Faculty of Engineering, Environment and Computing
6006CEM Machine Learning and Related Applications
Coursework Template

Comparing Different Machine Learning Models for Classification

Coventry GitHub Repository URL or **Coventry OneDrive URL** (mandatory):

https://github.coventry.ac.uk/tuna4/13963000-AT-s1

Name: Aung Myo Tun
Student ID: 13963000
Course: Bachelor of Computer Science (Year 3)

Table of contents

Contents

1) Literature Review: Chatbots and Conversational AI in NLP 3
1.1) Introduction 3
1.2) The Early Stages of Development 3
1.3) The role of Machine Learning in chatbots. 3
1.4) The Rise of Transformer Models 3
1.5) Training Chatbots 4
1.6) Applications of Chatbots 4
1.7) Challenges in Chatbot Development 4
1.8) Future Directions 5
1.9) Conclusion 5
2) Approach, Analysis, and Evaluation of Machine Learning Models for Classification 5
2.1) Introduction 5
2.2) Comparing the Approaches and Results of Existing Work 6
2.3) Importing Libraries and Loading the Dataset 6
2.4) Inspecting the Dataset 7
2.5) Cleaning the Data 8
2.6) Training and Testing Machine Learning Models 9
2.7) Evaluates the Performance of the Models 10
2.8) Hyperparameter Tuning for Model Optimisation 12
2.9) Hyperparameter Tuning and Model Evaluation 12
2.10) Conclusion 14

Academic Report

1) Literature Review: Chatbots and Conversational AI in NLP

1.1) Introduction

The software programs, in the likeness of human conversation using either text or voice interactions, have found their application in various fields, such as customer support, healthcare, and education. In these areas, the role of a chatbot is performed by quickly and helpfully answering questions. Over recent years, new breakthroughs in machine learning and NLP have granted them such intelligence and conversational capabilities to engage in meaningful and contextual conversations (Al-Amin et al., 2024).

1.2) The Early Stages of Development

The journey of chatbots began in the 1960s with ELIZA, one of the earliest conversational programs. ELIZA worked by using basic patterns and rules to respond to user input, but it did not actually "understand" the text it was processing (Ciesla, 2024). Later, more advanced systems like PARRY and ALICE used rule-based approaches with predefined scripts to produce answers. These worked fine for simple interactions but were bound due to limitations in handling complex language and context (Yellowfin BI, 2023).

1.3) The role of Machine Learning in chatbots.

Machine learning signified a great improvement in the development of chatbots. While earlier versions relied on fixed scripts, machine learning gave them the ability to learn from data. Supervised learning, in which models are trained on labeled examples, enabled the chatbot to identify the user's intent and provide more accurate responses (Takyar, 2020).

Additionally, the use of Natural Language Processing (NLP) helped chatbots process human language more effectively. For example, sentiment analysis can help a chatbot understand if a customer is upset or happy and named entity recognition can allow a chatbot to pick out important details like dates or names in a conversation (Dam et al., 2024).

1.4) The Rise of Transformer Models

The first significant stride in the realm of chatbots came with transformer-based models. These deep learning models, one example being the Generative Pre-trained Transformer from OpenAI, revolutionized the generation of text by chatbots. Unlike their predecessors, transformers have the capability to process and understand the whole context of a conversation, which makes their responses much more natural and relevant (Dam et al., 2024).

For instance, GPT-3 and GPT-4 are large language models trained on massive amounts of data. They can hold complex conversations and adapt to a wide range of topics. OpenAI's ChatGPT, based on this technology, is a good example of how transformers make chatbots capable of answering questions, solving problems, or even engaging in creative tasks like writing (Al-Amin et al., 2024).

1.5) Training Chatbots

Modern chatbots are trained using a two-step process:

1. **Pre-training**: The model learns from a wide range of text data, such as books, articles, and websites. During this phase, it learns general language patterns and facts.
2. **Fine-tuning**: The model is adjusted to perform specific tasks, such as answering customer queries or providing coding assistance. This phase ensures the chatbot's responses are more accurate and tailored to its use case (Ciesla, 2024).

1.6) Applications of Chatbots

Chatbots are now used in various industries to improve efficiency and provide better services:

- **Customer Support**: Chatbots can handle frequent questions, helping reduce waiting times and improving customer satisfaction (Takyar, 2020).
- **Healthcare**: They assist patients by scheduling appointments, offering mental health support, or providing symptom checks (Dam et al., 2024).
- **Education**: Chatbots help students by answering questions, offering tutoring, or creating personalized study plans (Al-Amin et al., 2024).
- **Programming**: Tools like GitHub Copilot assist developers by suggesting code snippets and debugging solutions (Ciesla, 2024).

1.7) Challenges in Chatbot Development

Challenges in Developing Chatbots
Despite their growth, chatbots are still facing quite a few of these challenges:

Understanding Context: It's difficult for chatbots to fully retain and understand the context of long conversations.

Response Bias: Chatbots may sometimes project biases present in the material they were trained on, leading to responses that are inappropriate or unfair.

Resource Requirements: Training advanced models like GPT-4 requires significant computational power, which can be expensive and environmentally impactful (Dam et al., 2024).

Addressing these challenges is essential to ensure chatbots remain useful, fair, and accessible.

1.8) Future Directions

In the future, researchers are going to improve the intelligence and efficiency of the chatbots, enhance their ability to understand the context-let alone in long conversations-to better follow and remember interactions. Also, developers are working on how to reduce biases in chatbot responses in order for them to be more ethical and inclusive. Future chatbots are likely to also use multimodal interaction, that is, combining text, voice, and images to offer a more natural and intuitive means of communication means. Efforts are being made to optimize resources by creating models that require less computational power, making them more accessible to smaller organizations (Al-Amin et al., 2024).

1.9) Conclusion

From simple rule-based systems to advanced deep learning-powered conversational agents, chatbots have come a long way. While they are already transforming industries like customer service and healthcare, challenges such as bias and resource constraints still need to be addressed. As these technologies continue to evolve, chatbots are set to play an even bigger role in our daily lives, enabling smoother and more intelligent interactions.

2) Approach, Analysis, and Evaluation of Machine Learning Models for Classification

2.1) Introduction

This project applies machine learning to solve a real-world classification problem. The dataset consists of 7027 entries and 65 features; some of them have missing values, hence indicating the complexity of working with real-world data. The target variable represents binary outcomes, where values correspond to one of two classes. Although the exact context is unknown, the dataset could be applied to tasks such as financial risk prediction or operational performance analysis (GeeksforGeeks, 2023). This project involves cleaning and preprocessing the data, training machine learning models, and assessing their performance.

Three models were chosen: Random Forest, Logistic Regression, and SVM, as among the most used in various classification tasks due to efficiency and reliability. According to GeeksforGeeks, 2023, hyperparameter tuning was done using GridSearchCV further for performance improvement. Metric evaluation includes accuracy, precision, recall, F1-score, and ROC-AUC; this is a guarantee of quality analysis, especially in approaches to class imbalance.
This report will cover: Following is what this report will cover:

1. Methods for data preprocessing.
2. How the best models were developed and tuned.

3. Comparison of results for each model.
4. Conclusions with suggestions for future improvements.

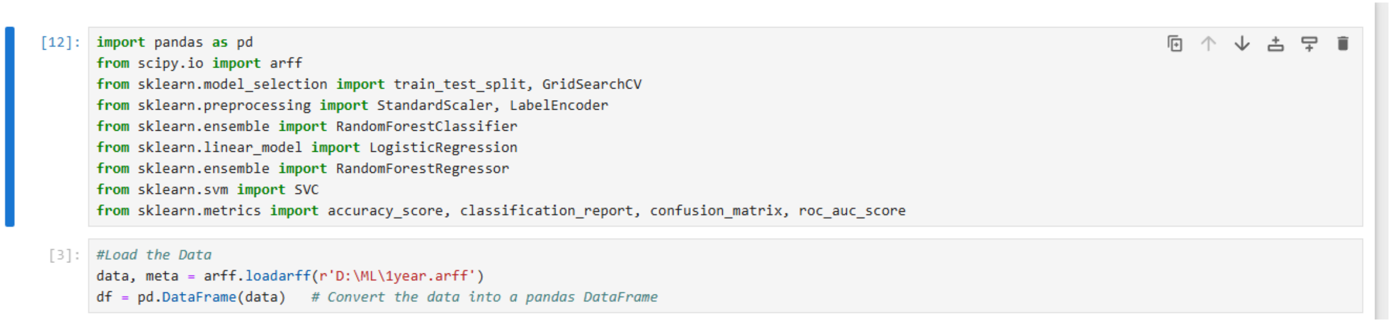
2.2) Comparing the Approaches and Results of Existing Work

In classification problems, popular machine learning models include Logistic Regression, Random Forest, and Support Vector Machines (SVM). Logistic Regression is a simple and interpretable model, often used as a baseline for comparison (GeeksforGeeks, 2023). Random Forest is reliable and performs well on large datasets and complex relationships, making it a robust choice (GeeksforGeeks, 2023). SVM excels at defining clear decision boundaries, which is useful for binary classification tasks (Scikit-learn, 2023).

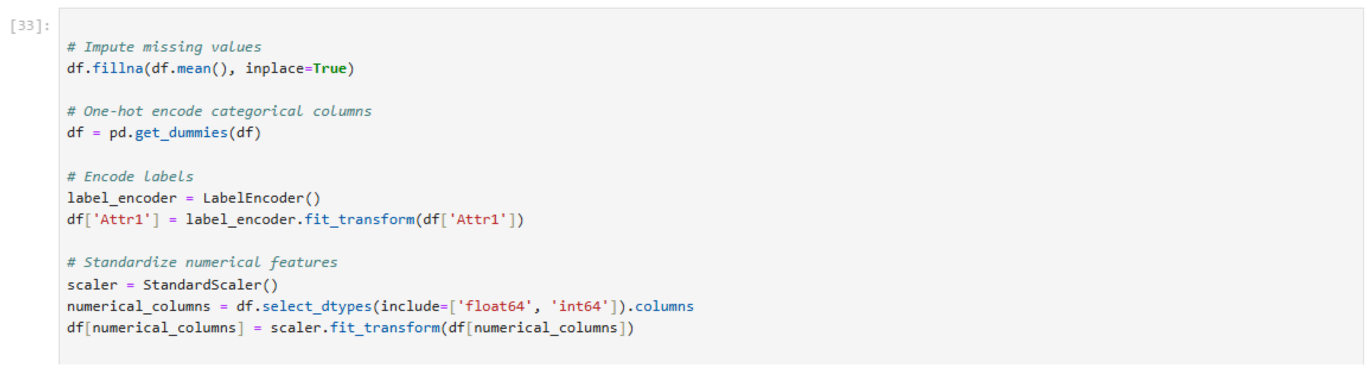
This project uses these models along with hyperparameter tuning to improve their performance. Unlike many typical implementations, this project focuses on metrics such as recall and F1-score, which are particularly valuable for evaluating models on imbalanced datasets (FutureMachineLearning, 2023).

2.3) Importing Libraries and Loading the Dataset

In this part of the code, we import the necessary libraries to handle the dataset and build machine learning models. The dataset, stored in .arff format, is loaded using arff.loadarff and then converted into a pandas DataFrame for easy processing. Key libraries like pandas are used for data handling, while machine learning tools such as RandomForestClassifier, LogisticRegression, and SVC are imported for model training. This step sets up the project by preparing the tools and loading the dataset into a usable format.



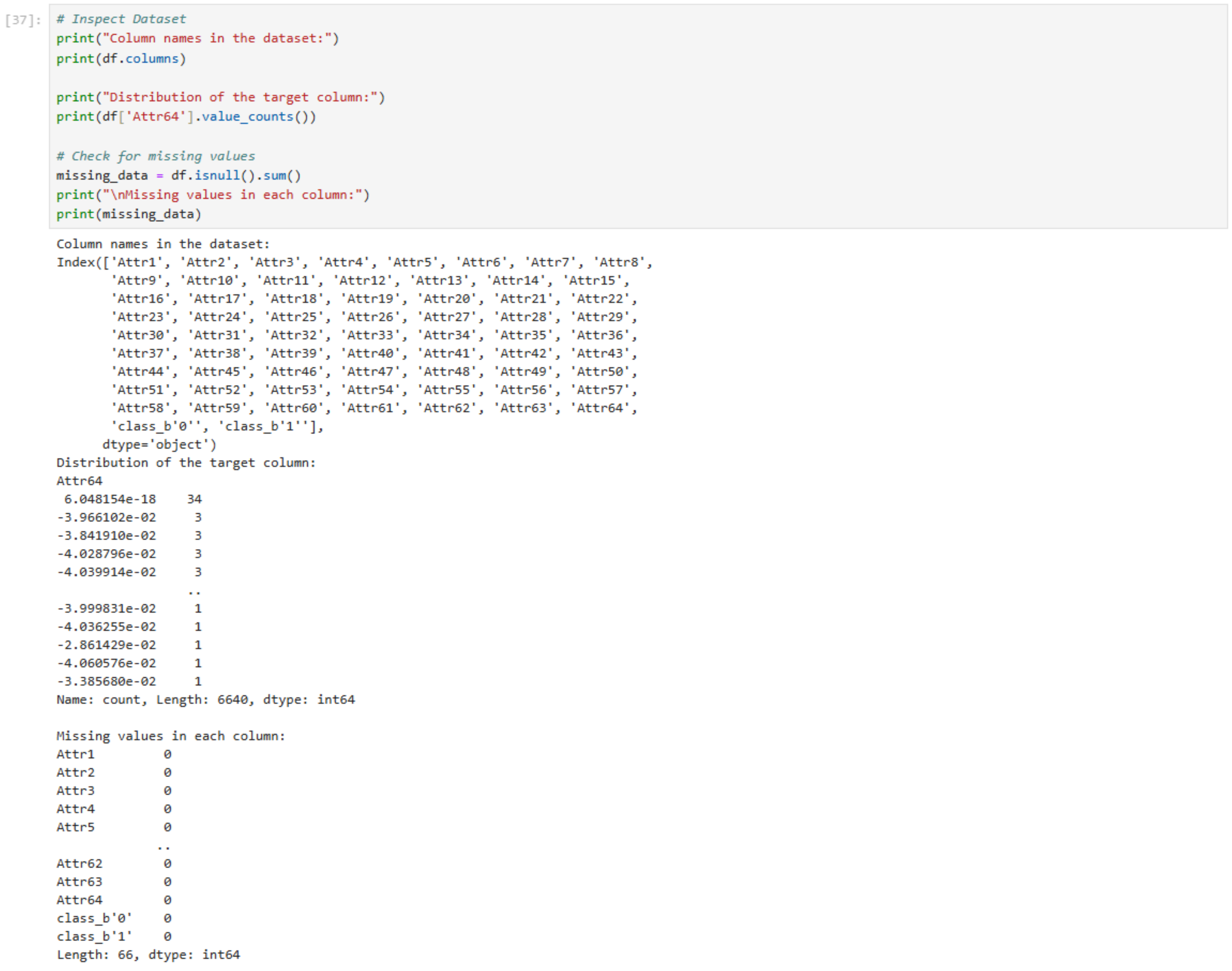
```
[12]: import pandas as pd
      from scipy.io import arff
      from sklearn.model_selection import train_test_split, GridSearchCV
      from sklearn.preprocessing import StandardScaler, LabelEncoder
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.linear_model import LogisticRegression
      from sklearn.ensemble import RandomForestRegressor
      from sklearn.svm import SVC
      from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score

 [3]: #Load the Data
      data, meta = arff.loadarff(r'D:\ML\1year.arff')
      df = pd.DataFrame(data)    # Convert the data into a pandas DataFrame
```



```
[33]:
# Impute missing values
df.fillna(df.mean(), inplace=True)

# One-hot encode categorical columns
df = pd.get_dummies(df)

# Encode labels
label_encoder = LabelEncoder()
df['Attr1'] = label_encoder.fit_transform(df['Attr1'])

# Standardize numerical features
scaler = StandardScaler()
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

2.4) Inspecting the Dataset

The data has been checked to understand the structure and make sure that it's ready for use. It has 66 columns, one of which is Attr64, presumably the target variable. Upon investigating Attr64, it became obvious that it contains a lot of unique numbers, indicating it is a continuous variable and not pre-prepared for classification. This will later need to be changed to work with the models. The dataset has no missing values, so it is complete and prepared for further steps that include preprocessing and modelling.

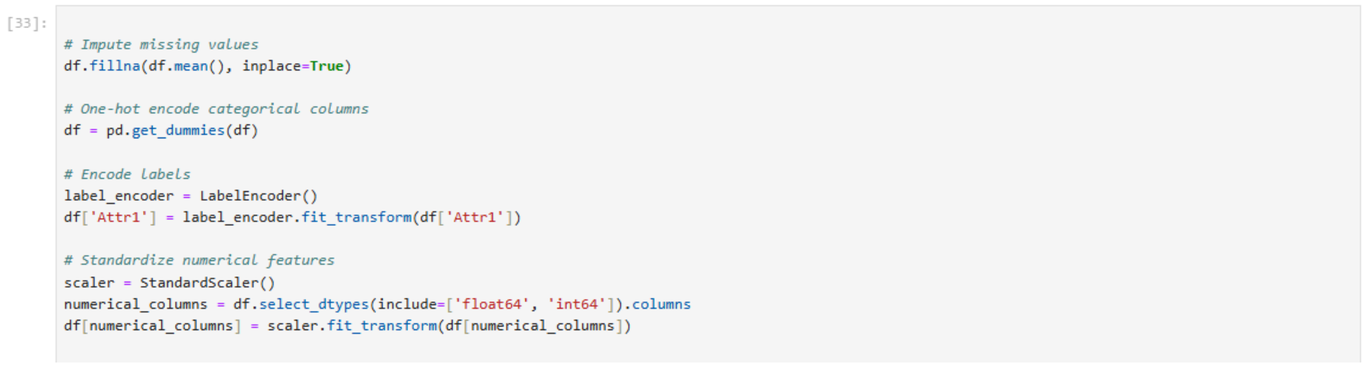


```
[37]: # Inspect Dataset
      print("Column names in the dataset:")
      print(df.columns)

      print("Distribution of the target column:")
      print(df['Attr64'].value_counts())

      # Check for missing values
      missing_data = df.isnull().sum()
      print("\nMissing values in each column:")
      print(missing_data)
```

```
Column names in the dataset:
Index(['Attr1', 'Attr2', 'Attr3', 'Attr4', 'Attr5', 'Attr6', 'Attr7', 'Attr8',
       'Attr9', 'Attr10', 'Attr11', 'Attr12', 'Attr13', 'Attr14', 'Attr15',
       'Attr16', 'Attr17', 'Attr18', 'Attr19', 'Attr20', 'Attr21', 'Attr22',
       'Attr23', 'Attr24', 'Attr25', 'Attr26', 'Attr27', 'Attr28', 'Attr29',
       'Attr30', 'Attr31', 'Attr32', 'Attr33', 'Attr34', 'Attr35', 'Attr36',
       'Attr37', 'Attr38', 'Attr39', 'Attr40', 'Attr41', 'Attr42', 'Attr43',
       'Attr44', 'Attr45', 'Attr46', 'Attr47', 'Attr48', 'Attr49', 'Attr50',
       'Attr51', 'Attr52', 'Attr53', 'Attr54', 'Attr55', 'Attr56', 'Attr57',
       'Attr58', 'Attr59', 'Attr60', 'Attr61', 'Attr62', 'Attr63', 'Attr64',
      'class_b'0'', 'class_b'1''],
      dtype='object')
Distribution of the target column:
Attr64
 6.048154e-18    34
-3.966102e-02     3
-3.841910e-02     3
-4.028796e-02     3
-4.039914e-02     3
                 ..
-3.999831e-02     1
-4.036255e-02     1
-2.861429e-02     1
-4.060576e-02     1
-3.385680e-02     1
Name: count, Length: 6640, dtype: int64

Missing values in each column:
Attr1         0
Attr2         0
Attr3         0
Attr4         0
Attr5         0
             ..
Attr62        0
Attr63        0
Attr64        0
class_b'0'    0
class_b'1'    0
Length: 66, dtype: int64
```

2.5) Cleaning the Data

This code block cleans and prepares the data for use in a machine learning model. For every missing value in a dataset, it replaces that with the mean of each column, hence there will not be any lost data. One-hot encoding turns categorical values into numbers; each category is given its binary column. And specifically, Attr1 column has been label-encoded to integers. StandardScaler standardizes all the numerical features so that their distributions have a mean of 0 and a standard deviation of 1. This can help all the features be on the same scale and prepare the data for models sensitive to feature magnitudes.

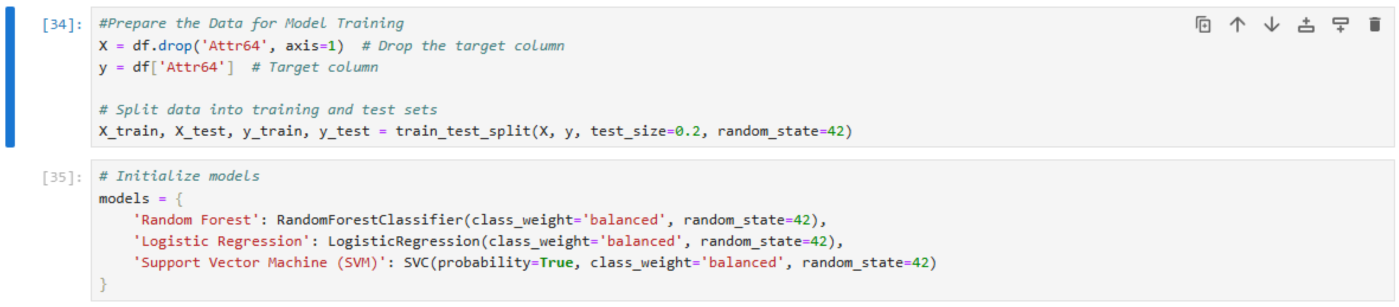

```
[33]:
# Impute missing values
df.fillna(df.mean(), inplace=True)

# One-hot encode categorical columns
df = pd.get_dummies(df)

# Encode labels
label_encoder = LabelEncoder()
df['Attr1'] = label_encoder.fit_transform(df['Attr1'])

# Standardize numerical features
scaler = StandardScaler()
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

2.6) Training and Testing Machine Learning Models

This part of the code prepares the data for training and testing machine learning models and sets up the models to be used.

1. Defining Features (X) and Target (y):
   - The target column Attr64 is separated from the rest of the dataset:
     - X contains all the feature columns except Attr64 (dropped using df.drop).
     - y contains only the target column Attr64.
   - This step ensures that the machine learning models can learn the relationship between the features (X) and the target (y).
2. Splitting the Data:
   - The train_test_split function splits the data into training and testing sets:
     - 80% of the data is used for training (X_train, y_train) to build the models.
     - 20% of the data is used for testing (X_test, y_test) to evaluate the models.
   - The random_state=42 ensures consistent splitting every time the code is run.
3. Setting Up Models:
   - Three machine learning models are initialised:
     - Random Forest: A robust model that builds multiple decision trees and combines their outputs.
     - Logistic Regression: A simple and interpretable model used for binary classification.
     - Support Vector Machine (SVM): A model that finds the optimal boundary to separate classes.
   - All models are set with class_weight='balanced', which helps handle the class imbalance in the target variable by giving more importance to the minority class.

This code splits the data into training and testing sets, making it ready for model training. It also sets up three machine learning models, Random Forest, Logistic Regression, and SVM, ensuring they can handle class imbalance for better performance on the minority class.



```
[34]: #Prepare the Data for Model Training
      X = df.drop('Attr64', axis=1)  # Drop the target column
      y = df['Attr64']  # Target column

      # Split data into training and test sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

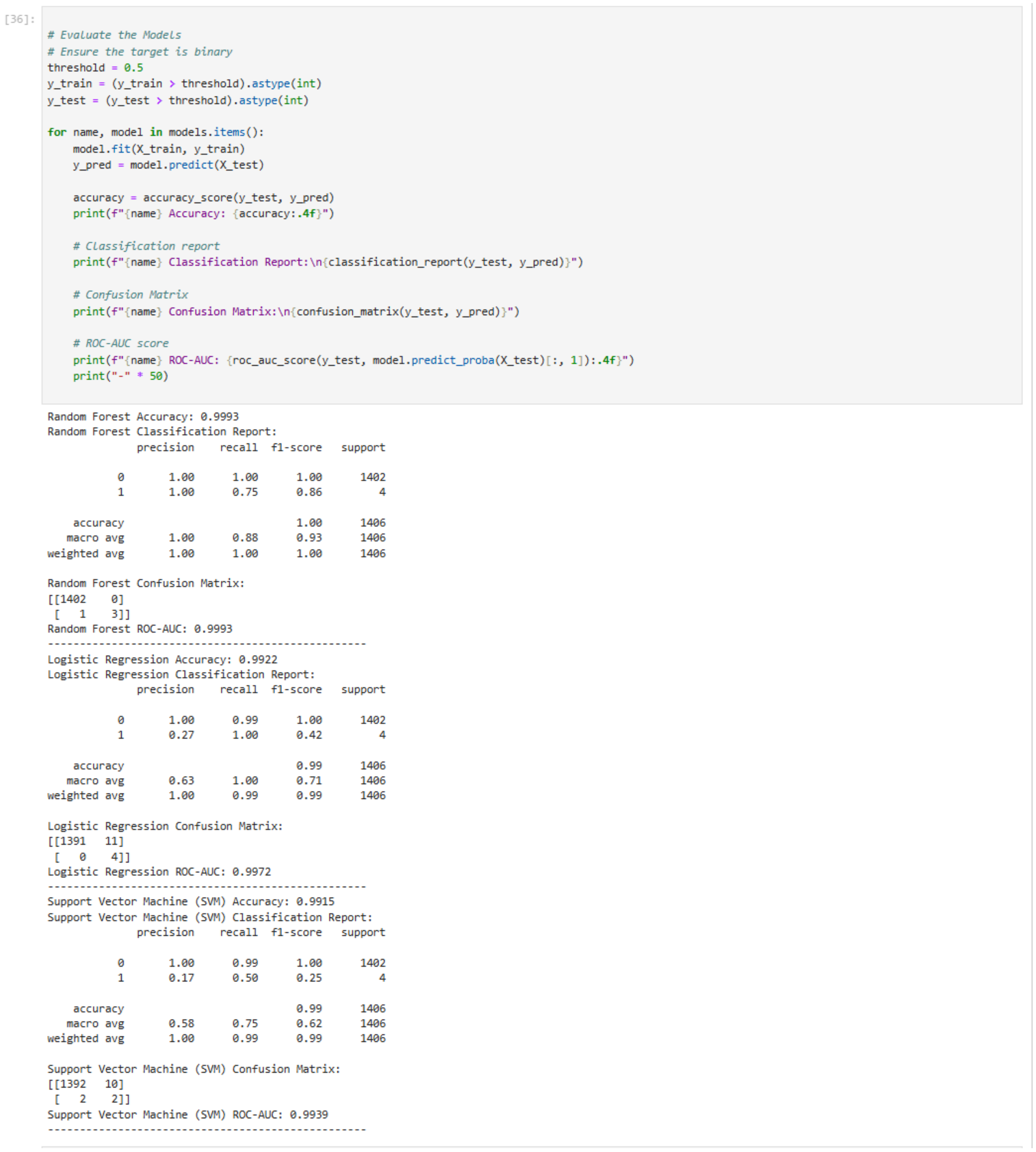
```
[35]: # Initialize models
      models = {
          'Random Forest': RandomForestClassifier(class_weight='balanced', random_state=42),
          'Logistic Regression': LogisticRegression(class_weight='balanced', random_state=42),
          'Support Vector Machine (SVM)': SVC(probability=True, class_weight='balanced', random_state=42)
      }
```

2.7) Evaluates the Performance of the Models

This part of the code evaluates the performance of the models after training them on the dataset. Here's a summary of what happens:

- **Binary Target Conversion**:
  - The target variable (Attr64) is converted into binary using a threshold of 0.5, ensuring it's suitable for classification.
- **Model Training and Evaluation**:
  - Each model (Random Forest, Logistic Regression, and SVM) is trained on the training set and tested on the test set.
  - Metrics like **accuracy**, **precision**, **recall**, **F1-score**, and **ROC-AUC** are calculated to assess the models' performance.
- **Results**:
  - **Random Forest**:
    - Achieved the highest accuracy (99.93%) and performed well across all metrics.
    - Balanced performance for both the majority and minority classes.
  - **Logistic Regression**:
    - Slightly lower accuracy (99.22%) but excellent recall for the minority class.
    - Precision was lower, indicating a trade-off.
  - **SVM**:
    - Good overall accuracy (99.15%) but struggled with recall for the minority class.
    - Shows more misclassifications in the minority class.

The evaluation highlights that Random Forest is the best-performing model, with strong scores across all metrics, while Logistic Regression and SVM show specific strengths and weaknesses.

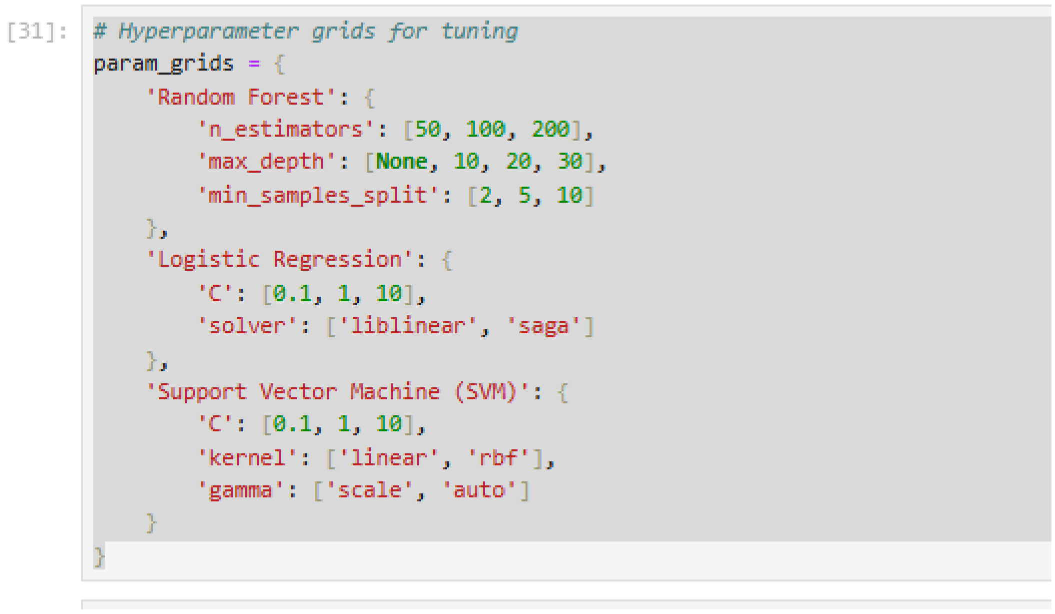


```
[36]:
# Evaluate the Models
# Ensure the target is binary
threshold = 0.5
y_train = (y_train > threshold).astype(int)
y_test = (y_test > threshold).astype(int)

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")

    # Classification report
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred)}")

    # Confusion Matrix
    print(f"{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")

    # ROC-AUC score
    print(f"{name} ROC-AUC: {roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]):.4f}")
    print("-" * 50)
```

```
Random Forest Accuracy: 0.9993
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1402
           1       1.00      0.75      0.86         4

    accuracy                           1.00      1406
   macro avg       1.00      0.88      0.93      1406
weighted avg       1.00      1.00      1.00      1406

Random Forest Confusion Matrix:
[[1402    0]
 [   1    3]]
Random Forest ROC-AUC: 0.9993
--------------------------------------------------
Logistic Regression Accuracy: 0.9922
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1402
           1       0.27      1.00      0.42         4

    accuracy                           0.99      1406
   macro avg       0.63      1.00      0.71      1406
weighted avg       1.00      0.99      0.99      1406

Logistic Regression Confusion Matrix:
[[1391   11]
 [   0    4]]
Logistic Regression ROC-AUC: 0.9972
--------------------------------------------------
Support Vector Machine (SVM) Accuracy: 0.9915
Support Vector Machine (SVM) Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1402
           1       0.17      0.50      0.25         4

    accuracy                           0.99      1406
   macro avg       0.58      0.75      0.62      1406
weighted avg       1.00      0.99      0.99      1406

Support Vector Machine (SVM) Confusion Matrix:
[[1392   10]
 [   2    2]]
Support Vector Machine (SVM) ROC-AUC: 0.9939
--------------------------------------------------
```

2.8) Hyperparameter Tuning for Model Optimisation

This part of the code sets up hyperparameter grids for tuning the models. For Random Forest, it tests different values for the number of trees, tree depth, and sample splitting. For Logistic Regression, it adjusts the regularisation strength and solver type. For SVM, it explores values for regularisation, kernel type, and gamma. These grids help optimise the models' performance by selecting the best combination of hyperparameters.



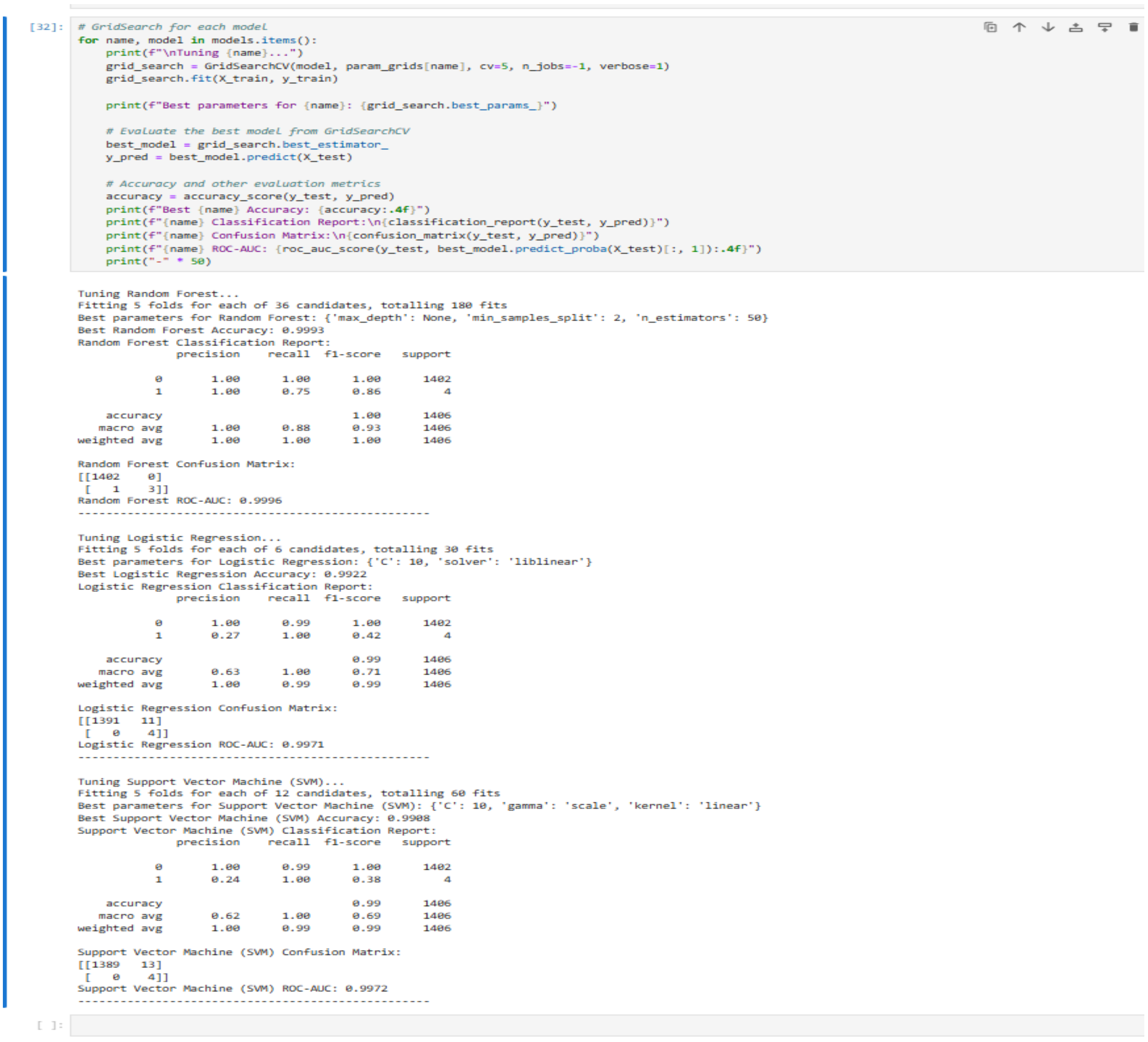
```
[31]: # Hyperparameter grids for tuning
param_grids = {
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    },
    'Logistic Regression': {
        'C': [0.1, 1, 10],
        'solver': ['liblinear', 'saga']
    },
    'Support Vector Machine (SVM)': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    }
}
```

2.9) Hyperparameter Tuning and Model Evaluation

This section of the code tunes the hyperparameters of each model (Random Forest, Logistic Regression, and SVM) using GridSearchCV to find the best settings for each. It then evaluates the performance of the models on the test data.

- **Random Forest**:
  - It achieved an accuracy of 99.93%, with a good recall (75%) for the minority class. The model also had a high ROC-AUC score of 0.9996, showing that it could effectively separate the two classes.
- **Logistic Regression**:
  - It achieved 99.22% accuracy, but its recall for the minority class was perfect (100%), while precision was lower (27%). This suggests that it focuses more on correctly identifying the minority class, even at the cost of false positives.

- **Support Vector Machine (SVM)**:
    - SVM achieved 99.08% accuracy. Like Logistic Regression, it had perfect recall for the minority class, but its precision was low (24%). The ROC-AUC score of 0.9972 indicates good overall performance, but there's room for improvement in precision.



```
[32]: # GridSearch for each model
for name, model in models.items():
    print(f"\nTuning {name}...")
    grid_search = GridSearchCV(model, param_grids[name], cv=5, n_jobs=-1, verbose=1)
    grid_search.fit(X_train, y_train)

    print(f"Best parameters for {name}: {grid_search.best_params_}")

    # Evaluate the best model from GridSearchCV
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Accuracy and other evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Best {name} Accuracy: {accuracy:.4f}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred)}")
    print(f"{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")
    print(f"{name} ROC-AUC: {roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1]):.4f}")
    print("-" * 50)
```

```
Tuning Random Forest...
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best parameters for Random Forest: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
Best Random Forest Accuracy: 0.9993
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1402
           1       1.00      0.75      0.86         4

    accuracy                           1.00      1406
   macro avg       1.00      0.88      0.93      1406
weighted avg       1.00      1.00      1.00      1406

Random Forest Confusion Matrix:
[[1402    0]
 [   1    3]]
Random Forest ROC-AUC: 0.9996
--------------------------------------------------

Tuning Logistic Regression...
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for Logistic Regression: {'C': 10, 'solver': 'liblinear'}
Best Logistic Regression Accuracy: 0.9922
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1402
           1       0.27      1.00      0.42         4

    accuracy                           0.99      1406
   macro avg       0.63      1.00      0.71      1406
weighted avg       1.00      0.99      0.99      1406

Logistic Regression Confusion Matrix:
[[1391   11]
 [   0    4]]
Logistic Regression ROC-AUC: 0.9971
--------------------------------------------------

Tuning Support Vector Machine (SVM)...
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters for Support Vector Machine (SVM): {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Best Support Vector Machine (SVM) Accuracy: 0.9908
Support Vector Machine (SVM) Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1402
           1       0.24      1.00      0.38         4

    accuracy                           0.99      1406
   macro avg       0.62      1.00      0.69      1406
weighted avg       1.00      0.99      0.99      1406

Support Vector Machine (SVM) Confusion Matrix:
[[1389   13]
 [   0    4]]
Support Vector Machine (SVM) ROC-AUC: 0.9972
--------------------------------------------------
```

```
[ ]:
```

2.10) Conclusion

Random Forest performed the best overall, with high accuracy, recall, and a strong ability to distinguish between classes. It handled the class imbalance well. Logistic Regression was good at identifying the minority class but struggled with precision, meaning it had more false positives. SVM also identified the minority class correctly but had low precision, which is not ideal for some applications. In conclusion, Random Forest was the most reliable model for this classification task, with the best balance between performance and precision. Both Logistic Regression and SVM showed strengths in recall for the minority class but could benefit from improved precision.

Bibliography

- Al-Amin, M., Ali, M. S., Salam, A., Khan, A., Ali, A., Ullah, A., Alam, M. N., & Chowdhury, S. K. (2024). History of generative Artificial Intelligence (AI) chatbots: past, present, and future development. *arXiv preprint arXiv:2402.05122*. Available at: https://arxiv.org/abs/2402.05122 (Accessed: 5 December 2024).
- Ciesla, K. (2024). *The Book of Chatbots: From ELIZA to ChatGPT*. Springer. Available at: https://link.springer.com/book/10.1007/978-3-031-51004-5 (Accessed: 5 December 2024).
- Dam, S. K., Hong, C. S., Qiao, Y., & Zhang, C. (2024). A Complete Survey on LLM-based AI Chatbots. *arXiv preprint arXiv:2406.16937*. Available at: https://arxiv.org/abs/2406.16937 (Accessed: 5 December 2024).
- Takyar, A. (2020). AI Chatbots – Challenges and Opportunities. *DZone*. Available at: https://dzone.com/articles/everything-around-ai-chatbots-challenges-and-oppor (Accessed: 5 December 2024).

- Yellowfin BI (2023). The History of Chatbots: A Timeline of Conversational AI. Available at: https://www.yellowfinbi.com/history-of-chatbots-timeline-of-conversational-ai (Accessed: 5 December 2024).GeeksforGeeks (2023). *Top 6 Machine Learning Algorithms for Classification*. Available at: https://www.geeksforgeeks.org/top-6-machine-learning-algorithms-for-classification/ (Accessed: 5 December 2024).
- FutureMachineLearning (2023). *Understanding the F1 Score: A Guide for Imbalanced Datasets*. Available at: https://futuremachinelearning.org/understanding-the-f1-score-a-guide-for-imbalanced-datasets/ (Accessed: 5 December 2024).
- Scikit-learn (2023). *Support Vector Machines — scikit-learn documentation*. Available at: https://scikit-learn.org/stable/modules/svm.html (Accessed: 5 December 2024).
- GeeksforGeeks (2023). *Top 6 Machine Learning Algorithms for Classification*. Available at: https://www.geeksforgeeks.org/top-6-machine-learning-algorithms-for-classification/ (Accessed: 5 December 2024).

Appendix A

Appendix B

Source Code

```
import pandas as pd
from scipy.io import arff
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
roc_auc_score

#Load the Data
data, meta = arff.loadarff(r'D:\ML\1year.arff')
df = pd.DataFrame(data)   # Convert the data into a pandas DataFrame

# Inspect Dataset
print("Column names in the dataset:")
print(df.columns)

print("Distribution of the target column:")
print(df['Attr64'].value_counts())

# Check for missing values
missing_data = df.isnull().sum()
print("\nMissing values in each column:")
print(missing_data)

# Impute missing values
df.fillna(df.mean(), inplace=True)

# One-hot encode categorical columns
```

```
df = pd.get_dummies(df)

# Encode labels
label_encoder = LabelEncoder()
df['Attr1'] = label_encoder.fit_transform(df['Attr1'])

# Standardize numerical features
scaler = StandardScaler()
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

#Prepare the Data for Model Training
X = df.drop('Attr64', axis=1)  # Drop the target column
y = df['Attr64']  # Target column

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
models = {
    'Random Forest': RandomForestClassifier(class_weight='balanced', random_state=42),
    'Logistic Regression': LogisticRegression(class_weight='balanced', random_state=42),
    'Support Vector Machine (SVM)': SVC(probability=True, class_weight='balanced',
random_state=42)
}


# Evaluate the Models
# Ensure the target is binary
threshold = 0.5
y_train = (y_train > threshold).astype(int)
y_test = (y_test > threshold).astype(int)

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")
    # Classification report
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred)}")

    # Confusion Matrix
    print(f"{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")

    # ROC-AUC score
    print(f"{name} ROC-AUC: {roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]):.4f}")
    print("-" * 50)


# Hyperparameter grids for tuning
param_grids = {
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    },
    'Logistic Regression': {
        'C': [0.1, 1, 10],
        'solver': ['liblinear', 'saga']
    },
    'Support Vector Machine (SVM)': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    }
}

# GridSearch for each model
for name, model in models.items():
    print(f"\nTuning {name}...")
    grid_search = GridSearchCV(model, param_grids[name], cv=5, n_jobs=-1, verbose=1)
    grid_search.fit(X_train, y_train)

    print(f"Best parameters for {name}: {grid_search.best_params_}")

    # Evaluate the best model from GridSearchCV
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)
    # Accuracy and other evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Best {name} Accuracy: {accuracy:.4f}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred)}")
    print(f"{name} Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")
    print(f"{name} ROC-AUC: {roc_auc_score(y_test, best_model.predict_proba(X_test)[:,
1]):.4f}")
    print("-" * 50)
```