MACHINE LEARNING PROJECT
UE21EC352B
FINAL ESA REVIEW

Mental Health Disorder Detection Using Machine Learning

Team: 12

Tejas V P	PES1UG21EC910
Akash Ravi Bhat	PES1UG21EC025
Anumula Balaji	PES1UG21EC052

Contents

Brief Background

The paper uses four machine learning models (Logistic Regression, K-Neighbors Classifier, Decision Tree Classifier and Bagging) to predict whether an individual has sought for any treatment for a mental health issue.

It also uses a CNN model to determine whether an individual is depressed or not, based on their activity score and timestamp data.

Problem Statement

We will be increasing the efficiency of the Machine Learning Model by increasing accuracy and precision. Most of the models had issues with the overfitting of data. So we are going to use techniques which avoid such a problem.

NOVELTY

Overfitting Reduction: Implemented techniques to decrease overfitting in Logistic Regression, enhancing model generalizability.

Accuracy Improvement: Achieved higher accuracy across all models, indicating more reliable predictions.

Model Expansion: Added four new models — Random Forest, Stacking, Boosting, and Neural Networks — expanding the predictive capabilities beyond the base paper's models.

Diverse Techniques: The inclusion of ensemble methods like Random Forest, Stacking, and Boosting introduces a variety of decision-making processes for better performance.

Work Done

We have implemented Logistic Regression model, k-NN model and Decision Tree Classifier.

The Ensemble models we implemented are Random Forest.

Methodology:

1) Extracting and Removing Unnecessary Features.

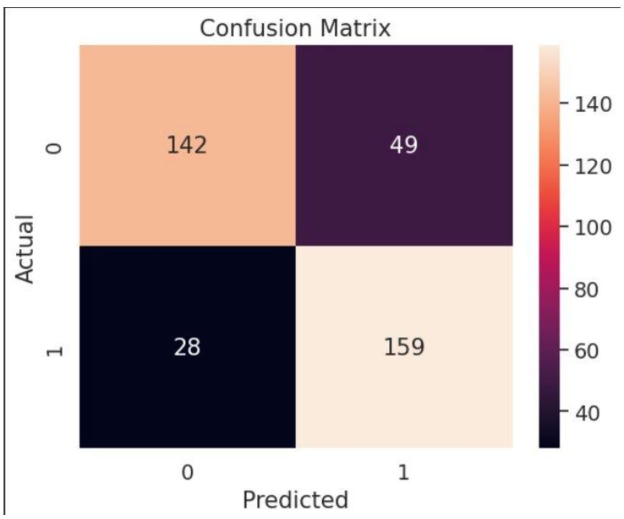
2) Implement various Classification models like Logistic Regression, k-NN, Random Forest and check for performance.

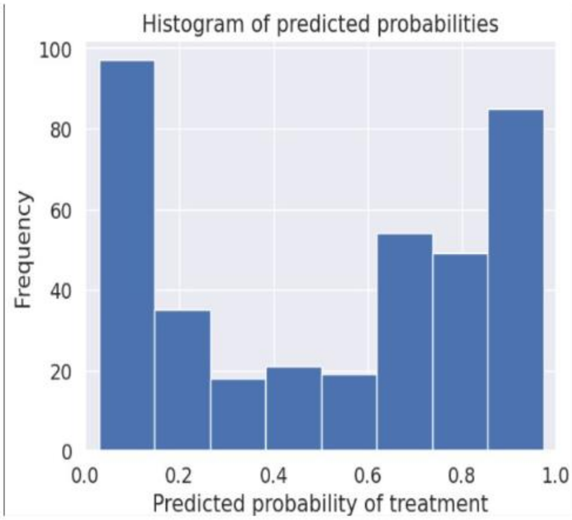
3) Implement an Ensemble Model based on the above algorithms.

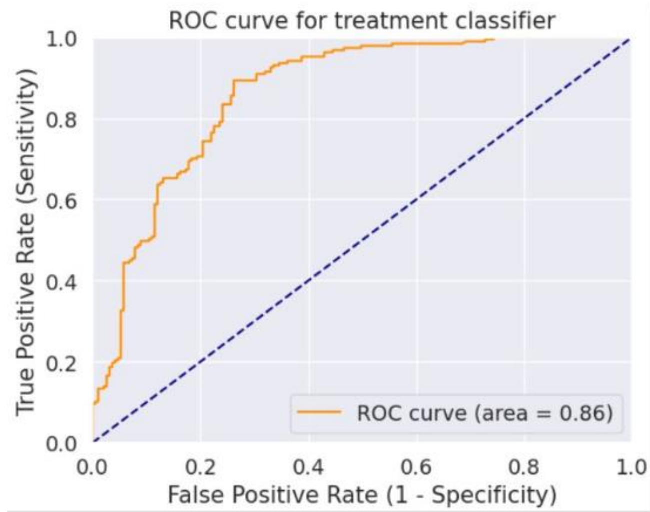
RESULTS

```
logisticRegression()
```

```
Accuracy: 0.7962962962962963
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```



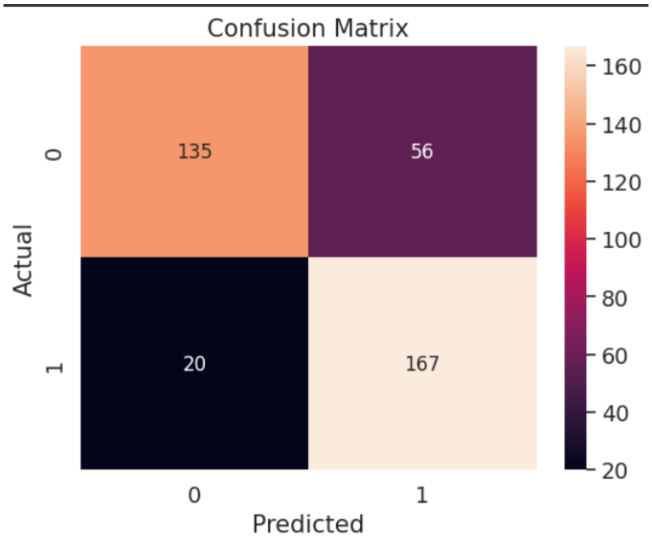




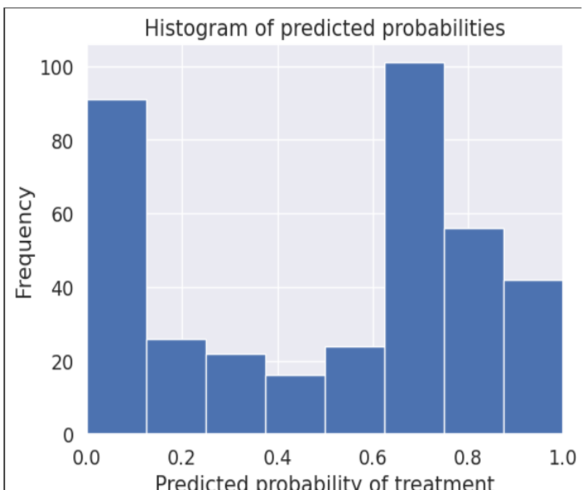
KNeighbors Classifier

```python
def Knn():
    # Calculating the best parameters
    knn = KNeighborsClassifier(n_neighbors=5)

    # define the parameter values that should be searched
    k_range = list(range(1, 31))
    weight_options = ['uniform', 'distance']

    # specify "parameter distributions" rather than a "parameter grid"
    param_dist = dict(n_neighbors=k_range, weights=weight_options)
    tuningRandomizedSearchCV(knn, param_dist)

    # train a KNeighborsClassifier model on the training set
    knn = KNeighborsClassifier(n_neighbors=27, weights='uniform')
    knn.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = knn.predict(X_test)

    accuracy_score = evalClassModel(knn, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['K-Neighbors'] = accuracy_score * 100
```

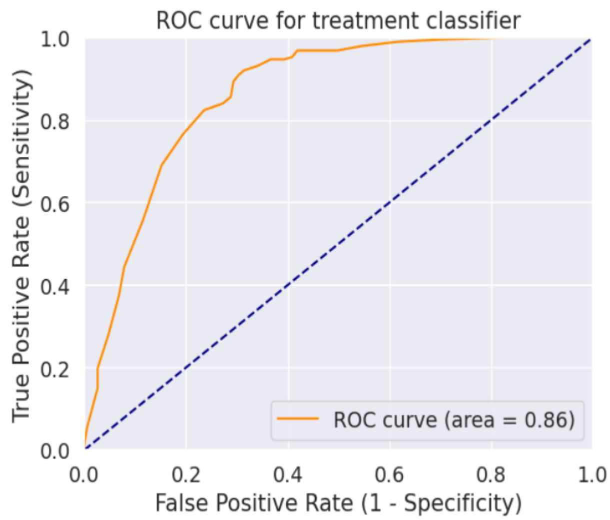
```
Rand. Best Score:  0.8201841269841269
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 15}
[0.823, 0.808, 0.823, 0.823, 0.823, 0.822, 0.812, 0.82, 0.823, 0.819, 0.823, 0.815, 0.823, 0.816, 0.822, 0.823, 0.816, 0.819, 0.819, 0.82]
Accuracy: 0.798941798941799
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```



Confusion Matrix
135
56
20
167
Actual
Predicted
0
1
160
140
120
100
80
60
40
20



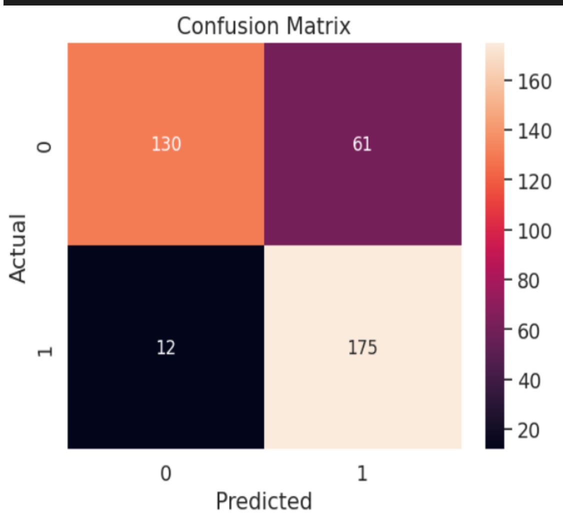
Histogram of predicted probabilities
Frequency
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

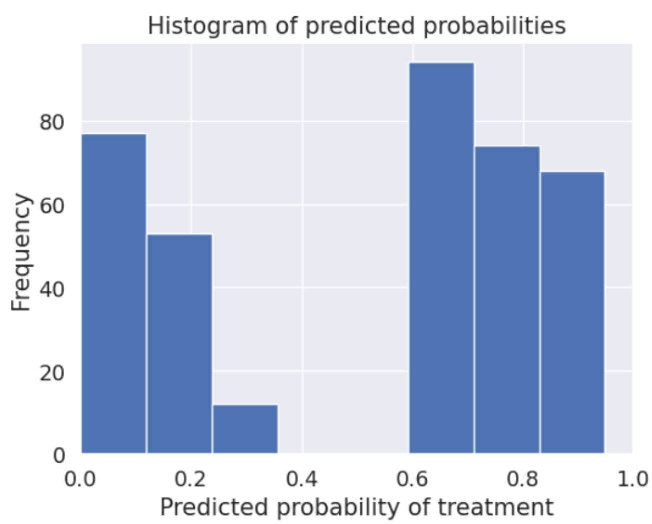


ROC curve for treatment classifier
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
ROC curve (area = 0.86)
0.0
0.2
0.4
0.6
0.8
1.0

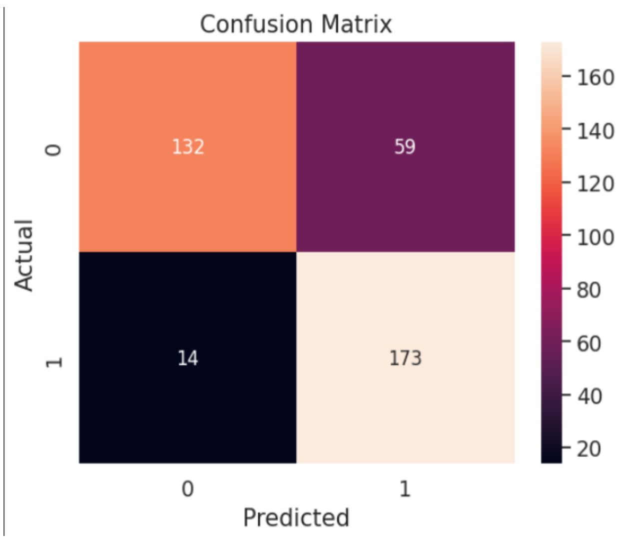
Decision Tree classifier

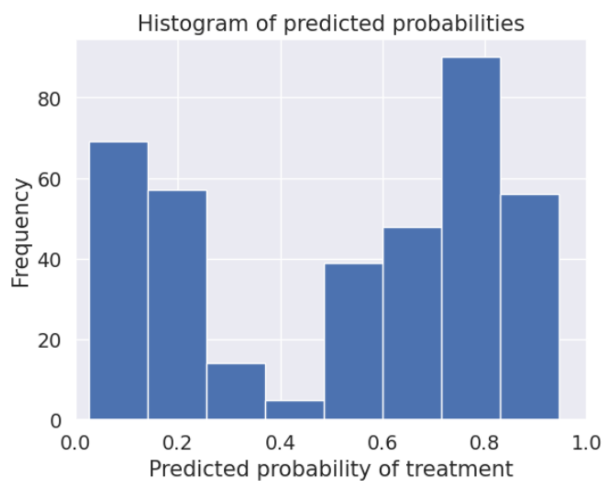
```
def treeClassifier():
    # Calculating the best parameters
    tree = DecisionTreeClassifier()
    featuresSize = feature_cols.__len__()
    param_dist = {"max_depth": [3, None],
              "max_features": randint(1, featuresSize),
              "min_samples_split": randint(2, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}
    tuningRandomizedSearchCV(tree, param_dist)

    # train a decision tree model on the training set
    tree = DecisionTreeClassifier(max_depth=3, min_samples_split=8, max_features=6, criterion='entropy', min_samples_leaf=7)
    tree.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = tree.predict(X_test)

    accuracy_score = evalClassModel(tree, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['Decision Tree Classifier'] = accuracy_score * 100
```

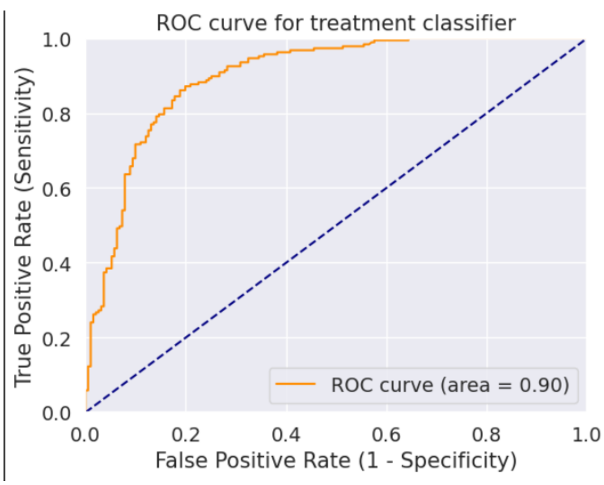




```python
def randomForest():
    # Calculating the best parameters
    forest = RandomForestClassifier(n_estimators = 20)

    featuresSize = feature_cols.__len__()
    param_dist = {"max_depth": [3, None],
              "max_features": randint(1, featuresSize),
              "min_samples_split": randint(2, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}
    tuningRandomizedSearchCV(forest, param_dist)

    # Building and fitting my_forest
    forest = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_split=2, n_estimators = 350, random_state = 1)
    my_forest = forest.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = my_forest.predict(X_test)

    accuracy_score = evalClassModel(my_forest, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['Random Forest'] = accuracy_score * 100
```







RESULT COMPARISION

MODEL	BASE PAPER	OUR OUTPUT
Logistic Regression	79.8	**81.75**
Knn Classifier	76.9	**80.42**
Decision Tree Classifier	72.8	**80.69**
Random Forest	78	**81.22**

Conclusion

In accordance with the base paper, we carefully examined the feature selection graph and pruned the non-essential features.

Subsequently, we proceeded to implement various standalone models, including SVM, k-NN, and Decision Tree Classifier.

Furthermore, we constructed Ensemble models, specifically Random Forest and Voting Classifier, leveraging the strengths of these individual models. Remarkably, our model yielded an accuracy rate of more than 80%, surpassing the approach outlined in the base paper.