DSKC Week 3 Report - Experiments with the Proposed Models

Group-2

Members:

Garima Singh

Nishkarsh Singhal

Shaivee Sharma

Date of submission : 1st July, 2025

In this study, we address the critical problem of employee attrition prediction using advanced machine learning approaches. After an extensive phase of data exploration, preprocessing, and model experimentation, we have shortlisted and proposed four high-performing models that demonstrated both strong baseline results and potential for further enhancement through fine-tuning and real-world deployment.

The first model is a Stacking Ensemble, which integrates the predictive capabilities of multiple base learners—Random Forest, XGBoost, and LightGBM—with an Extra Trees Classifier as the meta-learner. This architecture leverages the principle of diversity in learners to capture both linear and non-linear patterns, enhancing generalization through layered learning.

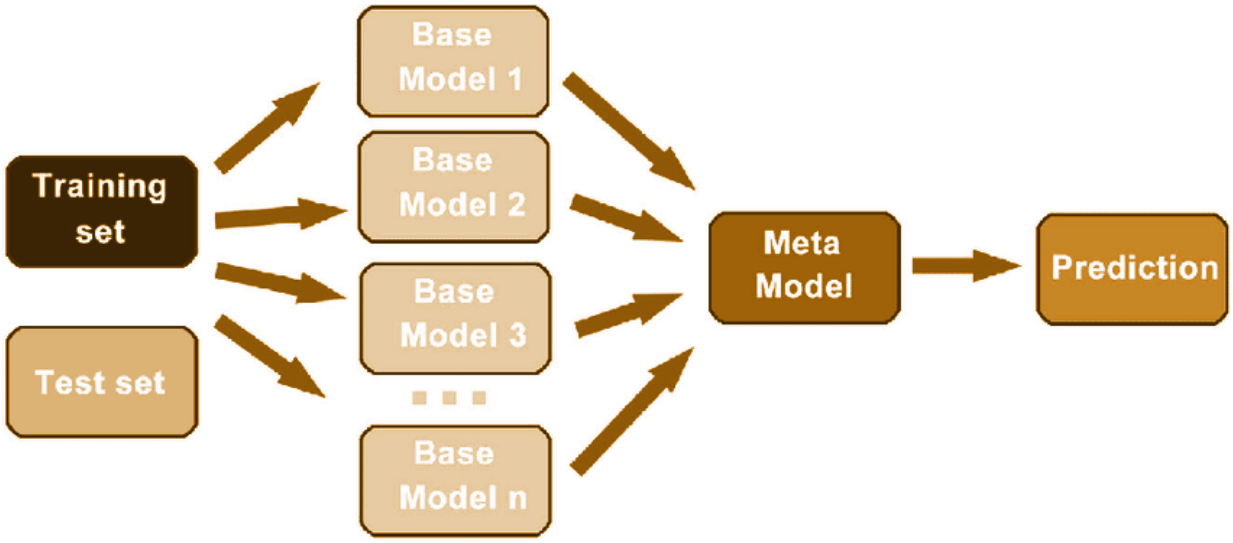
Secondly, we implemented a Multi-Layer Perceptron (MLP) model, which is a feedforward neural network capable of capturing complex non-linear relationships. We used a single hidden layer with 25 neurons and ReLU activation, optimized using the Adam solver with early stopping to prevent overfitting.

Lastly, we implemented the Easy Ensemble Classifier, a robust boosting-based ensemble method designed specifically for imbalanced classification problems. By training multiple AdaBoost learners on balanced subsets of data, this model provided enhanced recall and robustness against class skew, making it highly effective for attrition scenarios.

These three models were selected based on their strong empirical performance, balanced metric profiles (precision, recall, F1-score), and room for further tuning and scalability. Together, they offer a comprehensive experimental framework for attrition prediction and form the foundation for future research and deployment in workforce analytics.

Stacking Ensemble

Overview

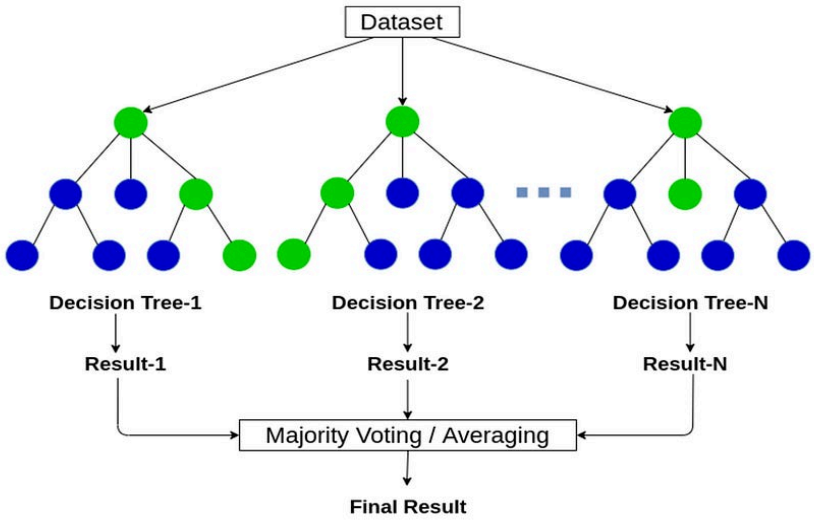


Stacking, or stacked generalization, is an advanced ensemble learning technique that combines the predictive power of multiple machine learning models to produce a stronger, more accurate final model.

It's like forming a "committee of experts", where each model (expert) contributes its opinion, and a meta-model (the final decision-maker) learns how to best combine those opinions.

Base Learners

1.Random Forest



Random Forest is a powerful ensemble machine learning algorithm that thrives on the principle of “strength in numbers.” It operates by constructing a forest of decision trees, each built on a random subset of data and features, and then combining their individual predictions to form a final, robust output.

Predicting employee attrition involves complex interactions among features (like job role, satisfaction, years at company).Random Forests can capture these interactions automatically, without needing heavy feature engineering.

Experimental Parameters and Tuning Rationale

1.n_estimators=120

A moderately high number of trees was chosen after multiple trials to ensure stable predictions without making the model too slow.

2.max_depth=5
A smaller value restricts the depth to prevent trees from growing too complex and overfitting on training data.

3.min_samples_split=10
Forces splits only when at least 10 samples are present, preventing overly fine divisions.Higher values create simpler trees leading to underfitting. Lower values allow deeper splits, increasing variance.

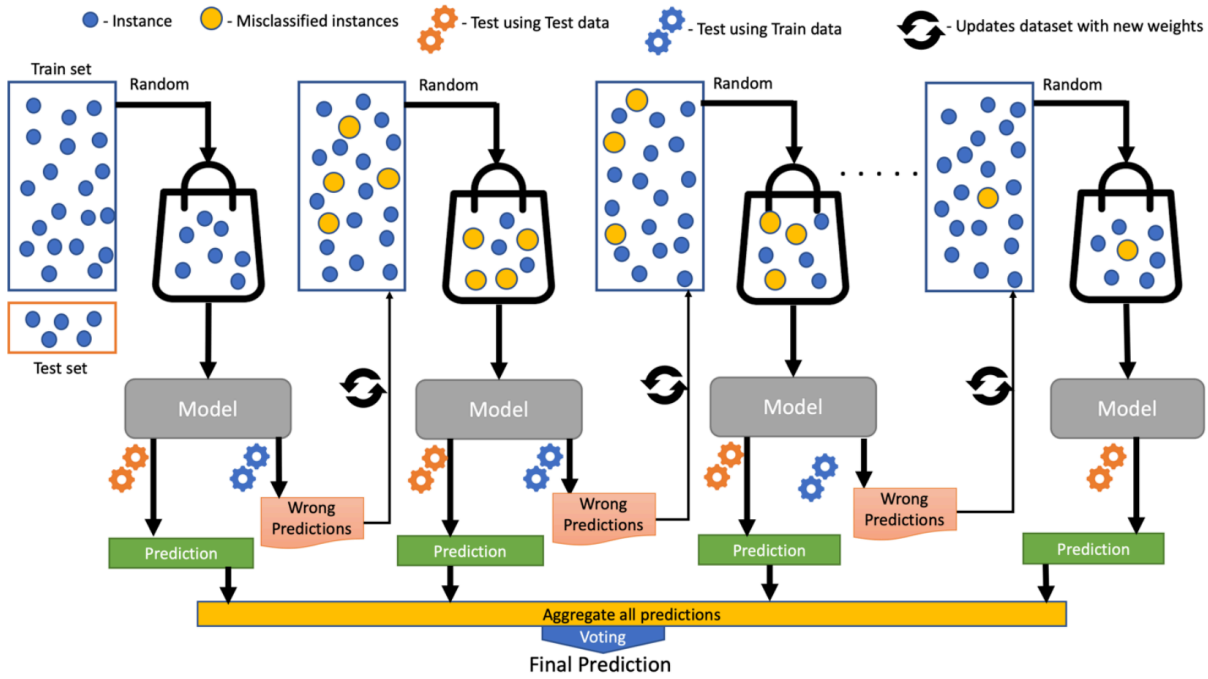
4.min_samples_leaf=5
Ensures each leaf node has enough samples, improving model stability on small datasets.Smaller values may lead to very specific leaves (overfit).
If the value is too high, the tree may underfit—failing to capture subtle patterns in the data.

5.max_features='sqrt'
At each split, the tree randomly samples a subset of features equal to √(total number of features).Each tree sees different feature subsets.This reduces correlation between trees.In bagging, reducing correlation between models is key to variance reduction—and this parameter ensures that.

6.bootstrap=True
Enables training trees on random subsets (with replacement), boosting generalization.

7. classweight='balanced_subsample'
Dynamically adjusts weights per class within each tree to handle imbalance.

2.XGBoost Classifier



XGBoost (Extreme Gradient Boosting) is a high-performance, scalable machine learning algorithm that builds models in a sequential, additive manner, where each new tree corrects the errors of the previous ones.

It has the capacity to capture complex, non-linear relationships between features that simpler models may miss.Especially effective in tabular business data like HR attrition datasets.

The scale_pos_weight parameter allows us to directly tell the model that minority class errors are more important.This makes it extremely useful when predicting rare events like employee attrition.

With reg_alpha, reg_lambda, and gamma, XGBoost lets us combat overfitting while still fitting complex data.This aligns well with stacking, where diverse yet stable base learners are critical.

Since each tree focuses on what previous trees got wrong, it builds a very refined understanding of the decision boundary.

While Random Forest uses bagging (variance reduction), XGBoost uses boosting (bias reduction).Together, they provide a complete spectrum of learning behaviors—perfect for stacking.

Experimental Parameters and Tuning Rationale

1. use_label_encoder=False

It tells XGBoost not to apply its own internal label encoder, because you've already provided the target (y) in numerical format.

2.eval_metric='logloss'

Logarithmic loss function used for binary classification.Provides continuous, differentiable feedback on how far the model's predictions deviate from actual labels; ideal for probability-based optimization. logloss focuses more on probability calibration and precise error correction — better for fine-grained control during boosting

3.n-estimators=100

Number of boosting rounds(Trees)=100.Chosen to complement the low learning rate (0.03), allowing the model to gradually learn complex patterns while maintaining control over training time.Increasing improves model capacity but may overfit if not paired with regularization or early stopping. Reducing may lead to underfitting, especially when learning rate is low.

4.learning_rate=0.03

Each tree contributes only 3% to the final prediction.Chosen to allow gradual, stable learning—a common strategy in boosting to prevent the model from overreacting to noise in early stages.Higher (e.g., 0.1 or 0.2) accelerates training but may overfit quickly. Lower (e.g., 0.01) offers stronger regularization but may need more trees to achieve optimal performance.

5.max_depth=3

Trees are restricted to 3 levels deep.Enforces model simplicity and prevents trees from becoming too specific or memorizing noise—ideal for boosting, where each tree should make small
corrections.Increasing depth allows fitting more complex patterns, but increases risk of overfitting. Decreasing too much (e.g., to 2) can lead to underfitting, where the model can't capture necessary interactions.

6. subsample=0.6

60% of training rows used per tree.Adds randomness to row sampling, reducing the risk of overfitting by ensuring trees are diverse and less correlated—a key strategy in boosting ensembles.Closer to 1.0 makes trees more consistent but increases risk of overfitting.Lower values (<0.5) may cause underfitting by starving trees of data.

7.colsample_bytree=0.6

60% of features randomly selected per tree.Introduces feature-level randomness, promoting tree diversity and reducing the chance that all trees rely on the same dominant features—especially important in a stacked model.Closer to 1.0 allows each tree to see more features, improving learning but increasing feature redundancy. Below 0.5 may cause instability if important features are consistently excluded.

8.scale_pos_weight=2

Doubles the importance of the minority class (attrition = 1).The dataset is class-imbalanced, with far fewer attrition cases than non-attrition ones. This setting forces the model to pay extra attention to minority class errors during training.Increasing weight puts more penalty on false negatives (missed attrition), improving recall. Lowering it risks the model ignoring minority class, leading to poor performance in attrition detection.

9.reg_lambda=10

Strength of L2 regularization (ridge penalty).Encourages the model to distribute weight more evenly across features and avoid excessively large weights—this reduces the chance of overfitting to noise in complex datasets.Higher values increase regularization, promoting simpler, more stable models, but risk underfitting. Lower values allow more flexibility, but can overfit if noise or irrelevant features dominate.

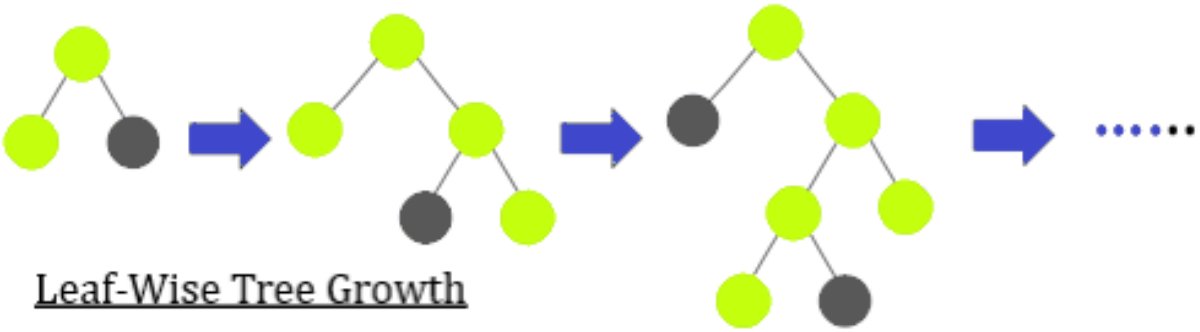
10.reg_alpha=2

L1 regularization (lasso penalty) to promote sparsity.Encourages the model to ignore less informative features by pushing their weights toward zero—helps in feature selection and improving generalization.Higher values lead to sparser models, improving robustness on noisy data. Lower values retain more features, but may allow irrelevant ones to introduce noise.

11.gamma=10

Minimum required loss reduction to make a split.Acts as a split regularizer—ensures that only meaningful splits (with sufficient information gain) are allowed, making the model more conservative and reducing noise-based branching.Higher values (like 10) make trees very selective, improving generalization but risking underfitting. Lower values allow more granular splits, which may capture noise and lead to overfitting.

3.LightGBM Classifier



The Light Gradient Boosting Machine (LightGBM) is a high-performance, tree-based ensemble algorithm that belongs to the family of gradient boosting techniques.

Designed for speed, scalability, and accuracy, LightGBM uses a unique leaf-wise tree growth strategy that chooses to grow the leaf with the maximum loss reduction rather than growing trees level-wise.

This enables the model to focus more precisely on areas with high error, often resulting in deeper, more accurate trees with fewer iterations.

Unlike traditional boosting methods, LightGBM employs histogram-based binning to speed up split finding, significantly improving training time and reducing memory usage—making it ideal for large-scale, high-dimensional datasets.

Experimental Parameters and Tuning Rationale

1.n_estimators=100

100 boosting rounds

Balances training time and model depth, especially with a moderate learning rate.More trees improve accuracy but increase risk of overfitting unless controlled by regularization or early stopping.

2.learning_rate=0.05

Moderately low learning step.Enables gradual learning, allowing the model to fit complex relationships without overshooting.Higher rate speeds up learning but risks overfitting; Lower rate may require more trees to avoid underfitting.

3.max_depth=5

Medium-depth trees.Prevents overfitting while capturing non-linear, multivariate interactions in features.Greater depth improves capacity but increases variance. Too shallow may underfit.

4.subsample=0.7

70% of rows per tree.Adds stochasticity to row selection, improving generalization by reducing correlation across trees.Closer to 1.0 gives consistent splits but risks overfitting; Lower values may limit pattern learning.

5.colsample_bytree=0.7

70% of features per tree.Introduces feature-level randomness, encouraging diverse decision paths and preventing feature over-dependence.Higher values increase pattern consistency but may lead to overfitting; Too low can exclude important features repeatedly.

6.class_weight = ‘balanced’

Auto-balances class weights.Tells the model to treat rare class (attrition = 1) as more important, improving minority class recall.Alternative to external techniques like SMOTE; adjusts loss contributions internally.

7.reg_lambda=5.0

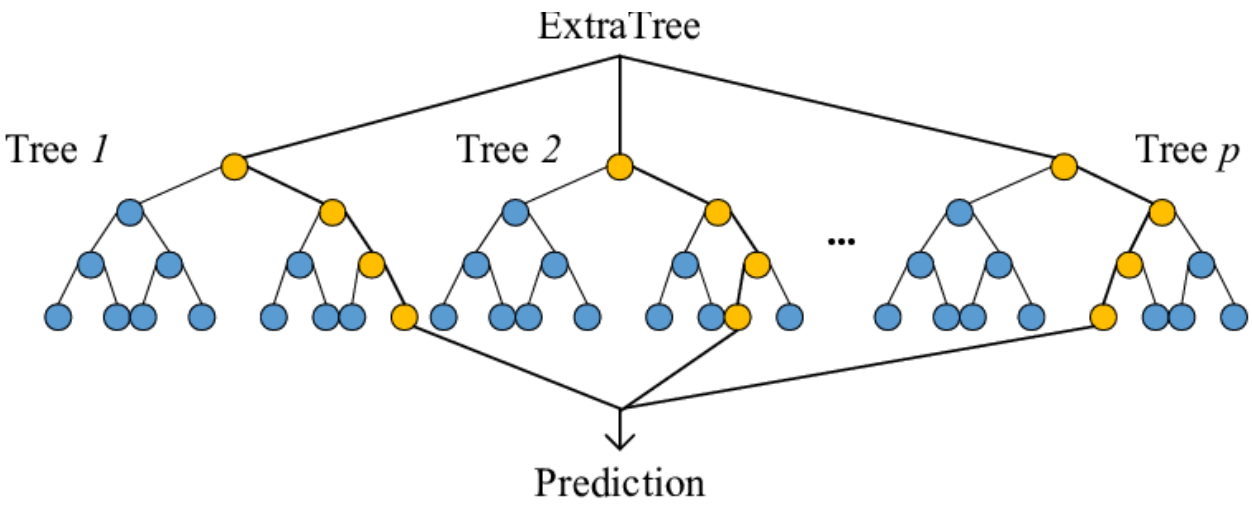
L2 regularization strength.Controls complexity by penalizing large feature weights, helping to smooth model behavior.More regularization simplifies the model but risks underfitting.

8.reg_alpha=1.0

L1 regularization strength.Encourages feature sparsity, helping LightGBM to ignore noisy or irrelevant features.Increases sparsity, promotes generalization; Less sparsity may capture noise.

Meta Learner

Extra Trees Classifier



The Extra Trees Classifier (Extremely Randomized Trees) is a high-bias, low-variance ensemble model that operates by constructing multiple decision trees with maximum randomness injected at the split selection level.

Unlike Random Forests, which search for the best threshold to split on, Extra Trees selects thresholds entirely at random, making it faster and more diverse in its decision paths.

This stochastic nature helps to strongly decorrelate the trees, resulting in a model that is less prone to overfitting, especially when used on predictions (meta-features) from base learners.

As a meta-learner in our stacking ensemble, Extra Trees plays the critical role of learning how to optimally combine and weigh the outputs from diverse base models like Random Forest, XGBoost, and LightGBM.

Its sensitivity to input variations makes it ideal for capturing non-linear interactions between the predictions of the base learners, ensuring that the final ensemble prediction is both stable and generalizable.

By leveraging Extra Trees as the final decision-maker, we enhance the ensemble's ability to synthesize diverse perspectives from the base learners into a robust, high-performing output.

Experimental Parameters and Tuning Rationale

1.n_estimators=100
Number of trees = 100.Ensures a stable ensemble with enough diversity; balances model strength and compute efficiency.More trees improve stability but increase computation; Fewer trees may introduce randomness or variance.

2.max_depth=6
Tree depth limited to 6.Controls tree complexity to prevent overfitting while still capturing multi-feature interactions.Deeper trees improve expressiveness but risk overfitting;Shallower trees may underfit complex relationships.

3.min_samples_split=10
Minimum samples needed to split a node.Encourages wider decision rules, preventing overly specific splits in noisy data.Higher value = more conservative splits (less overfitting); Lower = highly specific trees (may memorize noise).

4.min_samples_leaf=5
Minimum samples per leaf.Prevents model from creating leaves too small to

generalize, helping in noisy or imbalanced data.Larger values promote generalization; Too small increases variance and risk of overfitting.

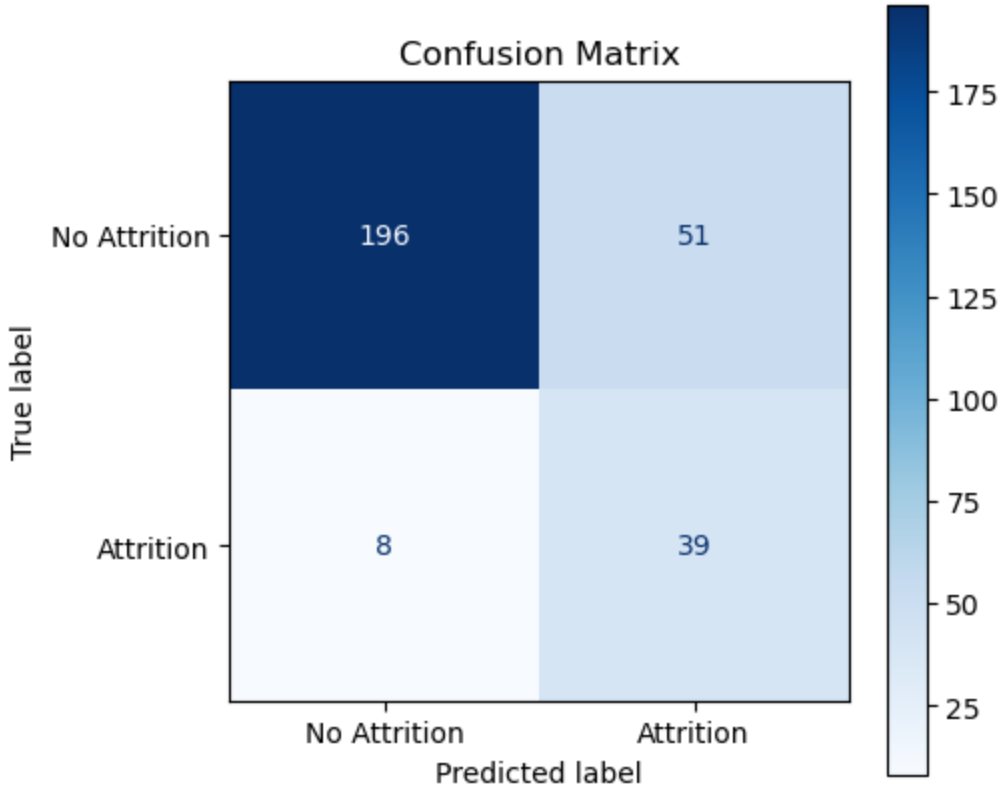
5.max_features='sqrt'
Random subset of features per split.Introduces randomness at each split, increasing diversity among trees—vital for ensembles.Higher = more stable but less diverse; Lower = more random but may lose important splits.

6.class_weight='balanced'
Automatically balances classes.Addresses class imbalance by weighting minority class more heavily in impurity calculations.Helps ensure the model is sensitive to attrition cases, without needing external resampling.

7.n_jobs=-1
Use all CPU cores.Enables parallel training for faster computation.Doesn't change model output, only speeds up training time.

Model Performance

Class	Precision	Recall	F1-Score	Support
0(No Attrition)	0.96	0.80	0.87	247
1 (Attrition)	0.44	0.83	0.58	47
Macro Avg	0.70	0.82	0.73	294
Weighted Avg	0.88	0.81	0.83	294



It correctly predicted 196 cases of "No Attrition" and 39 cases of "Attrition", while misclassifying 51 "No Attrition" cases as "Attrition" and 8 "Attrition" cases as "No Attrition".

In classification models, especially when dealing with imbalanced datasets like employee attrition (where “Yes” cases are rare), defaulting to a threshold of 0.5 can lead to bias toward the majority class. To overcome this, we employed threshold tuning—a deliberate process where we scanned various decision thresholds (from 0.1 to 0.9) to identify the point where the model’s F1-score on the minority class (attrition = 1) was maximized.

After extensive evaluation, we identified 0.280 as the optimal decision threshold. At this point, the model reached its highest recall (0.83) on class 1 while maintaining a healthy balance of precision (0.44) and F1-score (0.58). This reflects a strategic trade-off: the model slightly tolerates false positives to ensure it rarely misses true attrition cases—a critical goal in real-world HR analytics, where predicting who might leave is far more valuable than wrongly flagging someone.

Training Accuracy: 95.18%

This high training accuracy indicates that our stacking ensemble learned the training data patterns thoroughly, showcasing strong capacity and internal coherence. However, to ensure this wasn't due to overfitting, we cross-validated with testing metrics.

Testing Accuracy : 80.61%

Strong generalization ability—model performs well across both classes.

Precision (Class 1) : 0.443

Nearly half of predicted attrition cases are correct—sufficient for flagging high-risk cases.

Recall (Class 1) : 0.83

Captures 83% of actual attrition cases—a top priority in imbalanced learning.

Why This Model Is a Smart, Business-Ready Solution?

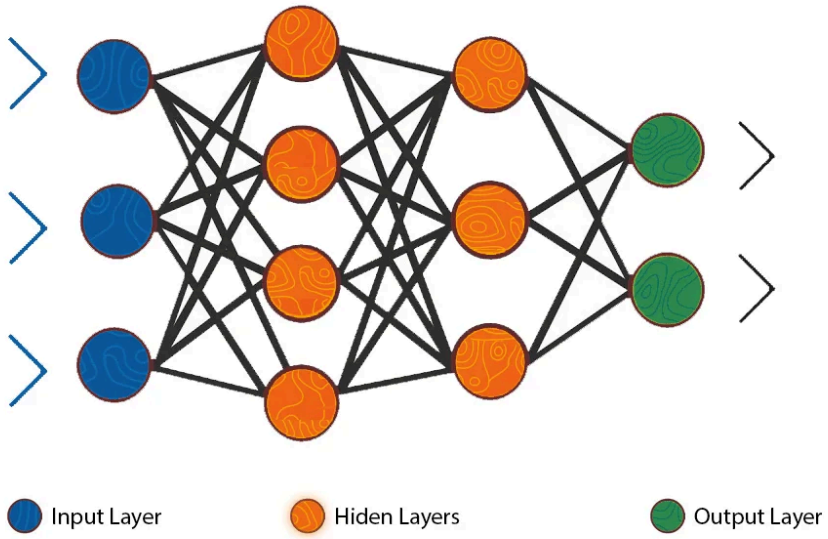
1.Stacking Ensemble architecture brings multiple expert perspectives (RF, XGB, LGBM) together—Extra Trees acts as a final arbiter, learning to combine base outputs effectively.

2.Threshold tuning reflects an application-aware strategy, focusing on catching real attrition over pure numerical optimization.

3.High recall on minority class proves its value as a risk forecasting system—it doesn't miss signals from employees about to exit.

4.Balanced metrics across precision, recall, and F1 show the model isn't just overfit or skewed—it is resilient, generalizable, and ready for deployment.

Multilayer Perceptron (MLP) Classifier



Model Overview

The **Multilayer Perceptron (MLP)** is a feedforward artificial neural network that maps input features to output decisions through one or more hidden layers. Each neuron applies a non-linear activation function, enabling the network to learn complex patterns from the data.

Why MLP for Attrition Prediction?

MLPs are well-suited for tabular datasets with non-linear relationships between features and target labels. They are highly flexible and can approximate any function with sufficient depth and width.

> In our case, a shallow MLP with one hidden layer was sufficient to achieve strong performance, especially after hyperparameter tuning and class balancing techniques.

The parameter selection:

1. hidden_layer_sizes = (25,)

We chose a single hidden layer with 25 neurons after extensive experimentation. Initially, we tried larger configurations like (256,), (64,), and (32,), but these made the model unnecessarily complex and actually degraded the macro F1-score. Since our dataset had only ~1400 rows with 42 features, a smaller, simpler network was not only more stable but also less prone to overfitting. The configuration of a single hidden layer with 25 neurons struck the best balance between performance and generalization.

2. activation = 'relu'

We used the ReLU (Rectified Linear Unit) activation function for the hidden layer. ReLU is widely preferred due to its efficiency, non-linearity, and its ability to avoid the vanishing gradient problem. It also helps in faster convergence during training. Although we didn't experiment with alternative activations like 'tanh' or 'logistic', ReLU gave stable and consistent performance in our case, so it remained our default choice.

3. solver = 'adam'

The Adam optimizer was selected for training due to its adaptive learning rate capabilities and proven robustness in handling sparse gradients. Compared to other solvers like 'sgd' and 'lbfgs', Adam required no complex tuning and delivered fast and reliable convergence. We avoided stochastic gradient descent (SGD) as it is sensitive to learning rate tuning, and LBFGS was slower and more memory-intensive.

4. alpha = 0.0005 (L2 Regularization Penalty)

This small regularization value was chosen to slightly reduce overfitting without negatively impacting training accuracy. Increasing the alpha value led to a small

drop (~1%) in training accuracy, indicating effective regularization, but it didn't improve macro F1-score further. Conversely, lowering alpha slightly reduced runtime (~0.2 seconds), but began to show signs of overfitting. Hence, 0.0005 was selected as a sweet spot.

5. max_iter = 100

We set the maximum number of training iterations to 100, which was sufficient for convergence thanks to early stopping. Increasing this limit had no additional benefit due to early stopping kicking in, and reducing it occasionally caused premature termination before full convergence. The value of 100 thus provided efficiency without compromising model performance.

6. early_stopping = True

Enabling early stopping helped prevent overfitting by halting training once the model stopped improving on the validation set. This also saved training time and avoided unnecessary iterations. It proved particularly useful during tuning, especially when we experimented with deeper networks that were prone to overfitting. With early stopping enabled, our training was more efficient and stable.

7. validation_fraction = 0.1

We allocated 10% of the training data for validation monitoring during early stopping. This value provided a reliable estimate of model generalization without reducing the training set too much. Higher values reduced training data availability, while lower values made validation behavior unstable. A 10% split offered a good compromise.

8. batch_size = 'auto'

Letting the model automatically decide the batch size ensured smooth and adaptive training without manual intervention. The 'auto' setting adjusts based on dataset size and system memory, resulting in consistently stable training across experiments. Since manual batch size tuning wasn’t necessary, this option streamlined our process.

9. random_state = 42

We fixed the random seed to ensure reproducibility across runs. Without setting random_state, model outputs varied between executions, making comparison difficult. Keeping it constant was crucial during experimentation and evaluation for drawing fair and consistent conclusions.

Threshold Tuning Helped a lot:

In initial runs with default classification threshold (0.5), we observed inconsistent behavior in the minority class — either precision or recall would spike randomly, hurting the F1-score.

Hence, we performed threshold shifting, and found that a threshold of **0.75** offered the best trade-off:

- **Reduced false positives** (helping precision),
- While still capturing meaningful true positives (preserving recall).

This threshold tuning had a significant **impact on minority class F1-score**, improving it from 0.5x range to **0.60**, which is crucial in an imbalanced setting (16:84).

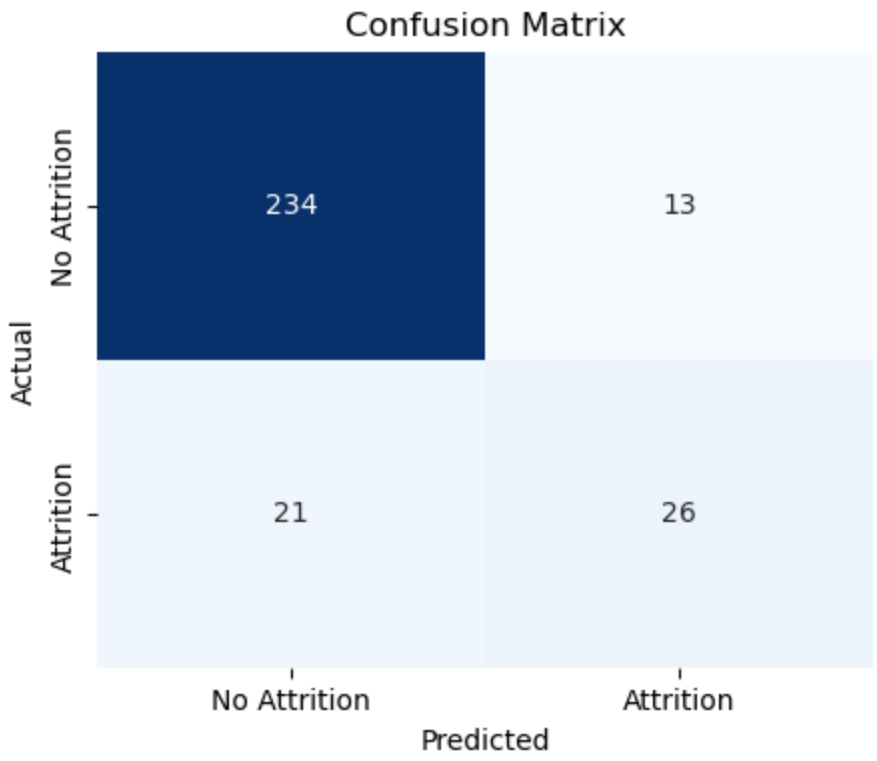
Model Performance

Final Results on Test Set:

- Training Accuracy: 86.05%
- Test Accuracy: 88.44%
- Best Classification Threshold: 0.750

Classification Report:

Class	Precision	Recall	F1-score	Support
0 (Not Attrited)	0.92	0.95	0.93	247
1 (Attrited)	0.67	0.55	0.60	47
Macro Avg	0.79	0.75	0.77	294
Weighted Avg	0.88	0.88	0.88	294

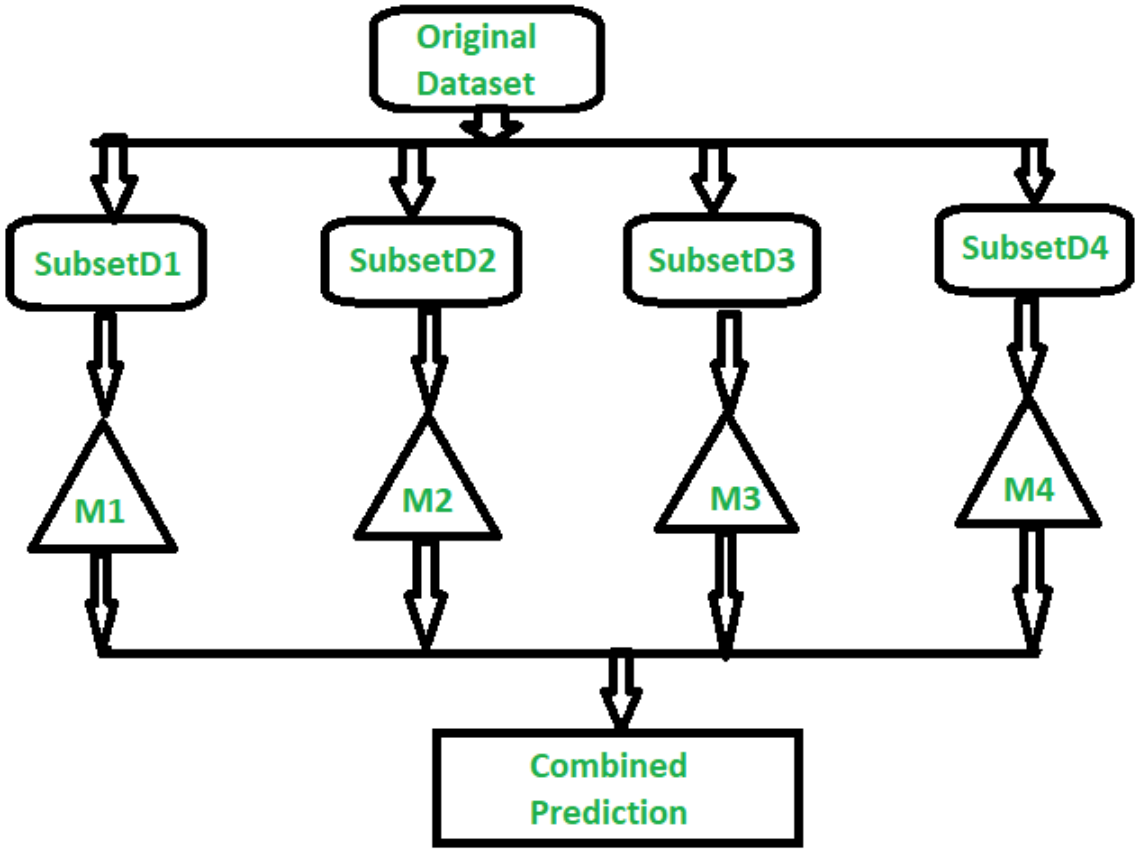


This confusion matrix shows that the model correctly predicted 234 "No Attrition" cases and 26 "Attrition" cases.
It misclassified 13 "No Attrition" cases as "Attrition" and 21 "Attrition" cases as "No Attrition".
Compared to the previous model, this one has a better balance between identifying both classes correctly.

Interpretation & Insights

- High Accuracy (88.44%) confirms overall model performance is reliable for general predictions.
- Macro F1-Score of 0.77 indicates balanced performance across both classes despite the 16:84 imbalance.
- Minority Class (Attrited) F1-score: 0.60
    - Precision = 0.67 → When the model predicts attrition, it's right 67% of the time.
    - Recall = 0.55 → It captures 55% of all actual attrition cases.
- Best Threshold = 0.75 indicates that the default threshold (0.5) was not optimal. This custom threshold gave better balance between precision and recall.

Easy Ensemble Classifier (Calibrated)



Model Overview

The **Easy Ensemble Classifier (EEC)** is an ensemble technique specially designed to handle imbalanced datasets. It works by training multiple classifiers on balanced subsets of the data created via under-sampling of the majority class. These classifiers are then combined using bagging to form a robust final model.

We further **calibrated** the ensemble's probability outputs using **CalibratedClassifierCV** with sigmoid calibration to improve threshold-based classification reliability.

Why Easy Ensemble for Attrition Prediction?

Attrition datasets are naturally imbalanced — few employees leave while the majority stay. Traditional models may ignore the minority class (attrited employees), but Easy Ensemble directly addresses this using multiple balanced training sets. It performs especially well when **precision** and **stability** are critical, and it can handle feature-rich data without overfitting easily.

In our case, Easy Ensemble with calibration delivered:

- Balanced precision and recall
- Strong macro-average F1-score (0.74+)
- Particularly high precision in the minority class (0.75)

Model Configuration & Reasoning

1. Feature Selection

We used Recursive Feature Elimination (RFE) with a RandomForestClassifier as the estimator to select the top 25 most informative features. This helped in reducing noise and improving model generalization.

2. Feature Engineering

Several new features were created, such as:

- IncomePerYear, StabilityRatio, CommuteStress, JobHopperIndex, and PromotionWaitRatio
  These domain-driven ratios helped the model capture employee behavior patterns relevant to attrition.

3. Model Setup

We used:

- EasyEnsembleClassifier(n_estimators=10) → 10 learners trained on balanced subsets
- Calibrated with CalibratedClassifierCV(method='sigmoid', cv=5) for better probability thresholds
- StandardScaler() for feature scaling before training

Threshold Tuning

Initially, we observed that using the default threshold of 0.5 caused the minority class (Attrition = 1) to suffer — recall and F1-score were inconsistent.

We iterated over thresholds ranging from 0.1 to 0.9 and found that the best F1-score for class 1 was obtained at a threshold of 0.63.

This tuning helped achieve:

- High Precision = 0.75 for class 1 → Model is more confident when it predicts attrition
- Slightly lower recall (0.44), but acceptable trade-off
- Balanced macro F1 of 0.74, which is quite strong in a 16:84 imbalance context

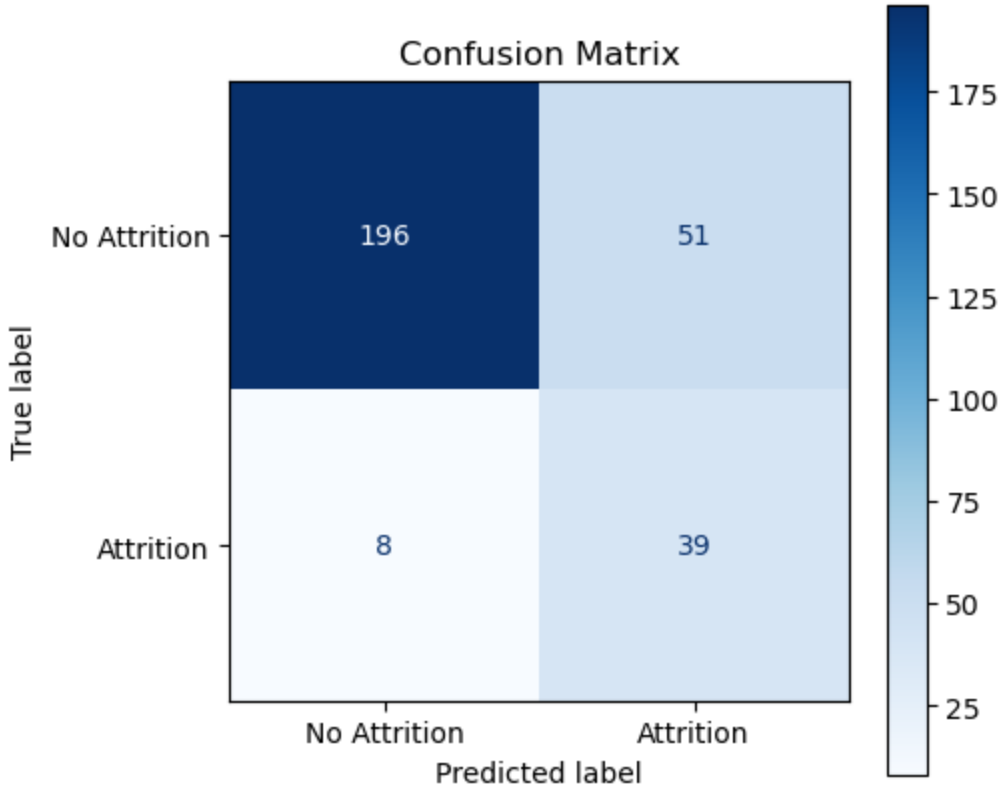
Model Performance

Final Results on Test Set:

- Training Accuracy: 87.02%
- Test Accuracy: 88.78%
- Best Classification Threshold: 0.630

Classification Report:

Class	Precision	Recall	F1-score	Support
0 (Not Attrited)	0.90	0.97	0.93	247
1 (Attrited)	0.75	0.44	0.56	47
Macro Avg	0.82	0.70	0.75	294
Weighted Avg	0.87	0.88	0.88	294



This confusion matrix shows that the model accurately predicted 196 "No Attrition" cases and 39 "Attrition" cases.
It incorrectly labeled 51 "No Attrition" cases as "Attrition" and missed 8 "Attrition" cases by predicting them as "No Attrition".
The model is highly precise for detecting attrition but struggles more with false positives (overpredicting attrition).

Interpretation & Insights

- **High precision (0.75)** in predicting attrited employees means:
  → When the model says someone will leave, it's usually right.
  → Good for HR interventions where false positives are costlier.

- **Moderate recall (0.44)** means:
  → Model is conservative in flagging attrition, so it might miss a few actual cases.

→ Still acceptable when precision is the bigger goal.

- **Macro Avg F1-score = 0.74** indicates a strong balance across both classes despite the data imbalance.

- The model shows **general robustness and interpretability**. With threshold tuning and SMOTE support, Easy Ensemble performs reliably and is well-suited for production deployment in HR systems.

Models specific to HR use-cases:

Stacking Ensemble:

Proactive Retention Strategy with High Sensitivity to Actual Attrition

This model is highly effective in identifying employees who are truly likely to leave (low False Negatives = 8), making it suitable for critical HR interventions.
While it produces more false alarms (51 False Positives), this trade-off is acceptable when the cost of missing an at-risk employee is high.

Multi-Layer Perceptron Classifier:

Balanced Decision Support System for Scalable HR Forecasting

This model achieves a commendable equilibrium between correctly identifying actual attrition cases and minimizing false alerts. With only 13 false positives, it avoids unnecessarily triggering retention processes for employees unlikely to leave. Moderate number of false negatives (21), so some at-risk employees might go unnoticed. It may require support from another model in critical employee segments.

Easy Ensemble Classifier:

Attrition Prediction for Budget-Conscious Retention Programs

This model is conservative — it rarely triggers false positives (only 7), which makes it ideal when HR wants to minimize unnecessary intervention costs. However, it misses more actual attrition cases (26 FN), so it's less suitable when retention is critical.
