Machine Learning Project

Ice Classification in Greenland

Authors: Chaker Meraihi & Shakil Rohimun
Degree Program: Master's in Quantitative Finance (M2QF)
Instructor: Prof. Mathilde Mougeot
Date: November 2024

Motivation

The goal of this study is to evaluate the performance of various machine learning models in building a predictive classifier to estimate the low or high quantity of ice at specific locations in Greenland, using infrasound recordings. This research aims to enhance our understanding of ice dynamics and contribute to more effective predictive methodologies.

Context

In this application, there are 4 potential target outputs corresponding to infrasound signals (Y1, Y3, Y4, Y5). The covariables are defined by 9 variables provided by the European Centre for Medium-Range Weather Forecasts (ECMWF).

The displacement of large volumes of air leads to low-frequency acoustic waves in the atmosphere. This so-called infrasound is inaudible to humans as it has frequencies lower than 20 Hz. In the raw data, the 4 output variables are quantitative infrasound records. For this classification study, the quantitative valued target signals will be transformed into binary information using an appropriate given threshold.

Exploratory Data Analysis

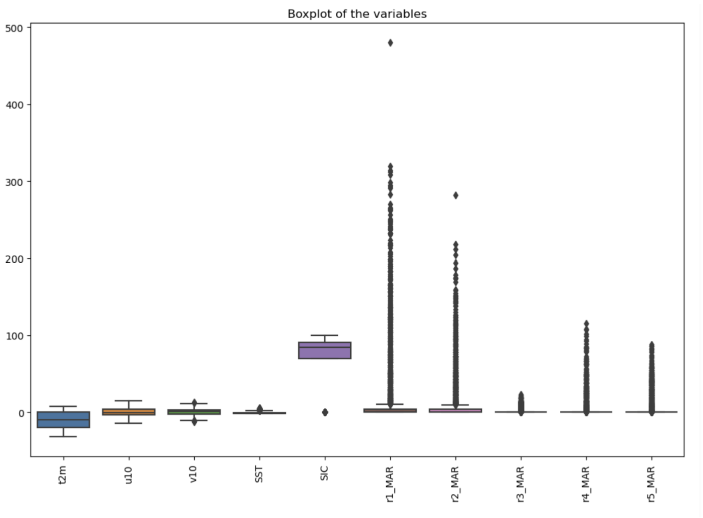

Figure 1.1: Boxplot of the variables.

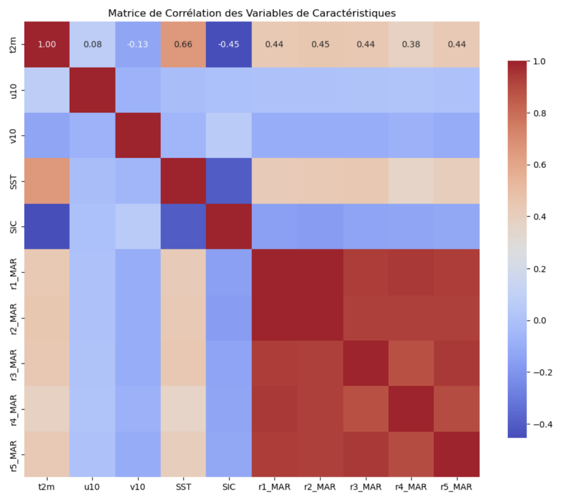

Figure 1.2: Correlation matrix of the variables.

We see a high correlation for the variables MAR (r1, r2, r3, r4, r5), it gives us the idea to try PCA or Supervised Principal Components (SPC).

Methodology: Models and Their Properties

In this project, we employ a selection of machine learning classification models to evaluate their performance in predicting the target variable. A brief presentation of each model, including its underlying principles and key characteristics, will be provided. This includes linear models such as Logistic Regression, ensemble methods like Random Forest and Gradient Boosting, and advanced algorithms such as XGBoost and LightGBM. Additionally, simpler algorithms like K-Nearest Neighbors and Naive Bayes are included for comparison. By combining diverse approaches, we aim to ensure a comprehensive evaluation and identify the most suitable model for the problem at hand. Below, we present the models selected for this study.

Logistic Regression

Logistic regression models the posterior probabilities of the classes while ensuring they sum to one and remain in $[0, 1]$. For K classes, the probabilities are:

$$\Pr(G = k|X = x) = \frac{\exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^T x)}, \quad k = 1, \dots, K-1, \tag{1.1}$$

$$\Pr(G = K|X = x) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\beta_{\ell 0} + \beta_\ell^T x)}. \tag{1.2}$$

For binary classification ($K = 2$), the log-likelihood for N observations is:

$$\ell(\beta) = \sum_{i=1}^{N} \left[y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta)) \right], \tag{1.3}$$

where $p(x_i;\beta)$ is the predicted probability for class 1. To maximize $\ell(\beta)$, the Newton-Raphson algorithm is used, leading to the iteratively reweighted least squares (IRLS) method.

Naive Bayes Classifier

The Naive Bayes classifier assumes that given a class $G = j$, the features X_k are independent. The class-conditional density is given by:

$$f_j(X) = \prod_{k=1}^{p} f_{jk}(X_k), \tag{1.4}$$

where $f_{jk}(X_k)$ represents the marginal density of feature X_k in class j. This simplifies density estimation, allowing the marginal densities to be estimated independently, often using kernel density estimates for continuous variables or histograms for discrete variables.
Despite its simplicity and the independence assumption, Naive Bayes often performs competitively with more sophisticated methods, especially in high-dimensional settings where variance reduction outweighs bias.

Nearest-Neighbor Methods

Nearest-neighbor methods use observations in the training set closest in input space to a given point x to estimate $\hat{Y}$. Specifically, the k-nearest neighbor estimate is:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{1.5}$$

where $N_k(x)$ denotes the k closest points to x in the training data, typically measured using Euclidean distance. For classification, the predicted class is determined by a majority vote among the k neighbors. For $k = 1$, the method corresponds to assigning the class of the single nearest neighbor.
The decision boundaries created by nearest-neighbor methods are highly flexible, adapting to local data clusters. As k increases, the predictions become smoother but may lose sensitivity to small-scale patterns in the data.

Support Vector Classifier

The support vector classifier (SVC) constructs an optimal separating hyperplane to maximize the margin between two classes. For perfectly separable data, it solves the optimization problem:

$$\min_{\beta,\beta_0} \|\beta\| \quad \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 \text{ for all } i. \tag{1.6}$$

For non-separable data, slack variables $\xi_i \geq 0$ are introduced to allow some points to be on the wrong side of the margin. The optimization problem becomes:

$$\min_{\beta,\beta_0,\xi} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i, \quad \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \text{ for all } i, \tag{1.7}$$

where C controls the trade-off between margin width and classification error.
The solution depends only on a subset of training points, called support vectors, which lie on or violate the margin. The decision function is:

$$\hat{G}(x) = \operatorname{sign}(x^T\hat{\beta} + \hat{\beta}_0). \tag{1.8}$$

SVC is a powerful method for classification and can handle overlapping classes by adjusting the cost parameter C.

Random Forests

Random forests are an ensemble method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. The algorithm works as follows:

1. For $b = 1$ to B (number of trees):
    (a) Draw a bootstrap sample of size N from the training data.
    (b) Grow a decision tree T_b on the bootstrapped data by:
        i. Randomly selecting m features out of p at each split.
        ii. Choosing the best split among the m features.
        iii. Recursively splitting nodes until a stopping criterion (e.g., minimum node size) is met.
2. Combine the predictions of all trees:
    - For regression: $\hat{f}_B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
    - For classification: Use majority voting among the B trees.

Random forests reduce overfitting by decorrelating trees through feature subsampling and achieve high performance with minimal tuning. The parameter m controls the number of features considered at each split, typically set to $\sqrt{p}$ for classification or $p/3$ for regression.

AdaBoost

AdaBoost (Adaptive Boosting) is an ensemble method that builds a strong classifier by combining weak classifiers iteratively. At each iteration, AdaBoost adjusts the weights of the training samples to focus on those that were misclassified by the previous classifier. The algorithm works as follows:

1. Initialize weights $w_i = \frac{1}{N}$ for all $i = 1, \dots, N$.
2. For $m = 1$ to M (number of classifiers):
    (a) Train a weak classifier $G_m(x)$ using the weighted dataset.
    (b) Compute the weighted error:
    $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}. \tag{1.9}$$
    (c) Compute the classifier weight:
    $$\beta_m = \frac{1}{2}\log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right). \tag{1.10}$$
    (d) Update the weights:
    $$w_i \leftarrow w_i \exp(-\beta_m y_i G_m(x_i)), \tag{1.11}$$
    and normalize so that $\sum_{i=1}^{N} w_i = 1$.
3. Combine the classifiers:
$$F(x) = \text{sign}\left(\sum_{m=1}^{M} \beta_m G_m(x)\right). \tag{1.12}$$

AdaBoost minimizes the exponential loss function:

$$L(y, F(x)) = \sum_{i=1}^{N} \exp(-y_i F(x_i)). \tag{1.13}$$

It is effective at reducing both bias and variance, and works well with a variety of weak classifiers, such as decision stumps.

Gradient Boosting

Gradient Boosting is a powerful ensemble learning method that builds models iteratively by optimizing a loss function using decision trees as weak learners. The algorithm works as follows:

1. Initialize the model with a constant prediction:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma), \tag{1.14}$$

   where L is the loss function.

2. For $m = 1$ to M (number of iterations):

   (a) Compute the negative gradient (pseudo-residuals):

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\bigg|_{f(x)=f_{m-1}(x)}. \tag{1.15}$$

   (b) Fit a regression tree $T_m(x)$ to the pseudo-residuals r_{im}.

   (c) Compute the optimal step size for each terminal region R_{jm}:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma). \tag{1.16}$$

   (d) Update the model:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}). \tag{1.17}$$

3. Output the final model:

$$\hat{f}(x) = f_M(x). \tag{1.18}$$

Gradient Boosting can handle different loss functions, such as squared error for regression or deviance for classification, making it highly flexible.

LightGBM and XGBoost

LightGBM and XGBoost are advanced gradient boosting frameworks that improve upon traditional Gradient Boosting with speed and accuracy enhancements:
LightGBM uses a histogram-based algorithm to discretize continuous features into bins, which reduces memory usage and speeds up training. The objective for a tree in LightGBM is to minimize:

$$\text{Objective} = \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|T\|, \tag{1.19}$$

where $\|T\|$ is the complexity of the tree and λ is the regularization parameter.

XGBoost introduces regularization terms to prevent overfitting and uses second-order derivatives to optimize the loss function. The objective function for XGBoost is:

$$\text{Objective} = \sum_{i=1}^{N} L(y_i, f(x_i)) + \sum_{k=1}^{K} \Omega(T_k), \tag{1.20}$$

where $\Omega(T_k) = \gamma T_k + \frac{1}{2}\lambda \|w\|^2$, T_k is the tree complexity, and λ and γ are regularization parameters.
Both frameworks support parallel computation and advanced features such as handling missing data and feature importance ranking.

Principal Components and Variants

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) provides a sequence of best linear approximations to data in $\mathbb{R}^p$. Consider observations $x_1, x_2, \ldots, x_N$ and represent them with a rank-q linear model:

$$f(\lambda) = \mu + V_q \lambda, \tag{1.21}$$

where μ is a location vector, V_q is a $p \times q$ matrix with orthogonal unit vectors as columns, and λ is a q-dimensional parameter vector. Fitting such a model minimizes the reconstruction error:

$$\min_{\mu, \{\lambda_i\}, V_q} \sum_{i=1}^{N} \|x_i - \mu - V_q \lambda_i\|^2. \tag{1.22}$$

The optimal μ and λ_i are:

$$\hat{\mu} = \bar{x}, \quad \hat{\lambda}_i = V_q^T (x_i - \bar{x}). \tag{1.23}$$

The matrix V_q consists of the top q eigenvectors of the covariance matrix of the centered data X. PCA identifies directions of maximum variance, where the principal components Z are projections of the data onto these directions.

Supervised Principal Components (SPC)

Supervised Principal Components (SPC) modifies PCA by selecting features correlated with the outcome variable before applying PCA. This enhances predictive power, especially when $p \gg N$. The procedure is as follows:

1. Compute univariate regression coefficients for each feature.
2. Select features with coefficients above a threshold θ.
3. Apply PCA to the selected features and use the first few principal components in the predictive model.

This supervision ensures that the principal components are aligned with the outcome, making them more effective for prediction tasks.

Model Performance Comparison

Threshold Selection Rationale

The thresholds used in this project, namely the *mean*, *mean + standard deviation (std)*, and the *90th percentile*, are selected to capture different aspects of ice concentration variability. The *mean* provides a balanced threshold to distinguish lower and higher concentrations based on central tendency, offering an interpretable and straightforward division. The *mean + std* highlights concentrations that significantly exceed average levels, capturing moderately high events that might indicate unusual conditions. Finally, the *90th percentile* focuses on extreme values, representing the top 10% of concentrations, which are crucial for identifying rare but impactful high-concentration scenarios. These thresholds allow for robust experimentation and ensure the analysis accounts for typical, moderately high, and extreme variations in ice concentration data.

The following section presents the results obtained using these thresholds, demonstrating their impact on the classification and predictive performance of the models. For the sake of brevity in this report, we only include the numerical results corresponding to the best threshold, which is the mean. However, we provide the ROC curves for all thresholds (mean, mean+std, and 90% quantile) to allow for a comprehensive visual comparison of the model performances across the different thresholds. The following section presents the numerical results for the best-performing threshold, alongside ROC curves for all tested thresholds to compare model performances visually.

Performance Analysis for Mean Threshold

Model	**Data Type**	**Accuracy**	**F1 Score**	**Recall**	**Precision**
Logistic Regression	Standard	0.963494	0.714286	0.625000	0.833333
K-Nearest Neighbors	Standard	0.970013	0.792793	0.785714	0.800000
Naive Bayes	Standard	0.954368	0.744526	0.910714	0.629630
SVM	Standard	0.968709	0.769231	0.714286	0.833333

Model	Data Type	Accuracy	F1 Score	Recall	Precision
Random Forest	Standard	0.971317	0.803571	0.803571	0.803571
Extra Trees	Standard	0.973924	0.824561	0.839286	0.810345
AdaBoost	Standard	0.971317	0.803571	0.803571	0.803571
Gradient Boosting	Standard	0.960887	0.736842	0.750000	0.724138
XGBoost	Standard	0.967405	0.770642	0.750000	0.792453
LightGBM	Standard	0.971317	0.803571	0.803571	0.803571
Logistic Regression	PCA	0.966102	0.740000	0.660714	0.840909
K-Nearest Neighbors	PCA	0.964798	0.761062	0.767857	0.754386
Naive Bayes	PCA	0.967405	0.793388	0.857143	0.738462
SVM	PCA	0.964798	0.752294	0.732143	0.773585
Random Forest	PCA	0.966102	0.771930	0.785714	0.758621
Extra Trees	PCA	0.967405	0.774775	0.767857	0.781818
AdaBoost	PCA	0.962190	0.728972	0.696429	0.764706
Gradient Boosting	PCA	0.956975	0.713043	0.732143	0.694915
XGBoost	PCA	0.967405	0.766355	0.732143	0.803922
LightGBM	PCA	0.962190	0.738739	0.732143	0.745455
Logistic Regression	SPC	0.966102	0.740000	0.660714	0.840909
K-Nearest Neighbors	SPC	0.964798	0.761062	0.767857	0.754386
Naive Bayes	SPC	0.967405	0.793388	0.857143	0.738462
SVM	SPC	0.964798	0.752294	0.732143	0.773585
Random Forest	SPC	0.966102	0.771930	0.785714	0.758621
Extra Trees	SPC	0.970013	0.788991	0.767857	0.811321
AdaBoost	SPC	0.962190	0.728972	0.696429	0.764706
Gradient Boosting	SPC	0.955671	0.706897	0.732143	0.683333
XGBoost	SPC	0.967405	0.766355	0.732143	0.803922
LightGBM	SPC	0.962190	0.738739	0.732143	0.745455

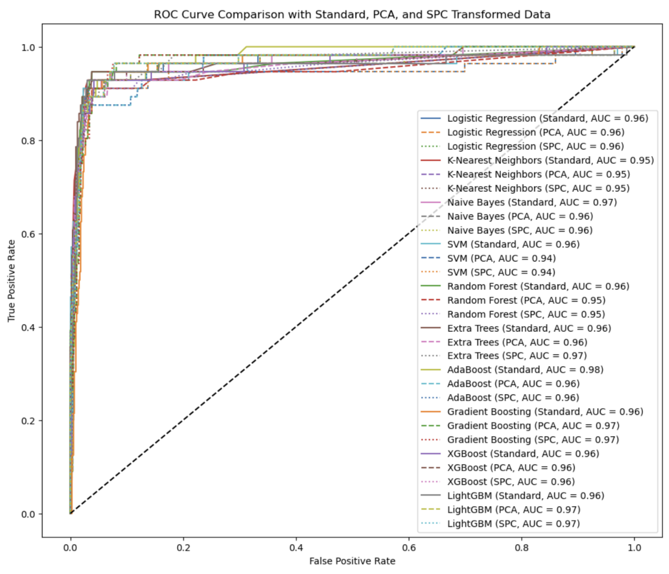

Figure 1.3: ROC Curve for mean threshold

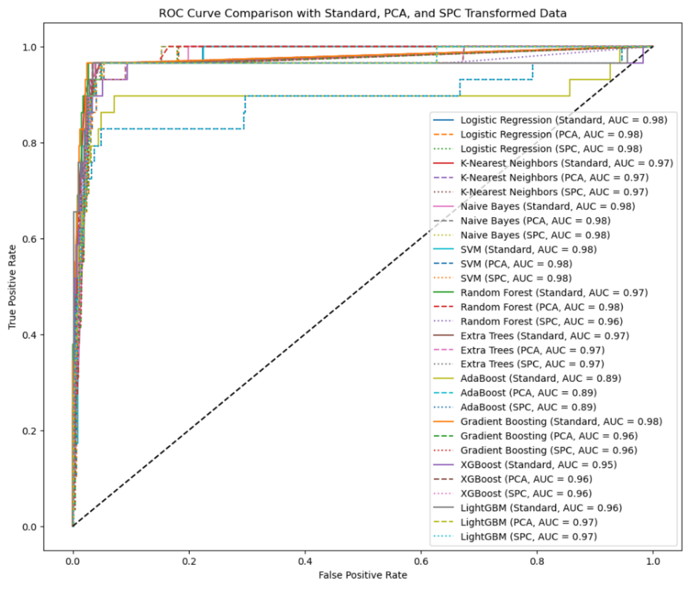

Figure 1.4: ROC Curve for mean+std threshold

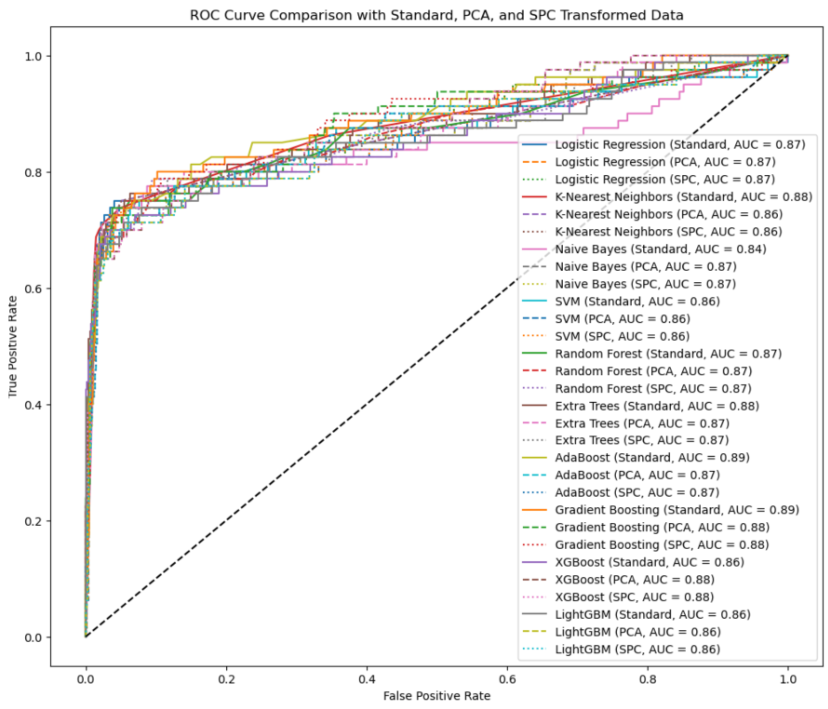

Figure 1.5: ROC Curve for 90% quantile threshold

Conclusion

In this analysis, several machine learning models were evaluated using three distinct data transformation techniques: Standard, PCA, and SPC. Additionally, the models were tested with three thresholds (**mean**, **mean + standard deviation**, and **90% quantile**) to assess their impact on classification and predictive performance. The main findings are as follows:

- Models such as **Random Forest**, **Extra Trees**, **AdaBoost**, and **LightGBM** consistently achieved high accuracy and F1 scores across all transformations and thresholds. Notably, **Extra Trees** (Standard, mean threshold) achieved the highest F1 score of 0.824.
- **Naive Bayes** demonstrated superior recall, particularly with higher thresholds, but at the cost of reduced precision, highlighting its effectiveness in identifying positive cases while producing more false positives.
- The choice of threshold significantly influenced model performance. Using the **mean threshold** provided a balanced approach, optimizing both precision and recall. The **mean + standard deviation** threshold was effective for identifying edge cases, improving precision but slightly lowering recall. The **90% quantile** threshold emphasized highly selective predictions, resulting in increased false negatives.
- PCA and SPC transformations resulted in minor variations in performance. PCA typically displayed marginally lower scores compared to Standard data, while SPC maintained performance close to PCA, suggesting that both techniques are useful for dimensionality reduction but may lead to minor trade-offs in predictive power.
- The ROC Curves (Figures 1.3, 1.4, 1.5) illustrate the impact of the thresholds on model performance. Models like **AdaBoost**, **LightGBM**, and **Extra Trees** achieved consistently high AUC scores across thresholds, underscoring their robustness and adaptability.

Overall, the findings suggest that **Extra Trees**, **AdaBoost**, and **LightGBM** are highly reliable for applications prioritizing both precision and recall. **Naive Bayes** is particularly useful in scenarios where recall is critical, such as detecting rare or critical events. The choice of threshold should align with the specific objectives of the application, with the **mean threshold** recommended for general use, while the **mean + standard deviation** and **90% quantile** thresholds are more suited to specialized tasks requiring high precision or selective predictions. These findings underscore the importance of carefully selecting thresholds and machine learning models tailored to specific objectives, paving the way for more precise ice classification methodologies in Greenland.

References

- Professor Mougeot, *Lecture Notes in Machine Learning.*
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd Edition, 2017.
- Kevin P. Murphy, *Probabilistic Machine Learning: Advanced Topics* (Draft).