RAMAH ALHARBI

جامعة جدة
University of Jeddah

Pattern Recognition Project

MAGIC Gamma Telescope

Section: I7

Student name: Ramah Nader Alharbi

Instructor: Safa ALsafari

1.Introduction: The MAGIC Gamma Telescope project represents a remarkable milestone in the field of astrophysics, focusing on the detection and analysis of high-energy gamma particles using ground-based atmospheric Cherenkov gamma telescopes. These elusive gamma rays originate from some of the most energetic and distant cosmic phenomena, offering invaluable insights into the universe's fundamental processes. In this project report, we delve into the implementation of advanced data analysis techniques to unravel the mysteries of high-energy gamma rays. Leveraging imaging methods and a diverse array of data analysis algorithms, including Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors (KNN), and Neural Networks, we aim to precisely classify and understand the gamma ray signatures recorded by the MAGIC telescope. The following sections detail our methodology, findings, and the implications of this research, as we seek to deepen our understanding of the universe's most enigmatic and energetic phenomena. Through this endeavor, we hope to contribute to the ever-expanding frontier of astrophysics and high-energy particle detection.

2.problem definition: Leveraging Data Analysis Techniques for Accurate Classification of High-Energy Gamma Rays. The MAGIC Gamma Telescope project faces the challenge of accurately classifying high-energy gamma rays detected by the ground-based atmospheric Cherenkov gamma telescope. These gamma rays originate from extreme cosmic phenomena, including black hole accretion, active galactic nuclei, and supernova remnants. However, the detection of these gamma rays is often impeded by background noise generated by cosmic rays in the Earth's atmosphere. The primary objective of this research is to implement advanced data analysis techniques, such as Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors (KNN), and Neural Networks, to precisely differentiate between primary gamma signals and background noise. By developing robust pattern regression models, the project aims to optimize the data analysis process and improve the accuracy of high-energy gamma ray classification. The successful classification of these gamma rays will lead to a deeper understanding of the universe's most energetic processes, unlocking valuable insights into the nature and origins of cosmic phenomena. Ultimately, this research contributes to the advancements in astrophysics and high-energy particle detection, paving the way for new discoveries in the field.

3.Data descriptions: The MAGIC Gamma Telescope project dataset consists of 19020 samples and 11 features. The dataset includes a of data types, including integers, floats and objects, providing a rich set of variables to analyze. Usage Patterns, Length, Width, Size, Conc, Conc, Asym, long of ray, Transmission of ray, Alpha, Distance and class of gamma or hadron, the project aims to optimize the data analysis process and improve the accuracy of high-energy gamma ray classification. We performed the following data analysis step such as Feature selection we use hyperparameter turning. Exploratory Data Analysis (EDA): Descriptive statistics we computed summary statistics. Data visualization, we used visualizations like histograms, pie chart...etc., to explore relationships between variables,
identify patterns. Model Training and Evaluation, Dataset splitting, we split the dataset into training and testing sets. Model selection: We experimented with various machine learning algorithms suitable for classification tasks, such as decision trees, SVM and Neural Network. We selected the model that exhibited the best performance based on evaluation metrics. Model evaluation: We assessed the performance of the trained model using evaluation metrics such as accuracy, precision, recall score. **4.Methods:**

this code snippet performs k-Nearest Neighbors (k-NN) classification using the KNeighborsClassifier from scikit-learn. It trains the k-NN model on the training data, makes predictions on the test data, and then prints a classification report showing various performance metrics for the model's predictions. Simple and easy to understand, requiring minimal assumptions about the data.
Works well with small to medium-sized datasets.

Naive Bayes model

```
[ ] from sklearn.naive_bayes import GaussianNB

[ ] nb_model = GaussianNB()
    nb_model = nb_model.fit(X_train, y_train)

[ ] y_pred = nb_model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

             0       0.63      0.43      0.51      1305
             1       0.75      0.87      0.80      2499

      accuracy                           0.72      3804
     macro avg       0.69      0.65      0.66      3804
  weighted avg       0.71      0.72      0.70      3804
```

This code snippet uses the Gaussian Naive Bayes algorithm for classification. It creates a Gaussian Naive Bayes classifier and trains it using the training data. Then, it makes predictions on the test data and prints a classification report with various performance metrics, evaluating the model's accuracy and performance. Can handle both binary and multi-class classification problems.
Works effectively with text classification tasks, such as spam filtering.

Log Regression model

```
[ ] from sklearn.linear_model import LogisticRegression

[ ] lg_model = LogisticRegression()
    lg_model = lg_model.fit(X_train, y_train)

[ ] y_pred = lg_model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

             0       0.65      0.71      0.68      1305
             1       0.84      0.80      0.82      2499

      accuracy                           0.77      3804
     macro avg       0.75      0.76      0.75      3804
  weighted avg       0.78      0.77      0.77      3804
```

This code uses Logistic Regression for classification. It creates a logistic regression classifier, trains it with the training data, makes predictions on the test data, and prints a classification report showing various performance metrics for the model's predictions. Fast and easy to implement, making it a good baseline model for binary classification tasks.
Interpretable results, allowing analysis of the impact of each feature on the classification.

SVM model

```
[ ] from sklearn.svm import SVC

[ ] svm_model = SVC()
    svm_model = svm_model.fit(X_train, y_train)

[ ] y_pred = svm_model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

             0       0.80      0.81      0.80      1305
             1       0.90      0.90      0.90      2499

      accuracy                           0.87      3804
     macro avg       0.85      0.85      0.85      3804
  weighted avg       0.87      0.87      0.87      3804
```

uses Support Vector Machine (SVM) for classification. It creates an SVM classifier, trains it with the training data, makes predictions on the test data, and prints a classification report showing various performance metrics for the model's predictions. Effective in high-dimensional spaces and works well with feature-rich datasets.Suitable for both linear and non-linear classification tasks, thanks to different kernel functions (e.g., linear, polynomial, radial basis function).Generalizes well to new data and is robust against overfitting.

Random forest model

```
[ ] from sklearn.ensemble import RandomForestClassifier
    RandomForest_model = RandomForestClassifier(n_estimators=100, random_state=42)
    RandomForest_model = RandomForest_model.fit(X_train, y_train)
    y_pred = RandomForest_model.predict(X_test)
    print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

             g       0.88      0.89      0.89      2444
             h       0.80      0.79      0.80      1360

      accuracy                           0.86      3804
     macro avg       0.84      0.84      0.84      3804
  weighted avg       0.86      0.86      0.86      3804
```

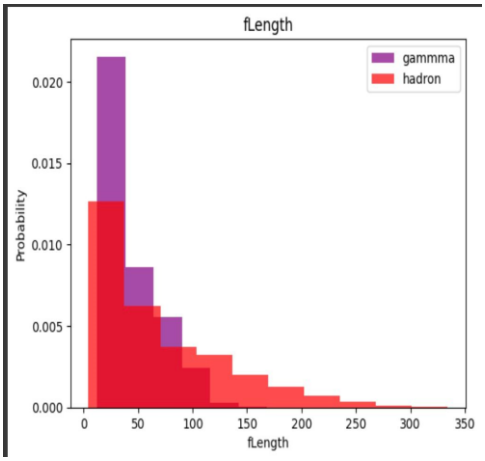
This code uses Random Forest for classification. It creates a Random Forest classifier with 100 decision trees, trains it with the training data, makes predictions on the test data, and prints a classification report showing various performance metrics for the model's predictions. Reduces the risk of overfitting due to ensemble learning and bagging. Provides feature importance, allowing insights into feature relevance.
Works well with imbalanced datasets and avoids the need for feature scaling.

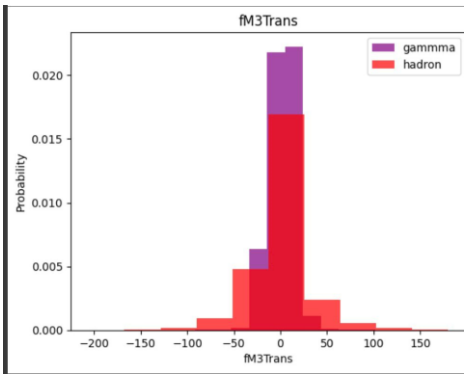
```
[ ] def train_model(X_train, y_train, num_nodes, dropout_prob, lr, batch_size, epochs):
        nn_model = tf.keras.Sequential([
            tf.keras.layers.Dense(num_nodes, activation='relu', input_shape=(10,)),
            tf.keras.layers.Dropout(dropout_prob),
            tf.keras.layers.Dense(num_nodes, activation='relu'),
            tf.keras.layers.Dropout(dropout_prob),
            tf.keras.layers.Dense(1, activation='sigmoid')
        ])

        nn_model.compile(optimizer=tf.keras.optimizers.Adam(lr), loss='binary_crossentropy',
                         metrics=['accuracy'])
        history = nn_model.fit(
          X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=0
        )

        return nn_model, history
```

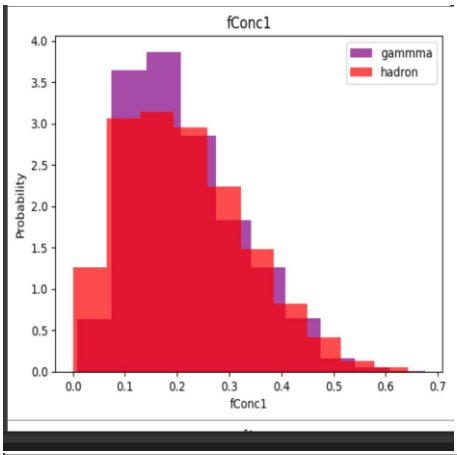
This code defines two functions and implements a training loop to find the neural network model with the lowest validation loss for a binary classification problem using TensorFlow. The plot_history(history) function is responsible for visualizing the training history, displaying the loss and accuracy during training and validation epochs. The train_model(X_train, y_train, num_nodes, dropout_prob, lr, batch_size, epochs) function creates, compiles, and trains a neural network model with the specified hyperparameters. The training loop iterates over various combinations of hyperparameters, trains the models, and evaluates their performance on the validation set. It keeps track of the model with the least validation loss, and at the end, it stores this model in the least_loss_model variable. This approach helps to find the optimal hyperparameter configuration that results in the best-performing model for the given classification task.

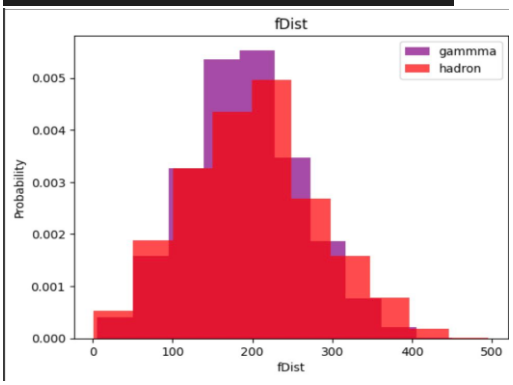
6.Visualization:

6.1histogram

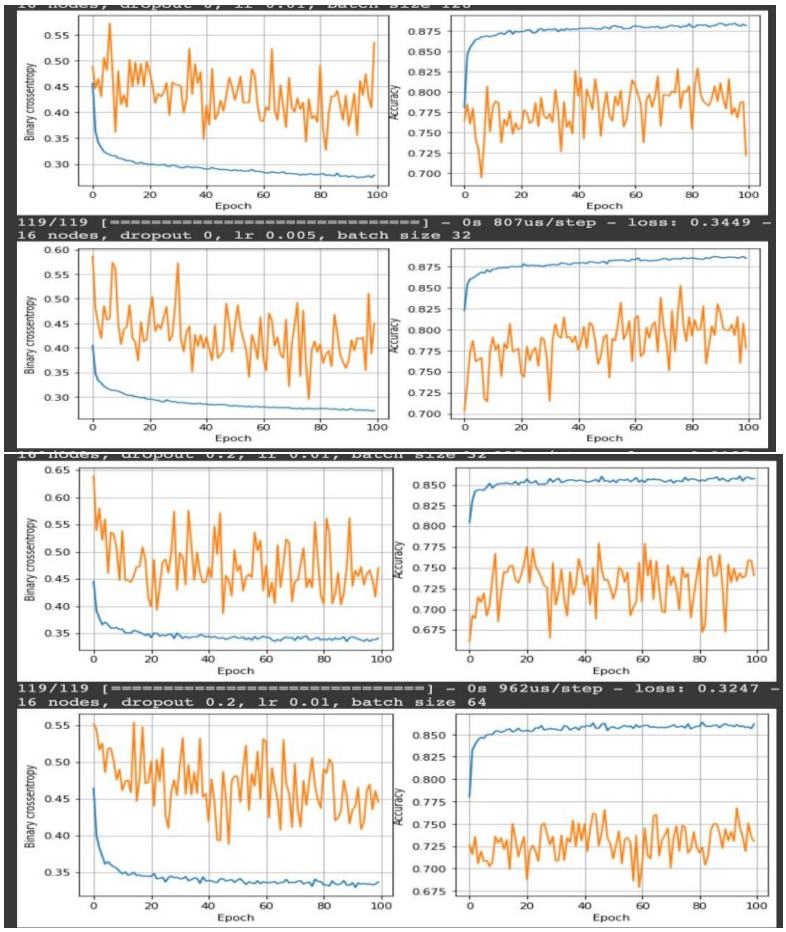




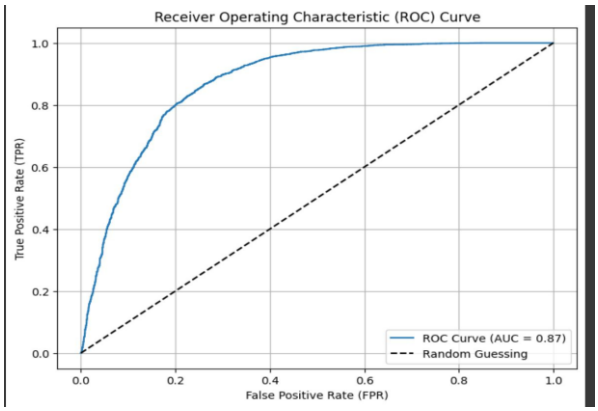




plots histograms for each feature (label) in the dataset, showing the distribution of values for the two classes: "gamma" (blue) and "hadron" (red).
Observe the shapes of the histograms for each class. Differences in the distributions may indicate that certain features have predictive power for distinguishing between the classes.
Overlapping or Separation of Distributions:
Check if the histograms of the two classes overlap significantly or if they are well-separated.
If the distributions largely overlap, it might indicate that the feature is not highly informative in classifying the two classes.
On the other hand, well-separated distributions suggest that the feature has discriminative power and can help distinguish between the classes.
Outliers and Skewness:
Look for extreme values or outliers in the distributions that might indicate data errors or rare instances.
Assess the skewness of the distributions; a highly skewed distribution may need preprocessing or transformation to improve model performance.
Class Imbalance:
Observe if there is a significant difference in the number of samples between the "gamma" and "hadron" classes.
Class imbalance can affect the model's performance, and you may need to address it using techniques like oversampling, undersampling, or class weighting during model training.
Feature Importance:
Identify features that show clear differences in their distributions between the classes.
Features with distinct distributions might be more important in classification, and they could be good candidates for inclusion in the model.
Feature Scaling Consideration:
Consider whether certain features have different scales. Standardizing or normalizing the features might be necessary, especially for algorithms sensitive to feature scales (e.g., SVM, k-NN).
Feature Relationships:
Explore relationships between features by comparing their distributions for each class.
Identify features that exhibit similar or different patterns across classes, which may provide insights into the underlying data characteristics.
Overall, the graphs help you understand the data and the potential relationships between features and classes. They provide valuable insights into the dataset's characteristics, which can guide you in selecting appropriate features and building a more accurate classification model.



Training Loss and Validation Loss (Left Plot):
Look for the trend of the training loss and validation loss over epochs.
If the training loss is decreasing but the validation loss is increasing or stagnating, it might indicate overfitting, meaning the model is memorizing the training data and not generalizing well to new data. In this case, regularization techniques or reducing model complexity may help.
If both training and validation losses are high and not decreasing, it could indicate underfitting, meaning the model is not learning the underlying patterns in the data. You may need to increase model complexity or train for more epochs to improve performance.
Training Accuracy and Validation Accuracy (Right Plot):
Observe the trend of the training accuracy and validation accuracy over epochs.
If the training accuracy continues to increase while the validation accuracy plateaus or decreases, it suggests overfitting. Consider regularization or other techniques to prevent overfitting.
If both training and validation accuracies are low and not improving, it might indicate underfitting. Adjust model complexity or hyperparameters accordingly.



the graph shows the Receiver Operating Characteristic (ROC) curve for the Support Vector Machine (SVM) classifier in a binary classification task. The curve represents the classifier's performance in distinguishing between positive and negative classes at different threshold levels. A higher AUC (Area Under the Curve) and a curve closer to the top-left corner indicate better classifier performance, while an AUC close to 0.5 suggests a classifier that is not better than random guessing.

8.Conclusions: In conclusion, this project explored different machine learning algorithms for a binary classification task. After evaluating the models, it was found that Support Vector Machine (SVM) and Neural Network achieved the highest accuracy. The study demonstrated the significance of hyperparameter tuning and the need to select appropriate algorithms based on dataset characteristics. Future research may focus on advanced algorithms and feature engineering to further enhance classification performance. Overall, the project highlights the value of applying diverse techniques in solving real-world classification problems.