Ministry of Science and Higher Education
of the Russian Federation

Faculty of Digital Transformations

Subject Area (Major): 01.04.02

Applied Mathematics and Informatics

The study of the comparative efficiency of machine learning algorithms in solving specific problem

Author:
Jorge Javier Sosa Briseno
sosa_brijj@icloud.com

Supervisor:
Dr. Gladilin P.E.

St. Petersburg
2024

Contents

1 Introduction

The course challenge involves using the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology to develop a solution that integrates data science and artificial intelligence with cutting-edge computational tools. This requires obtaining a reliable database to perform an analysis, aiming to predict a specific variable. In this case, the challenge is to predict whether a person who boarded the Titanic survived or not.

The first step to address the problem was to identify the issue, as well as the requirements of the challenge and the tools needed to meet these requirements.

Initially, it was necessary to download the Titanic passengers database, which includes key features such as name, ticket number, gender, age, and so on.

The second step was to use a platform that facilitates the application of mathematical tools and artificial intelligence, providing a better opportunity to solve the problem collaboratively. For this, Python was chosen as the programming language, and the development was carried out on the Google Colab platform using a file format known as Jupyter Notebook.

It was also identified that the first step in generating predictions about the survival or death of Titanic passengers is cleaning the database. This is because there are variables that do not provide useful information about the event that led to the deaths of many passengers (specifically, who boarded the lifeboats and who did not).

2 Literature Review

The field of machine learning and artificial intelligence has seen rapid advancements over the past few decades, with applications ranging from predictive modeling to neural network implementations. In this study, we focus on a range of supervised learning algorithms to predict the survival of Titanic passengers. The selection of models and methodologies is grounded in established research and practical utility in the domain of classification problems.

2.1 Gradient Boosting and XGBoost

Gradient Boosting is a widely used ensemble technique that builds a series of models to correct the errors of its predecessors. Research has shown that gradient boosting algorithms, such as XGBoost, are particularly effective in handling structured datasets with missing values and complex relationships [1]. XGBoost extends Gradient Boosting by introducing regularization techniques and parallel computation, making it computationally efficient for large-scale datasets [2].

2.2 Decision Tree Classifiers and Random Forests

Decision Trees are interpretable models that recursively split data into subsets based on feature values. However, they are prone to overfitting, especially in the absence of regularization. Random Forests mitigate this issue by creating multiple decision trees and aggregating their results, leading to improved accuracy and robustness [3]. These models are often used as a baseline for classification problems due to their simplicity and efficiency.

2.3 Neural Networks and Multi-Layer Perceptrons (MLP)

Neural networks, particularly Multi-Layer Perceptrons (MLP), have gained significant attention due to their ability to learn non-linear patterns. The backpropagation algorithm, a cornerstone of supervised deep learning, enables these models to iteratively adjust their weights to minimize error [4]. Recent applications highlight the adaptability of MLPs to a wide range of tasks, including binary and multi-class classification problems.

2.4 Evaluation Metrics and Data Handling

The selection of appropriate evaluation metrics is crucial for comparing model performance. Metrics such as accuracy, precision, recall, and F1-Score provide insights into the trade-offs between false positives and false negatives [5]. Additionally, the handling of missing data plays a critical role in the success of machine learning models. Studies emphasize the importance of strategies like imputation or preserving null values to maintain data integrity and model reliability [6].

2.5 Practical Applications and Future Work

The methodologies discussed in this study are aligned with contemporary practices in predictive modeling. Gradient Boosting and Random Forests are widely adopted in structured data scenarios, while neural networks remain at the forefront of tasks requiring flexibility and nonlinearity. Future work could explore advanced ensemble techniques or deep neural networks to further enhance predictive capabilities.

3 Code Structure

3.1 Directory

This section uses a Google Colab environment, a platform provided by Google for running code in the cloud using Jupyter Notebooks. The purpose of this section is to mount (connect) the Google Drive account to Google Colab to access and work with files stored in Google Drive from the Colab environment.

3.2 Data Cleaning

To efficiently analyze and properly clean the data, we utilized Python libraries such as **Pandas**, **Numpy**, and **Matplotlib**.

As the first cleaning step, we decided to remove the following categories: **ID**, **Name**, **Parch**, **Ticket**, **Cabin**, and the relationships between passengers (*parents, siblings, spouses*). This decision was based on the hypothesis that the most determining factors for survival were **age**, **gender**, and **class**, considering the historical priority given to women and children when boarding lifeboats. The **Ticket Number** was considered irrelevant since it merely represents a record of ticket sales and does not correlate with a passenger survival.

Next, we converted the categorical variables **Sex** (*"sex"*) and **Embarkation** (*"embarked"*) into numerical variables to establish mathematical relationships. Male was replaced with `0` and female with `1`, while the embarkation points (*"S"*),(*"C"*),(*"Q"*) were converted to `0`, `1`, and `2`, respectively.

Additionally, we used the `dropnan()` function to remove undefined data from the dataset, as they provide no useful information for analysis.

Similarly, the `dropnull()` function was used to remove rows containing null values to minimize noise during analysis.

From this point, we started working with the cleaned data to calculate statistics that enabled further analysis:

- We performed counts and percentage distributions of categories like **gender**, **embarkation**, **surviving men**, **deceased men**, **surviving women**, and **deceased women**.
- We analyzed quartiles and quintiles for the frequencies of passenger ages, noting that the most common age range was between **20 and 40 years old**.
- A correlation analysis was conducted to identify which variables were most strongly related to survival.

We found that the variables most correlated with survival were **gender** and the **fare paid for the ticket**. The correlation table indicated a significant relationship (close to 1) between survival, gender, and class. However, age showed a weaker correlation of approximately **9%**, contrary to expectations.

3.3 Modeling

In this stage, we present the process followed in the project, focusing on the generation and comparison of machine learning models to address the posed challenge. This phase explores different types and configurations of models to identify the most effective ones, with decisions documented in detail.

A study of classification models was conducted, identifying several models suitable for our dataset to predict a binary outcomeâĂŤin this case, the survival or death of a Titanic passenger. The models selected include:

- `KNeighborsClassifier`
- `Support Vector Classifier (SVC)`

- `LogisticRegression`
- `DecisionTreeClassifier`
- `GaussianNB`
- `RandomForestClassifier`
- `GradientBoostingClassifier`
- `MLPClassifier`

Among these, we aim to select only three models. A selection process was conducted to choose the best-performing models, and the preparation of the dataset for each model, including training and prediction splits, was documented.

The continuation of the project builds upon the data cleaning completed in the previous deliverable. The next step involved importing the necessary libraries for utilizing the prediction models of interest. The following code demonstrates this process.

4 Experimental research

4.1 Libraries for Classification Models

K-Neighbors Classifier: Uses the k-nearest neighbors method for classification. Predictions are based on the classes of the nearest examples in the feature space. It requires tuning the number of neighbors and possibly other distance metrics.

```
from sklearn.neighbors import KNeighborsClassifier
```

Code Listing 1: *Importing KNeighborsClassifier*

SVC (Support Vector Classifier): A classifier that finds the optimal separating hyperplane between classes. It can use various kernel functions to transform the data into a higher-dimensional space. It is useful for both linear and non-linear classification problems.

```
from sklearn.svm import SVC
```

Code Listing 2: *Importing Support Vector Classifier (SVC)*

Logistic Regression: Performs classification using the logistic function to model the probability of belonging to a class. It is effective for binary and multi-class classification problems when classes are linearly separable.

```python
from sklearn.linear_model import LogisticRegression
```

Code Listing 3: *Importing Logistic Regression*

Decision Tree Classifier: Builds a decision tree that recursively splits the feature space into purer regions. It is easily interpretable but can be prone to overfitting if not properly controlled.

```python
from sklearn.tree import DecisionTreeClassifier
```

Code Listing 4: *Importing Decision Tree Classifier*

Gaussian NB (Naive Bayes): Implements the Gaussian NaÃŕve Bayes classifier, assuming features are independent and normally distributed. Despite its simplicity, it can perform surprisingly well in many cases.

```python
from sklearn.naive_bayes import GaussianNB
```

Code Listing 5: *Importing Gaussian Naive Bayes*

Random Forest Classifier: Creates multiple decision trees and combines their predictions to improve model robustness and accuracy. It can handle both categorical and numerical features and is less prone to overfitting than a single tree.

```python
from sklearn.ensemble import RandomForestClassifier
```

Code Listing 6: *Importing Random Forest Classifier*

Gradient Boosting Classifier: Builds a sequence of models that correct the errors of the previous model. It is useful for classification and regression problems and tends to produce high-quality models.

```python
from sklearn.ensemble import GradientBoostingClassifier
```

Code Listing 7: *Importing Gradient Boosting Classifier*

Decision Tree Classifier (Repeated): Constructs a decision tree that recursively splits the feature space into purer regions. It is easily interpretable but can be prone to overfitting if not properly controlled.

```python
from sklearn.neural_network import MLPClassifier
```

Code Listing 8: *Importing Multi-Layer Perceptron Classifier (MLP)*

Cross_val_score Library: Used in a function (not yet applied in the project) to generalize analyses with graphs and predictions. Additionally, `accuracy_score` and `metrics` are imported to gather information about the models and their predictions.

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

Code Listing 9: *Importing Cross Validation and Metrics Libraries*

4.2 Model Implementation

For each model, different tests were performed: using various hyperparameters and two different datasets. One dataset excluded passengers with null data, while the other predicted the null values using the k-nearest neighbors algorithm to avoid discarding those data points. The following procedure was followed:

4.3 Model Implementation and Results

For each test, the hyperparameters were adjusted with the objective of finding the best hyperparameters to achieve better predictions.

The following function is designed to create a meta-analysis of multiple algorithms to predict the death of a person on the Titanic. However, it has not yet been implemented for use.

Next, an array of models was created to store the models. This array is used in a loop to prepare, train, and predict with each of the imported models. Additionally, a precision analysis was performed using the `accuracy_score` library.

The scores of the models are as follows:

- **K-Neighbors Classifier (KNN)**: 0.643357
- **Support Vector Classifier (SVC)**: 0.671329
- **Logistic Regression (LR)**: 0.755245
- **Decision Tree Classifier (DT)**: 0.713287
- **Gaussian NB (GNB)**: 0.755245
- **Random Forest Classifier (RF)**: 0.769231

- **Gradient Boosting Classifier (GB)**: 0.783217
- **Multi-Layer Perceptron Classifier (MLP)**: 0.762238

It is evident that the models with the best accuracy are **Gradient Boosting**, **Multi-Layer Perceptron**, and **Decision Tree Classifier**. These will be among the models used for analysis and prediction. However, further adjustments to the parameters will be made to improve the prediction results.

From this point, the procedures shown in the previous diagram were followed to optimize the hyperparameters for our selected models. These include **Gradient Boosting**, **Extreme Gradient Boosting (XGBoost)**, and a neural network.

For Gradient Boosting specifically, we selected the best hyperparameters and conducted an analysis of the significance of input variables. This resulted in the following graph, indicating that the `embarked` variable does not need to be included as it does not appear to correlate with the outcome of passenger deaths.

```
# Initialize the model with a learning_rate of 0.01
GB = GradientBoostingClassifier(learning_rate=0.01,
    subsample=0.6)

# Fit the model to the training data
GB.fit(X_NonNull_train, y_NonNull_train)

# Predict the test data
y_NonNull_pred = GB.predict(X_NonNull_test)

# Generate the confusion matrix
confusion_matrix = metrics.confusion_matrix(y_NonNull_test,
    y_NonNull_pred)
```

Code Listing 10: *Implementing Gradient Boosting Model*

4.4 Results and Variable Importance

The results obtained are as follows:

- **Accuracy**: 0.8111888111888111
- **Precision**: 0.7735849056603774
- **Recall**: 0.9647058823529412
- **F1 Score**: 0.8586387434554973

A graph was implemented to visualize the importance of each variable for prediction. Based on the graph, the variable `Embarked` could be discarded as it appears to have low importance.

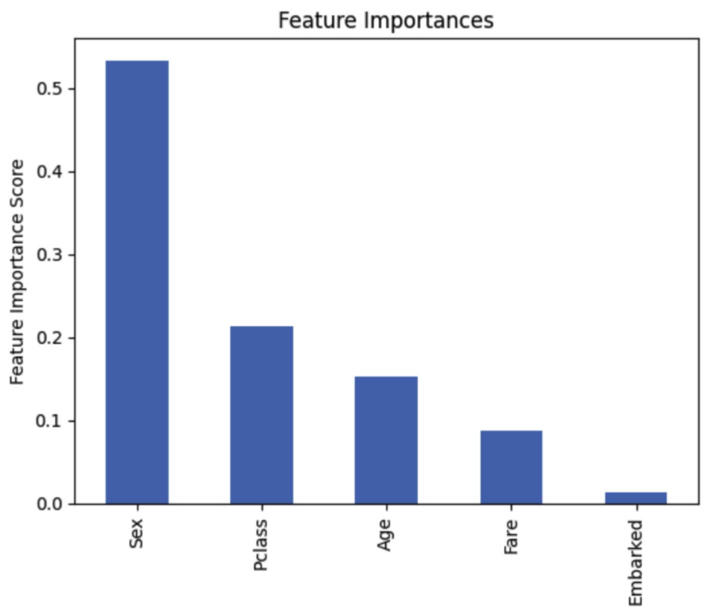

Figure 1: *Importance*

4.5 Application of Xtreme Gradient Boosting

Next, the Xtreme Gradient Boosting model was applied. This model builds a decision-making system based on a branching tree, or k-tree. It offers a variation of the gradient boosting method with better computational efficiency, as it is more efficient than the previously mentioned method due to its parallel computation capability.

```
# Initialize the XGBoost model with specified
    hyperparameters
xGB = xgb.XGBClassifier(learning_rate=0.1, subsample=0.2,
                         min_child_weight=2, reg_alpha=1.5)

# Fit the model to the training data
xGB.fit(X_NonNull_train, y_NonNull_train)

# Predict the test data
y_NonNull_pred = xGB.predict(X_NonNull_test)

# Generate the confusion matrix
confusion_matrix = metrics.confusion_matrix(y_NonNull_test,
    y_NonNull_pred)

# Calculate the accuracy
accuracy = (confusion_matrix[0, 0] + confusion_matrix[1, 1])
     / (
    confusion_matrix[0, 0] + confusion_matrix[0, 1] +
    confusion_matrix[1, 0] + confusion_matrix[1, 1])
```

Code Listing 11: *Implementing XGBoost Classifier*

4.6 Improved Metrics and Neural Network Implementation

It can be observed that the metrics for precision and accuracy in prediction results improved:

- **Accuracy**: 0.8391608391608392
- **Precision**: 0.8522727272727273
- **Recall**: 0.8823529411764706
- **F1 Score**: 0.8670520231213872

4.7 Neural Network Implementation - Multi-layer Perceptron Classifier (MLP)

This classifier implements the supervised deep learning algorithm MLP, which is trained using the backpropagation method. It is composed of multiple layers of neurons: input layers, hidden layers, and output layers, each connected through weighted connections [**?**]. This model was chosen to predict fatalities on the Titanic due to its current popularity in society, as it forms the basis of new technologies that we are using, such as modern AI applications. The following steps were taken to implement the model:

```python
# Create a neural network classifier with 100 hidden layers
    and a max iteration of 1000
mlp_model = MLPClassifier(hidden_layer_sizes=(100, 50),
    max_iter=1000)

# Adjust the model to the training data
mlp_model.fit(X_NonNull_train, y_NonNull_train)

# Calculate the score of the neural network
mlp_score = mlp_model.score(X_NonNull_test, y_NonNull_test)
```

Code Listing 12: *Implementing Neural Network Classifier (MLP)*

- **MLP Score**: 0.7342657342657343

A variable was created to regularize the data using `DecisionTreeClassifier` with parameters from the layers of the neural network.

```python
# Configure the decision tree with regularization parameters
regularized_tree = DecisionTreeClassifier(max_depth=5,
    min_samples_split=5)

# Fit the model to the training data
regularized_tree.fit(X_NonNull_train, y_NonNull_train)

# Calculate the score of the decision tree
regularized_tree_score = regularized_tree.score(
    X_NonNull_test, y_NonNull_test)
```

Code Listing 13: *Implementing Regularized Decision Tree*

This regularization improved the model's accuracy to **0.8181818181818182**.

After selecting the best models, various tests were conducted using different hyperparameters and two distinct datasets: one where passengers with null data were removed, and another where null values were predicted using the k-nearest neighbors algorithm to avoid eliminating those data points. For each model, three tests were performed with different parameters to achieve the best performance.

The dependent and independent variables of the dataset were determined, as well as the training and testing data. Next, an array of models was created to store the models, which were then used in a loop to prepare, train, and predict with each of the imported models. Additionally, a precision analysis was conducted using the `accuracy_score` library. Below are the scores of the models:

Model	**Accuracy**
K-Neighbors Classifier (KNN)	0.643357
Support Vector Classifier (SVC)	0.671329
Logistic Regression (LR)	0.755245
Decision Tree Classifier (DT)	0.713287
Gaussian NB (GNB)	0.755245
Random Forest Classifier (RF)	0.769231
Gradient Boosting Classifier (GB)	0.783217
Multi-Layer Perceptron Classifier (MLP)	0.762238

Table 1: *Accuracy of the Models*

It can be observed that the models with the best accuracy are: Gradient Boosting, Multi-Layer Perceptron, and Decision Tree Classifier. These models will be used for further analysis and prediction. However, some parameter adjustments will be made to improve the prediction results.

Once the best models were identified, three tests were performed per learning model for each dataset. The best accuracy results obtained for each model are presented below:

Model	**Removing Null Values**	**Preserving Null Values**
Gradient Boosting	0.82	0.78
Xtreme Gradient Boosting	0.84	0.80
Multi-Layer Perceptron	0.76	0.79
Decision Tree	0.82	0.84

Table 2: *Comparison of Model Accuracy with and without Null Values*

4.8 Analysis and Observations

Specifically, in Gradient Boosting, we selected the best hyperparameters and performed a significance analysis of the input variables. This analysis provided the following graph, which indicates that the `embarked` variable is not necessary, as it does not appear to be related to the survival outcomes.

We observed that the results differ depending on the dataset used. However, there is no clear pattern indicating that one dataset is consistently better than the other. Nevertheless, in most models, the dataset where null values were preserved (and predicted) showed higher model accuracy. Additionally, it is worth noting that the Decision Tree Classifier demonstrated the highest accuracy among all classifiers.

4.9 Conclusion and Model Performance Summary

Throughout this phase, an exhaustive process of selection, implementation, and comparison of machine learning models was carried out with the aim of predicting the survival of Titanic passengers. A variety of classification algorithms were studied and evaluated in different configurations to determine which ones offer the best performance in terms of evaluation metrics.

Among the analyzed models, two approaches stood out: the **Xtreme Gradient Boosting (XGBoost)** algorithm and the use of neural networks through the **Multi-Layer Perceptron Classifier (MLP)**. Both methods demonstrated promising results in terms of accuracy, recall, and F1-Score. The XGBoost algorithm, which implements a tree-based decision system, excelled due to its ability to handle complex datasets and generate precise predictions. In contrast, the MLP neural network exhibited a strong capacity to learn patterns within the data and adapt to non-linear relationships, contributing to its high performance.

It is important to highlight that the best model performances were observed when working with datasets that did not contain null values. Removing records with missing data allowed the models to focus on meaningful relationships within the available data, resulting in more accurate predictions. Furthermore, various hyperparameter values and regularization techniques were experimented with. Constant iterations helped identify optimal combinations that maximized the accuracy and generalization of the models.

5 Conclusion

This study successfully demonstrated the importance of systematic model selection, hyperparameter tuning, and dataset preparation in achieving high-performing machine learning models. The analysis highlighted that the choice of algorithms and handling of missing data play a crucial role in improving model accuracy.

The results show that both tree-based models like XGBoost and neural network-based approaches like MLP can achieve high accuracy depending on their configurations. This highlights the need for a tailored approach to model selection based on the dataset's characteristics and the problem at hand. Future work may explore further improvements by incorporating ensemble methods or advanced neural network architectures to enhance prediction robustness and scalability.

6 Annexes

The code for this project is implemented in a Google Colab notebook. You can access the notebook using the following link:

Click here to view the Google Colab Notebook

References

[1] J.H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine.* The Annals of Statistics, 29(5), 2001.

[2] T. Chen and C. Guestrin. *XGBoost: A Scalable Tree Boosting System.* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785 and 794, 2016.

[3] L. Breiman. *Random Forests.* Machine Learning, 45(1), 5-32, 2001.

[4] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning representations by back-propagating errors.* Nature, 323, 533-536, 1986.

[5] M. Sokolova and G. Lapalme. *A systematic analysis of performance measures for classification tasks.* Information Processing and Management, 45(4), 427-437, 2009.

[6] R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data.* John Wiley & Sons, 1987.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. Recuperado de `https://www.deeplearningbook.org`

[8] F. Pedregosa et al. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825-2830, 2011.

[9] Kaggle. *Titanic - Machine Learning from Disaster.* Recuperado de `https://www.kaggle.com/competitions/titanic`

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009. Recuperado de `https://web.stanford.edu/~hastie/ElemStatLearn/`

[11] K.P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012.

[12] C.M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[13] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770-778, 2016.

[14] Y. Zhang and Q. Yang. *A Survey on Multi-Task Learning.* IEEE Transactions on Knowledge and Data Engineering, 34(1), 2021.

[15] UCI Machine Learning Repository. *Automobile.* Recuperado de `https://archive.ics.uci.edu/dataset/10/automobile`

[16] R. Kohavi. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.* International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[17] F. Chollet. *Keras.* GitHub repository, 2015. Recuperado de `https://github.com/keras-team/keras`