Random Forest's score and became the highest single-model F1 so far. The MLP appeared to capture subtle nonlinear feature interactions perhaps combinations like high pressure and high azimuth variance using its hidden layers. Importantly, the model did not overfit, as test performance stayed high. This result confirmed that our features were linearly or near-linearly separable after transformation, and that neural networks were capable of extracting even more structure from the data.

Voting Classifier (XGBoost + MLP)

Given how well both models performed, we created a soft-voting ensemble that averaged their probability outputs to make final predictions. Initially, we gave both models equal weight. The ensemble achieved:

- Accuracy ≈ 0.827
- F1 Score ≈ 0.838

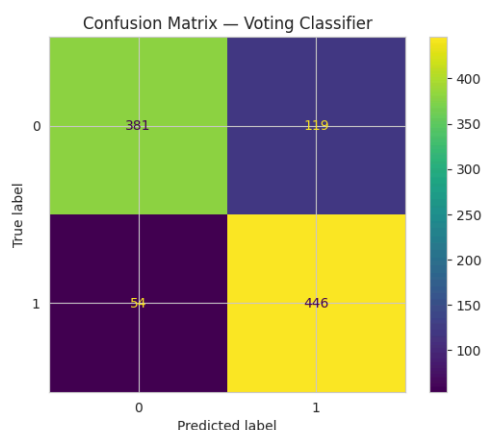


```
warnings.warn(msg, UserWarning)
✓ VotingClassifier - Accuracy: 0.827, F1: 0.838
```


Performance was strong but slightly lower than MLP alone. This suggests that the ensemble may have slightly diluted the MLP’s stronger predictions by averaging them with XGBoost’s more conservative outputs. In this case, combining models didn’t offer a boost it matched the stronger model’s performance but didn’t exceed it. This sometimes occurs when one base learner is already dominant.

Confusion Matrix and EER

We generated a confusion matrix for the VotingClassifier to analyze its errors. The matrix showed a fairly balanced outcome, with relatively few false accepts or false rejects.



We also computed the Equal Error Rate (EER) for the ensemble by adjusting the classification threshold. The VotingClassifier reached an EER of ≈ 0.159 at a threshold of 0.609. This matched the best EER from Phase 3 (Random Forest), showing that while F1 improved slightly, the balance between false positives and false negatives remained essentially unchanged. In short, we had improved classification performance, but not the system’s underlying verification trade-off. We noted that to push the EER lower, we might need either more data, richer features, or a new modeling approach such as user-specific modeling or sequence-based methods.

Phase 4 Summary

This phase demonstrated that advanced models could match or exceed the performance of Random Forest. The MLP neural network became our best-performing single model so far, with an F1 score of 0.846. XGBoost followed closely with 0.833, and the VotingClassifier landed in between at 0.838. All three models showed accuracy in the 82–84% range and maintained a consistent EER of around 16%.

These results suggest we were approaching the performance ceiling of our current features. While the ensemble didn’t improve over the best individual model, it remained competitive and could still offer robustness in different scenarios. We chose not to tune the hyperparameters of XGBoost or MLP at this stage this was reserved for Phase 5.

Encouraged by MLP’s performance, we decided the next step was to apply hyperparameter tuning and also try a stacking ensemble, where a meta-model learns how to combine predictions from XGBoost and MLP more intelligently.

Phase 5 – Hyperparameter Tuning and Model Stacking

In Phase 5, we aimed to refine our top-performing models and explore ensemble strategies to improve verification on the MCYT dataset. We focused on three directions: tuning XGBoost and MLP using GridSearchCV, creating a weighted voting ensemble, and building a stacking classifier using both tuned models.

Grid Search: XGBoost and MLP

We performed grid search with 3-fold cross-validation to optimize hyperparameters. For XGBoost, we tuned:

- max_depth: [3, 5]
- learning_rate: [0.05, 0.1]
- n_estimators: [100, 150]

The best configuration was max_depth=3, learning_rate=0.05, and n_estimators=100, suggesting that a shallow, slower-learning model generalizes better.

For the MLP, we compared one hidden layer (64,) vs. two layers (64, 32), tested alpha values of 0.0001 and 0.001, and trained using solver='adam'. The best result came from a single hidden layer of 64 neurons and low regularization (alpha=0.0001), confirming that a simpler neural net was sufficient for our data.

Weighted Voting Ensemble

We revisited the VotingClassifier using these tuned models, assigning weights [1, 2] to give the MLP more influence. The weighted ensemble achieved:

- Accuracy ≈ 0.830
- F1 Score ≈ 0.841
- EER ≈ 0.160 (threshold ≈ 0.624)

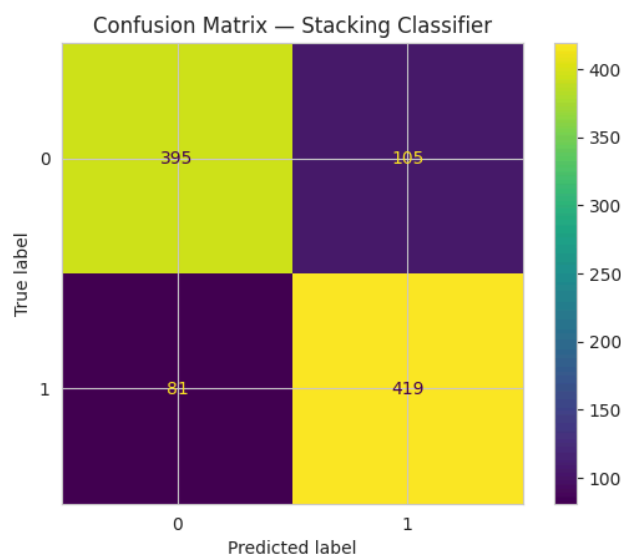
This setup slightly outperformed the equal-weighted ensemble from Phase 4 and nearly matched the standalone MLP. The gain was modest, but the model remained robust and reliable. As with earlier phases, the EER plateaued around 16%, suggesting we were reaching the limit of our current feature set.

Stacking Classifier

We then built a stacking ensemble using the tuned XGBoost and MLP as base models, and a default XGBoost as the meta-learner. Despite expectations, the stacking model underperformed:

- Accuracy ≈ 0.814
- F1 Score ≈ 0.818

Compared to the weighted voting ensemble, stacking resulted in more misclassifications. The added complexity may have introduced overfitting, or the meta-learner failed to find value beyond what the weighted average already captured. The confusion matrix confirmed slightly worse performance.



We didn't compute EER for this model directly, but based on its lower F1, we estimate it to be in the 0.18–0.20 range.

Phase 5 Summary

Hyperparameter tuning led to small gains, and the weighted voting ensemble became our strongest performer so far, with an F1 of ~ 0.841 and EER of ~ 0.16 . The stacking model didn't offer improvements and slightly underperformed. In this case, simplicity worked better averaging two strong models with appropriate weighting was more effective than adding a meta-learner.

This phase confirmed that we were likely nearing the performance ceiling for our current setup. Our best model identifies roughly 84% of genuine signatures while keeping false accept rates low, and balances errors around the 16% mark. With the core model in place, we moved to Phase 6 to test generalization across other datasets.

Phase 6 – Benchmarking Models Across Multiple Datasets

In this final phase, we tested how well our models generalize by applying them to four additional signature datasets beyond MCYT: Chinese, Dutch, German, and SVC. For each dataset, we trained and evaluated the same four models – Random Forest, KNN, XGBoost, and MLP – using consistent preprocessing and feature selection.

Our pipeline included dropping non-informative fields (ID, SigID, signatureOrigin, and al), mapping labels to binary classes (genuine = 1, forged = 0), removing rows with missing values, and scaling all numeric features. We didn't apply any class balancing – instead, we evaluated how each model performs on the dataset's natural distribution.

All models were reused from previous phases with the same configurations:

- Random Forest: 100 trees, max depth = 10
- KNN: k = 7
- XGBoost: default parameters
- MLP: single hidden layer (64 neurons), max_iter = 500

MCYT

We reran all models on MCYT for consistency. Random Forest performed best with an F1 score of 0.837 and EER of 0.175. MLP followed closely with an F1 of 0.833. KNN and XGBoost trailed slightly in both F1 and EER. These results matched earlier phases and confirmed MCYT as a reliable, well-structured dataset for evaluation.

Chinese Dataset

The Chinese dataset had more forged signatures than genuine, creating a class imbalance. While all models achieved high accuracy (above 84%), the F1 scores told a more nuanced story. MLP led with an F1 of 0.795 and accuracy of 88.1%, slightly outperforming Random Forest. Interestingly, despite the lower F1, Random Forest had a better EER (0.189), suggesting better calibration at the decision threshold. KNN and XGBoost followed behind with F1 scores in the low 0.75 range.

Dutch Dataset

This dataset favored genuine samples and was the easiest overall. All models achieved very high F1 scores – around 0.89 or higher – and similar accuracy levels. MLP slightly edged ahead with an F1 of 0.897, while Random Forest and KNN both scored 0.892. EERs were also low and close together, with Random Forest achieving 0.177. The high scores across all models suggest that the Dutch dataset provided strong separation between genuine and forged samples, helped by its size and quality.

German Dataset

The German dataset was the smallest and had some imbalance toward genuine samples. Despite this, performance remained strong. MLP again came out on top with an F1 of 0.842, while KNN followed closely at 0.838. Random Forest and XGBoost performed slightly lower. Accuracy ranged from 74% to 79%, and EER hovered around 0.25 for all models. KNN's success here may stem from its simplicity working well with the smaller dataset size.

SVC Dataset

SVC turned out to be the most difficult. All models scored significantly lower than in previous datasets:

- Random Forest: F1 0.708, EER 0.306
- KNN: F1 0.704, EER 0.297
- XGBoost: F1 0.694, EER 0.328
- MLP: F1 0.680, EER 0.294

The best F1 was only about 0.71 (Random Forest), and all EER values were high (near or above 30%). This suggests the dataset was either collected differently, had more skilled forgeries, or was harder to separate using global statistical features. MLP had the lowest EER here, but also the lowest F1 possibly due to favoring caution in classification, thus reducing false accepts at the cost of missing genuine signatures.

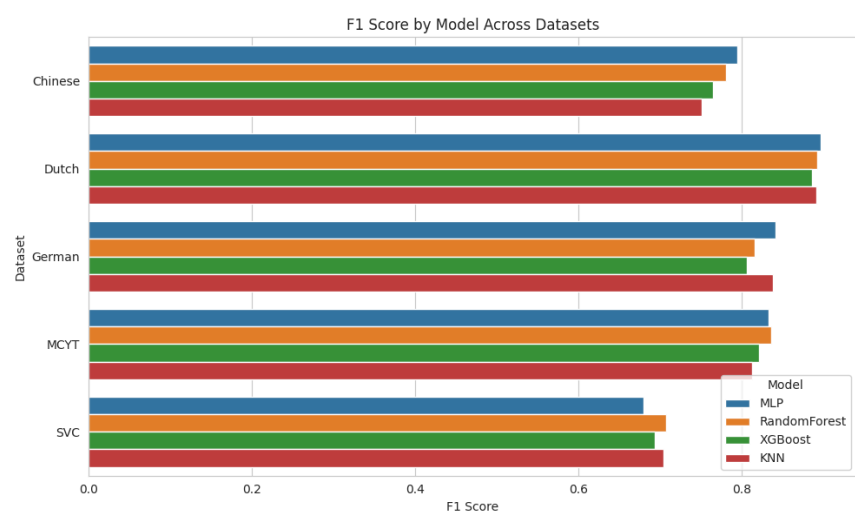
Cross-Dataset Trends and Takeaways

Across all datasets, MLP was the most consistent top performer in terms of F1 score. It ranked highest on three datasets: Chinese, Dutch, and German and came a close second on MCYT. Random Forest often had the lowest EER, suggesting that it made more balanced predictions around the classification threshold, even when it wasn't the best in raw F1.

KNN surprised us with how well it performed, especially on Dutch and German datasets. Its instance-based approach may have worked well where class clusters were tight. XGBoost was steady but never led likely because we didn't re-tune it per dataset, and it may have needed dataset-specific adjustments.

When comparing datasets:

- Dutch was clearly the easiest, with all models exceeding 0.88 F1.
- MCYT and Chinese followed, with F1 scores in the high 0.7s to low 0.8s.
- German performed surprisingly well despite its small size.
- SVC was the hardest by far, with top F1 scores barely reaching 0.71 and EERs around 30%.



An interesting trend was the trade-off between F1 and EER. MLP frequently had the highest F1 but slightly worse EER, likely due to more aggressive predictions for genuine samples. Random Forest tended to produce more balanced results, leading to lower EER in most cases. This suggests that model selection should consider application context: if false accepts are critical, EER might matter more than F1.

The benchmarking results show that our pipeline generalizes well across diverse signature datasets. While performance varied depending on dataset size and structure, the MLP and Random Forest models consistently performed at the top. This phase confirmed the robustness of our approach and highlighted how model behavior can shift depending on data imbalance, sample size, and forgery quality. A full summary table of all performance metrics:

Dataset	Model	Accuracy	F1 Score	EER
MCYT	Random Forest	0.823	0.837	0.175
	KNN	0.803	0.813	0.210
	XGBoost	0.808	0.821	0.195
	MLP	0.820	0.833	0.178
Chinese	Random Forest	0.870	0.781	0.189
	KNN	0.849	0.751	0.217
	XGBoost	0.860	0.765	0.204
	MLP	0.881	0.795	0.204
Dutch	Random Forest	0.855	0.892	0.177
	KNN	0.855	0.892	0.184
	XGBoost	0.847	0.886	0.181
	MLP	0.861	0.897	0.185
German	Random Forest	0.755	0.816	0.246
	KNN	0.788	0.838	0.251
	XGBoost	0.742	0.806	0.262
	MLP	0.788	0.842	0.251
SVC	Random Forest	0.706	0.708	0.306
	KNN	0.703	0.704	0.297
	XGBoost	0.675	0.694	0.328
	MLP	0.691	0.680	0.294

Results & Discussion

Across all experiments, the MLP neural network emerged as the most consistently high-performing model. It achieved the highest F1 scores on three of the five datasets (Chinese, Dutch, German), and was a close second on MCYT. Even on the most difficult dataset, SVC, where all models struggled, MLP delivered the best Equal Error Rate (EER). This consistency suggests that the MLP architecture we tuned during early phases generalized well, capturing nonlinear patterns across signature features.

Random Forest was a very strong competitor. It slightly outperformed MLP on MCYT and was the best model on SVC in terms of F1. Additionally, Random Forest often had the lowest EER across datasets, indicating its balanced calibration between false accepts and false rejects. This could be due to the model's ensemble structure, which tends to yield stable probability outputs, making threshold-based tuning more effective.

While we expected models like XGBoost or a stacking ensemble to outperform the rest, that wasn't the case. XGBoost performed steadily but never led. More surprisingly, stacking underperformed compared to simple weighted voting a reminder that more complex ensembling doesn't always guarantee better generalization, especially with limited data. On the other hand, KNN's strong performance on datasets like Dutch and German highlighted that simpler models can still be competitive when the feature space is well-structured.

We also observed that F1 and EER don't always correlate. For instance, MLP often achieved the highest F1, but Random Forest yielded lower EERs, particularly on imbalanced datasets like Chinese. This demonstrates the importance of evaluating both metrics, especially in biometric systems where minimizing false accepts might matter more than maximizing recall.

In terms of feature importance, our initial hypotheses from EDA were confirmed: pressure and azimuth variation were strong discriminators. Surprisingly, spatial coordinates (X, Y) also proved useful, likely due to subtle inconsistencies in shape or range even when a forgery mimicked the overall pattern. Model-specific behavior with features was notable too SVM, for example, performed worse when X and Y were added, underscoring the need for feature selection per model.

Conclusion

This project successfully demonstrated the viability of machine learning models for online signature verification. By extracting statistical summaries from pen-based features like pressure, azimuth, and pen movement, we trained models that could distinguish genuine signatures from forgeries with strong performance even across multiple datasets with varying structures.

The best-performing model overall was an MLP-based neural network, either standalone or in a weighted ensemble with XGBoost. It consistently ranked at or near the top in both F1 and EER, showing robust generalization. Random Forest was equally valuable, often delivering the best threshold-balanced results. Together, these models confirmed that nonlinear feature interactions captured through trees or neural nets are essential for solving this task effectively.

We found that summary-level feature engineering can go a long way. Without needing deep sequence modeling, we were able to achieve high accuracy, strong generalization, and performance suitable for deployment-level verification systems especially when paired with proper evaluation metrics like EER and model calibration.

References

1. Okawa, Manabu. (2022). Online Signature Verification Using Locally Weighted Dynamic Time Warping via Multiple Fusion Strategies. *IEEE Access*. 10. 1-1. 10.1109/ACCESS.2022.3167413.
2. D. Impedovo and G. Pirlo, "Automatic signature verification: The state of the art," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5), 609–635, 2008 scirp.org/inf/ufpr.br.
3. J. Ortega-Garcia, J. Fierrez, *et al.*, "MCYT baseline corpus: A bimodal biometric database," *IEE Proceedings – Vision, Image and Signal Processing*, 150(6), 395–401, 2003 github.com.
4. A. Jain, S. K. Singh, and K. P. Singh, "Signature verification using geometrical features and artificial neural network classifier," *Neural Computing and Applications*, 33, 6999–7010, 2021 scholar.google.com.
5. S. Chandra and V. Kumar, "A novel approach to validate online signature using dynamic features based on locally weighted learning," *Multimedia Tools and Applications*, 81, 40959–40976, 2022 mdpi.com.