ST443 Group 1 Project

Abstract

This report investigates the performance of machine learning models across two high-dimensional classification tasks. Task 1 addresses multi-class land-type classification using pixel-level image features, while Task 2 focuses on feature selection for binary decision classification based on neural activity recordings. Both tasks begin with exploratory data analysis, covering data structure, distributions, and feature characteristics. A range of models is then trained and evaluated using multiple performance metrics. Task 1 also includes a brief real-world application, whereas Task 2 emphasises identifying informative features in a highly sparse, high-dimensional dataset.

1 Task 1: multi-class classification

1.1 Methodology

The high-dimensional pixel dataset was preprocessed by detecting invalid entries and extracting all spectral-band features. We then split the data into an 80% training set (with a small subsample used for particular models due to memory constraints) and a 20% test set using stratified sampling to preserve class proportions. All models were implemented using `scikit-learn` pipelines with standardisation. We additionally evaluated PCA versions of all models, using 10 principal components fitted on the training set.

Each classifier—*Discriminant Analysis*, *Logistic Regression*, *k-NN*, *Tree-based Models*, and *SVM*—was first evaluated using 5-fold stratified cross-validation, computing *Accuracy*, *Balanced Accuracy*, *Macro-F1*, *Macro-AUC*, and confusion matrices. The models were then retrained on the complete 80% training set and assessed once on the untouched 20% test set to obtain unbiased performance estimates and identify candidates for the final glacier–ice binary classification task.

1.2 Visualisation and summary statistics

The dataset has dimensions 215,604 × 223, with 218 spectral-band features and no missing values or duplicates. Since reflectance values must lie within [0,1], we checked for invalid entries and found about 0.84% outside this range; these were clipped to maintain consistency. Approximately 5.17% of values were flagged as outliers under a 3-sigma rule. But given the small proportion and the fact that extreme reflectance (e.g., from bright snow) can occur naturally, we retained them.

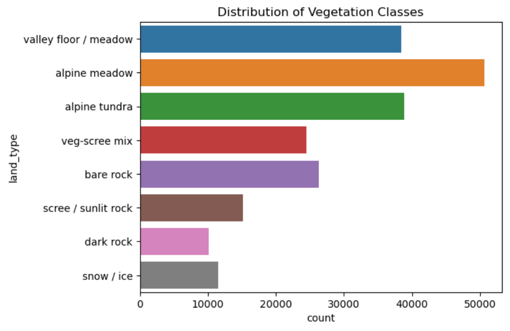


Figure 1: Class-wise distribution

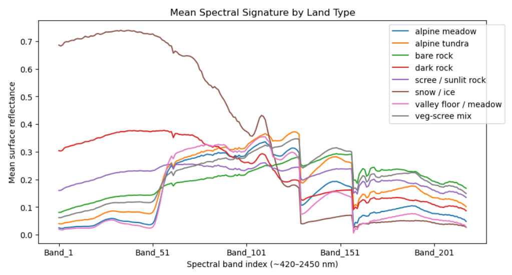


Figure 2: Mean Spectra

For visual exploration, we first examined class distributions and inter-class relationships. From Figure 1, *alpine meadow* is the most frequent class, accounting for about a quarter of all observations, while *valley floor* and *alpine tundra* are also common, together making up around 58% of the dataset. This imbalance highlights the need for careful stratification in cross-validation, as uniform splits could cause minority classes to vanish in some folds. Figure 2 shows that classes with distinct spectral curves should be easier to separate, whereas overlapping curves indicate spectrally similar classes where linear methods may struggle, and nonlinear models may perform better. The smooth variation across neighbouring bands further suggests strong correlation among features, supporting the use of PCA or regularisation in later modelling.

To examine these correlations more closely and assess whether PCA is appropriate for this feature space, we plot the correlation heatmaps for adjacent bands and for all spectral bands:

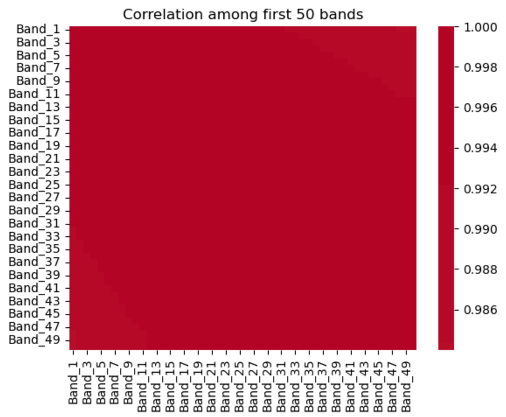


Figure 3: Adjacent correlation heatmap

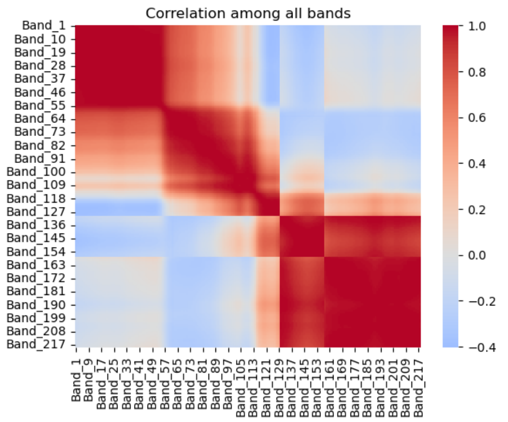


Figure 4: All bands correlation heatmap

Figure 3 shows strong correlations among adjacent spectral bands, meaning many channels capture almost the same information. In contrast, the blue regions in Figure 4 represent bands from different spectral ranges, showing lower or even negative correlation. This block-like pattern highlights substantial multicollinearity among the features. It motivates the use of PCA to compress redundant bands into a smaller set of orthogonal components while retaining most spectral variation.

As the last step before modelling, we applied PCA and clustering as exploratory methods. PCA reduces highly correlated, high-dimensional band data to a smaller set of informative components, revealing broad structure. A test PCA with four components explains 99.87% of the variance, but K-means on the first two components offers limited interpretive value, with clusters arbitrarily separated (Figure 5). Therefore, we use PCA solely for dimensionality reduction before modelling.

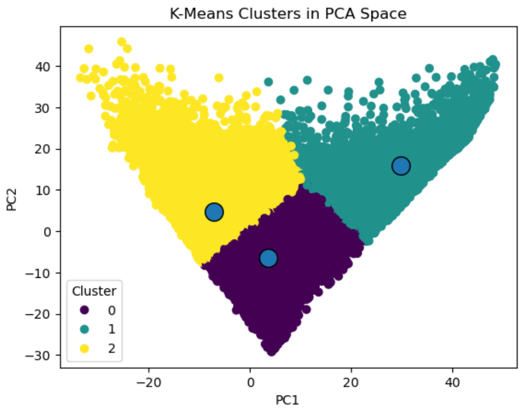


Figure 5: K-Means clustering for the 2 testing principal components

1.3 Model development

In this part of the investigation, we aim to predict vegetation type from pixel reflectance values. We consider several types of classifiers for this multi-class classification problem. Categorical labels are encoded numerically using `scikit-learn` 's `LabelEncoder`. Our methodology is as follows: we evaluate each model using cross-validation on the training set, select the best-performing models, and then retrain them on the full training portion before evaluating performance on the test set. The test set is used only for final reporting.

1.3.1 Inputs and evaluation framework

We use all available spectral-band measurements as the feature matrix *X* and the land-type labels as the target variable *y*. Before fitting models, we standardise all features to ensure comparability across bands, and we evaluate each classifier on both the raw features and a reduced-dimensional representation obtained via PCA with 10 components.

To reliably assess model performance, we use 5-fold stratified cross-validation on the training set. Stratification preserves class proportions in each fold, which is especially important given the imbalanced land-type distribution observed earlier. For each model, we compute several performance metrics: *Accuracy*, *Balanced Accuracy*, *Macro-F1*, and *Macro-AUC*. The *Macro* versions average performance across classes, making them more sensitive to imbalance. After cross-validation, models are retrained on the full 80% training split and evaluated once on the untouched 20% test set to obtain unbiased performance estimates.

1.3.2 Linear and Quadratic Discriminant Analysis (LDA & QDA)

LDA assumes that each class follows a multivariate Gaussian distribution with its own mean vector but a shared covariance matrix, leading to linear decision boundaries. The discriminant function is:

$$\delta_k(x) = x^\top \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^\top \Sigma^{-1}\mu_k + \log \pi_k,$$

QDA relaxes the LDA assumption by allowing each class to have its own covariance matrix. This yields more flexible quadratic decision boundaries. The discriminant function is:

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k) + \log \pi_k,$$

Both classifiers predict based on the rule of:

$$\psi_(x) = \arg\max_k \ \delta_k(x).$$

Furthermore, we applied a regularisation parameter of 0.01 to QDA to ensure a non-singular covariance matrix, avoiding model crashes due to the highly correlated features.

1.3.3 Logistic Classifier

As one of the most fundamental models in machine learning, the *Logistic Classifier* estimates the probability of each class given the input features. In the binary case, it models the log-odds of class membership as a linear function of the predictors. For multi-class problems, logistic regression uses the softmax function to model probabilities across all classes:

$$P(y=k \mid x) = \frac{\exp(\beta_k^\top x)}{\sum_{j=1}^K \exp(\beta_j^\top x)}.$$

In addition, to ensure parameter convergence, we trained the classifier for 2000 iterations, using the `lbfgs` solver, which provides faster, more stable optimisation.

1.3.4 Tree-type models

Gradient Boosting Decision Trees (GBDT) build an additive ensemble of shallow trees, where each new tree is trained to correct the residual errors of the previous ones. By iteratively minimising a differentiable loss via gradient descent in function space, GBDT captures nonlinear relationships and class-specific patterns. Although it is more sensitive to hyperparameters, it often performs well on high-dimensional, structured data. We chose the GBDT with 100 trees, a learning rate of 0.15 to ensure stable operation and regularisation, shallow trees with max_depth=2 to prevent overfitting, and used only 70% of samples per tree to reduce variance:

$$F_m(x) = F_{m-1}(x) + \nu\, h_m(x), \quad m = 1, \ldots, M,$$

$$F_0(x) = \arg\min_c \sum_i \ell(y_i, c),$$

An alternative was the *Random Forest* model, which leveraged multiple simple decision trees to reduce variance when averaging the trees. In this case, we set the forest to have 100 trees and a minimum sample split of 2.

1.3.5 Nearest Neighbour Classifier (k-NN)

K-Nearest Neighbours is a simple, non-parametric classification algorithm that predicts the label of a new sample by examining the labels of its k closest training samples under a chosen distance metric (usually Euclidean distance). In high-dimensional settings like task 1, distances between points become less informative, thereby reducing k-NN's discriminative power:

$$\psi_{\mathrm{kNN}}(x) := \arg\max_{k} \sum_{i=1}^{K} \mathbf{1}\left\{ Y^{(i)} = k \right\},$$

However, due to the high dimensionality of the data, k-NN ran out of memory. To address this while preserving fair comparability across models, we trained k-NN on a randomly selected subset of the training data. Crucially, the final evaluation was still performed on the same 20% untouched test set used by all other models, ensuring consistency in evaluation and avoiding data leakage. Additionally, we chose k=7 to balance model stability and flexibility, and used distance weights to reduce the influence of irrelevant, far-away neighbours.

1.3.6 Support Vector Machine (SVM)

Support Vector Machines classify data by finding the optimal separating hyperplane that maximises the margin between classes. Only a subset of the training data—the support vectors—determines this boundary, making SVM robust and effective in high-dimensional feature spaces:

$$\psi_{\mathrm{SVM}}(x) = \mathrm{sign}\left(\beta_0^* + \sum_{i \in \mathcal{S}} \alpha_i^* \, y_i \, k(x, x_i) \right),$$

To ensure comparability across models, the SVM used the same training–testing configuration as k-NN. We set the SVM misclassification penalty to C=20; such a large value forces the classifier to fit the training data more tightly by penalising errors more heavily, which is helpful in our setting because the classes become more separable after PCA. We also set $k(x, x_i)$ as the Gaussian (RBF) kernel, with the kernel coefficient γ set to 'scale', which ensures the decision boundary remained smooth despite the large Euclidean distances between spectral signatures.

1.4 Model performance

As mentioned in the methodology, we split the data into two parts: 80% for training (with a further subset for SVM and k-NN) and 20% for testing, and used 5-fold cross-validation to prevent overfitting. Now that we have fitted all the desired models, we evaluate their performance, all computed on the same untouched 20% test set.

Model	Acc	BalAcc	AUC	F1
Logistic	0.9922	0.9911	0.99995	0.99116
LDA	0.8613	0.8598	0.99080	0.86179
QDA	0.9217	0.9183	0.99558	0.91886
RF	0.9849	0.9832	0.99987	0.98353
GBDT	0.9475	0.9521	0.99842	0.95305
KNN	0.9613	0.9595	0.99869	0.96081
SVM	0.9904	0.9895	0.99995	0.98986

(a) Raw Models

Model	Acc	BalAcc	AUC	F1
Logistic_PCA10	0.9916	0.9923	0.99994	0.99248
LDA_PCA10	0.8406	0.8331	0.98853	0.83837
QDA_PCA10	0.9311	0.9284	0.99637	0.92789
RF_PCA10	0.9764	0.9729	0.99966	0.97388
GBDT_PCA10	0.9534	0.9451	0.99828	0.94638
KNN_PCA10	0.9624	0.9605	0.99869	0.96172
SVM_PCA10	0.9892	0.9888	0.99993	0.98918

(b) PCA-10 Models

Table 1: Performance comparison of models with and without PCA.

As shown in Table 1a, most raw models performed very well across all four metrics, though with some noticeable differences. Logistic Regression, SVM, and RF consistently achieved high Accuracy, Balanced Accuracy, AUC, and F1 score, suggesting that they effectively handled the data's high dimensionality. In contrast, LDA and QDA performed weaker across most metrics, likely due to their stronger assumptions on class covariance that do not hold for this dataset. GBDT and k-NN fell between the top-performing models, performing reasonably but not reaching their level.

After applying PCA with 10 components (Table 1b), the overall ranking of models remained similar. Logistic Regression and SVM stayed stable and strong, while RF showed a slight drop—as expected, since tree-based models do not always benefit from linear dimension reduction. LDA and QDA continued to struggle after PCA, suggesting that the generative approach to classification may not be suitable for this dataset, especially if the features do not follow class-conditional Multivariate Normal assumptions. In contrast, k-NN performance remained broadly consistent.

In the end, we implemented a `mypredict()` function to train our final chosen model on the full labelled dataset and generate predictions for the provided external test set. This function automates the complete pipeline for Task 1, including preprocessing, model fitting, prediction, and output.

1.5 Application: glacier-ice detection

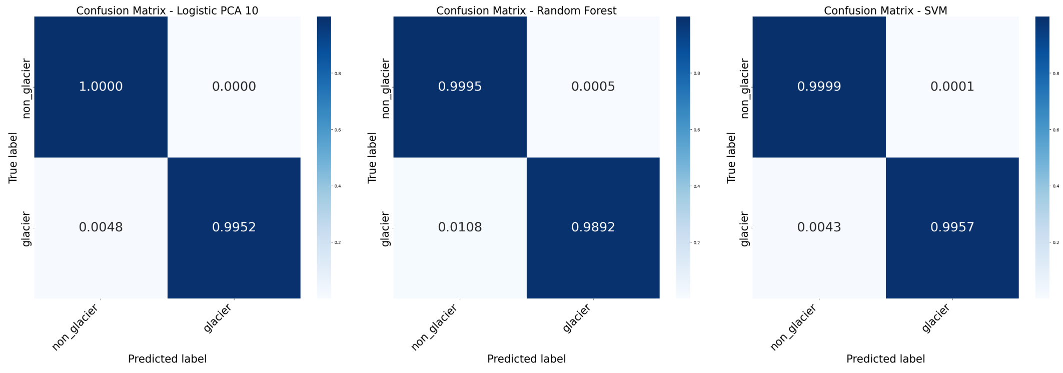
For this task, we relabelled *glacier* as the positive class and merged all other land types into the negative class, reducing the original multi-class problem to a binary one. The *F1 score*—the harmonic mean of precision and recall—provides a balanced measure of performance by penalising both false positives and false negatives, making it more suitable than accuracy or *AUC* when identifying a comparatively rare class. The three best-performing models were PCA10 Logistic Regression (0.99248), raw SVM (0.98904), and raw Random Forest (0.98353).

We reused the same 80/20 stratified train–test split and preprocessing from Task 1: features were standardised using parameters fitted on the training set. For the binary task, each model produced glacier-ice probabilities thresholded at 0.5, and performance was evaluated on the same held-out 20% test set. The classifiers have the same configurations as before: 2000 iterations for Logistic Regression, an RBF kernel with C=20 for SVM, and 100 trees (minimum split 2) for Random Forest. The key difference is the binary target, which allows use of the standard (binary) F1 score and enables SVM to train on the full dataset rather than a subsample, since the two-class formulation is far less computationally demanding than the multi-class case.

Model	F1
Logistic_PCA10	0.997177
RF_Raw	0.990224
SVM_Raw	0.996745

Table 2: *F1 scores*

Table 2 summarises the F1 scores of the three evaluated models. Logistic_PCA10 achieved the highest score (0.9972), indicating that PCA preserved nearly all discriminative information. Raw SVM performed similarly well (F1 = 0.9967), while Raw RF obtained a slightly lower score (0.9902) but still showed strong predictive ability. Overall, the slight performance differences suggest that the underlying signal is highly separable and that multiple model classes can achieve excellent Accuracy. We may gain more intuitive insight from their corresponding confusion matrices:



(a) PCA10 Logistic (b) Raw RF (c) Raw SVM

Figure 6: Comparison of confusion matrices for the three models.

All three confusion matrices (Figure 6) show that the models correctly classify almost all samples in both classes, consistent with their very high F1 scores. PCA10 Logistic Regression and the raw SVM achieve nearly perfect separation, with misclassification rates below 0.01% for both classes. RF performs similarly across the non-glacier class but shows a slightly higher false-negative rate for glacier pixels, consistent with its relatively lower F1 score. Overall, the matrices confirm that all models are highly accurate, with only minor differences in their handling of glacier misclassifications.

1.6 Summary: Task 1

Task 1 investigated multi-class land-type classification using a high-dimensional spectral dataset. After confirming data validity and exploring the structure of the spectral bands, we observed strong correlations across adjacent wavelengths and class-dependent spectral patterns, motivating the use of standardisation and PCA. A range of linear, nonlinear, and ensemble-based models was developed and evaluated under a consistent 5-fold stratified cross-validation framework, using Accuracy, Balanced Accuracy, Macro-F1, and Macro-AUC as performance metrics.

Across all models, Logistic Regression, SVM, and Random Forest achieved the strongest results, both with raw inputs and with PCA-compressed features, indicating that the dataset is highly separable and well-suited to a variety of classifiers. PCA helped simplify the feature space without sacrificing predictive performance, particularly for linear models.

For the final approach, we selected the PCA10 Logistic Regression classifier because it consistently delivered the best validation and test performance while remaining simple, stable, and easy to reproduce. The final `mypredict()` method applies the stored scaler and PCA transform to new data, runs the trained classifier, and outputs the land label.

For the glacier–ice binary task, the top models from the multi-class setting (PCA10 Logistic Regression, SVM, and Random Forest) were retrained on the binary labels. All achieved near-perfect performance, with PCA10 Logistic Regression performing marginally best ($F1score = 0.997177$). This indicates that glacier ice has a particularly distinctive spectral signature, allowing even simple models to separate it reliably.

2 Task 2: feature selection

The neural dataset is high-dimensional and likely to contain many weakly informative or correlated features, which motivates feature selection to reduce noise and improve generalisation. Unlike the previous classification task, this task focuses specifically on selecting informative subsets of features. Feature-selection tools ("selectors") fall into three categories—*Embedded*, *Filter*, and *Wrapper* methods—but wrapper approaches are computationally infeasible for a dataset of this size. We therefore restrict our analysis to embedded and filter selectors.

2.1 Methodology

Our analysis followed a structured pipeline consisting of data preparation, feature selection, model training, and evaluation. After creating a stratified 8:2 train–test split, all models were implemented using `scikit-learn` pipelines to ensure that the required procedures were applied consistently across methods. This structure allowed us to isolate the effect of the selector while keeping the classifier fixed. We keep our core logic for this task as:

$$Selector \rightarrow Classifier$$

We evaluated several feature-selection methods—L1, Random Forest, XGBoost, and Mutual Information—covering both embedded and filter-based approaches, each tested across multiple feature counts. All pipelines were assessed using stratified 5-fold cross-validation on the training data, followed by evaluation on the held-out test set. Balanced accuracy and confusion matrices served as the primary performance metrics, and a unified evaluation framework enabled consistent comparison across models and feature numbers $k \in \{20, 50, 100, 200, 400\}$. We also ran an Optuna-based hyperparameter search to examine potential improvements from jointly tuning selector and classifier parameters; this was conducted separately and did not affect the main modelling pipeline.

2.2 Exploratory Data Analysis (EDA)

The high-dimensional ($n < p$) neural spike dataset, with 11,191×683 entries, contained no missing values, duplicates, negative entries, or non-integer counts. We observed 536 neuron features that never fired (all zeros); these were retained, as silent neurons may simply be inactive during the recorded time window. Because the observations are discrete spike counts, the data are non-Normal. Outlier detection using the IQR rule—flagging values outside $Q_1 - 3IQR$ to $Q_3 + 3IQR$—identified no outliers. To illustrate the distributional characteristics of the spike data, we examine sample histograms and correlation structure:

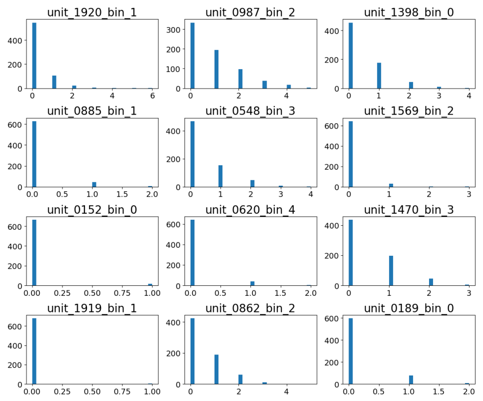


Figure 7: Sample distribution of features

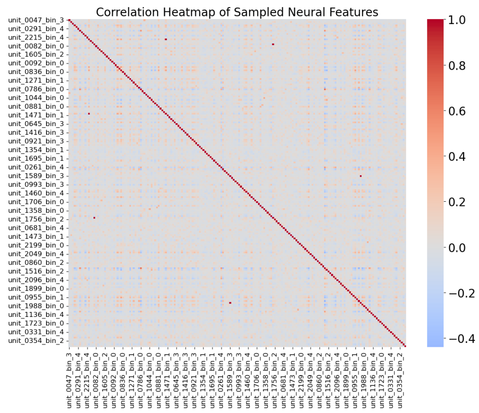


Figure 8: Sample heatmap of features

Figure 7 shows histogram examples from twelve feature channels, which are generally right-skewed and contain many zeros—approximately 74.03% of all entries by computation. The 200-sample correlation heatmap (Figure 8) also shows weak correlation structure, with mostly grey tones and only a few isolated coloured blocks; strong correlations appear only along the diagonal, reflecting individual feature variances. These observations suggest that a substantial proportion of features are likely uninformative. This motivates and justifies the application of feature selection and dimensionality-reduction techniques in the following analysis.

2.3 Model development

During model development, each feature selector is combined with the fixed classifier to form a pipeline, which is evaluated across multiple feature counts. All preprocessing, feature selection, and model fitting steps were encapsulated within pipelines, ensuring no data leakage during cross-validation. This design guarantees that selectors are fitted only on training folds, making performance estimates unbiased and the workflow fully reproducible.

2.3.1 Inputs and evaluation framework

We use the full neural feature matrix as the input X and the binary class labels as the target variable y. To ensure comparability across feature-selection methods, each selector is paired with the same downstream classifier—Logistic Regression with L2 regularisation. This model is chosen because it is fast, stable, and well-suited to high-dimensional data, allowing differences in performance to be attributed directly to the feature selector rather than to the choice of classifier. For every method, we evaluate multiple feature-set sizes by varying the number of selected features k.

Model performance is assessed using 5-fold stratified cross-validation on the training set, ensuring equal class balance across folds. For each selector–classifier pipeline, we compute Balanced Accuracy as the primary metric, as it accounts for class imbalance and gives equal weight to both classes. After cross-validation, each pipeline is retrained on the full training split and evaluated once on the untouched test set to produce an unbiased estimate of predictive performance. Confusion matrices further illustrate how each model behaves under different feature-selection configurations.

2.3.2 Embedded Selectors

Embedded selectors perform feature selection as part of the model training process itself. Instead of ranking features independently, these approaches learn feature importance directly from a fitted model—typically through regularisation penalties or tree-based splitting criteria. They are appealing in this task for two reasons. First, they are computationally more feasible than wrapper methods like *Forward/Backward Selection*, which would require repeatedly training thousands of models in a prohibitively ample search space. Second, they leverage the modelling structure to capture nonlinear interactions and hierarchical importance patterns that filter methods cannot detect.

In our implementation, we use three embedded selectors: L1 logistic regression, Random Forest, and XGBoost. Specifically:

L1 Selector

- Penalty: `l1`
- Solver: `liblinear`
- Iterations: `2000`

Random Forest Selector

- Number of trees: `200`
- Minimum samples to split: `2`

XGBoost Selector

- Booster: “`gbtree`”
- Maximum depth: `3`
- Learning rate: 0.3
- L1 regularisation (α): `1`

2.3.3 Filter Selector

Here we introduce another type of selector: *Filter*. Filter Selectors evaluate each feature independently based on its statistical relationship with the target variable, without using a classifier (only in the selection step; a classifier is still needed later to produce classification results). Because they operate directly on the data rather than through model training, filter methods are computationally efficient in high-dimensional settings. Common Filter Selectors are: *ANOVA F-test*, *Chi-square Test*, etc.

In this analysis, we employ Mutual Information (MI) as our filter-based selector. MI measures the dependency between each feature and the class label, capturing both linear and nonlinear associations. In practice, MI is estimated using a k-nearest-neighbours (kNN) density estimator, where the choice of *n_neighbors* affects the smoothness and stability of the scores:

$$I(X;Y) = \mathbb{E}\left[\log \frac{p(X,Y)}{p(X)p(Y)}\right].$$

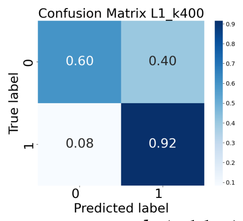
We apply MI through `SelectKBest`, and a pipeline that standardises the data to ensure reproducibility, ranks features by their MI scores, and then trains the common Logistic Regression classifier on the top k selected features. We also set $n\_neighbors = 10$ (instead of the default 3) to stabilise k-NN density estimation and improve performance. A larger neighbourhood reduces variance in the MI scores, leading to more reliable feature rankings across runs.

2.4 Model performance

Overall, the four selectors show distinct behaviours as the number of selected features increases:

L1 Selector

model	bal. acc.	#feat
L1_k20	0.5825	20
L1_k50	0.6843	50
L1_k100	0.6605	100
L1_k200	0.7345	200
L1_k400	0.7555	400



RF Selector

model	bal. acc.	#feat
RF_k20	0.6060	20
RF_k50	0.6672	50
RF_k100	0.6525	100
RF_k200	0.7187	200
RF_k400	0.6697	400

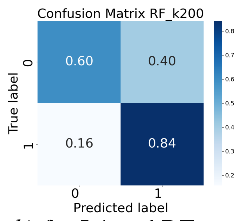
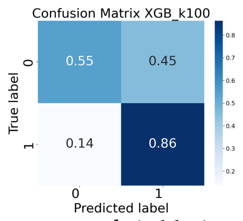


Figure 9: Balanced accuracy over k (tables) and confusion matrices (best k) for L1 and RF.

XGB Selector

model	bal. acc.	#feat
XGB_k20	0.5440	20
XGB_k50	0.6445	50
XGB_k100	0.7054	100
XGB_k200	0.6749	200
XGB_k400	0.6484	334



MI Selector

model	bal. acc.	#feat
MI_k20	0.5941	20
MI_k50	0.6445	50
MI_k100	0.6539	100
MI_k200	0.6647	200
MI_k400	0.6276	400

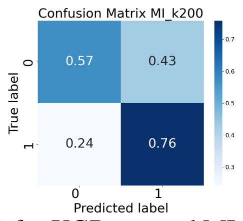


Figure 10: Balanced accuracy over k (tables) and confusion matrices (best k) for XGBoost and MI.

The L1 selector performs best, reaching a balanced accuracy of 0.7555 at $k = 400$, and its confusion matrix shows clear separation with 92% recall for class 1 and low false-negative rates. Random Forest also performs well, particularly at $k = 200$, with its confusion matrix showing strong recall for class 1 (84%). XGBoost exhibits its strongest performance at $k = 100$, and its confusion matrix similarly reflects high class 1 recall (86%). At the same time, Mutual Information remains the weakest method across all k, achieving a maximum accuracy of 0.665 and only moderate recall in its confusion matrix. Overall, embedded methods — particularly L1 — are more effective on this dataset, whereas simple filter-based selection struggles to capture the informative structure.

It is also noteworthy that most selectors exhibit diminishing effects as the feature count increases, such as MI's score dropping from 0.6647 at $k = 200$ to 0.6276 at $k = 400$. The following chart better illustrates this trend:

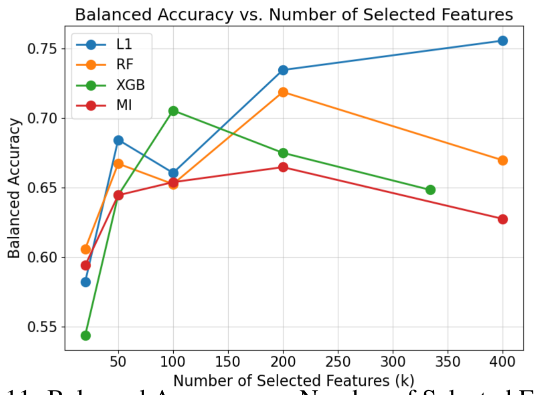


Figure 11: Balanced Accuracy vs. Number of Selected Features

We observe a clear diminishing pattern: accuracy improves rapidly when increasing k from 20 to around 100–200, but beyond this point, additional features offer little benefit and can even reduce performance. This suggests that the most informative structure is captured early, whereas larger feature sets introduce noise rather than improving discrimination.

2.5 Optuna intervention

Our main modelling framework varies the number of selected features ($k \in \{20, 50, 100, 200, 400\}$) while keeping selector hyperparameters at their defaults. This provides a clear and consistent baseline, but it may not fully capture interactions between selector settings and the classifier. As a side investigation, we also ran a small `optuna` search (30 trials per model) using TPE to see whether joint hyperparameter tuning could offer further improvements.

Metric	**L1 Selector + Logistic**	**RF Selector + Logistic**	**XGB Selector + Logistic**	**MI Selector + Logistic**
Balanced Accuracy	0.7436	0.7159	0.7079	0.6367
# Selected Features	400	100	400	400

Best Hyperparameters	solver = `liblinear`
	$C_{\text{selector}} = 1.1073$
		n_estimators = 200
		max_depth = 8
		max_features = `sqrt`
			max_depth = 4
			$\eta = 0.0439$
			n_estimators = 150
			subsample = 0.8452
			colsample_bytree = 0.6461
			$\alpha = 0.0958$, $\lambda = 2.9784$
				mi_n_neighbors = 7

Table 3: Optuna-tuned models.

The Optuna search offered only limited improvement over the baseline results. L1 (0.7436) again emerged as the strongest selector, reinforcing the data's sparse yet informative structure. RF (0.7159) remained the next-best method, favouring smaller k, whereas MI (0.6367) remained the poorest model due to its sensitivity to noise. XGBoost (0.7079) was the only model to benefit, with a slight increase of about 0.25%. Overall, the tuned models closely reflected the previously observed patterns.

2.6 Summary: Task 2

Task 2 examined how different feature selection methods perform on a high-dimensional ($n < p$) neural dataset. Across all selectors, accuracy improved quickly from $k = 20$ to about $k = 100$, after which several methods plateaued or declined, showing diminishing returns. L1 was the strongest method, reaching a balanced accuracy of 0.7555 at $k = 400$, consistent with a sparse but informative signal structure. Random Forest performed well, peaking at 0.7187 for $k = 200$, whereas XGBoost achieved its best result at $k = 100$ (0.7054) and then declined as more features were added, indicating sensitivity to redundancy. Mutual Information was consistently weaker, with a maximum of 0.6647, reflecting the limitations of a univariate filter in a correlated setting.

As a potential improvement, we used `optuna` to tune the selector hyperparameters jointly. The tuned models showed only minor changes and retained the same ranking as the baseline pipelines, suggesting that the original workflow was already close to optimal and that most of the proper structure was captured by varying k alone. Overall, embedded selectors—particularly L1—proved the most effective, while increasing the number of features beyond a moderate threshold added little additional value.

A Appendix 1: Acknowledgement: The use of generative AI tools

We acknowledge the use of GPT-5 (OpenAI, `https://openai.com/gpt-5`) to generate, debug, and assess portions of the Python code in this investigation, particularly during feature selector construction and `optuna` implementation.