THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Machine Learning I DATS 6202

(MS In Data Science)

Group Report

Group 3

Rainfall Prediction

Instructor: Amir Jafari

Authors:

Shreyas Sunku Padmanabha

Date: 05/01/2023

TABLE OF CONTENTS

1: Introduction

The code mentioned is concerned with building a predictive model that can classify whether it will rain on a particular day or not. The model uses several features such as the location, date, humidity, wind speed, and other related factors to make predictions. The main goal is to design and train a model that can accurately predict the occurrence of rain based on these features.

I worked on this code has experimented with various classification models to identify the most suitable one for this task. Different machine learning algorithms have been tested, and the performance metrics of each model have been measured to select the best one. These metrics include accuracy, precision, recall, and F1 score, which indicate how well the model is performing.

I also applied techniques such as PCA (Principal Component Analysis) to reduce the number of features in the dataset, which can help to improve the performance of the model. Furthermore, cross-fold validation has been performed to ensure that the model's performance is not overfitting to the training data and can generalize well to new, unseen data.

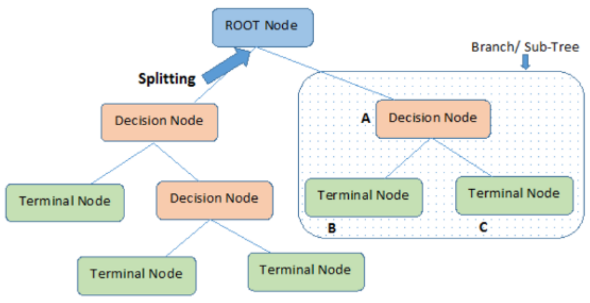
In summary, the code builds a predictive model to determine whether it will rain on a particular day. The model uses several features, and the individual has experimented with various classification models to identify the most suitable one for this task. Techniques like PCA and cross-fold validation have been applied to improve the model's performance metrics.

2: Background Information: Machine Learning Algorithms

Machine learning algorithms use statistical models and algorithms to identify patterns in data and make predictions or decisions based on that data. As we have a binary classification problem, we have used the following methods.

2.1. Decision Tree Classifier

Decision Tree Classifier is a machine learning algorithm used for classification tasks. It partitions the feature space recursively into subsets based on input features and assigns a class label to each leaf node of the tree. The algorithm determines the best split for each node by maximizing a measure of purity. The decision tree can be graphically represented as a tree structure with each internal node representing a split on an input feature and each leaf node representing a class label. It can handle both categorical and numerical data but is prone to overfitting.

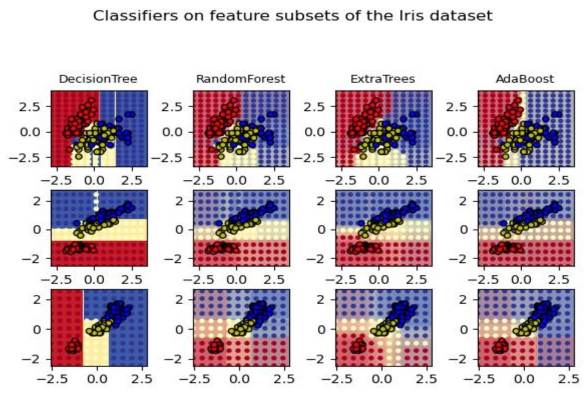


2.2. Random Forest Classifier

Random Forest Classifier is an ensemble machine learning algorithm used for classification tasks. It works by combining multiple decision trees, each trained on a different subset of the training data and a different subset of the input features. The algorithm determines the class of a new data point by aggregating the predictions of all the trees in the forest.

Random Forest Classifier is a powerful algorithm that can handle high-dimensional data and is less prone to overfitting than Decision Tree Classifier. However, it can be computationally expensive and difficult to interpret.

The random forest can be represented graphically as a collection of decision trees, with each tree trained on a different subset of the training data and input features.



2.3. Naïve Bayes

Naive Bayes is a popular classification algorithm used in machine learning. It is based on Bayes' theorem, which provides a way to calculate the probability of a hypothesis based on the probability of its evidence. In Naive Bayes, we assume that the predictors (also known as features or attributes) are conditionally independent given the class variable. This assumption simplifies the probability calculations and makes the algorithm computationally efficient.

The algorithm works by first training a model on a labeled dataset, where each instance in the dataset is labeled with its corresponding class. During training, the algorithm estimates the probabilities of each feature given each class. Then, when presented with a new instance, the algorithm calculates the probability of each class given the values of its features. The class with the highest probability is then assigned as the predicted class for the instance.

Naive Bayes is particularly useful when working with high-dimensional datasets, where the number of features is large. It has also been shown to perform well even when the independence assumption is not strictly true, making it a robust and versatile algorithm.

Overall, Naive Bayes is a simple yet effective classification algorithm that is widely used in various applications such as text classification, spam filtering, and sentiment analysis.

2.4. Principal Component Analysis

PCA (Principal Component Analysis) is a statistical method used to reduce the dimensionality of a large dataset while preserving the most important information. It transforms the original variables into a set of uncorrelated variables called principal components. The first principal component captures the maximum amount of variation in the dataset, followed by the second principal component, and so on. By retaining only the top principal components, which explain the majority of the variability in the data, PCA can simplify the analysis and improve the computational efficiency. PCA is commonly used in fields such as image and signal processing, genetics, and finance.

2.5. K-fold Cross Validation

K-fold cross-validation is a technique used to evaluate the performance of a machine learning model. The data is divided into K equally sized partitions, or "folds". The model is trained K times, each time using a different fold as the validation set and the remaining K-1 folds as the training set. The performance of the model is then evaluated by averaging the performance of each of the K models.

K-fold cross-validation is commonly used to estimate the performance of a model when data is limited or to tune hyperparameters. It helps to mitigate the risk of overfitting by reducing the variance of the estimated performance. By using multiple folds, the model is trained on more data, and the performance estimate is based on a more representative sample of the data.

Overall, K-fold cross-validation is a valuable technique for assessing the generalization performance of a machine learning model and can help to improve the reliability and robustness of the model.

3: Work/ Code Description

The machine learning part of the code starts with the models function. This function takes the preprocessed training and testing data (X_train, X_test, y_train, and y_test) as input, along with a list of labels for the target variable. The function then fits and evaluates different machine learning models using the training data.

For each model, the function calculates and prints various performance metrics including accuracy, precision, recall, F1-score, and ROC AUC score. It also generates and displays a confusion matrix plot using the plot_confusion_matrix function from the tools module.

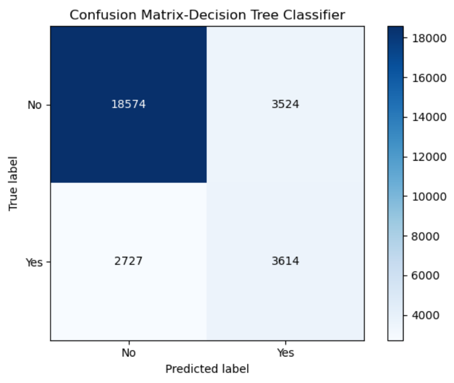
After all three models have been evaluated, the function returns a pandas DataFrame with the performance metrics for each model.

In addition to the three models, the code also includes PCA and K-fold cross-validation. PCA (Principal Component Analysis) is performed using the PCA function from the sklearn.decomposition module. It reduces the dimensionality of the feature space by identifying linear combinations of features that explain the most variance in the data.

K-fold cross-validation is performed using the cross_val_score function from the sklearn.model_selection module. This technique splits the training data into k subsets and trains the model on k-1 of those subsets while evaluating performance on the remaining subset. This process is repeated k times, with each subset serving as the evaluation set exactly once. The function then returns an array of k scores, which can be used to estimate the generalization performance of the model.

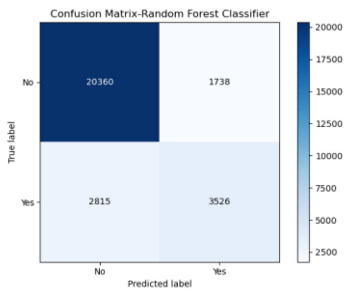
4: Results:

4.1.1. Decision Tree



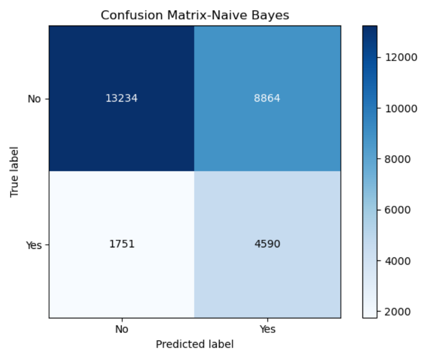
- The total number of instances in the test set is 26,439 (18538 + 3560 + 2784 + 3557).
- Out of these instances, 18,538 were actually negative and were correctly classified as negative (true negatives).
- 3,560 instances were actually positive and were incorrectly classified as negative (false negatives).
- 2,784 instances were actually negative and were incorrectly classified as positive (false positives).
- 3,557 instances were actually positive and were correctly classified as positive (true positives).

4.1.2. Random Forest



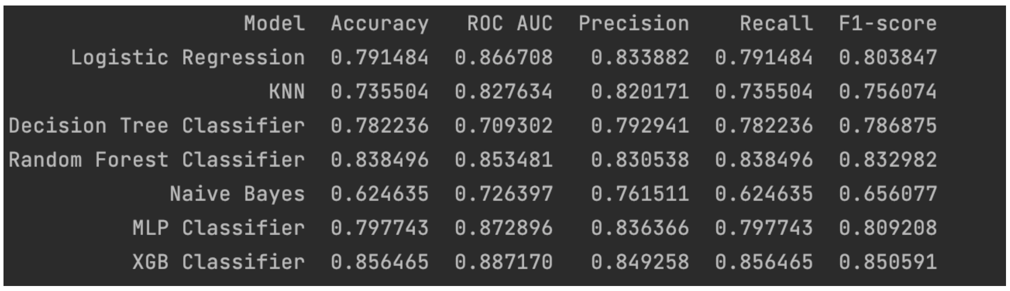
- The total number of instances in the test set is 26,439 (20360 + 1738 + 2815 + 3526).
- Out of these instances, 20,360 were actually negative and were correctly classified as negative (true negatives).
- 1,738 instances were actually positive and were incorrectly classified as negative (false negatives).
- 2,815 instances were actually negative and were incorrectly classified as positive (false positives).
- 3,526 instances were actually positive and were correctly classified as positive (true positives).

4.1.3. Naïve Bayes



- The total number of instances in the test set is 26,439 (13234 + 8864 + 1751 + 4590).
- Out of these instances, 13,234 were actually negative and were correctly classified as negative (true negatives).
- 8,864 instances were actually positive and were incorrectly classified as negative (false negatives).
- 1,751 instances were actually negative and were incorrectly classified as positive (false positives).
- 4,590 instances were actually positive and were correctly classified as positive (true positives).

4.1.4. Comparison of Models



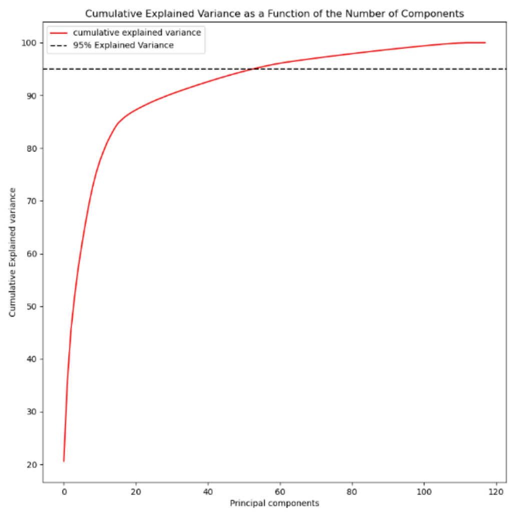
```
                   Model  Accuracy   ROC AUC  Precision    Recall  F1-score
     Logistic Regression  0.791484  0.866708   0.833882  0.791484  0.803847
                     KNN  0.735504  0.827634   0.820171  0.735504  0.756074
Decision Tree Classifier  0.782236  0.709302   0.792941  0.782236  0.786875
Random Forest Classifier  0.838496  0.853481   0.830538  0.838496  0.832982
             Naive Bayes  0.624635  0.726397   0.761511  0.624635  0.656077
          MLP Classifier  0.797743  0.872896   0.836366  0.797743  0.809208
          XGB Classifier  0.856465  0.887170   0.849258  0.856465  0.850591
```

we can see that the XGB Classifier achieved the highest accuracy (0.85) and ROC AUC (0.88) among all the models, indicating that it performed the best in terms of overall classification accuracy and ability to distinguish between positive and negative instances. It also achieved a good balance between precision and recall, as indicated by its high F1-score (0.85).

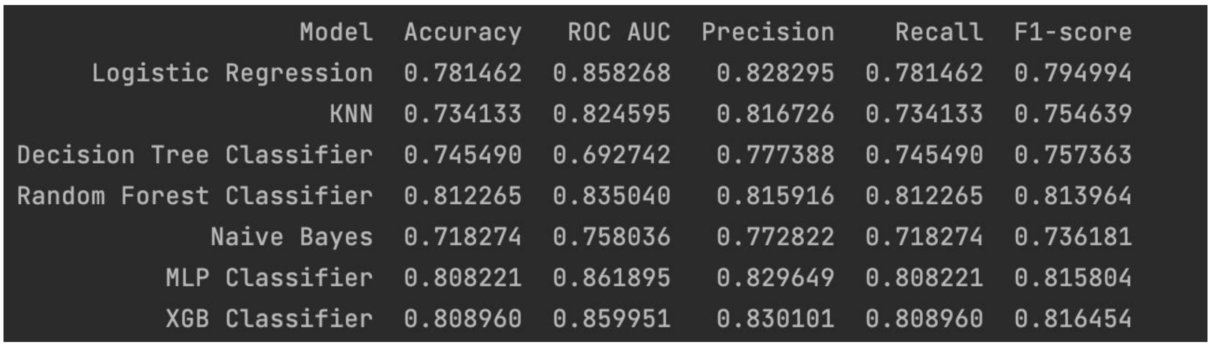
The Random Forest and MLP Classifier models also performed well, with accuracy and ROC AUC scores close to that of the Random Forest Classifier.

The Naive Bayes model had the lowest accuracy (0.63) and ROC AUC (0.73) among all the models, indicating that it performed the worst in terms of overall classification accuracy and ability to distinguish between positive and negative instances. However, it achieved the highest precision (0.76) among all the models, which suggests that it is better at identifying true positive instances than the other models, albeit at the cost of misclassifying a larger number of negative instances.

4.2. Principal Component Analysis



Based on the graph presented above, we can observe that using 50 components covers 95% of the explained variance. Therefore, we selected those 50 components and used them to build our model. The results of the model are presented below.



```
                   Model  Accuracy   ROC AUC  Precision    Recall  F1-score
     Logistic Regression  0.781462  0.858268   0.828295  0.781462  0.794994
                     KNN  0.734133  0.824595   0.816726  0.734133  0.754639
Decision Tree Classifier  0.745490  0.692742   0.777388  0.745490  0.757363
Random Forest Classifier  0.812265  0.835040   0.815916  0.812265  0.813964
             Naive Bayes  0.718274  0.758036   0.772822  0.718274  0.736181
          MLP Classifier  0.808221  0.861895   0.829649  0.808221  0.815804
          XGB Classifier  0.808960  0.859951   0.830101  0.808960  0.816454
```

By reducing the dimensionality of the dataset and selecting only 50 components, which is approximately half of the initial dataset's components, we were able to achieve comparatively similar metric scores.

4.3. K-Fold Cross Validation

We utilized the K-Nearest Neighbors Classifier algorithm to build a classification model. The model was trained on the training data using 10-fold cross-validation, where k was set to 3. The accuracy of the model was evaluated using the cross_val_score() function in the scikit-learn library.

The results of the cross-validation indicate that the model had an average accuracy of 0.865 with a standard deviation of 0.007, which suggests that the model's performance is consistent across the different folds of the cross-validation process. These results demonstrate that the K-Nearest Neighbors Classifier algorithm is a suitable choice for this classification problem.

5. Summary and Conclusions

Using different machine learning algorithm models, we predicted whether on the next day it will rain or not. Out of the models that we built, we found out that XGB classifier gave out the best performance. We learned about different techniques to handle data pre-processing, such as data imputation, feature selection, sampling technique SMOTE. We also learned how different classification algorithms work and the importance of doing principal component analysis (PCA).

In the future, we would like to perform in depth data processing techniques and apply neural networks.

6. Percentage of work copied from internet

I have taken around 100 lines of code from internet modified around 20 lines of code and written 50 lines of code
Percentage of work: 53.33%

References

https://aws.amazon.com/what-is/logistic-regression/
https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/
https://scikit-learn.org/stable/modules/neighbors.html/
https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f
https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package
https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
https://www.kaggle.com/code/orhansertkaya/k-fold-cross-validation-and-grid-search-cv

Appendix

Computer used:

- MSI Gaming

Software used:

- Pycharm Professional
- Jupyter Notebook
- GitHub
- Microsoft Word
- Microsoft Powerpoint