MACHINE LEARNING PROJECT REPORT: SUICIDE DETECTION IN TEXT

Team: Isabela MORA, Mathieu MAURY, Paul MILLIEN, Yannis MICHOUX
Subject: Natural Language Processing (NLP) / Binary Classification

December 2, 2025

GitHub Repository: https://github.com/isabelamoraesilv/Machine-Learning-Project

---

1 Business Scope

Business Case and Objective

In the digital era, social media platforms have become primary outlets for individuals to express personal distress. Mental health crises, specifically suicide, remain a global leading cause of death. However, the sheer volume of content generated online makes manual monitoring by human moderators effectively impossible.

The Objective: The goal of this project is to develop an automated Machine Learning system capable of detecting suicidal ideation in unstructured text data with high accuracy.

Link to Specialization: This project sits at the intersection of **Natural Language Processing (NLP)** and **Content Safety**. By automatically flagging concerning posts, this tool can assist mental health organizations and social media platforms in early intervention strategies, prioritizing critical cases for human review. This addresses the urgent business need for scalable, real-time safety mechanisms in digital communities.

2 Problem Formalization and Methods

The problem is formalized as a **Supervised Binary Classification** task.

- **Input (X):** Unstructured text strings (user posts derived from Reddit).
- **Output (y):** A binary label: `suicide` (1) or `non-suicide` (0).

Methods Overview

We employed a standard NLP pipeline to transform raw text into predictive insights:

1. **Text Preprocessing:** Cleaning noise and normalizing raw text to ensure model consistency.
2. **Feature Extraction:** Converting text strings into numerical vectors using **TF-IDF** (Term Frequency-Inverse Document Frequency) to capture the weight of significant words.
3. **Dimensionality Reduction:** Using **TruncatedSVD** (Latent Semantic Analysis) to compress sparse matrices for computationally expensive models like Gradient Boosting.

4. **Classification:** Training and comparing a diverse set of algorithms, from linear models to neural networks.

5. **Feature Extraction Comparison:** In addition to TF-IDF, we implemented a **Bag of Words (BoW)** approach using `CountVectorizer`. This allows us to verify if the raw frequency of specific alarmist terms (e.g., "kill", "die") provides a stronger signal than the penalized weighting of TF-IDF.

3 Algorithm Description

We selected a range of algorithms to compare their efficacy in handling high-dimensional sparse text data:

1. **Logistic Regression (Baseline):** A linear classifier selected for its interpretability and efficiency. It serves as our baseline performance metric.

2. **Multinomial Naïve Bayes:** A probabilistic classifier based on Bayes' theorem. It is a traditional standard for text classification due to its handling of discrete frequency counts.

3. **Linear SVM (Support Vector Machine):** Highly effective in high-dimensional spaces, it attempts to find the optimal hyperplane that separates the two classes with the maximum margin.

4. **Random Forest:** An ensemble method using bagging (bootstrap aggregating) of decision trees. It captures complex interactions but is prone to overfitting on sparse data.

5. **Gradient Boosting:** An ensemble boosting method that builds trees sequentially, where each new tree corrects the errors of the previous ones.

6. **Bagging Classifier:** A meta-estimator that fits base classifiers (in our case, Logistic Regression) on random subsets of the original dataset to reduce variance.

7. **Multi-Layer Perceptron (MLP):** A Neural Network used to capture potential non-linear relationships that simple linear models might miss.

4 Limitations

- **Contextual Nuance:** The TF-IDF approach relies on word frequency and ignores the sequential order of words. Consequently, the models may struggle with sarcasm, complex negation, or metaphorical language compared to modern Transformer models (like BERT).

- **Data Source Bias:** The dataset is sourced exclusively from Reddit ("SuicideWatch" vs "Teenagers"). The language used is specific to this demographic and platform, which potentially limits the model's ability to generalize to other contexts (e.g., clinical notes or Twitter).

- **Dimensionality Reduction:** For the Gradient Boosting model, we reduced features from 5,000 to 100 using SVD to manage computational costs. This compression inevitably leads to some loss of semantic information.

5 Methodology

Data Description and Exploration

- **Source:** The dataset was acquired via Kaggle (`nikhileswarkomati/suicide-watch`).
- **Volume:** The raw dataset contains approximately **232,074** entries.
- **Structure:** It consists of two columns: `text` (the user post) and `class` (the target label).

Data Preprocessing & Cleaning

We implemented a robust cleaning function to maximize the signal-to-noise ratio:

1. **Noise Removal:** Regex was utilized to strip URLs, HTML tags, numbers, and punctuation.
2. **Normalization:** All text was converted to lowercase.
3. **Stopwords & Lemmatization:** We removed common English stopwords (e.g., "the", "is") using the NLTK library and applied **Lemmatization**. Unlike stemming, which chops words, lemmatization converts words to their dictionary root (e.g., "better" $\rightarrow$ "good"), reducing vector sparsity.

Missing Values

We performed a null check (`df.isnull().sum()`). While the dataset was largely clean, we included a safety step `df.dropna()` to prevent runtime errors during vectorization.

Imbalanced Data

We analyzed the target distribution:

- **Non-suicide:** $\sim 50\%$
- **Suicide:** $\sim 50\%$

Conclusion: The dataset is perfectly balanced. Therefore, we did **not** need to apply resampling techniques (like SMOTE), and **Accuracy** remains a valid evaluation metric alongside F1-Score.

Outliers

We analyzed the distribution of text length.

- *Observation:* Some texts were extremely short (e.g., "help"), providing insufficient context for a model, while others were excessively long (noise).
- *Action:* We filtered the dataset to keep texts between **20 and 3,000 characters**.

Data Splitting

We split the data into **Training (80%)** and **Testing (20%)** sets.

- **Stratification:** We used `stratify=y` to ensure the perfect class balance (50/50) was strictly maintained in both the training and testing subsets.

6 Algorithm Implementation and Hyperparameters

Vectorization

We used **TfidfVectorizer** limited to `max_features=5000`. This constraint was chosen to balance model performance with memory efficiency, preventing the "Curse of Dimensionality."

Hyperparameter Tuning

To address the obstacle of parameter optimization, we utilized **GridSearchCV** with 3-fold Cross-Validation on the Logistic Regression model.

- **Grid:** We tested different Regularization strengths (`C`: 0.1, 1, 10) and solvers.
- **Result:** The optimal parameters found were `{'C': 10, 'solver': 'lbfgs'}`.

Bag of Words vs. TF-IDF Experiment

To validate the effectiveness of our TF-IDF vectorization, we conducted a comparative experiment using a **Bag of Words (BoW)** representation. We trained our best-performing classifier (Logistic Regression with $C = 10$) on features generated by simple occurrence counting. We maintained the same vocabulary size constraint ($N = 5000$) to ensure a fair comparison, isolating the impact of term weighting versus raw frequency.

Overfitting Control

We implemented a custom evaluation function `check_overfitting_and_robustness`. This function calculates the gap between Train Accuracy and Test Accuracy.

- **Threshold:** If the gap exceeded 5%, a warning was flagged.
- **Neural Network Strategy:** We used `early_stopping=True` in the MLP classifier. This automatically halts training when validation scores stop improving, preventing the network from memorizing the training data.

7 Results

We evaluated all models based on **Accuracy**, **Precision**, **Recall**, and **F1-Score**. Below is the summary of the final performance on the test set.

Rank	Model	Accuracy	F1 (Macro)	Observations
1	Bagging (LogReg)	93.44%	0.93	Best Performance, highly stable
2	Neural Network (MLP)	93.38%	0.93	Captured non-linearities well
3	Linear SVM	93.37%	0.93	Efficient for high-dimensional text
4	LogReg (Tuned TF-IDF)	93.35%	0.93	Strong baseline with weighting
5	LogReg (Bag of Words)	92.90%	0.93	Slightly lower than TF-IDF
6	Naïve Bayes	90.20%	0.90	Fast, but lower precision
7	Gradient Boosting	87.87%	0.88	Suffered from dim. reduction (SVD)
8	Random Forest	85.88%	0.86	Signs of overfitting (Train > Test)

Table 1: Final Model Comparison including Bag of Words

Overfitting/Underfitting Analysis

- **Linear Models (SVM, LogReg):** These showed almost no overfitting (Train/Test gap $< 1\%$). They generalized extremely well to the unseen test data.
- **Random Forest:** Exhibited the largest gap (Train $\sim 88.5\%$, Test $\sim 85.9\%$). While not catastrophic, it indicates the model was beginning to memorize noise rather than learning generalized language patterns.

8 Discussion and Conclusion

Evaluation and Comparison

Our analysis demonstrates that **linear models** (Logistic Regression, SVM) and the **Bagging ensemble** of linear models significantly outperformed complex tree-based models (Random Forest, Boosting) on this dataset.

This is a common phenomenon in NLP when using TF-IDF: the feature space is sparse and high-dimensional. Linear models are mathematically well-suited to separate classes in such high-dimensional spaces. Tree-based models often struggle with sparse matrices unless heavily tuned or used with dense embedding techniques.

The **Neural Network (MLP)** performed equivalently to the best linear models but required significantly more training time and computational resources.

8.1 Feature Engineering Analysis

The comparison between TF-IDF and Bag of Words revealed that TF-IDF yields a slight performance advantage (approx. $+0.45\%$). This confirms that penalizing common words across the corpus helps the model focus on more discriminative terms specific to the suicidal context. However, the high performance of BoW demonstrates that the mere **presence** of specific high-risk keywords is already an extremely strong predictor for this specific classification task, even without sophisticated weighting schemes.

Conclusion on the Business Case

We successfully tackled the business objective. We have produced a **Bagging Logistic Regression model** with **93.4% accuracy**.

- **Reliability:** The model is robust, as verified via Cross-Validation.
- **Deployment:** It is lightweight compared to Transformers, allowing for low-latency real-time monitoring.
- **Impact:** This tool can effectively filter thousands of posts per second, flagging high-risk content for immediate human review, potentially saving lives.

9 References and External Sources

1. **Dataset:** Nikhileswar Komati, "Suicide Watch Dataset", Kaggle. https://www.kaggle.com/datasets/nikhileswarkomati/suicide-watch
2. **Scientific Paper:** Sida Wang and Christopher D. Manning. 2012. *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification.* In Proceedings of ACL 2012. (Justification for MLP and Linear Baselines).
3. **Libraries:** Scikit-learn, NLTK, Pandas, Seaborn documentation.

A Appendix: Figures

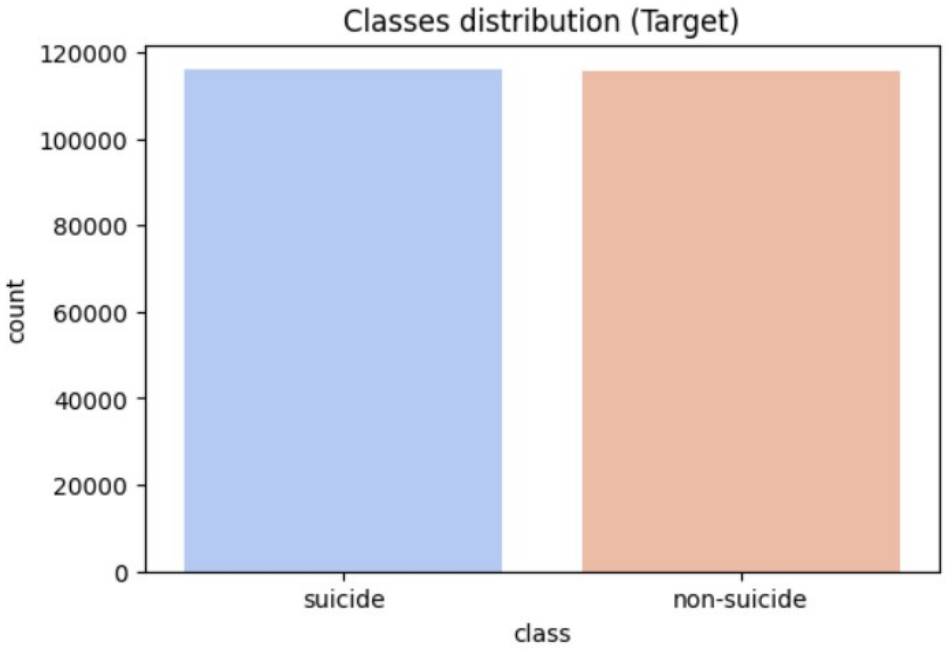

Figure 1: Classes distribution bar chart.

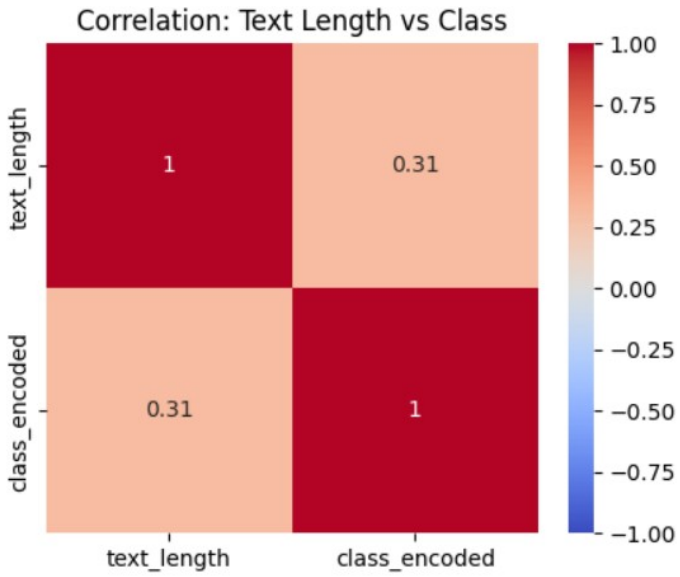

Figure 2: Correlation matrix

Logistic Regression (Baseline) results :
Accuracy: 0.9334
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.92	0.94	0.93	23164
suicide	0.94	0.92	0.93	23175
accuracy			0.93	46339
macro avg	0.93	0.93	0.93	46339
weighted avg	0.93	0.93	0.93	46339

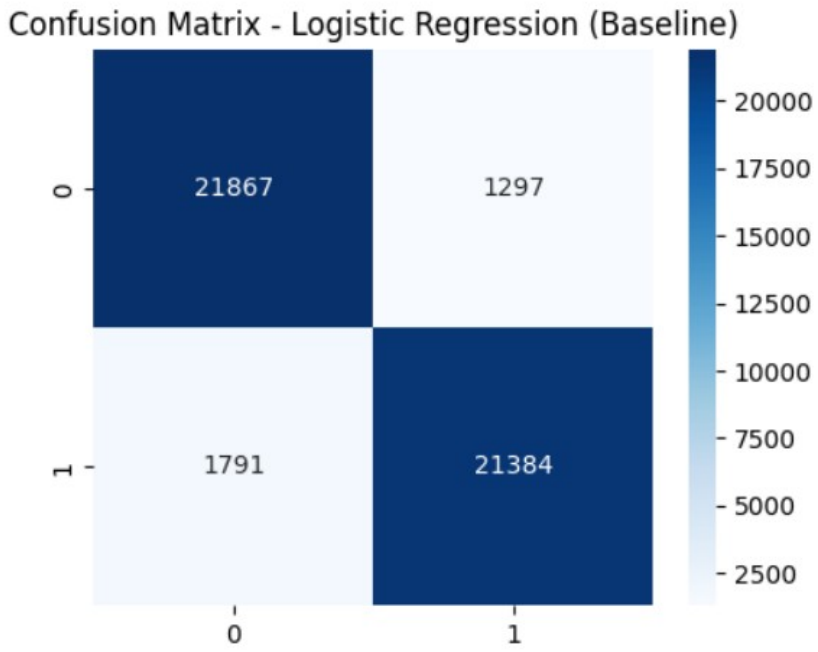

Figure 3: Logistic regression results.

Naïve Bayes results :
Accuracy: 0.9020
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.94	0.86	0.90	23164
suicide	0.87	0.95	0.91	23175
accuracy			0.90	46339
macro avg	0.91	0.90	0.90	46339
weighted avg	0.91	0.90	0.90	46339

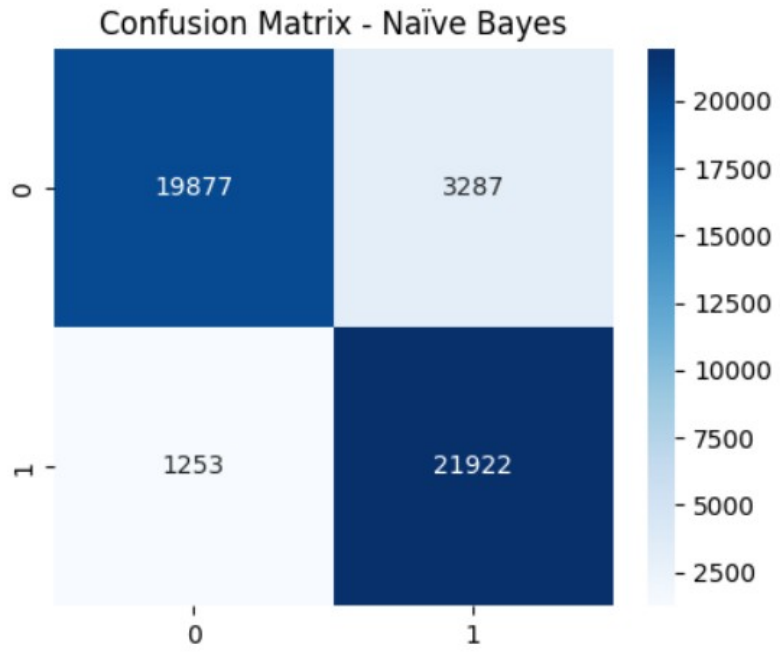

Figure 4: Naive Bayes results.

Linear SVM results :
Accuracy: 0.9337
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.92	0.94	0.93	23164
suicide	0.94	0.92	0.93	23175
accuracy			0.93	46339
macro avg	0.93	0.93	0.93	46339
weighted avg	0.93	0.93	0.93	46339

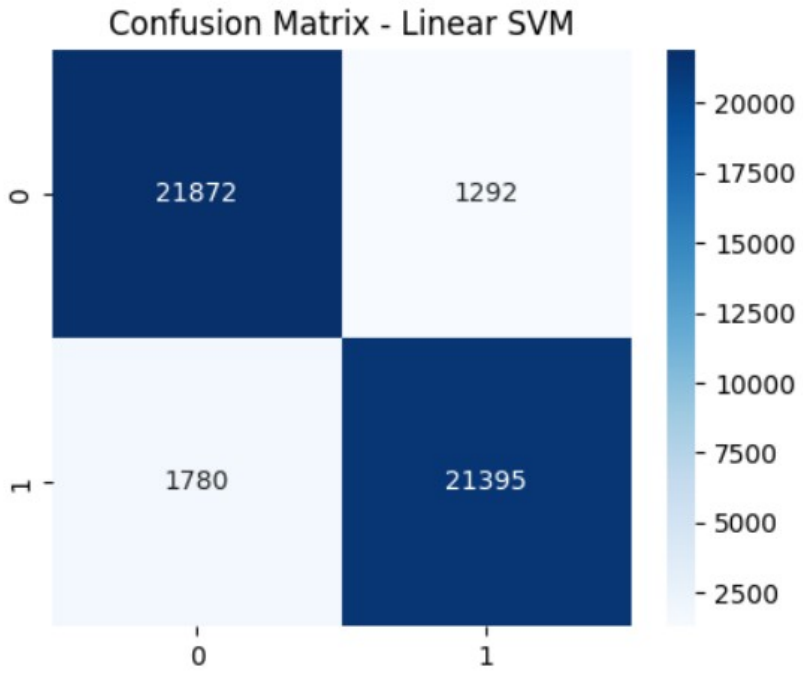

Figure 5: Linear SVM results.

Random Forest results :
Accuracy: 0.8588
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.83	0.91	0.87	23164
suicide	0.90	0.81	0.85	23175
accuracy			0.86	46339
macro avg	0.86	0.86	0.86	46339
weighted avg	0.86	0.86	0.86	46339

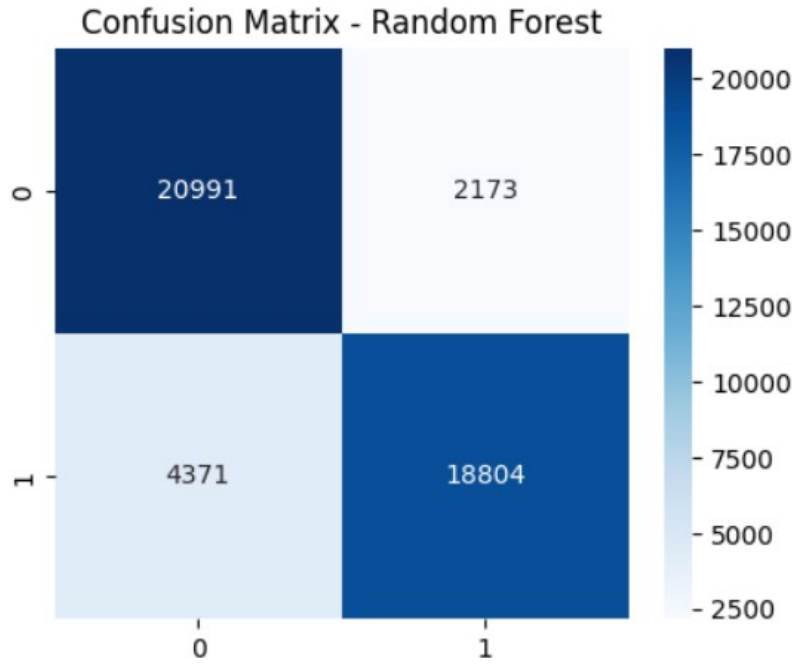

Figure 6: Random forest results.

```
Random Forest results :
Accuracy: 0.8588
Classification Report:
```

	precision	recall	f1-score	support
non-suicide	0.83	0.91	0.87	23164
suicide	0.90	0.81	0.85	23175
accuracy			0.86	46339
macro avg	0.86	0.86	0.86	46339
weighted avg	0.86	0.86	0.86	46339

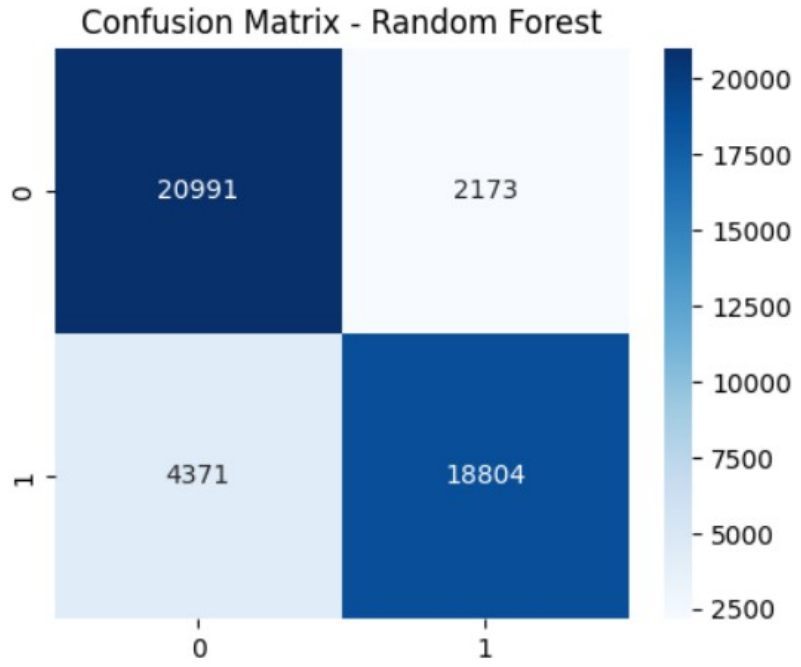

Figure 7: Tuned Logistic regression results.

Bagging results :
Accuracy: 0.9344
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.93	0.94	0.93	23164
suicide	0.94	0.92	0.93	23175
accuracy			0.93	46339
macro avg	0.93	0.93	0.93	46339
weighted avg	0.93	0.93	0.93	46339

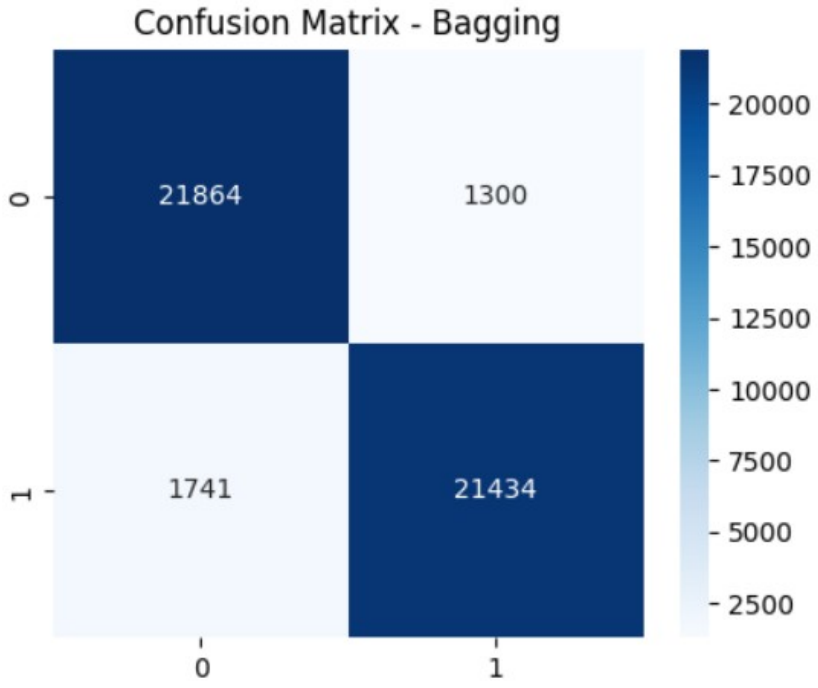

Figure 8: Bagging results.

Gradient Boosting results :
Accuracy: 0.8787
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.85	0.92	0.88	23164
suicide	0.92	0.83	0.87	23175
accuracy			0.88	46339
macro avg	0.88	0.88	0.88	46339
weighted avg	0.88	0.88	0.88	46339

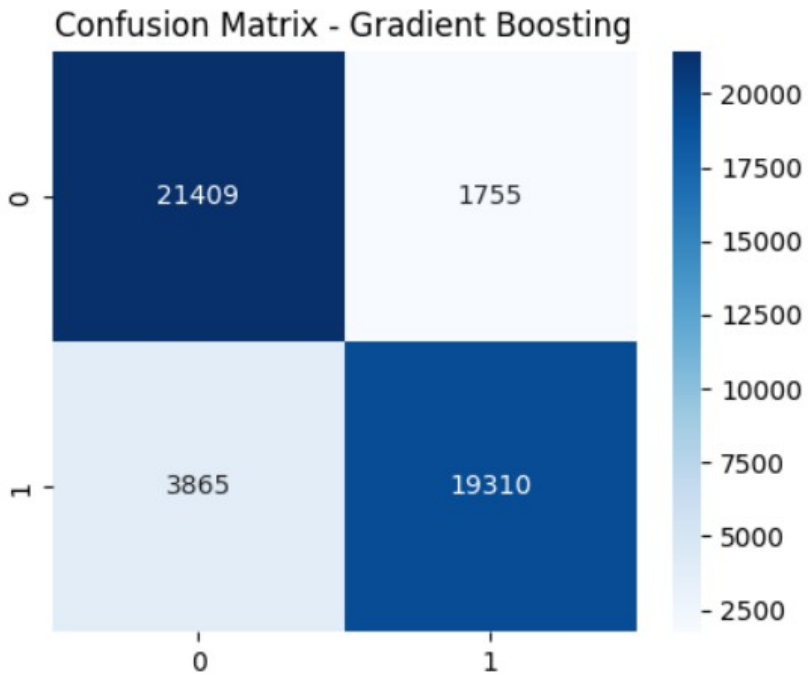

Figure 9: Gradient boost results.

Neural Network (MLP) results :
Accuracy: 0.9338
Classification Report:

	precision	recall	f1-score	support
non-suicide	0.93	0.93	0.93	23164
suicide	0.93	0.93	0.93	23175
accuracy			0.93	46339
macro avg	0.93	0.93	0.93	46339
weighted avg	0.93	0.93	0.93	46339

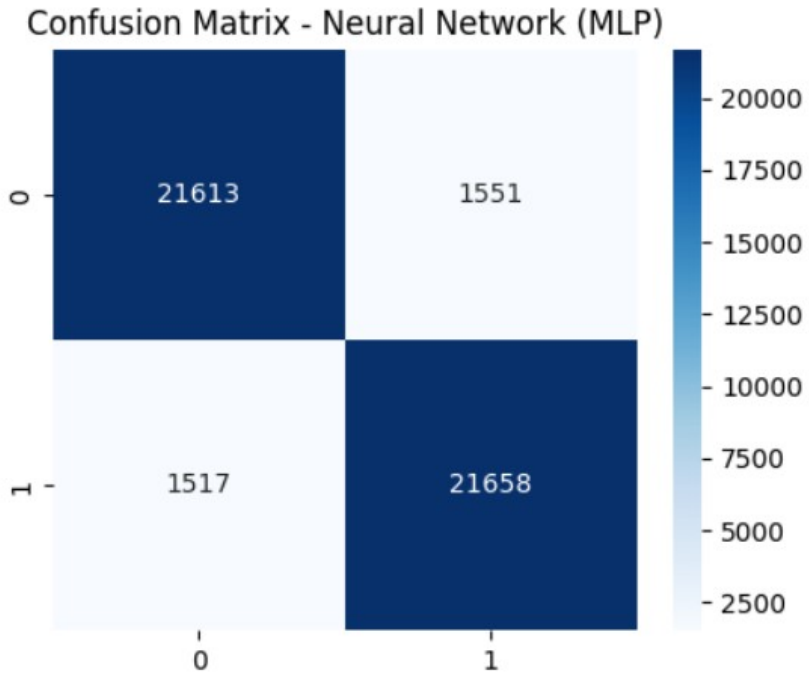

Figure 10: Neural networks MLP results.

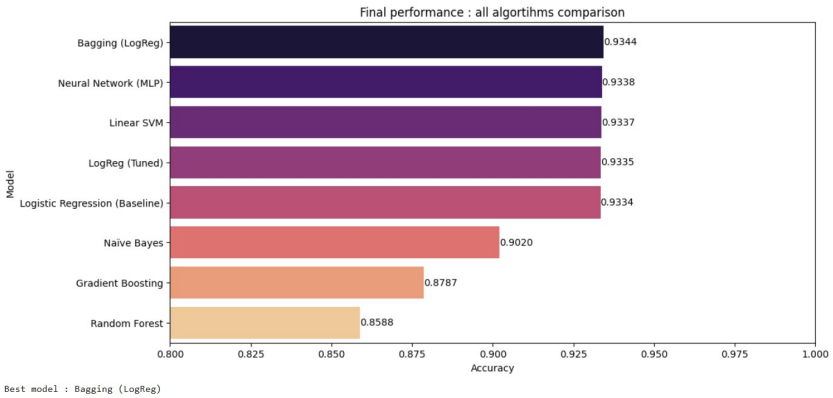

Figure 11: Final comparison.