Classifying Eye States with EEG: A Machine Learning Approach

Electroencephalography, or EEG, measures electrical activity in the brain. It's used in diagnosing brain disorders and has exciting applications in Brain-Computer Interfaces (BCIs).

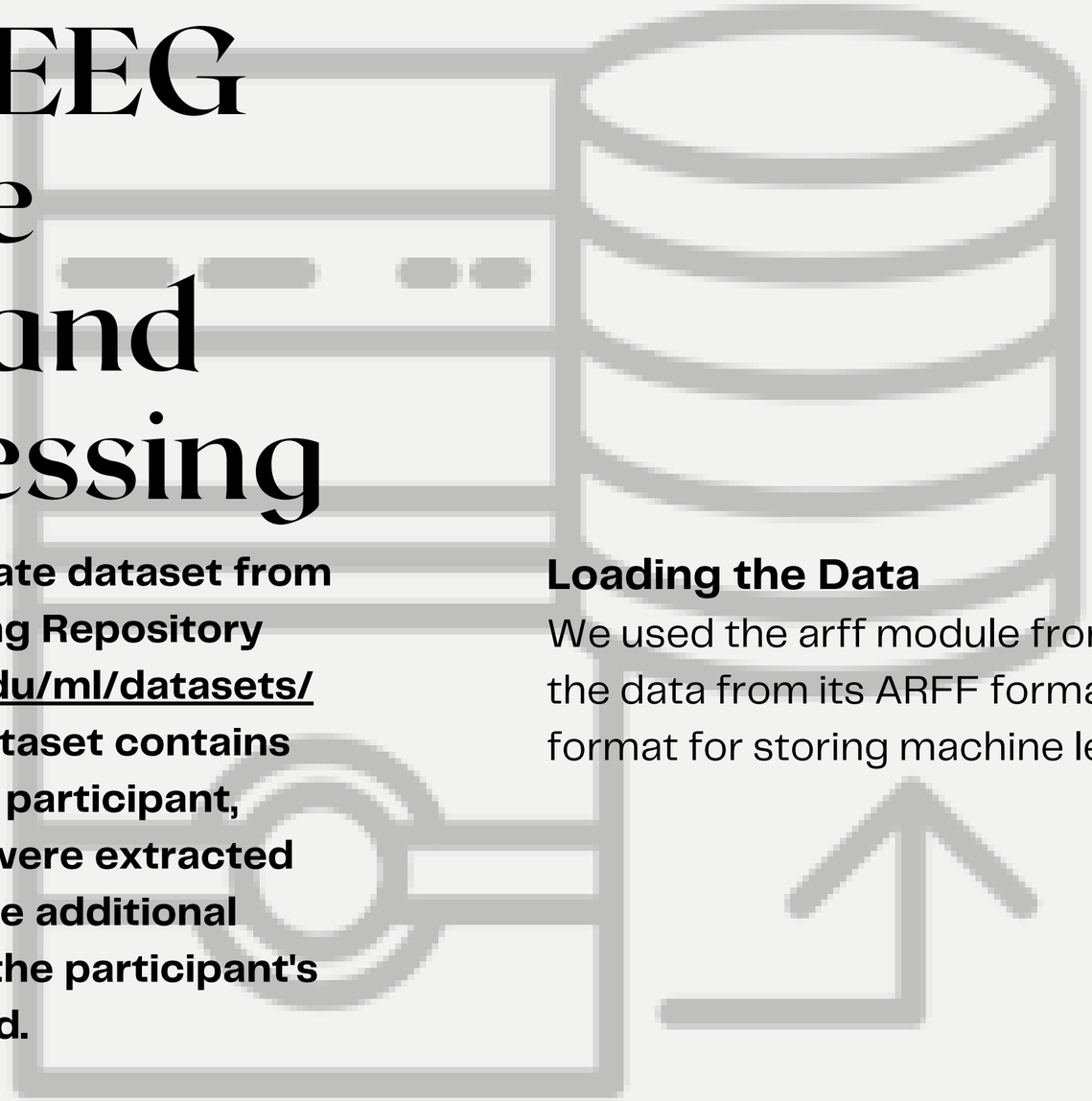
Mustafa Kaya Bozbel
Melih Yıldız
Muhammed Tekin

Purpose of Project

We'll focus on using EEG to classify eye states, a crucial step for BCI systems that aim to control devices or applications just by thinking about looking at them.

The UCI EEG Eye State Dataset and Preprocessing

We used the EEG Eye State dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State). This dataset contains recordings from a single participant, where various features were extracted from their EEG signal. The additional label indicates whether the participant's eyes were open or closed.



Loading the Data

We used the arff module from scipy.io to load the data from its ARFF format, a common format for storing machine learning data.

Preprocessing Steps

1. Separating Features and Target: We separated the EEG features (all columns except the last) from the target variable (last column) for easier manipulation.

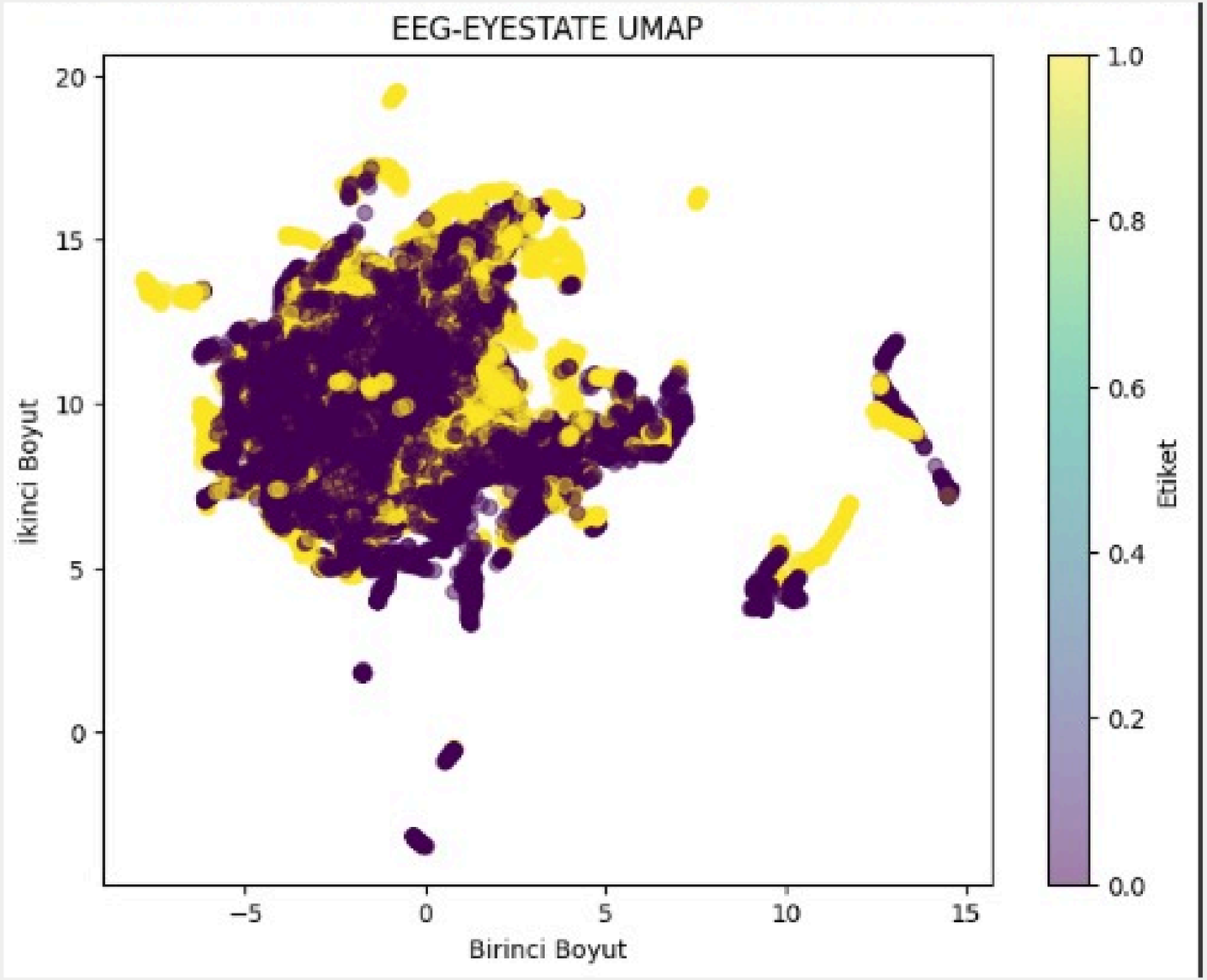
2. Missing Value Imputation: We addressed any missing values in the features using SimpleImputer, which replaces them with the mean value of each feature.

3. Label Encoding: Since the target variable represents categories (open/closed eyes), we used LabelEncoder to convert these categories into numerical values for machine learning models.

4. Train-Test Split: Finally, we split the data into training and testing sets using train_test_split to train our models and evaluate their performance on unseen data.

Features

-AF3
-F7
-F3
-FC5
-T7
-P7
-O1
-O2
-P8
-T8
-FC6
-F4
-F8
-AF4
-Eye Detection (Target)

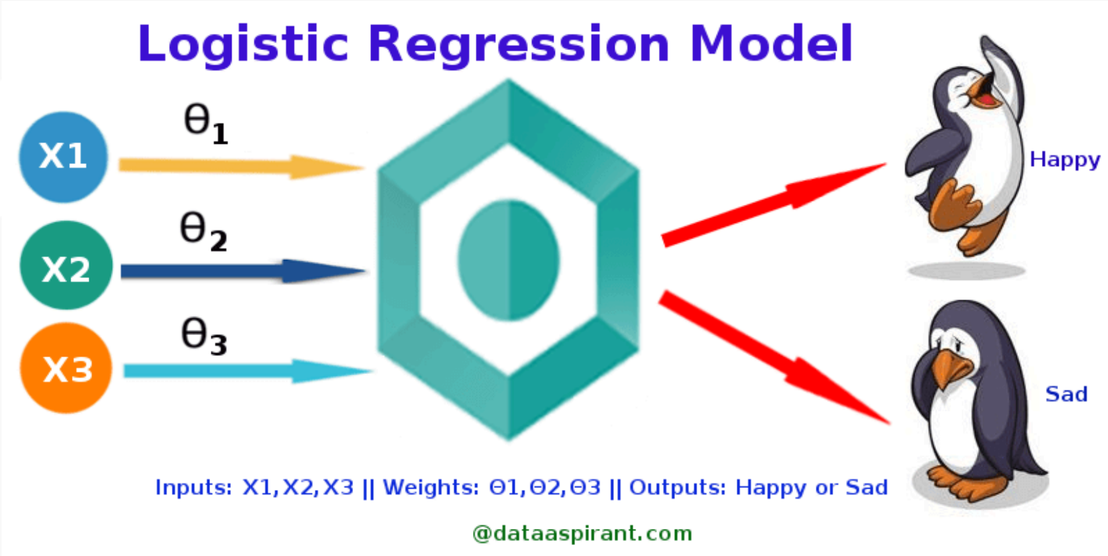


Visualization of Dataset

The Models That We Used

The Models That We Used

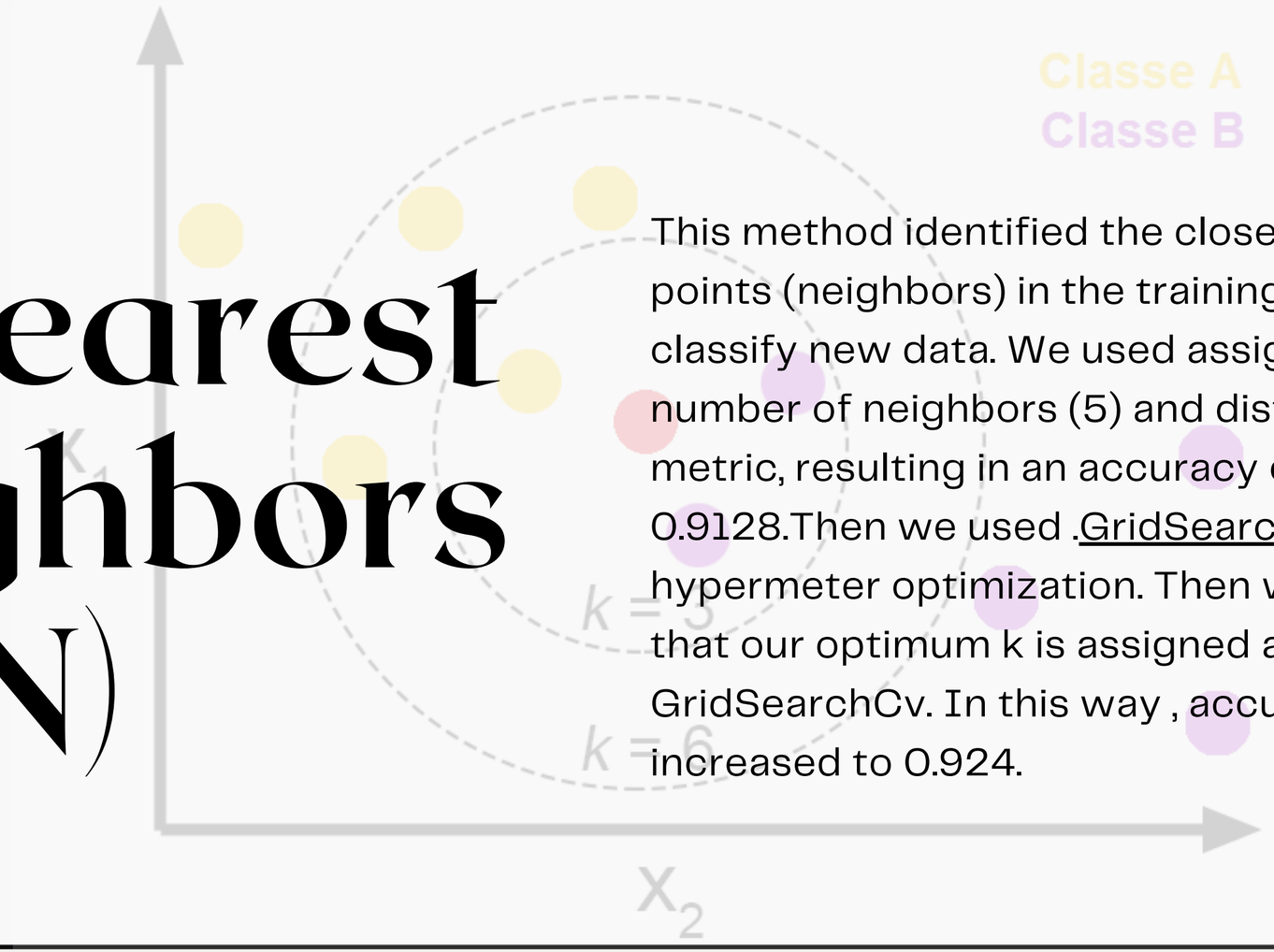
Logistic Regression



This linear model achieved an accuracy of 0.5804 , providing a baseline for classification performance.

The Models That We Used

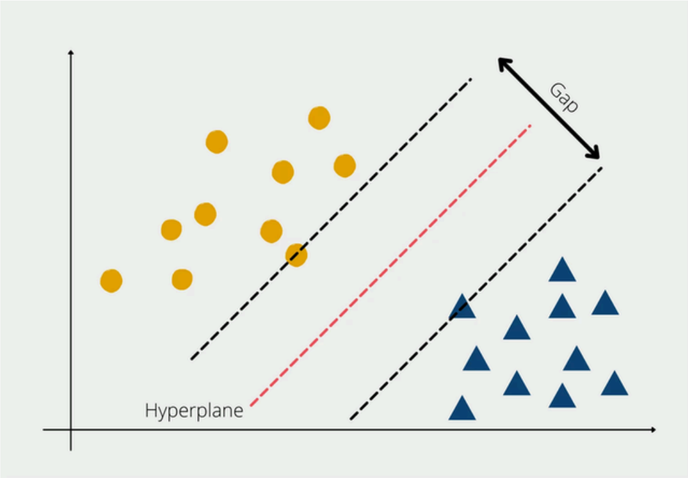
K-Nearest Neighbors (KNN)



This method identified the closest data points (neighbors) in the training set to classify new data. We used assigned the number of neighbors (5) and distance metric, resulting in an accuracy of 0.9128.Then we used .GridSearchCV for hypermeter optimization. Then we realized that our optimum k is assigned as 3 by GridSearchCv. In this way , accuracy increased to 0.924.

The Models That We Used

Support Vector Machine (SVM)

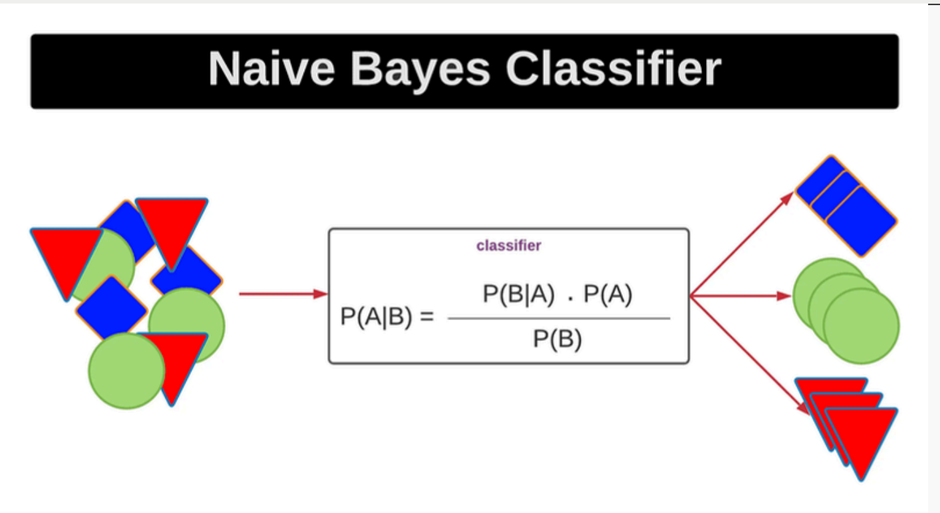


We explored two types of SVMs

- Linear SVM: This achieved an accuracy of 0.6068.
- RBF kernel SVM: This non-linear SVM achieved an accuracy 0.7156.

The Models That We Used

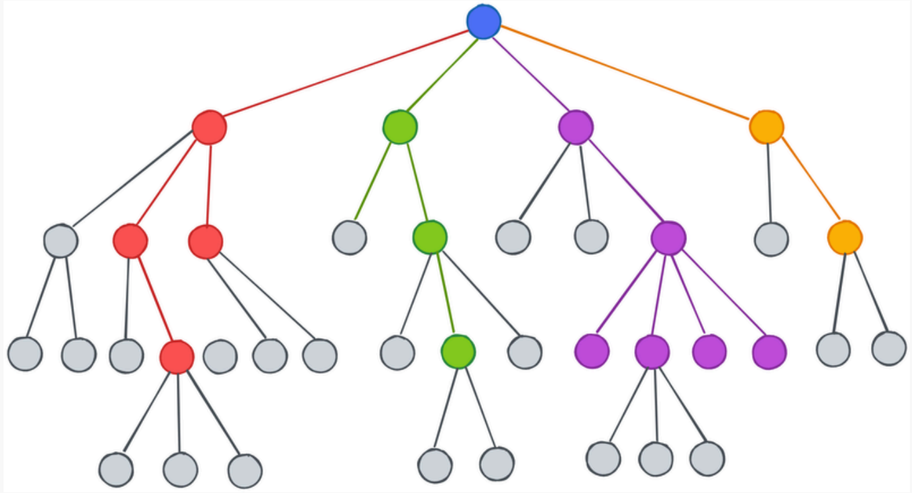
Naive Bayes



This probabilistic model achieved an accuracy of 0.5100. It assumes features are independent, which may not always hold true for EEG data.

The Models That We Used

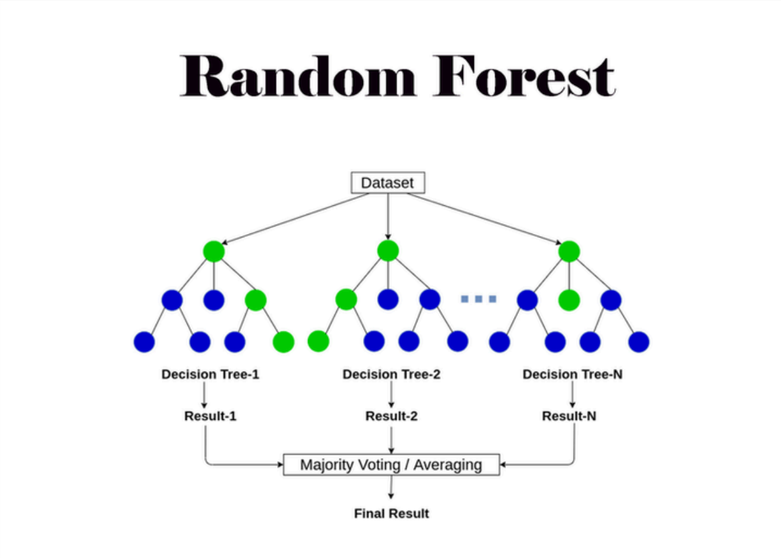
Decision Tree



This tree-based model learned a set of rules to classify data. It achieved an accuracy of 0.8307.

The Models That We Used

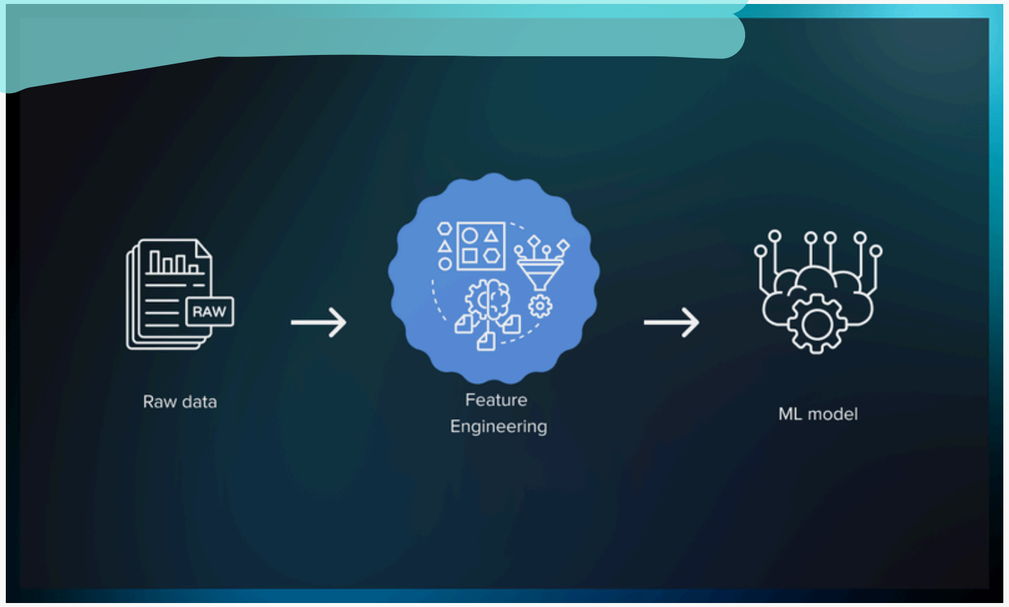
Random Forest



This ensemble method combines multiple decision trees for improved performance. It achieved the highest accuracy among the evaluated models at 0.8945.

Feature Engineering

Feature Engineering



Techniques Applied:

1. Principal Component Analysis (PCA)
2. Feature Selection Methods
3. Statistical Transformations

Results:

- No significant improvement in accuracy
- The original feature set was optimal for our model

Conclusion:

- Feature engineering did not enhance model performance for our dataset

Techniques

Principal Component Analysis (PCA):

PCA is a dimensionality reduction technique used to reduce the number of features in a dataset, It transforms the original features into a new set of features called principal components, which are uncorrelated to each other.

Feature Selection Methods:

Feature selection involves selecting a subset of relevant features from the original set of features. Techniques include:

- Filter Methods
- Wrapper Methods
- Embedded Methods

Polynomial Features:

Polynomial features are created by raising existing features to a power and/or by creating interaction terms between features. This technique allows linear models to fit more complex, nonlinear relationships by increasing the feature space.

KNN Hyperparameter Tuning

KNN Hyperparameter Tuning

1. Used **GridSearchCV** to find optimal parameters
2. Tested various values for k, weights, and distance metrics

Initial KNN Model:

- Hyperparameters:
  - k: 5
  - Metric: Minkowski
- Accuracy: 91.2%

Performance Metrics

Accuracy Score: 91.28%
Precision Score:91.60%
Recall Score:89.13%
F1 Score:90.35%

Optimal KNN Model:

- Hyperparameters:
  - k: 3
  - Metric: Minkowski
- Accuracy: 92.4%

Performance Metrics

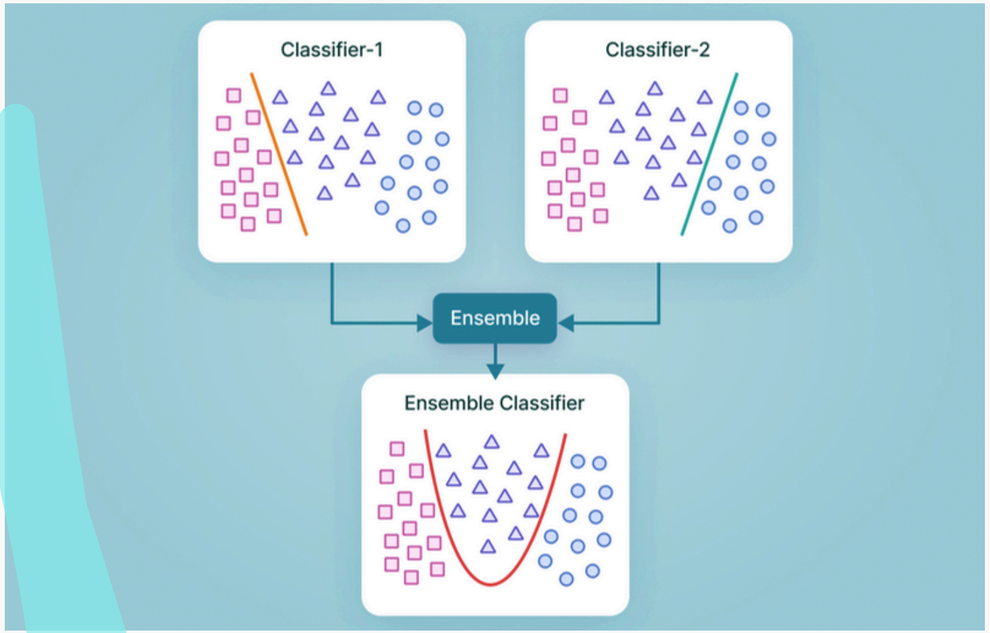
Accuracy Score: 92.45%
Precision Score:92.37%
Recall Score:91.02%
F1 Score:91.69%

GridSearchCV

A model is built separately with all combinations for the hyperparameters and their values that are desired to be tested in the model, and the most successful hyperparameter set is determined according to the specified metric.

Ensemble Methods

Ensemble Methods



Ensemble Techniques Tested:

1. Bagging
2. Boosting
3. Stacking

- Overall, ensemble methods provided strong performance but did not surpass the performance of the optimized KNN model

Stacking

Results:

- Results:

```
Stacking Classifier Accuracy: 0.9135514018691588
Stacking Classifier Metrics:
Precision: 0.9161676646706587
Recall: 0.8927789934354485
F1 Score: 0.9043221278167713
```

- Stacking does better because it combines different types of models, making predictions stronger and more accurate

Bagging

Results:

- Results:

```
Bagging Classifier Accuracy: 0.8871829105473965
Bagging Classifier Metrics:
Precision: 0.9195775792038993
Recall: 0.825674690007294
F1 Score: 0.870099923136049 2
```

AdaBoost

Results:

- Results:

```
AdaBoost Classifier Accuracy: 0.6765687583444593
AdaBoost Classifier Metrics:
Precision: 0.6998011928429424
Recall: 0.513493800145879
F1 Score: 0.5923432898611696
```

Best Model Selection Using AutoML

AutoML

Automated Machine Learning, aims to make the machine learning process more accessible and efficient by automating key steps.

We used the **AutoGluon** library for this purpose. Here's a brief overview of the steps we followed:

The best-performing model identified by AutoGluon was **KNeighborsDist**

The best-performing model identified by AutoGluon was KNeighborsDist

which achieved an impressive accuracy of 99.75%. This model uses the K-Nearest Neighbors algorithm with specific hyperparameters:

- weights : 'distance'
- n_neighbors: '5'
- metric : 'minkowski'

Confusion Matrix:

- True Positives: High
- True Negatives: High
- False Positives: Low
- False Negatives: Low

Performance Metrics:

- Accuracy: 99.75%
- Precision: High
- Recall: High
- F1 Score: High

- These results demonstrate the power of AutoML in identifying the best model and hyperparameters for our EEG classification task.

Conclusion

Conclusion

We developed a machine learning model to classify EEG signals into eyes open and eyes closed states.Firstly we choose K-Nearest Neighbors model with the k value of 3.We use the default weight which is ‘uniform’. Our accuracy is =91.2% The K-Nearest Neighbors (KNN) model achieved the highest accuracy of 92.4% after hyperparameter tuning. Ensemble Methods and Feature Engineering didn’t work well. AutoML identifying KNeighborsDist as the best model. This work demonstrates the potential of machine learning for EEG analysis and lays the groundwork for future research in brain-computer interfaces and neurological studies.

Mustafa Kaya Bozbel
Melih Yıldız
Muhammed Tekin

May 21, 2024

Thank you!

Have a good day!