# Exercise 1 - Classification

VU Machine Learning - Group 73

Olsacher (01520348) Ponesch (11818774) Winkler (12013078)

November 10, 2025

## 1 Introduction

In this machine learning project, four tabular datasets were analyzed. Two were provided through Kaggle challenges: **Loan Grading** and **Breast Cancer**. The remaining two,**Myocardial Infarction Complications** and **Indoor Localization using Wi-Fi Access Point signals**, were independently selected by our group. This combination spans diverse problem domains such as credit risk assessment, clinical outcome prediction, and signal-based localization, allowing us to compare how preprocessing strategies and model behavior vary across fundamentally different data types.

**General Workflow**

1. **Descriptive statistics & problem framing:** The target and feature variables were identified, class balance and distributions were reviewed, and basic data-quality checks (e.g., outliers, duplicates) were done.
2. **Preprocessing (dataset-dependent):** Based on step 1, datasets were preprocessed. Missing or sentinel values were handled, categorical variables were encoded, numeric variables were scaled/normalized, and, where appropriate, features were filtered or dimensionality was reduced. The extent of preprocessing was determined by the dataset characteristics.
3. **Modeling with three classifiers and grid search:** On the preprocessed data, $k$-Nearest Neighbors (KNN), Random Forest, and XGBoost were trained. For each classifier, a grid search over key hyperparameters was performed with stratified validation.
4. **Evaluation and model adaptation:** The best configurations were selected using **F1** for binary tasks and **F1-macro** for multiclass tasks. These metrics were used to drive grid-search selection, and dataset-specific properties adaptations were made to obtain the best results.

## 2 Predictions

### 2.1 General Information about the used Classifiers

- **$k$-Nearest Neighbors (k-NN)** classifies a new instance by locating the $k$ most similar training instances according to a chosen distance or similarity measure, and assigns the majority class among them.
- **Random Forest** is an ensemble of decision trees trained on bootstrapped samples with randomized feature selection at each split. Predictions are aggregated across trees, reducing variance and improving robustness compared to a single tree.

- **XGBoost (Extreme Gradient Boosting)** is a gradient-boosted decision tree ensemble. Trees are added sequentially, with each new tree focusing on correcting the residual errors of the previous ones.

**Model Selection Rationale.** We deliberately chose these three models to cover complementary learning strategies–instance-based (kNN), bagging (Random Forest), and boosting (XGBoost)–and to explore how preprocessing and hyperparameter tuning affect different algorithmic biases.

We included **kNN** because we expected it to perform poorly on at least some datasets, particularly the MI dataset with many missing and mixed-type features. This allowed us to test how sensitive simple distance-based models are to scaling and imputation.

**Random Forest** was selected as a reliable all-rounder that typically performs well on any tabular data with little tuning. It handles nonlinearity, mixed feature types, and redundant variables efficiently, making it an excellent reference model for comparison.

Finally, we chose **XGBoost** as a powerful, high-ceiling method. It is often regarded as one of the best-performing algorithm for structured data when tuned carefully. However, due to computational and time constraints, we could not explore an extensive hyperparameter space, so its performance here likely underrepresents its full potential.

## 2.2 Hyperparameter configurations for Tuning

Table 1: Hyperparameter grids explored per model (3-fold CV, refit: F1_macro).

| Model | Hyperparameter | Values |
|---|---|---|
| KNN | `n_neighbors` | [2, 3, 6, 15] |
| | `weights` | [`uniform`, `distance`] |
| | `p` | [1, 2] |
| Random Forest | `n_estimators` | [100, 500, 1000, 1500] |
| | `max_depth` | [None, 10, 100] |
| XGBoost | `n_estimators` | [100, 500, 1000, 1500] |
| | `learning_rate` | [0.05, 0.1, 0.2] |
| | `max_depth` | [5, 10, 15, 20] |

## 2.3 Choice of Evaluation Metrics

In this exercise, **F1** was used for binary targets and **F1-macro** for multiclass tasks. *Precision* quantifies the reliability of positive predictions, $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$, while *Recall* (sensitivity) captures the ability to find actual positives, $\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$. Since increasing one of these quantities can depress the other, the **F1** score combines them via the harmonic mean,

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},$$

penalising models that perform well on only one of the two dimensions. For multiclass problems, **F1-macro** averages the per-class F1 scores uniformly, thus giving minority classes equal influence and avoiding dominance by the majority class. By contrast,

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$$

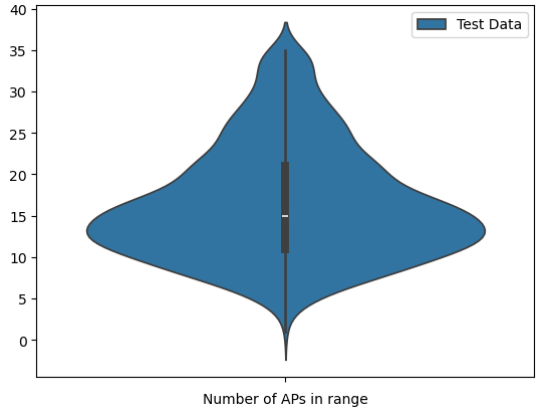
, can remain deceptively high under imbalance because correct predictions on common classes mask systematic errors on rare ones. Consequently, F1/F1-macro was preferred to obtain a more faithful summary of performance across classes and decision thresholds.

# 3 Dataset 1: UJIIndoorLoc

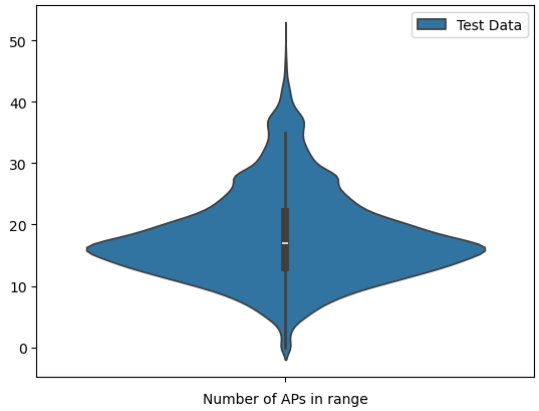
## 3.1 Description

The **UJIIndoorLoc** dataset contains 19,937 training and 1,111 validation/test samples. Each record has 520 WiFi RSSI features (**WAP001**–**WAP520**), numeric in $[-104, 0]$ dBm and `100` if an AP is out of range. Targets include **Building ID** (nominal), **Floor** (ordinal) and **Space ID** (nominal). Typical tasks are (i) regression for coordinates and (ii) **classification** of the triplet (**Building**, **Floor**, **Space**), which is unique (735 combinations; 123 distinct Space IDs). The dataset contains no missing values.

In Figure 1 and Figure 2, the in-range WAP counts are illustrated with violin plots. Under the initial train–test split (Figure 1), the training and test sets exhibit different distributions, which subsequently resulted in poorer predictions. By contrast, the new split shows a more similar distribution between training and test sets, leading to a more reliable evaluation.

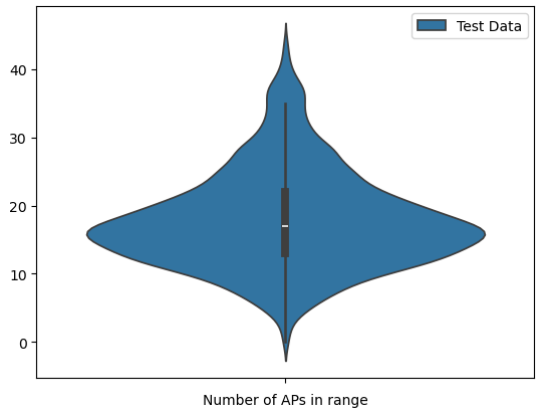



(a) initial Test Dataset (Mean: 16.47, Std: 6.89)

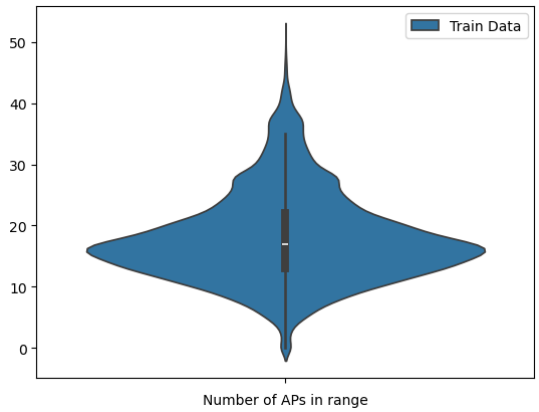



(b) initial Train Dataset (Mean: 17.99, Std: 7.33)

Figure 1: Initial Test/Train-Split: WAPs in Range Test vs. Train




(a) new Test Dataset (Mean: 17.85, Std: 7.23)




(b) new Train Dataset(Mean: 17.92, Std: 7.73)

Figure 2: New Test/Train-Split: WAPs in Range Test vs. Train

Furthermore, Figure 3 shows a slightly left-skewed distribution of the signal values (dBm), with a longer tail toward weaker signals. This pattern can be regarded as natural for Wi-Fi scans: many access points are detected, yet only a few have very strong signals, while most are received at moderate to low strength.

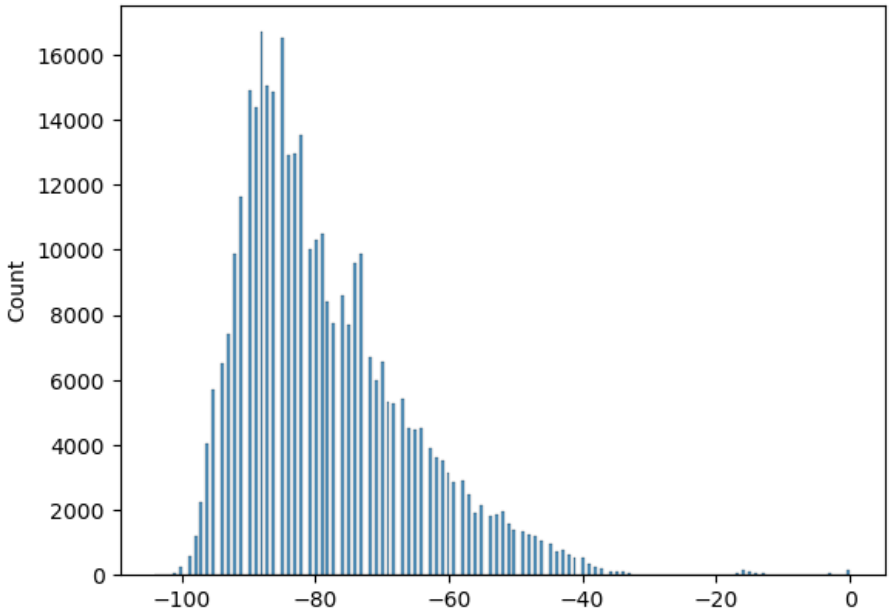



Figure 3: Unscaled signalvalue distribution in dBm

### 3.2 Preprocessing

The dataset consisted solely of numerical features (as noted above), which made it well suited to all selected classifiers. For **k-NN**, key steps included the treatment of *"not in reach"* values (encoded as `100` in the raw data) and the scaling of dB features. The `StandardScaler` from `scikit-learn` was applied to normalize the features, as dB-valued inputs are handled well by this transformer.

The *"not in reach"* entries were recoded to `NaN` and subsequently imputed to $-110\,\mathrm{dB}$ using `SimpleImputer` for k-NN-classifier. By imputing with a value far below the smallest observed measurement, these cases were intended to be interpreted by k-NN as very weak (yet present) signals.

### 3.3 Training & Evaluation

Several prediction scenarios were evaluated. First, a composite code from `BuildingID` and `Floor` was constructed and used as the target variable. Later, because `SpaceID` values were absent in the public test set (the column contained zeros), a new internal train–test split was created from the original, sufficiently large training set. This enabled training a classifier for a composite code of `BuildingID`, `Floor`, and `SpaceID` to obtain a more precise location prediction. The internal split contained valid `SpaceID` values and therefore allowed the training and evaluation of a dedicated "`SpaceID` model".

For the selection of the best-fitting hyperparameters of each classifier, the original train–test split with the composite label of `BuildingID` and `Floor` was used, due to computational limitations. For the XGBoost classifier, computation using grid search, including `SpaceID` in the target variable, was very slow on the available hardware. Consequently, hyperparameter evaluation (Grid Search) was done with a reduced target space (BuildingID, Floor) and only two folds (for XGBoost, three fold for the other). The Results of the Grid Search are shown in Figure 4, Figure 5 and Figure 6.

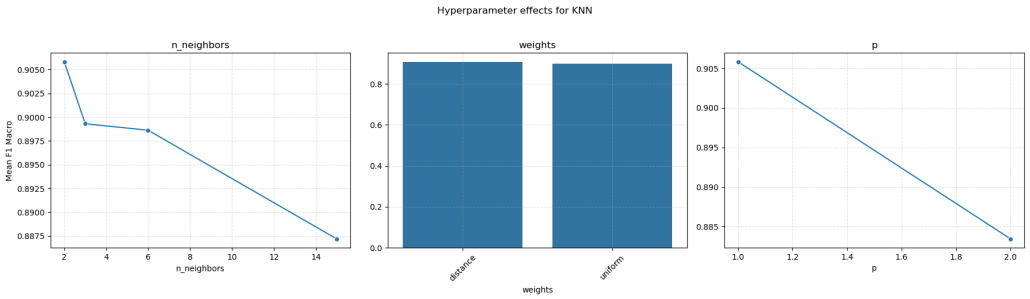



Figure 4: Grid Search Results: KNN

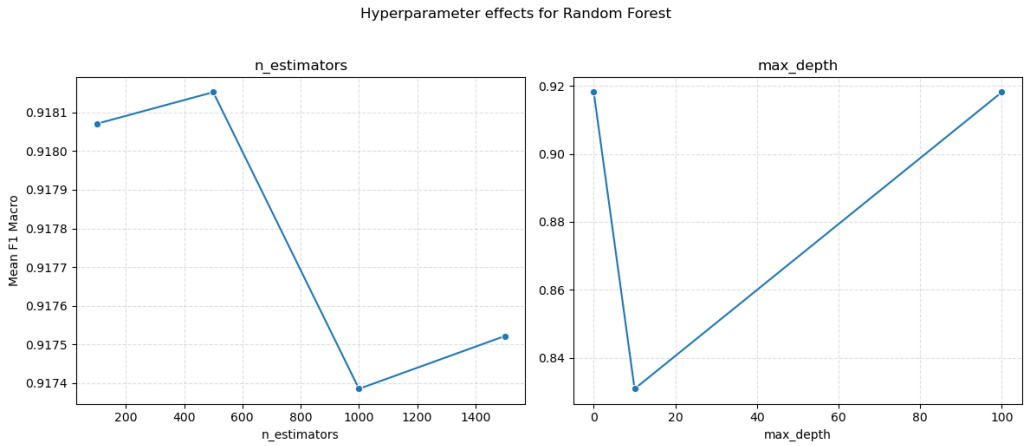



Figure 5: Grid Search Results: Random Forest

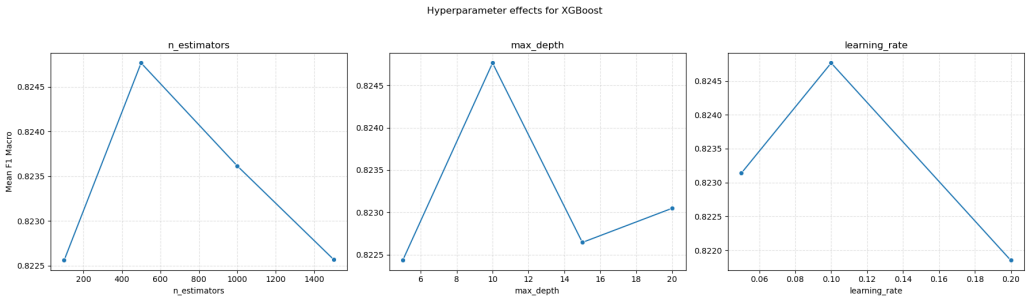



Figure 6: Grid Search Results: XGBoost

Table 2: Best performing hyperparameter grids explored per model (3-fold CV(2 for XGBoost), refit: F1_macro).

| Model | Hyperparameter | Value |
|---|---|---|
| KNN | `n_neighbors` | 2 |
| | `weights` | `uniform` == `distance` |
| | `p` | 1 |
| Random Forest | `n_estimators` | 500 |
| | `max_depth` | None |
| XGBoost | `n_estimators` | 500 |
| | `learning_rate` | 0.1 |
| | `max_depth` | 10 |

Using these settings, several single training runs were then performed on different data variants for further assessment. The following variations were considered:

- the initial train–test split (as provided),
- a new split created with `train_test_split` from `scikit-learn` on the combined initial data to improve the distribution of in-range WAPs,
- with vs. without feature scaling for **k**-NN,
- with vs. without including `SpaceID` in the target for all classifiers.

In these runs, the default Random Forest settings were found to yield better predictions than the best grid-search configurations. Further investigation showed that using `max_features = log2` produced the strongest results. Consequently, the subsequent tests were carried out with these Random Forest settings.

This observation also suggested that a Principal Component Analysis (PCA) might be beneficial, since limiting the effective feature set appeared to help the Random Forest classifier. Therefore, a PCA was performed. However, performance decreased substantially, with an F1-macro-Score of approximately 0.60. Consequently, this approach was not investigated further.

Table 3 reports the F1-macro scores for the different configurations. The mean F1-macro scores from the grid search appear comparatively low forXGBoost, which is likely due to the use of two-fold cross-validation compared to the initially provided single train-test split (which is more balanced regarding the ratio test/train). With only two folds, the validation fold is large, which can yield lower average scores.

Table 3: F1-macro (multiclass) across data variants and classifiers.

| **Split** | **Target** | **Scaling (KNN)** | **KNN** | **Random Forest** | **XGBoost** |
|---|---|---|---|---|---|
| Init Grid Search | B+F | Yes | 0.910 | 0.918 | 0.825 |
| Init | B+F | No | 0.875 | 0.912 | 0.892 |
| Init | B+F | Yes | 0.843 | – | – |
| New | B+F | No | 0.992 | 0.994 | 0.993 |
| New | B+F | Yes | 0.991 | – | – |
| New | B+F+SpaceID | No | 0.734 | 0.833 | 0.804 |
| New | B+F+SpaceID | Yes | 0.716 | – | – |

Notes: *Init* = initial train–test split, *New* = new `train_test_split`. *B+F* = composite of `BuildingID` and `Floor`; *B+F+SpaceID* adds `SpaceID`. Dashes (–) indicate configurations not evaluated.

# 4 Dataset 2: Myocardial Infarction Complications

## 4.1 Description

This dataset comprises 1,700 myocardial infarction (MI) patient records with features collected at four time points: at admission, and 24, 48, and 72 hours thereafter. In total, 111 mixed-type features cover demographics (e.g., age, sex), clinical history, and in-hospital measurements. Missing values occur in 109 of the 111 features; only two are complete.

The dataset provides 12 targets: 11 binary endpoints and one 8-class categorical endpoint. In this work, we restrict prediction to a single binary endpoint (*dead vs. alive*). This choice reflects severe class imbalance in `LET_IS` (several classes are strongly underrepresented) and time constraints. We further limit modeling to the 72-hour time point. Given the extent of

missingness and skewness, fully addressing the multi-class and multi-timepoint setting would exceed the available scope.

### 4.2 Data cleaning

We conducted a manual review of every variable and assigned explicit types: continuous laboratory and physiologic measurements, integers (counts or bounded numeric categories), categorical variables split into ordinal (inherent progression) versus nominal (unordered), and binary flags cast to Boolean. No duplicate entires were found in either the feature or target tables. Widespread missingness was observed across most fields.

For discrete event/action variables where absence is plausibly indicated by non-documentation, we applied the clinical convention "If it is not documented, it did not happen." Missing values in `R_AB_1_n`, `R_AB_2_n`, `R_AB_3_n` (pain relapses) and `NOT_NA_1_n` (NSAID use) were set to 0, and missing `IBS_NASL`, `NOT_NA_KB`, `LID_KB`, `NA_KB` (heredity or EMS drug administration flags) were set to `FALSE`, reflecting that such interventions would normally be recorded if present.

We dropped `KFK_BLOOD` (Serum CPK) due to extreme missingness and limited contemporary diagnostic utility versus troponin. Prehospital blood pressure fields `S_AD_KBRIG` and `D_AD_KBRIG` were removed because of pervasive non-documentation and inconsistent capture. All remaining missing values are deferred to structured imputation within the preprocessing pipeline, which operates separately on continuous, integer, ordinal, nominal, and binary feature subsets.

### 4.3 Preprocessing

We implemented a modular scikit-learn pipeline to isolate the effect of imputation and scaling choices on model performance. A ColumnTransformer splits features into four branches: (i) numeric (continuous + integer), (ii) ordinal categorical, (iii) nominal categorical, and (iv) binary flags. For numerics, we vary the imputer between `KNNImputer(`$k$`)`, `mean`, or `median` and optionally apply `StandardScaler`. Ordinal variables are imputed with `most_frequent` and encoded via `OrdinalEncoder` with fixed category order. Nominal variables are imputed with `most_frequent` and one-hot encoded (`handle_unknown=ignore`). Binary variables are cast to float and imputed either with `KNNImputer` or `most_frequent`. Features are further constrained by timepoint-specific availability (e.g., t0h, t24h, t48h, t72h). All steps are fit within the pipeline to avoid leakage during train/test splits and cross-validation. We sweep preprocessing configurations (numeric imputer, binary imputer, scaling, and $k$) across models (KNN, Random Forest, XGBoost) and compare metrics (accuracy, balanced accuracy, precision, recall, F1) to quantify their impact.

#### 4.3.1 Preprocessor evaluation

We evaluated twelve modular preprocessing configurations at $t_{72\text{h}}$, defined by the Cartesian product of three components: numeric imputer $\in \{\text{KNN}, \text{mean}, \text{median}\}$, binary imputer $\in \{\text{KNN}, \text{most\_frequent}\}$, and scaling $\in \{\text{on}, \text{off}\}$, with $k = 5$ for the KNN imputer. A full enumeration is beyond the scope of this report; the best-performing configurations for the `LET_IS_BINARY` target are summarized in Table 4.

Table 4: Strongest preprocessing configurations at $t_{72\text{h}}$ for `LET_IS_BINARY` (Test F1 score).

| **Model** | **Numeric** | **Binary** | **Scaling** | $k$ | **Timepoint** | **Test F1** |
|---|---|---|---|---|---|---|
| Random Forest | mean | KNN | on | 5 | $t_{72\text{h}}$ | 0.948 |
| XGBoost | mean | most_freq. | on | 5 | $t_{72\text{h}}$ | 0.951 |
| KNN | mean | KNN | on | 5 | $t_{72\text{h}}$ | 0.924 |

It's not shown here, but as expected, for tree-based models, scaling had no measurable effect on Random Forest or XGBoost, since both rely on feature-split thresholds rather than Euclidean distances. The combination without scaling performed as good as the one shown in the table. In contrast, the KNN classifier benefitted noticeably from feature scaling. We also plotted confusion matrices for the best performing model, but including these would not fit into this report.

## 4.4 Training & Evaluation

### 4.4.1 Train–Validation Split

We partition the data into training (80%) and test (20%) using a stratified split that preserves the class distribution of given target variable (for us the `LET_IS_BINARY`. The binary target is then trained/evaluated on these fixed indices. After splitting, we retain only the features admissible at the chosen timepoint (e.g., t72h). All preprocessing (imputation, encoding, scaling) is fit exclusively on the training split within a scikit-learn `Pipeline` to avoid leakage, and the fitted pipeline is applied to the test set.

### 4.4.2 Hyperparameter Tuning

For the MI dataset, we evaluated the same hyperparameter grids as in the previous experiments. Preprocessing parameters were fixed to the best-performing configuration identified in the initial preprocessor evaluation. The target variable was the binary indicator of myocardial infarction at $t = 72$ hours. A 3-fold grid search was performed with multi-metric scoring (accuracy, precision, recall, and macro F1), refitting each estimator on the macro F1 score before final evaluation on the hold-out test set. This ensured a consistent and fair comparison of hyperparameter influences across models.

### 4.4.3 Effects of Hyperparameters

We assessed the effect of each hyperparameter by plotting the mean test F1 score against the respective parameter while fixing all others at their best values (Figures 7–9).

- **Random Forest (Figure 7):** The model achieved its best performance with a smaller number of estimators ($n\_estimators = 100$) and unrestricted depth (`max_depth=None`). Increasing the number of estimators beyond this threshold provided negligible improvement. The performance decreased slightly for intermediate depths (e.g., `max_depth`=10) but recovered for very deep trees, suggesting that the model benefits from higher complexity on this dataset.
- **XGBoost (Figure 8):** The F1 score improved as both `n_estimators` and `max_depth` increased, peaking at `max_depth`=15 and `n_estimators`=500. A higher learning rate ($\eta = 0.20$) yielded the best results, indicating that faster convergence outweighed the risk of overfitting in this setting.
- **KNN (Figure 9):** The optimal configuration was found with few neighbors ($k = 2$), as performance decreased steadily with larger $k$. Both weighting strategies (`distance` and `uniform`) performed nearly identically, with a slight edge for `distance`. Manhattan distance ($p = 1$) again outperformed Euclidean distance ($p = 2$), consistent with results from other datasets.

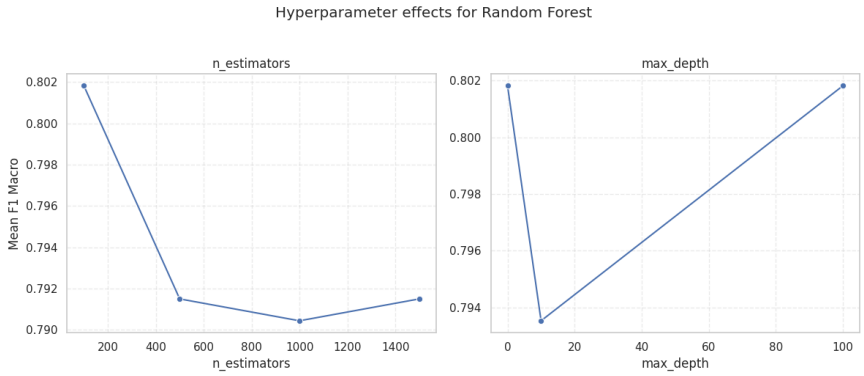



Figure 7: Hyperparameter effects for Random Forest on the MI dataset. The plots show mean test F1 versus `n_estimators` and `max_depth`, with other settings fixed at their best values (**Note:** `max_depth=0` ⇒ `max_depth=None`).

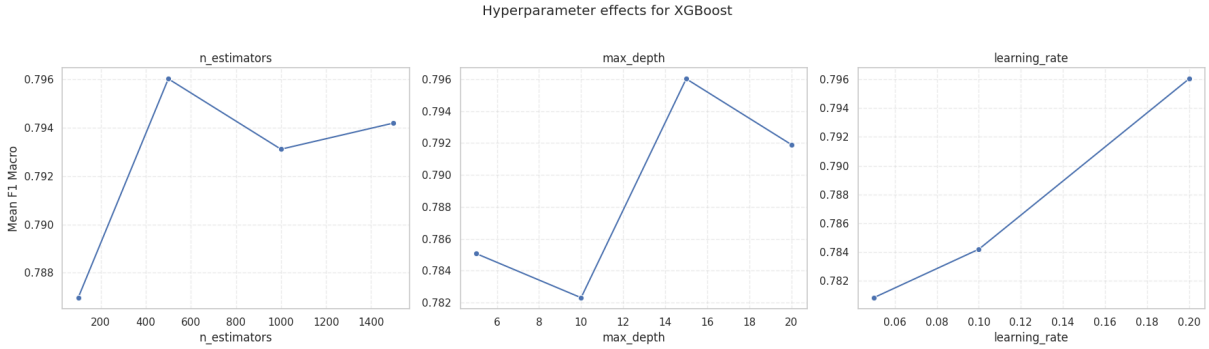



Figure 8: Hyperparameter effects for XGBoost on the MI dataset. The plots show mean test F1 versus `n_estimators`, `max_depth`, and `learning_rate`, fixing remaining parameters at their best values (**Note:** `max_depth=0` ⇒ `max_depth=None`).

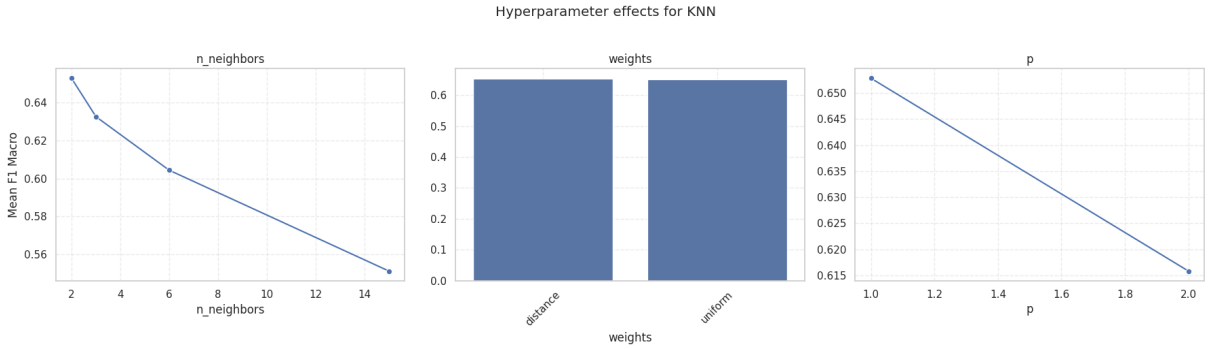



Figure 9: Hyperparameter effects for KNN on the MI dataset. The plots show mean test F1 versus `n_neighbors`, `weights`, and `p`, holding the other hyperparameters at their best settings.

## 4.5 Best Performing Model

For the MI dataset, we used the same preprocessing configuration as established in the preprocessor evaluation: numerical features were imputed with the mean, binary features with the KNN imputer ($k = 5$), and feature scaling was enabled only for KNN. Using these settings, the best hyperparameter configurations for each model were selected from the grid search and

refitted on the training data before evaluation on the held-out test set. Table 5 summarizes the results and hyperparameters.

Table 5: Best model performance on the MI dataset (test set).

| **Model** | **F1 Macro** | **Acc.** | **Best Params** |
|---|---|---|---|
| Random Forest | 0.802 | 0.912 | $n\_est$=100, $max\_depth$=None |
| XGBoost | 0.796 | 0.915 | $n\_est$=500, $max\_depth$=15, $\eta$=0.20 |
| KNN | 0.653 | 0.865 | $k$=2, $w$=dist, $p$=1 |

*Preprocessing:* numeric = mean, binary = KNN / most_freq., scaling = on for KNN

Both Random Forest and XGBoost achieved similar results, with macro F1 scores around 0.80 and test accuracy of about 0.912. Their class-wise performance was balanced, showing strong recall for the positive ("True") class while maintaining reasonable precision for the minority ("False") class. The KNN model, although performing significantly worse overall (F1 = 0.65, accuracy = 0.81), still demonstrated good recall for the majority class but struggled with imbalanced decision boundaries.

Overall, the **Random Forest** model achieved the best trade-off between interpretability, robustness, and predictive accuracy, and was therefore selected as the final model for this dataset.

#### 4.5.1 Hold-out vs. k-fold Cross-Validation

The hold-out evaluation trains the tuned Random Forest once on an 80% stratified training split and tests on the remaining 20%, which is fast but sensitive to how the data are split. Stratified 10-fold cross-validation instead refits the full preprocessing + model pipeline on nine folds and validates on the remaining one, yielding more stable and reliable performance estimates.

On the MI dataset, the hold-out test reached an accuracy of 0.91 and an F1 score of 0.81. Across ten folds, the Random Forest achieved a mean accuracy of $0.91 \pm 0.01$ and mean F1 of $0.95 \pm 0.01$, indicating stable generalization and low variance between folds. While the hold-out score is slightly lower, both methods are consistent, suggesting minimal overfitting.

Table 6: Hold-out vs. 10-fold cross-validation results for Random Forest.

| **Metric** | **Hold-out** | **10-fold Mean ± SD** |
|---|---|---|
| Accuracy | 0.912 | $0.905 \pm 0.012$ |
| Balanced Accuracy | 0.765 | $0.766 \pm 0.036$ |
| F1 Score | 0.805 | $0.945 \pm 0.007$ |

## 5 Dataset 3: Breast Cancer

### 5.1 Description

This dataset can be used to predict whether a patient experiences "recurrence-events" or "no-recurrence-events", corresponding to a malignant or benign diagnosis. It consists of 285 labeled training and 284 unlabeled test samples. The dataset contains **no missing values** and **no duplicate entries**.

Before preprocessing, the dataset includes 32 feature variables: a non-predictive **ID** (integer), the target variable **class** (boolean), and 10 numerical attributes describing various properties of cell nuclei (e.g. texture or radius). Each attribute is represented by its **mean** (see Figure 10),

**standard error**, and **worst** value. In the training set, 175 samples have `class=True` (61.4%) and 110 samples have `class=False`.

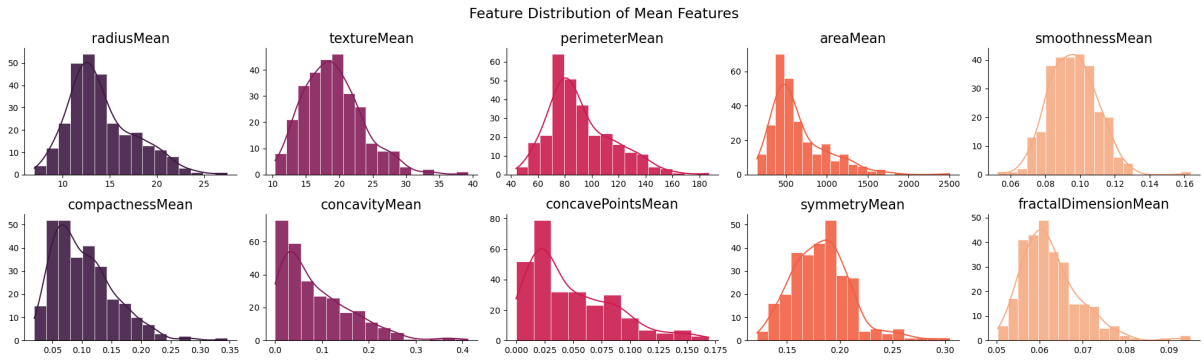



Figure 10: Breast Cancer Dataset: Distribution of Mean Features

## 5.2 Preprocessing

The preprocessing was straightforward, as the dataset contains only numerical features. We encoded the boolean target variable as integer and removed the non-predicitve ID.

Subsequent steps included scaling the feature variables using either `StandardScaler` or `MinMaxScaler` from `scikit-learn`, removing outliers using the interquartile range (IQR) method, and dropping features that were highly correlated above a specified threshold.

We tried adding additional engineered features (e.g. ratio between standard error and mean, or shape/geometry ratios), but these did not improve the performance and are no further considered within this report.

The processing pipeline is implemented using a `ColumnTransformer`, which scales the numerical features when required (see Table 7). This transformer is then combined with the classifiers into a single `Pipeline` for streamlined training and evaluation.

| Preprocessing Step | Options |
|---|---|
| Scaling | **False**: no scaling, **"Standard"**: StandardScaler(), **"MinMax"**: MinMaxScaler() |
| Outliers | **-1**: keep outliers, **0.01, 0.05**: quantile value |
| Correlation | **-1**: keep all features, **0.9, 0.95**: threshold value |

Table 7: Different configurations for preprocessing

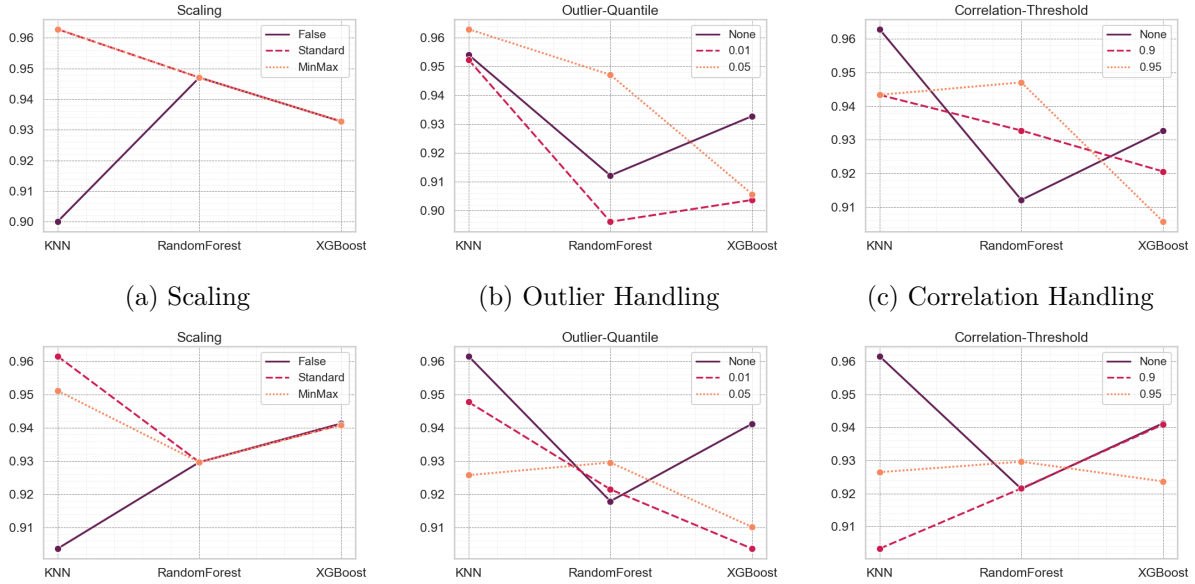
## 5.3 Training & Evaluation

To identify the optimal preprocessing configuration and best-fitting hyperparameters for each classifier, we performed a grid search for every classifier across all combinations of preprocessing configurations (see Table 7).

### 5.3.1 Effects of Preprocessing Steps

The **effects of different preprocessing and cleaning steps** are illustrated in Figure 11. Among the preprocessing strategies, feature **scaling** has a notable impact only on K-Nearest Neighbors (KNN), improving the mean F1 score by up to 6 percentage points. This is expected, as KNN relies on distance calculations, which are sensitive to the scaling of features, whereas tree-based models such as Random Forests and XGBoost are largely invariant to feature scaling.

**Outlier removal** and **redundant feature elimination** show more variable effects across classifiers. For KNN and XGBoost, these steps generally lead to decreased performance, likely because removing extreme values or features inadvertently reduces the richness of the data that the model can exploit. In contrast, Random Forests sometimes benefit from these cleaning steps, with minor improvements in F1 score, possibly due to reduced noise and feature redundancy. These results highlight that preprocessing and cleaning steps are highly model-dependent: distance-based methods are sensitive to scaling, while tree-based methods can tolerate unscaled and redundant features, but may still be affected by the removal of potentially informative outliers.




(a) Scaling (b) Outlier Handling (c) Correlation Handling

(d) Scaling (e) Outlier Handling (f) Correlation Handling

Figure 11: Effect of different preprocessing configurations for each classifier by comparing the mean F1 score of the best result per classifier and configuration. Subfigures A–C show results where the training dataset was split 80–20, while subfigures D–F show results using a 90–10 training–evaluation split.

### 5.3.2 Effects of Hyperparameters

For **KNN** (Figure 12), distance weighting outperforms uniform weighting. Performance decreases with higher values of p, meaning Manhattan distance performs better than Euclidean distance, and it declines slightly when using more than 7 neighbors.
**Random Forest** (Figure 13) shows a minor performance improvement when the number of estimators exceeds 1000, while maximal depth has little to no effect.
For **XGBoost** (Figure 14), the tested hyperparameters do not substantially influence performance on the Breast Cancer dataset. This is likely because the dataset is relatively small, so changes in hyperparameters do not produce strong or consistent effects on model performance.

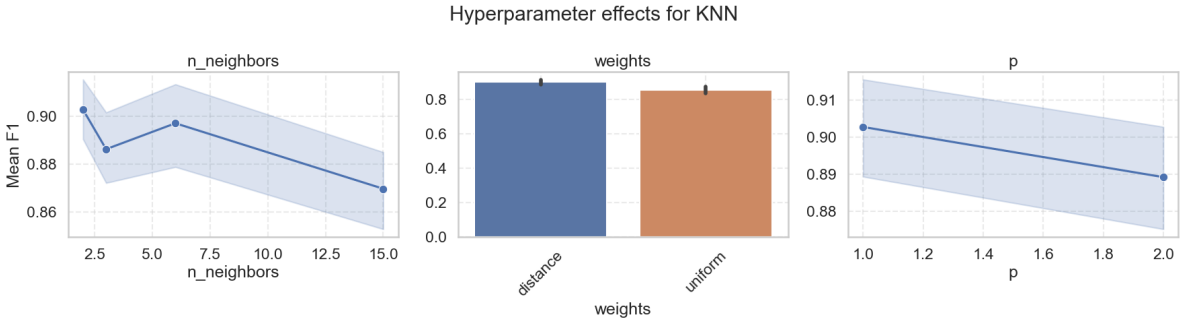



Figure 12: Hyperparameter effects for KNN on the Breast Cancer dataset. The plots show mean test F1 versus `n_neighbors`, `weights`, and `p`. Shaded areas represent variability across different preprocessing configurations.

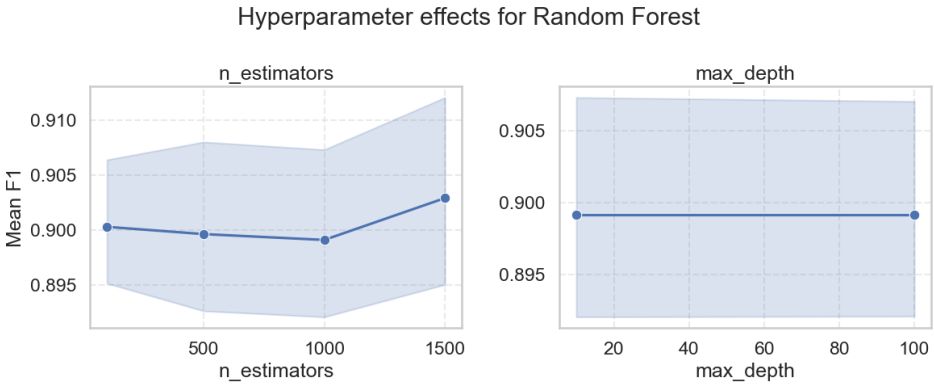



Figure 13: Hyperparameter effects for Random Forest on the Breast Cancer dataset. The plots show mean test F1 versus `n_estimators`, and `max_depth`. Shaded areas represent variability across different preprocessing configurations.

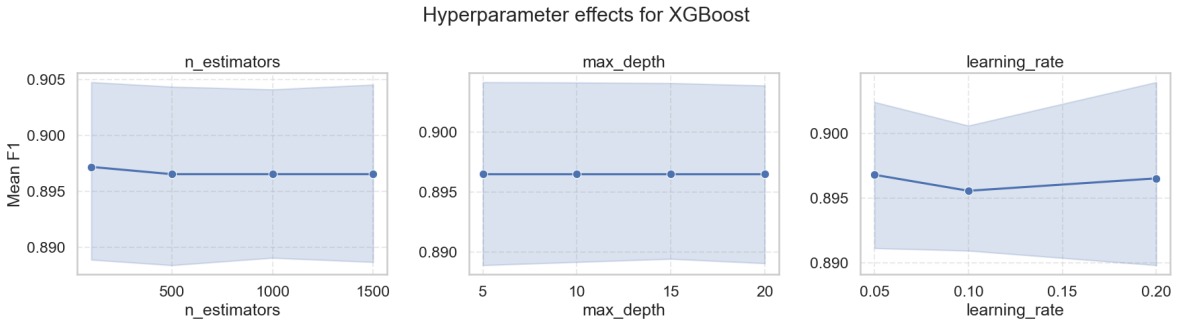



Figure 14: Hyperparameter effects for XGBoost on the Breast Cancer dataset. The plots show mean test F1 versus `n_estimators`, `max_depth`, and `learning_rate`. Shaded areas represent variability across different preprocessing configurations.

#### 5.3.3 Cross-Validation

| Split Ratio | KNN | Random Forest | XGBoost |
|---|---|---|---|
| 80:20 | 0.9628 | 0.9471 | 0.9327 |
| 90:10 | 0.9616 | 0.9296 | 0.9413 |

Table 8: Comparison of best mean F1 scores for different training dataset split ratios for different classifiers.

# 6 Dataset 4: Loan Grade Prediction

## 6.1 Description

The **Loan** dataset is a multiclass classification problem (grades A–G) with 10,000 training instances, 10,000 test instances, 91 features, and one target variable. Both splits are clean: no missing values and no duplicate rows. Before preprocessing there were 69 float features, 9 integer features, and 14 categorical (object) features. Several attributes encode information that would realistically be available only after loan issuance; nonetheless, all features were retained to maximize Kaggle leaderboard performance.

We produced (but omit here due to page limits) the following diagnostic plots: target class distribution, correlation heatmap of top grade-correlated features, pairwise feature (pairplot) relationships and boxplots for outlier inspection across grades. These supported feature retention (no aggressive outlier removal) and guided final model selection.

## 6.2 Cleaning

The loan dataset required only light cleaning. Binary categorical fields (`pymnt_plan`, `initial_list_status`, `hardship_flag`, `debt_settlement_flag`) were converted to boolean features with a `_bin` suffix and the original columns were dropped. Ordinal categories were encoded as integers to preserve order (`term` $\rightarrow$ 36/60, `emp_length` $\rightarrow$ 0–10, `verification_status` $\rightarrow$ 0–2). Date information was consolidated by creating `issue_d_months_since_reference` = (`issue_d_year` $-2012)\cdot 12+$ `issue_d_month` and `credit_history_age` in months between the earliest credit line and the issue date, after which the original year/month columns were removed. The constant `policy_code` (always 1.0) was dropped, and there were no missing values or duplicates. After preprocessing, the training set still contains 10,000 rows and a feature space dominated by numerical variables with a few boolean and a small number of remaining categorical variables, which are one-hot encoded in the modeling pipeline. Highly correlated features (some $\geq 0.95$, even 1.0) were identified but retained because removing them degraded validation performance and the Random Forest model handled redundancy well. Outliers observed in boxplots of the top correlated features were also kept given the robustness of tree-based models.

## 6.3 Preprocessing

After cleaning, we removed the identifier (`ID`), separated features ($X$) from the target ($y =$ `grade`) and built a preprocessing pipeline with a `ColumnTransformer`: numerical features are either standardized with `StandardScaler` (only when required **and** benefitial, e.i., for KNN) or passed through unchanged; categorical features are **One–Hot encoded** to ensure consistent columns and robustness to unseen categories. Boolean binary features are passed through without modification. This transformer is composed with each classifier in a single `Pipeline` to keep preprocessing consistent across training, validation, and test. For models requiring integer labels (**XGBoost** and **KNN** in the multiclass setup), `grade` is label-encoded only for fitting

and predictions, and predictions are mapped back to the original grade labels for evaluation and submission.

## 6.4 Training & Evaluation

### 6.4.1 Train–Validation Split

Since the competition test set had no labels and was scored on Kaggle, we created an internal stratified 80/20 train–validation split. After grid search with cross-validation, models were re-evaluated on this hold-out set to estimate generalization before retraining on the full training data for submission.

### 6.4.2 Hyperparameter Tuning

Grid search with 3-fold cross-validation was performed using unified preprocessing–estimator pipelines to avoid leakage. Numerical features were standardized only for KNN, categorical features one-hot encoded with `handle_unknown=ignore`, and binary features passed through unchanged. We used multi-metric scoring (accuracy, precision, recall, Macro F1) and refit on Macro F1. Hyperparameter grids per model are given in the parameter table above; KNN and XGBoost required integer-encoded labels.

### 6.4.3 Effects of Hyperparameters

Figure 15–17 show mean test F1 across hyperparameters. For KNN, performance improved up to $k = 6$ with Manhattan distance ($p = 1$) and distance weighting. XGBoost peaked at `n_estimators` $= 100$, `max_depth` $= 5$, and `learning_rate` $= 0.1$. Random Forest performed best with `max_depth=0` $= 10$ and roughly $n_{\text{estimators}} = 1000$, with little gain beyond that depth or size. (**Note:** `max_depth=0` denotes `None`.)

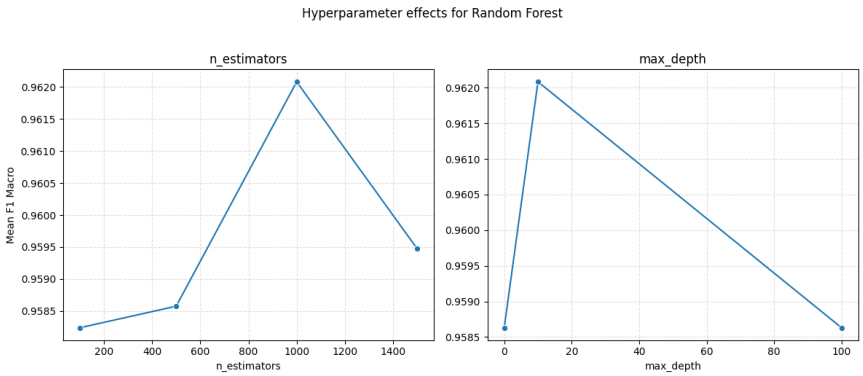



Figure 15: Hyperparameter effects for Random Forest on the Loan dataset.

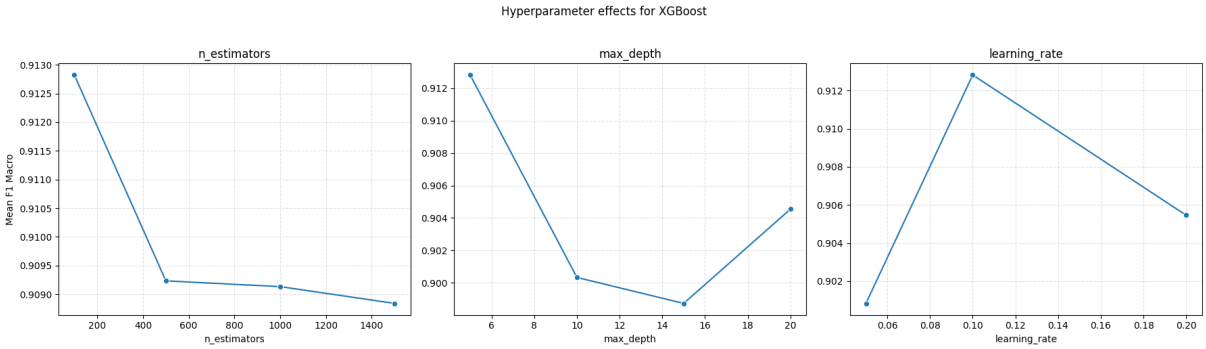



Figure 16: Hyperparameter effects for XGBoost on the Loan dataset.

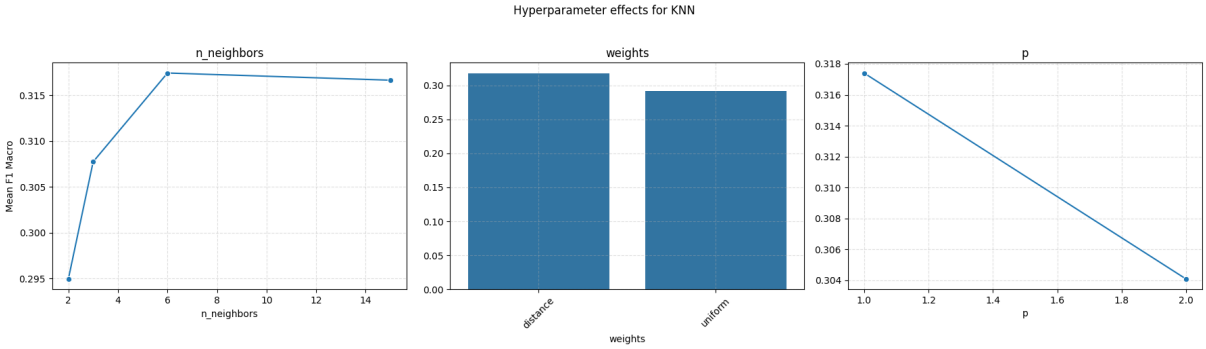



Figure 17: Hyperparameter effects for KNN on the Loan dataset.

### 6.4.4 Effect of Scaling on KNN

Using tuned hyperparameters, we compared KNN with and without numeric feature scaling. Standardization improved both balance and overall performance: macro F1 increased from 0.20 to 0.32 and accuracy from 0.32 to 0.48. Hence, scaling was retained in the final pipeline.

## 6.5 Best Performing Model and Kaggle Submission

Random Forest achieved the highest validation F1 and was chosen for the Kaggle submission. A refined 5-fold grid search explored $n_{\texttt{estimators}} \in [950, 1100]$ and $\texttt{max\_depth} \in \{9, 10, 11, 12\}$. Although a slightly smaller configuration (975/9) appeared optimal in CV, it underperformed on Kaggle, so the final model used $\texttt{max\_depth} = 10$, $n_{\text{estimators}} = 1000$. Retraining on all data yielded a public leaderboard score of **0.9938**.

Table 9: Summary of best models (Loan dataset, validation set).

| **Model** | **Accuracy** | **F1 (Macro)** | **Key Params** |
|---|---|---|---|
| Random Forest | 0.99 | 0.97 | $n_{\text{est}} = 1000$, $\texttt{max\_depth} = 10$ |
| XGBoost | 0.99 | 0.85 | $n_{\text{est}} = 100$, $\texttt{max\_depth} = 5$, $\eta = 0.1$ |
| KNN (scaled) | 0.48 | 0.32 | $k = 6$, $p = 1$, distance-weighted |

Random Forest achieved near-perfect performance on the internal validation set ($F1 = 0.97$, $Acc = 0.99$), far exceeding XGBoost ($F1 = 0.85$) and KNN ($F1 = 0.32$). It therefore served as the final model for submission and interpretation.

# 7 Conclusion

In this assignment, we applied three classification algorithms–k-Nearest Neighbors (KNN), Random Forest, and XGBoost–across four heterogeneous datasets: indoor localization (UJIIndoorLoc), clinical outcomes after myocardial infarction (MI), breast cancer recurrence prediction, and loan grade prediction. The experiments demonstrated how data modality, preprocessing, and model characteristics jointly determine performance.

Table 10 summarizes the best-performing configurations and metrics for all datasets. Across nearly all cases, Random Forest and XGBoost consistently achieved the highest F1 or F1-macro scores, confirming their robustness and generalization ability. In contrast, KNN was more sensitive to feature scaling and data dimensionality, performing well only on smaller, dense numerical datasets.

Table 10: Best results for all datasets and models (test/validation set).

| **Dataset** | **Model** | **Acc.** | **F1** | **Key Params** | **Scaling** |
|---|---|---|---|---|---|
| UJIIndoorLoc | RF | 0.907 | 0.918 | $n = 500$, depth=None | Yes |
| | XGB | 0.845 | 0.825 | $n = 500$, $\eta = 0.1$, d=10 | Yes |
| | KNN | 0.912 | 0.906 | $k = 2$, $p = 1$, unif. | Yes |
| MI Comp. | RF | 0.912 | 0.805 | $n = 100$, depth=None | No |
| | XGB | 0.915 | 0.796 | $n = 500$, $\eta = 0.2$, d=15 | No |
| | KNN | 0.865 | 0.653 | $k = 2$, $p = 1$, dist. | Yes |
| Breast Ca. | RF | 0.96 | 0.95 | $n = 1000$, depth=10 | Yes |
| | XGB | 0.95 | 0.93 | $n = 1500$, $\eta = 0.12$, d=20 | No |
| | KNN | 0.97 | 0.96 | $k = 2$, $p = 1$, dist. | Yes |
| Loan | RF | 0.99 | 0.97 | $n = 1000$, depth=10 | No |
| | XGB | 0.99 | 0.85 | $n = 100$, $\eta = 0.1$, d=5 | No |
| | KNN | 0.48 | 0.32 | $k = 6$, $p = 1$, dist. | Yes |

Overall, ensemble-based methods (Random Forest, XGBoost) proved superior across domains–delivering near-perfect results on structured tabular data such as Loan Grades and UJIIndoorLoc, and strong generalization on clinical datasets with high missingness such as MI Complications. KNN performed competitively on low-dimensional numeric data but degraded sharply in high-dimensional or sparse settings. Feature scaling was critical for KNN yet irrelevant for tree-based models.