**Student ID:** 23073695

**Name:** DONA ANN JIJI

**GitHub Link:** <https://github.com/donaann84/MLNN>

**Ensemble Learning in Machine Learning: Stacking vs Bagging vs Boosting**

Ensemble learning is a powerful machine learning paradigm where multiple models—often referred to as **base learners** or **weak learners**—are combined to solve a problem. The core philosophy behind ensemble learning is that a group of diverse models can outperform a single strong model if they are properly constructed and combined. This technique is especially valuable when aiming to improve predictive performance, reduce variance, mitigate bias, or enhance generalization across various machine learning tasks.

According to Zhou (2012), ensemble methods have become a mainstream technique in modern machine learning and are fundamental to many top-performing models in real-world applications and competitions [(Zhou, 2012)](https://doi.org/10.1007/978-3-642-38652-7).

**What is Bagging?**

**Bagging** (short for Bootstrap Aggregating) is a technique where multiple instances of a single base learning algorithm are trained on different bootstrap samples of the data. These models are trained in parallel, and their outputs are combined using methods like majority voting (for classification) or averaging (for regression).

The primary goal of bagging is to **reduce variance** by stabilizing predictions. Since each model is trained on a slightly different dataset, they tend to overfit differently. By averaging their results, we effectively cancel out their errors, leading to improved generalization. A well-known example of bagging is the **Random Forest**, which combines multiple decision trees trained on bootstrapped subsets of features and samples.

Breiman (1996) introduced bagging as a method to improve the stability and accuracy of machine learning algorithms [(Breiman, 1996)](https://doi.org/10.1023/A:1018054314350).

**What is Boosting?**

**Boosting** is another ensemble technique, but unlike bagging, it builds models **sequentially**. Each new model attempts to correct the errors made by the previous one, focusing more on the data points that were misclassified. This sequential learning enables boosting to reduce **bias** while still maintaining a relatively low variance.

The process involves weighting samples so that misclassified observations receive higher weight in subsequent rounds. Popular boosting algorithms include **AdaBoost**, **Gradient Boosting**, and **XGBoost**, all of which have been widely adopted in practice.

Freund and Schapire (1997) developed AdaBoost and demonstrated its success in reducing both bias and variance [(Freund & Schapire, 1997)](https://doi.org/10.1006/jcss.1997.1504).

**What is Stacking?**

**Stacking**, or stacked generalization, is a more sophisticated ensemble method where the predictions of several base learners are used as input to a **meta-model**. Unlike bagging and boosting, which rely on a single type of base learner, stacking often combines **heterogeneous models** (e.g., logistic regression, SVM, decision trees) to leverage their diverse strengths.

The base learners are trained on the original dataset, and the meta-model is trained on their predictions. The idea is that the meta-model can learn how to best combine the predictions of the base models, potentially correcting their individual biases.

Wolpert (1992) introduced the concept of stacked generalization as a method to minimize generalization error [(Wolpert, 1992)](https://doi.org/10.1016/S0893-6080(05)80023-1).

**When to Use Which?**

Each ensemble technique has its own strengths and is suited to different scenarios:

* Use **Bagging** (e.g., Random Forest) when your base model has **high variance**, like decision trees. It’s ideal when you want a robust model that performs consistently across different subsets of data.
* Use **Boosting** (e.g., XGBoost, AdaBoost) when your model suffers from **bias**, such as linear models on nonlinear problems. It’s especially powerful in tasks requiring high accuracy.
* Use **Stacking** when you want to combine **multiple types of models** to create a strong learner. It works best when your base models are diverse and complementary.

**Summary Comparison Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Bagging** | **Boosting** | **Stacking** |
| Training Style | Parallel | Sequential | Parallel + Meta-model |
| Reduces Bias? | ❌ | ✅ | ✅ |
| Reduces Variance? | ✅ | ✅ | ✅ |
| Base Models | Homogeneous | Homogeneous | Heterogeneous |
| Example Algorithms | Random Forest | AdaBoost, XGBoost | StackingClassifier |

In the next section, we’ll demonstrate all three techniques using a structured dataset (e.g., customer churn). We’ll train models using Scikit-learn and XGBoost, visualize their performance using ROC curves and confusion matrices, and interpret the pros and cons of each approach.

Let’s dive into the code!

**Dataset Overview: Telco Customer Churn**

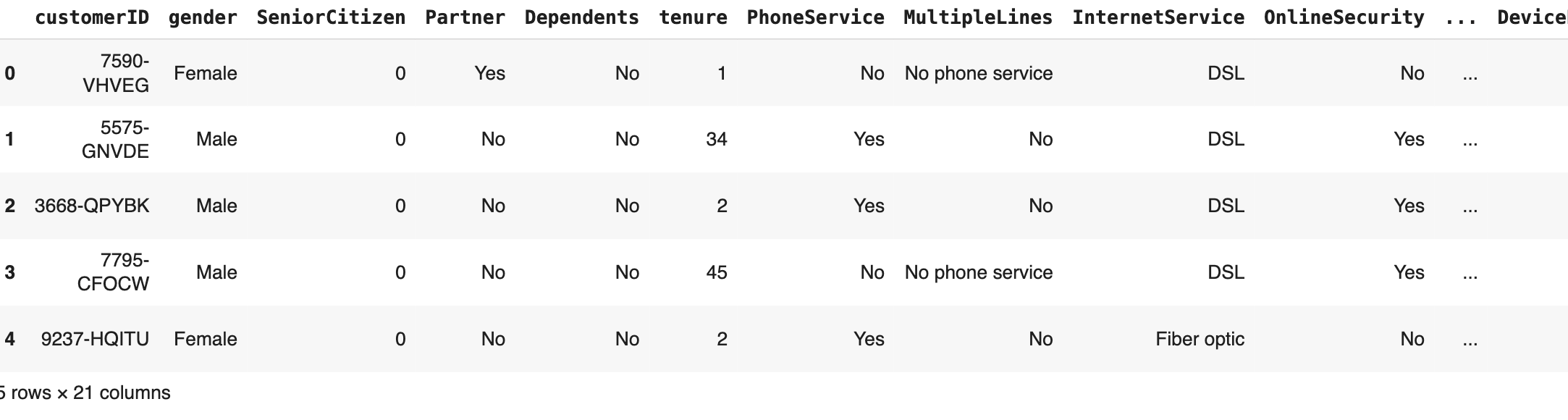
For this tutorial, I am using the **Telco Customer Churn** dataset from Kaggle. This dataset contains various customer demographic and service usage attributes to predict whether a customer will churn (i.e., leave the telecom service). It includes both numerical and categorical variables such as MonthlyCharges, Contract, InternetService, and the target label Churn.

After preprocessing (e.g., converting TotalCharges to numeric and encoding categorical variables), this dataset becomes an excellent testbed to apply and compare ensemble techniques.

**Step-by-Step Coding Explanation**

**Step 1: Data Loading and Exploration**

We load the dataset using pandas and immediately inspect the first few rows. This gives a sense of the features involved and allows us to spot any immediate data quality issues.



**Step 2: Data Preprocessing**

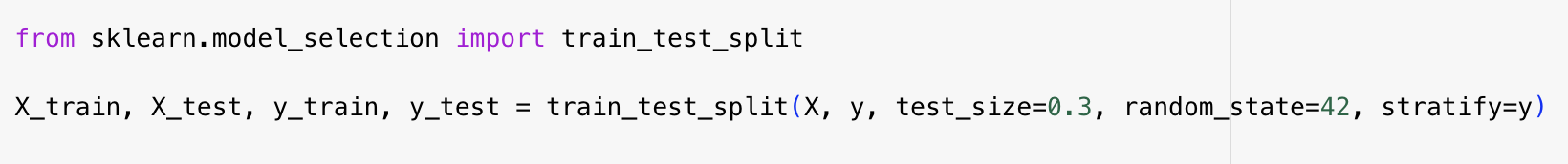
We drop the customerID column (which is not predictive), convert TotalCharges to numeric (handling coercion errors), drop null values, and perform one-hot encoding for categorical variables. The target column Churn is converted to binary (1 for churned, 0 for retained).

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 3: Train-Test Split**

Using stratified sampling to maintain class balance, we split the data into training and testing sets. This ensures fair evaluation across models.



**Step 4: Bagging with Random Forest**

We train a RandomForestClassifier on the training data and make predictions on the test set. We evaluate its performance using a classification report. Bagging works well here due to the ability of trees to handle nonlinear patterns and variance in data.

A screenshot of a computer code

AI-generated content may be incorrect.

**Step 5: Boosting with XGBoost**

Next, we train an XGBClassifier, which is a powerful implementation of gradient boosting. It sequentially improves on weak learners by focusing on misclassified examples. We again evaluate the output using classification metrics.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 6: Stacking**

We implement StackingClassifier using a combination of Random Forest and SVM as base learners, with logistic regression as the meta-learner. The stacking model leverages the diverse strengths of its components.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 7: ROC Curves**

We generate ROC curves for all three models using predicted probabilities. This visually shows the trade-off between true positive and false positive rates.

A graph of a curve

AI-generated content may be incorrect.  
*ROC curve comparing ensemble methods. Random Forest and Stacking have an AUC of 0.82, while XGBoost follows closely with 0.81.*

**Step 8: Confusion Matrices**

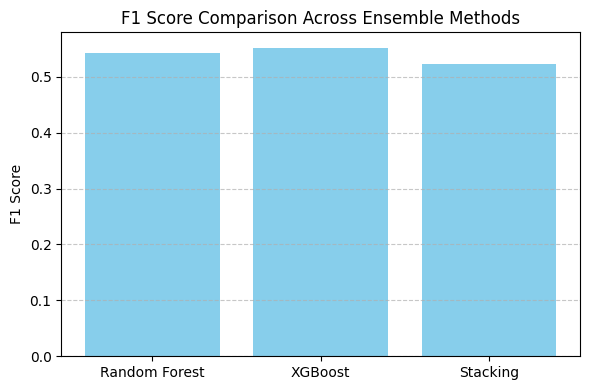
Side-by-side confusion matrices for all models allow comparison of how each model classifies positive and negative churn outcomes.

A chart with a yellow and purple squares

AI-generated content may be incorrect.  
*Confusion matrices for Random Forest, XGBoost, and Stacking. Each matrix highlights true positives and false positives.*

**Step 9: F1 Score Comparison**

We compute and plot F1 scores to understand the harmonic balance between precision and recall for each model.

  
*Bar chart comparing F1 scores across ensemble models. XGBoost slightly leads, indicating better balance in classification.*

**Final Summary and Conclusion**

After evaluating the three ensemble learning methods—Bagging (Random Forest), Boosting (XGBoost), and Stacking (Random Forest + SVM)—on the Telco Customer Churn dataset, we can draw the following conclusions:

* **Random Forest** (Bagging) showed consistent and balanced performance. It achieved an AUC of 0.82 and offered strong generalization by reducing variance through model averaging. It had good performance in correctly identifying churn but slightly lower recall compared to boosting.
* **XGBoost** (Boosting) performed very well, with a slightly lower AUC of 0.81 but the highest F1-score. Its sequential training allowed it to effectively reduce both bias and variance, making it especially effective in recognizing true churners. It had the most balanced precision and recall trade-off.
* **Stacking**, while powerful in theory, did not outperform the simpler models in this case. Despite combining multiple classifiers and achieving a respectable AUC of 0.82, its F1-score was slightly lower than XGBoost. This may be due to the complexity of tuning meta-learners and limited base model diversity.

From a practical perspective:

* Use **Random Forest** when you need a fast, stable, and interpretable model.
* Choose **XGBoost** when you seek maximum predictive accuracy and are comfortable with additional complexity.
* Consider **Stacking** when your dataset and models offer diverse perspectives and you can afford extra computation.

Overall, XGBoost emerged as the most effective technique in this case study, but each method has its merits depending on the specific problem context.

**GitHub and Accessibility**:

The full tutorial notebook, code files, and instructions are available in the [GitHub repository](https://github.com/donaann84/MLNN) and are designed to be screen reader-friendly, keyboard-navigable, colorblind-safe, and caption-annotated.

**References**

* Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123–140. <https://doi.org/10.1023/A:1018054314350>
* Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119–139. <https://doi.org/10.1006/jcss.1997.1504>
* Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241–259. <https://doi.org/10.1016/S0893-6080(05)80023-1>
* Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms*. Springer. <https://doi.org/10.1007/978-3-642-38652-7>