Analysis of Probabilistic Models in Machine Learning

Introduction

The goal of this project is to analyze a dataset provided in CSV format using machine learning techniques, specifically probabilistic techniques and embedded methods.

The probabilistic techniques to be used are: Naive Bayes and Discriminant Analysis. The embedded techniques to be used are: Bagging and Boosting.

The dataset contains a total of 91 variables and 3000 observations. The objective is to predict the target variable, which is a categorical variable with three possible values: 0, 1, and 2. All data is numeric. The variable names are random, and the meaning of each one is unknown.

Method

First, data preprocessing is carried out, as mentioned in the introduction, the dataset contains 91 variables. For data preprocessing, feature selection methods will be used.

Next, the following machine learning techniques are applied:

- Gaussian Naive Bayes
- Augmented Naive Bayes
- Linear and Quadratic Discriminant analysis

- Bagging
- Random Forest
- Ada Boost
- Gradient Boosting

Feature Selection

The feature selection methods we will use can be classified as follows:

- Filtering methods
    - Basic techniques: detection of constant, quasi-constant, and duplicate variables.
    - Statistical techniques: ANOVA and mutual information.
- Wrapper methods: Sequential Feature Selection (heuristic).
- Embedded methods: Lasso.

Filtering methods: statistical techniques.

For correlation between variables, we explore the number of variables that are correlated with each other and the strength of their correlation. If two variables are perfectly correlated, one is redundant and can be removed.

For ANOVA and mutual information, we use the SelectPercentile function from sklearn.feature_selection to select the features with the highest percentile values in the statistical tests. This function is useful when dealing with a large number of features and wanting to retain only a certain fraction of them. We use this method to select the features that fall within a specified percentile of the highest scores. In other words, we will select the top 90% of the original features with the highest mutual information or ANOVA scores with the target variable. The percentile to choose will be experimentally determined.

ANOVA is a statistical technique used to determine if there are significant differences between the means of two or more groups. It is used to compare the variance between different groups of features with the variance within each group. If the variance between groups is significantly higher than the variance within the groups, then it can be concluded that the feature has a significant effect on the target variable and is therefore an important feature.

Mutual information is the additional information about a variable provided by another variable.

Models

The Naive Bayes and Discriminant Analysis models are fitted using the different training sets that result from applying feature selection. This is because these models have very few parameters that can be modified, so to improve their performance, the practices are limited to preprocessing. The most important aspect is to apply feature selection methods that eliminate potential relationships. Additionally, cross-validation has been applied..

Naive Bayes

Within probabilistic techniques, we will use Naive Bayes, specifically the Gaussian Naive Bayes algorithm, as our variables are quantitative.

Several models have been tested:

- The initial models were tested without adjusting parameters, focusing only on identifying which training set with feature selection fits best. The dataset with the highest accuracy among these will be used in subsequent tests.
- Second model: adjust the prior probabilities of the classes and the variance by selecting the best hyperparameter using Sklearn's GridSearchCV function.
- Third model: calibrate the model using the isotonic and sigmoid methods.

The model is tested with the different datasets as before. However, in this case, with more parameters to choose from, the best options will be explored.

Augmented Naive Bayes

Augmented Naive Bayes (ANB), also known as Tree Augmented Naive Bayes (TAN), is an improvement on the Naive Bayes algorithm. While Naive Bayes assumes that all features are independent given the class, ANB allows for limited dependency between features.

The hyperparameters for which we will explore the best values are:

- `methods`: refers to the different network structure learning methods that can be used.
- `priors`: this parameter refers to the different types of priors that can be used for parameter learning.

- `discretizationStrategy`: the different strategies that can be used to discretize continuous variables.

The result is a model that can capture some dependencies between features, which can improve classification accuracy compared to Naive Bayes, especially when there are strong dependencies among the features.

Discriminant Analysis

Similar to Naive Bayes, there are very few hyperparameters to adjust. We have the choice of using either Quadratic Discriminant Analysis (QDA) or Linear Discriminant Analysis (LDA). The linear method imposes the constraint that there is a single covariance matrix for all variables, unlike the quadratic method, where there is a separate covariance matrix for each class. This single covariance matrix is estimated from the different covariance matrices. The quadratic method allows for classifying non-linear models, but sometimes it is not feasible because the number of parameters to estimate exceeds the number of observations. Therefore, the linear method is used in such cases. We will perform both Quadratic and Linear Discriminant Analysis and compare their results.

Bagging

Bagging is a method where the results of several base models are averaged. The idea is that combining multiple models creates a final model that is more powerful than any individual one. In Bagging, feature selection has little impact on the method's performance; in fact, it may benefit from having the complete dataset to detect complex patterns. For this reason, only tests with the original dataset are performed.

The models we will adjust in Bagging are:

- Standard bagging.
- Ramdom Forest.

Bagging has a set of parameters to modify in order to see how the model improves:

- The base estimator can be any model, but we will try with:
    - KNN. When choosing this model, the number of neighbors to be selected must be considered.
    - Decisio trees.

- Number of estimators: the number of decision trees or the number of different KNN models.

- Number of variables to be taken from the training set to train each base model, a parameter called `max_features`

- Estimate the generalization error with OOB (Out-Of-Bag). It is a measure to assess how well the model generalizes for those observations that have been systematically left out, not intentionally but due to the sampling process.

The OOB error is a measure that can help estimate how much my model fails to generalize. Since OOB takes those observations that have never appeared in the Bootstrap samples and trains with them, the error made on these observations forms the OOB score. We will use this both in this section and in Random Forest.

Random Forest is a type of bagging that uses trees. The parameters to choose in this case are fairly well predefined in the `RandomForestClassifier()` function of sklearn. We will adjust the `max_features` parameter, with the recommended value being the integer part of the square root of the number of variables for classification. `max_features` indicates the number of variables to be taken at each node level to decide the node split. If `max_features` is set to None, we would have a standard Bagging model.

Boosting

In Boosting, we will use AdaBoost. AdaBoost is a boosting algorithm that combines many weak classifiers to create a strong classifier. It assigns weights to the instances in the training set and adjusts them in each iteration to focus on the instances that have been misclassified. The final prediction is the weighted sum of the predictions from the weak classifiers. Therefore, in boosting, unlike bagging, we take a weak classifier and improve it over successive iterations.

For Boosting, we apply the AdaBoost and Gradient Boosting algorithms.

In both models, we need to select the loss function to optimize. We choose exponential in AdaBoost and Log Loss in Gradient Boosting to avoid heavily penalizing outliers. Also, for both models, shallow trees are preferred, so the `max_depth` parameter, common to both, will be explored within a small range of values (1-10).

We will also apply Gradient Boosting. The hyperparameters we are going to explore are as follows:

- Learning rate for gradient descent.
- Number of stages to perform (`n_estimators`).

Results

Feature Selection

The following table shows the results of applying feature selection:

Classification	Technique	Deleted Variables
Filtering methods	Constant features	0
	Quasi-constant features	15
	Duplicate features	0
	Features with mutual information	10
	ANOVA	10
Wrapper methods	SFS Naive Bayes	46
	SFS LDA	46
	SFS QDA	46
Embedded methods	Lasso	50

Table 1. Number of variables removed by each method using different feature selection techniques.

As mentioned earlier, for ANOVA and features with mutual information, we used the SelectPercentile function from sklearn.feature_selection to select the features with the highest percentile values in statistical tests. The percentile chosen for both tests was 90.

Naive Bayes

Figure 1 shows the results of an initial model without parameter tuning, except for the choice of the training set. The dataset that appears to work best is SFS, so it is used in the subsequent models.

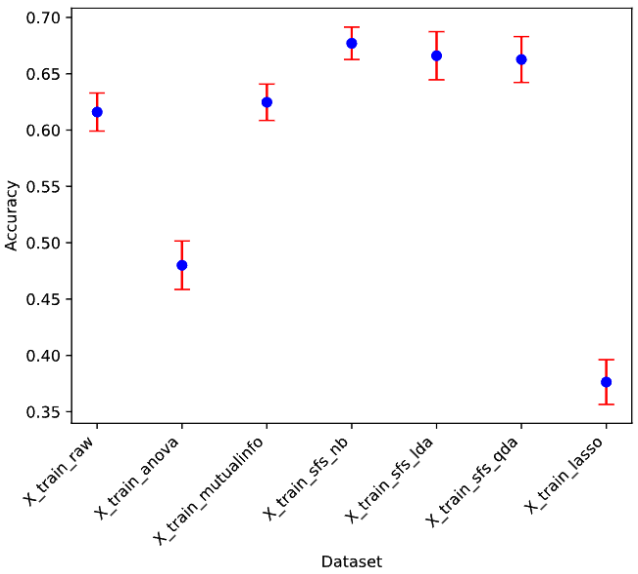

Figure 1. *Accuracy obtained in the Naive Bayes models with default parameters.*

By adjusting the priors and var_smoothing parameters, we find that the best settings are those used by the default model. Applying the two calibration methods mentioned, we obtain better results with sigmoid and manage to improve the original model.

Augmented Naive Bayes

We present the possible values and results of the hyperparameters mentioned in the Method section that we said we would use:

`methods`	`piors`	`discretizationStrategy`
Chow-Liu: 0.6745	Smoothing: 0.7385	quantile: 0.7385
NaiveBayes: 0.5890	BDeu: 0.7385	uniform: 0.704
TAN: 0.5925	NoPrior: 0.7385	kmeans: 0.656
MIIC: 0.7385		
GHC: 0.7385		
Tabu: 0.7385		

Table 2. *Accuracy results after testing with different parameter values.*

The best parameters for the model were: MIIC, Smoothing, and quantile-based discretization strategy.

Discriminant Analysis

In Figure 4, we see the combined results of applying Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA). QDA fits better than LDA, which is expected since QDA can adapt to non-linear data, which is likely the case with our dataset.

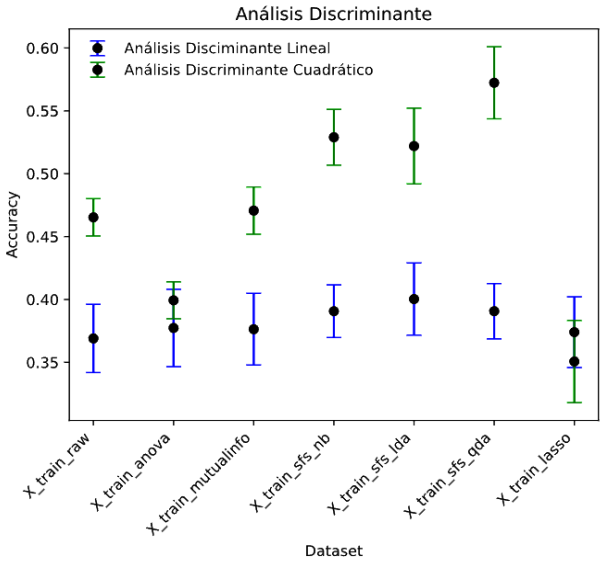

Figure 2. *Accuracy results for the different datasets in Linear Discriminant Analysis (green lines) and Quadratic Discriminant Analysis (blue lines). The lines represent the variance.*

Bagging

We have tried, in addition to tuning with decision trees, to do so with KNN, but the results were very poor. In the case of decision trees, first, the Bagging model with default parameters achieved an accuracy of 0.982 on the training set and 0.827 on the test set. As for the number of estimators in relation to their OOB score, the best number is 450 (Figure 3).

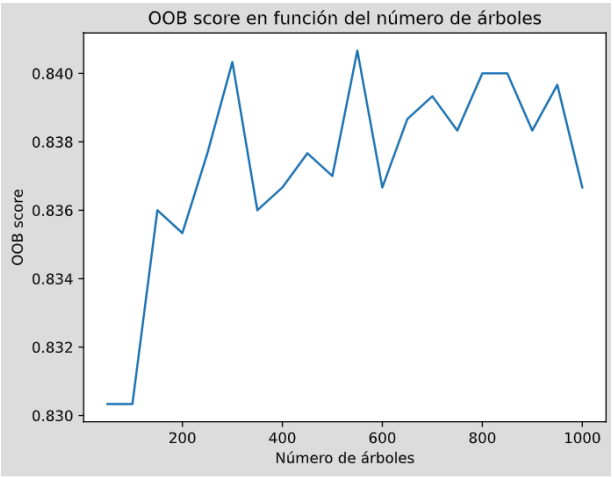

Figure 3. *OOB score when using different numbers of estimators with Bagging.*

Random Forest

We present the possible values we tested for the hyperparameters we were going to tune in Random Forest:

- `max_features` = sqrt, log2 y None
- `criterion` = gini, entropy
- `n_estimators` = Range of values from 50 to 500.

The best results we obtained were: `max_features = None`, `criterion = entropy`, and `n_estimators = 300. However, the differences between using 200 or 500 estimators are relatively small, on the order of hundredths (see Figure 4).

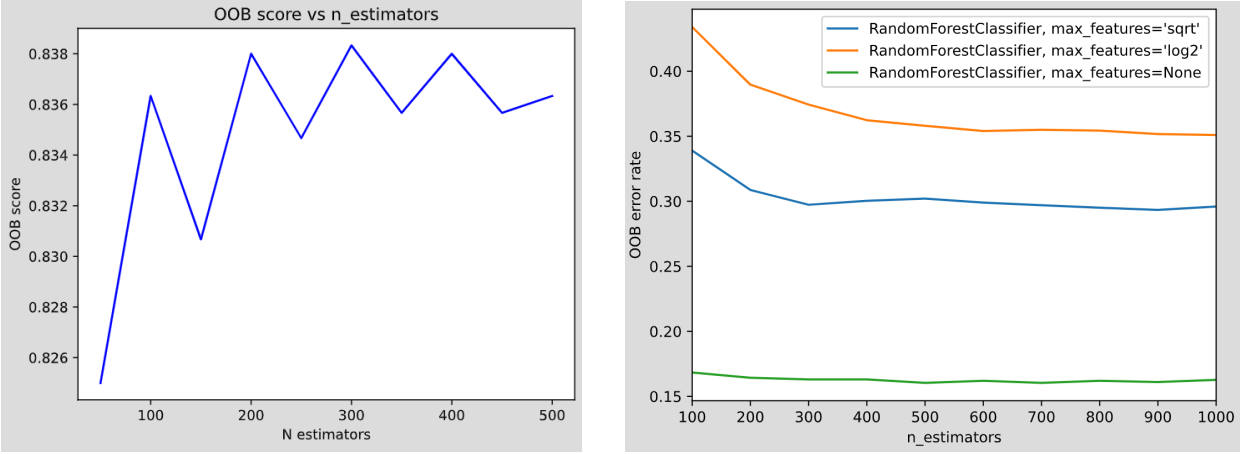

Figure 4. *Left: OOB score when using different numbers of estimators with Random Forest. Right: OOB error when using different numbers of estimators and different types of `max_features`.*

Boosting

As an initial model, we used the decision tree stump, which has a depth equal to the number of classes we want to predict. In this case, we want to predict 3 classes, so 2 levels are sufficient, as 2 levels allow us to classify up to 4 classes. However, with this depth, the results are very poor, and the model barely learns. Therefore, we tried using greater max_depth values.

AdaBoost

In this model, we defined a small learning rate (0.05) and adjusted the model from there. Using the decision tree stump, we tested which algorithm worked best, obtaining the following results:

	SAMME	***SAMME.R***
Train accuracy	0.775	0.717
Test accuracy	0.7735	0.7135

Table 3. *Accuracy results for the train and test sets when applying different algorithms.*

To improve the model, we introduced more estimators, but this did not lead to any improvement. Therefore, we decided to change the base tree to a maximum depth of 4 and 7, and these are the results we obtained, respectively:

Train accuracy: 0.893, Test accuracy: 0.83

Train accuracy: 0.93, Test accuracy: 0.8495

At the cost of introducing bias, we have managed to significantly improve the test error.

When we analyze the features used by AdaBoost to learn, we notice that two of them are of great importance, as shown in the figure below.

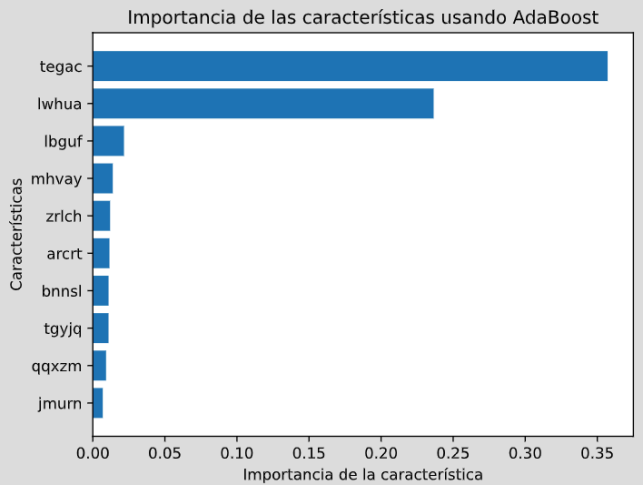

Figure 5. *Feature importance using the AdaBoost algorithm.*

We don't know what they mean, but they seem to be highly representative, and together they account for more than half of the total importance among all features.

GradientBoosting

Unlike AdaBoost, which uses the exponential function to adjust the loss, here we use the logarithmic function, which penalizes large errors less.

In this case, we obtain different features when analyzing it:

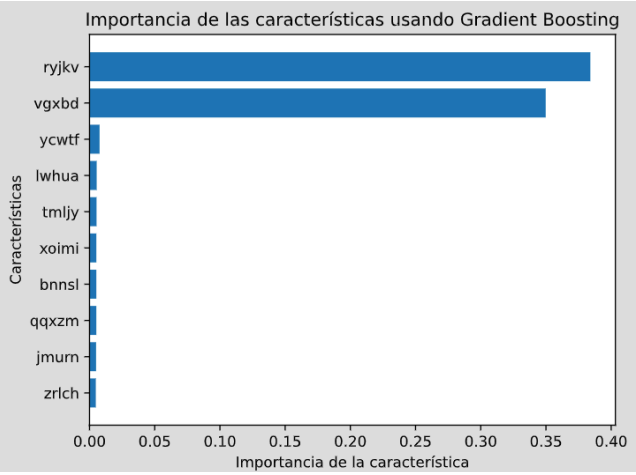

Figure 6. *Feature importance using the Gradient Boosting algorithm.*

Again, there are two features that are very important, accounting for more than 60% of the importance, and these are completely different from the ones selected by AdaBoost.

Results table

Classifier	***Best params***	***Train accuracy***	***Test accuracy***
Gaussian Naive Bayes	Calibration = sigmoid	0.717	0.7
Linear Discriminant Analysis	-	0.435	0.4
Quadratic Discriminant Analysis	-	0.797	0.56
Augmented Naive Bayes	learning_method = MIIC, prior = Smoothing, discretizationStrategy = quantile	0.706	0.738
Bagging	n estimators = 750	1	0.847
Random Forest	max_features = None, n estimators = 300	1	0.844

AdaBoost	algorithm = SAMME max_depth = 7 (decission tree) learning_rate = 0.05	0.933	0.8495
Gradient Boosting	max_depth = 7 learning_rate = 0.05	0.985	0.848

Table 4. *Results of all the models applied.*

Discussion

To adjust the first three models (NB, LDA, and QDA), most of the work involved preprocessing. We saw a significant improvement after performing feature selection compared to using raw data, with the best result achieved through sequential feature selection. On the other hand, in Naive Bayes, the default parameters gave the best results. Finally, calibrating the model helped improve the baseline model.

In the case of Enhanced Naive Bayes, we performed a selection of relevant parameters (as indicated in the table) and adjusted them sequentially. It is worth noting that we could achieve the same accuracy with different parameters, so we arbitrarily chose one of those that gave the same result.

For Bagging and Random Forest, we obtained very similar results. This happens because Random Forest improves as it is trained with a higher number of features, which theoretically should not happen; for classification models, the optimal value for `max_features` is usually `sqrt`. However, in this case, the theory did not hold, and the best model was obtained with `None`, meaning all features were used. For this reason, Random Forest became equivalent to Bagging. In Bagging, since the only parameter we need to adjust is the number of estimators (reducing other parameters leads to worse results), we conducted a comparison to evaluate whether continuing to increase the number of estimators is worth the computational cost. The results indicate that with 750 estimators, we get the best result (though we could keep increasing it indefinitely). However, with 350 estimators, we significantly reduced the computational cost, and the accuracy difference only decreased by about 0.002.

Given what happened with the previous models (Random Forest became equivalent to Bagging), we assumed that all features might be of great importance when training AdaBoost and GradientBoosting. In the case of AdaBoost, we initially used the decision tree stump and found that the SAMME algorithm gave the best results. Afterward, since

increasing the number of estimators did not reduce the error, we decided to modify the decision tree stump. We tested maximum depths of 4 and 7, and the result, as expected, was a reduction in bias. However, while this change caused variance to spike, we achieved a significantly lower test accuracy compared to the decision tree stump, where we achieved an accuracy of 0.77.

In the case of GradientBoosting, we changed the loss function to logarithmic and applied everything we learned from adjusting AdaBoost. We used the same parameters and obtained very similar results. Additionally, we tested modifying certain parameters to reduce variance, such as lowering the learning rate and increasing the number of estimators, as well as other parameters not adjusted in AdaBoost, but the results were worse or similar.

Conclusion

The first conclusion we can draw from this project is that ensemble models are clearly more accurate than probabilistic models. When properly tuned, they achieve much better accuracy, but they take significantly longer to train. We could say that if the goal is to maximize the number of correct predictions, ensemble models would be the best choice. However, if we prioritize training speed at the cost of a slightly worse prediction, we would use a probabilistic model, such as Gaussian Naive Bayes in this case.

On the other hand, focusing on the results of this project, we see that all the variables in the dataset are highly important, as we obtained the best results using all of them across all the ensemble models. When we decided to use a subset of them, as we did with Random Forest, the results were significantly worse.