



CLASSIFICATION GLIOMAS INTO LOW GRADE AND HIGH GRADE USING MACHINE LEARNING

DEFENSE PRESENTATION

GROUP MEMBERS:

Fatima Khan **Registration #** 21PWCSE2015

M.Zubair Qazi **Registration #** 21PWCSE2032

M.Jalal **Registration #** 21PWCSE2049

Supervisor: Dr. Athar Sethi

Recap

- Previously we completed following tasks:
 - ✓ Dataset prep-Data augmentation
 - ✓ Model training and testing
 - ✓ Classification using SVM-Support Vector Machine
 - ✓ Classification using decision tree
 - ✓ Classification using gradient boosting
- Among three of them, Gradient Boosting achieved highest accuracy of 88%.

Outline

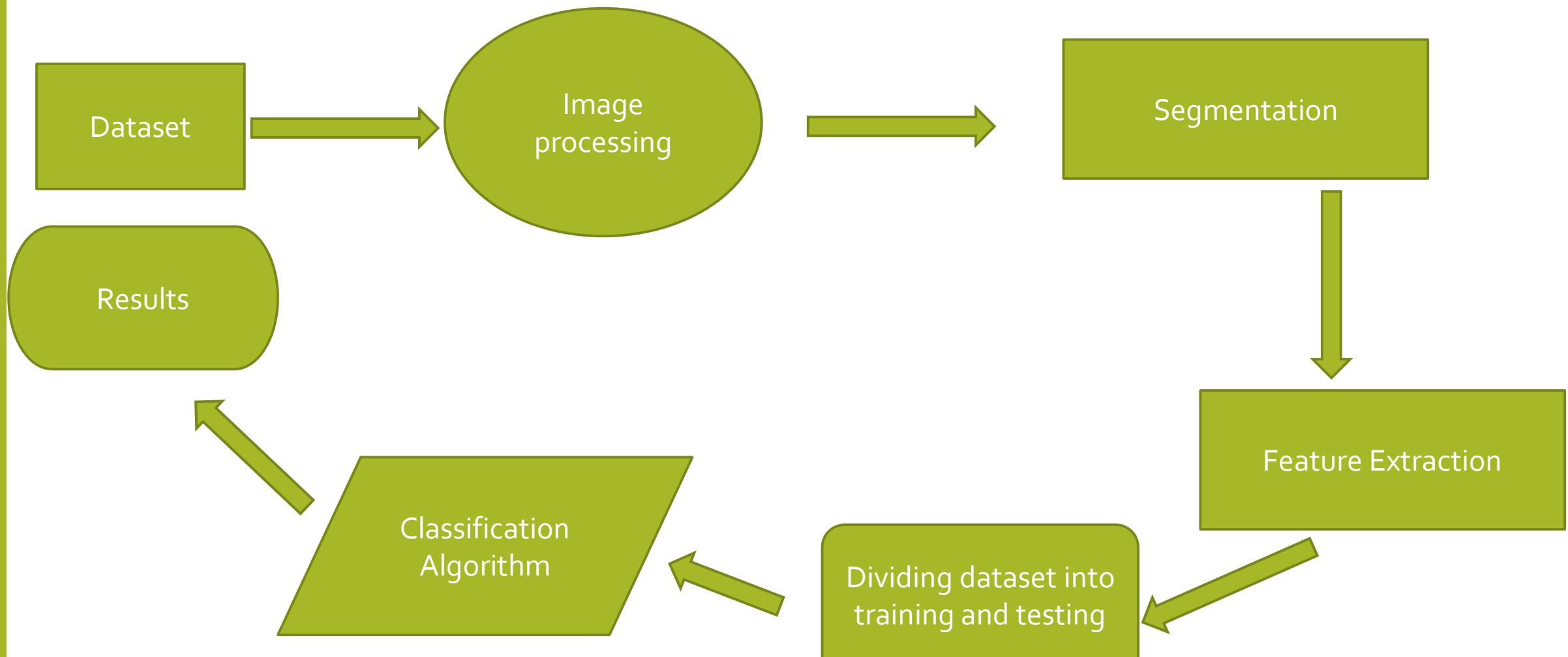
- 1. Classification using Logistic Regression**
- 2. Classification using Naïve Bayes**
- 3. Classification using Linear Regression**
- 4. Classification using K-Nearest Neighbors**
- 5. Classification using Random Forest**
- 6. Classification using Cat –Boost**
- 7. Classification using voting classifier**
- 8. Final Result and comparison with literature**



Problem Statement

- *How to develop a machine learning-based approach for accurate, non-invasive classification of Gliomas tumors?*

Flow of our Project



Logistic Regression

- Used Logistic Regression as the classifier.
- Applied BayesSearchCV from the skopt library for hyperparameter tuning.
- Achieved Train Accuracy: 85% and Test Accuracy: 84%.

Code:

```
!pip install scikit-optimize
from sklearn.linear_model import LogisticRegression
from skopt import BayesSearchCV
from sklearn.metrics import accuracy_score

# Initialize regression object; class_weight refers to the count ratio of each class per total class
logistic_regression = LogisticRegression(random_state = 42, class_weight = {0: df['Grade'].value_counts(normalize=True)[1],
                                                               1: df['Grade'].value_counts(normalize=True)[0]})

# Apply hyperparameter tuning
param = {
    'tol': [0.0001, 0.001, 0.01],
    'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
    'C': [0.5, 1, 1.5],
    'fit_intercept':[False, True],
}

bayes_search = BayesSearchCV(estimator=logistic_regression, search_spaces=param,
                             cv=10, n_jobs=-1, n_iter=30, scoring='roc_auc')
bayes_search.fit(x_train, y_train)
print("Parameter Terbaik:", bayes_search.best_params_)
```

Output:

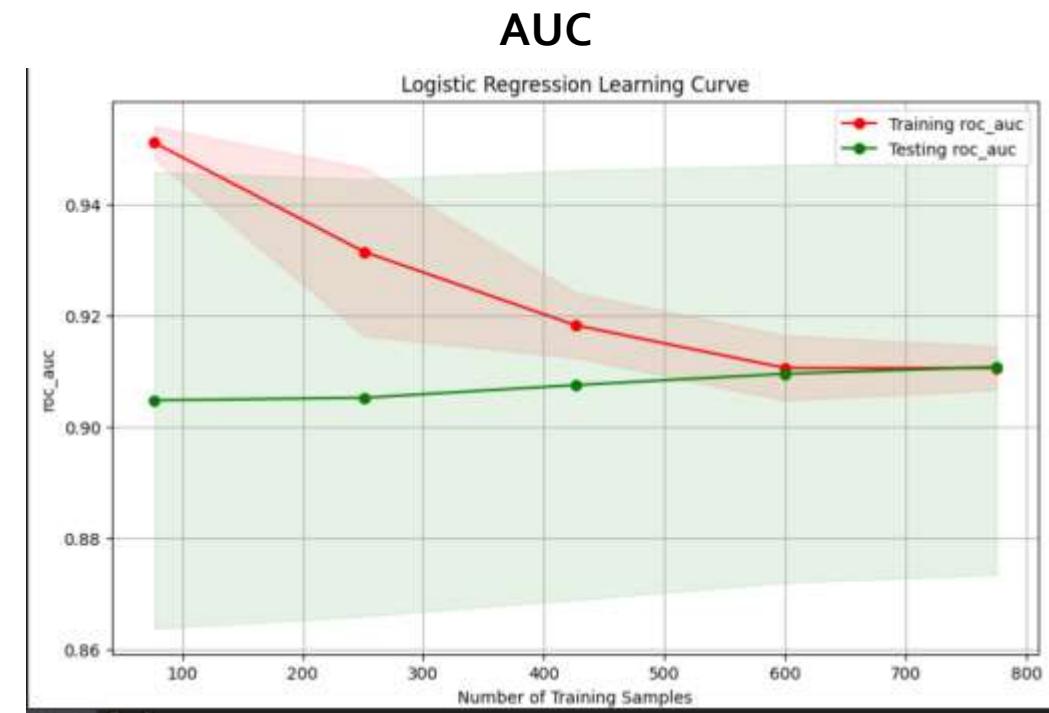
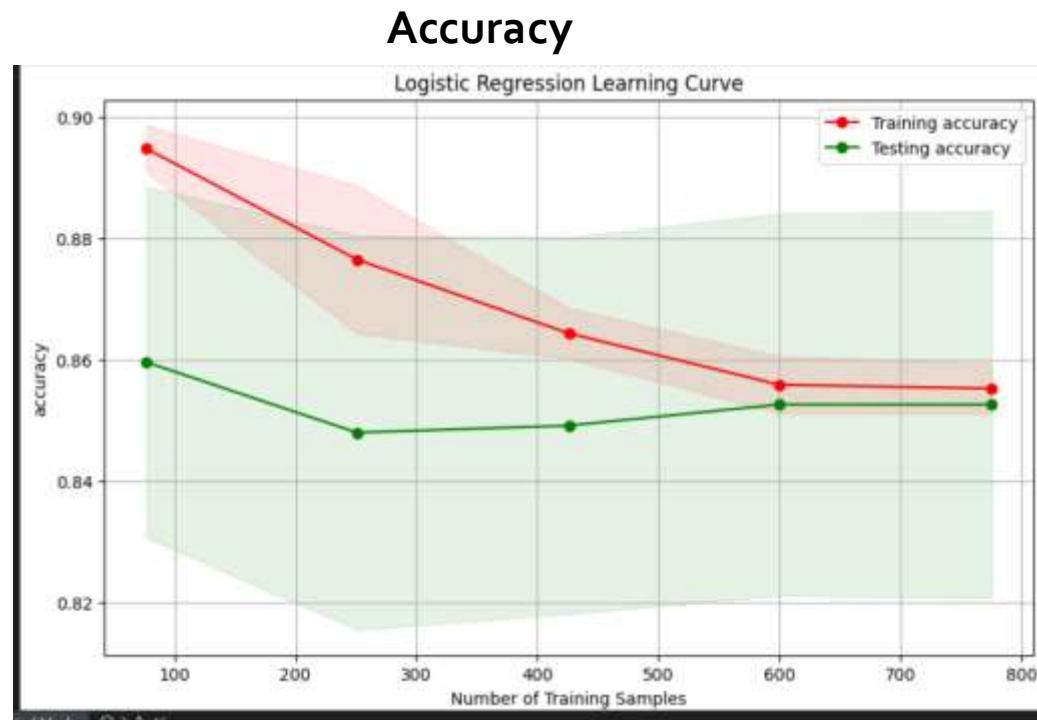
```
: warnings.warn("The objective has been e
Parameter Terbaik: OrderedDict([('C', 1.0),
Train Accuracy : 85.34107402031931 %
Test Accuracy : 84.97109826589595 %
```

Confusion Matrix and accuracy

```
True Positive: 67
True Negative: 80
False Positive: 20
False Negative: 6
```

```
Accuracy Data Test: 0.8497109826589595
Precision Data Test: 0.7701149425287356
Recall Data Test: 0.9178082191780822
F1-Score Data Test: 0.8375
```

ROC-AUC and Accuracy Graphs for LR



Naive Bayes

Code and output:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV

naive_model = GaussianNB()

# Apply hyperparameter tuning to find optimum other parameters
param = {'priors':[[0.03,0.97],[0.04,0.96],[0.05,0.95],[0.1,0.9]],
         'var_smoothing':[1e-14,1e-13,1e-12,1e-11,1e-10,1e-9,1e-8]
        }
naive_model = GaussianNB()
grid_search = GridSearchCV(estimator=naive_model, param_grid=param,
                           cv=10,n_jobs=-1,scoring='roc_auc')
grid_search.fit(x_train, y_train)
print("Parameter Terbaik:", grid_search.best_params_)

# Fit the model
naive_model = GaussianNB(**grid_search.best_params_)
naive_model.fit(x_train, y_train)

# Calculating accuracy
print(f"Train Accuracy : {accuracy_score(y_train, naive_model.predict(x_train))*100} %")
print(f"Test Accuracy : {accuracy_score(y_test, naive_model.predict(x_test))*100} %")

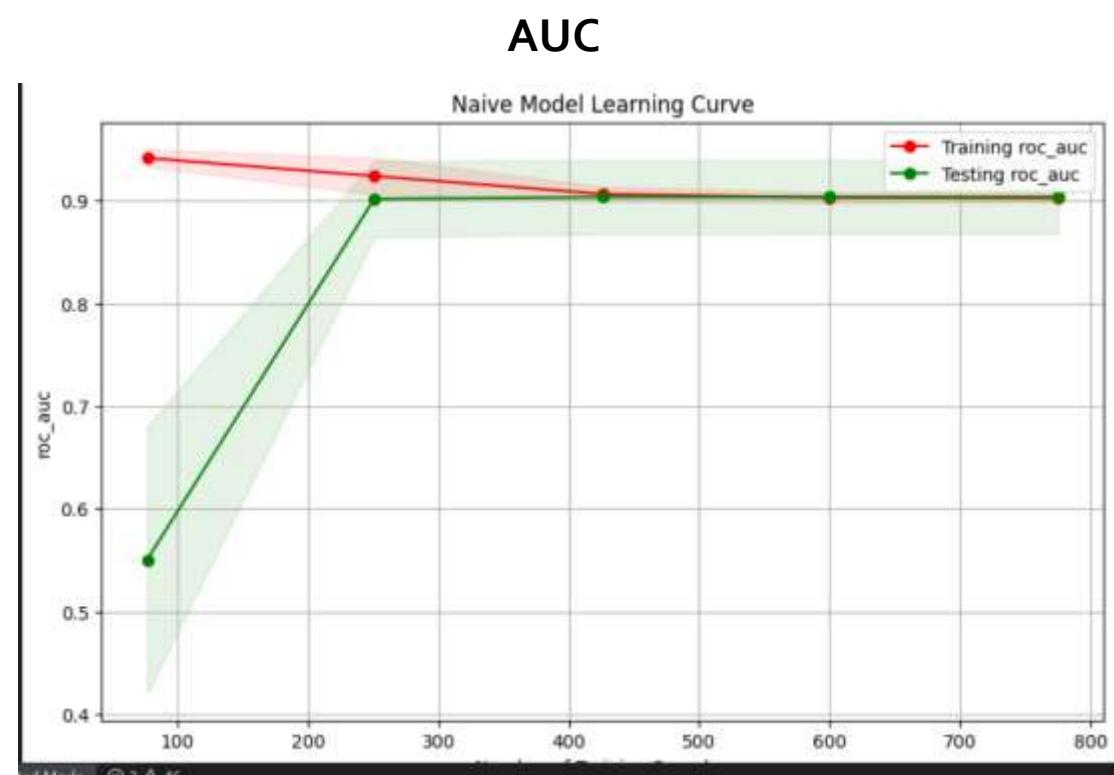
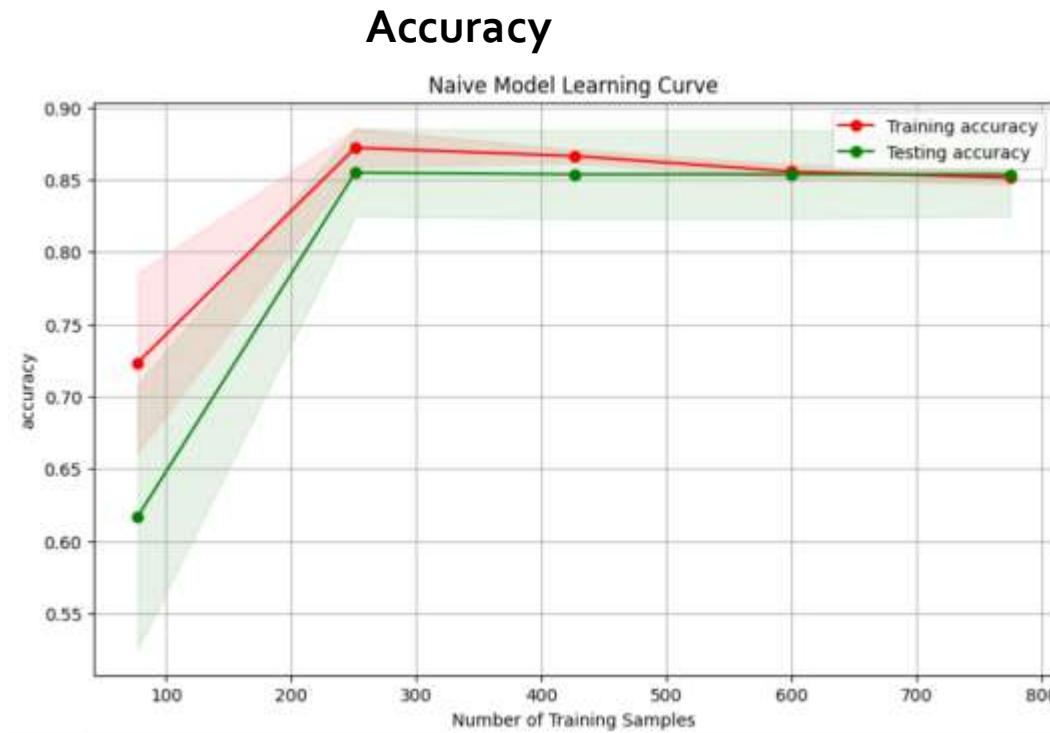
Parameter Terbaik: {'priors': [0.03, 0.97], 'var_smoothing': 1e-14}
Train Accuracy : 85.19593613933236 %
Test Accuracy : 84.97109826589595 %
```

Confusion Matrix and classification Report

True Positive: 66
True Negative: 81
False Positive: 19
False Negative: 7

Accuracy Data Test: 0.8497109826589595
Precision Data Test: 0.7764705882352941
Recall Data Test: 0.9041095890410958
F1-Score Data Test: 0.8354430379746836

ROC-AUC and Accuracy Graphs for Naïve Bayes



K-Nearest Neighbors

Code:

```
from sklearn.neighbors import KNeighborsClassifier

# Find optimum n_neighbor
error = []
for i in range(1,31,1):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    preds = knn.predict(x_test)
    error.append(np.mean(preds!=y_test))
n_neighbor = error.index(np.min(error))+1
print(f'optimum n_neighbor: {n_neighbor}')

# Apply hyperparameter tuning to find optimum other parameters
param = {
    'p':[1,0,1.5,2,0],
    'leaf_size': [20,30,40],
    'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],
}
knn = KNeighborsClassifier(n_neighbors=n_neighbor)
bayes_search = BayesSearchCV(estimator=knn, search_spaces=param,
                             cv=10,n_jobs=-1,n_iter=30,scoring='roc_auc')
bayes_search.fit(x_train, y_train)
print("Parameter Terbaik:", bayes_search.best_params_)

# Fit the model
knn = KNeighborsClassifier(**bayes_search.best_params_,n_neighbors=n_neighbor)
knn.fit(x_train, y_train)
```

Output:

```
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimize
  warnings.warn("The objective has been evaluated "
Parameter Terbaik: OrderedDict([('algorithm', 'auto'), ('leaf_
Train Accuracy : 87.08272859216255 %
Test Accuracy : 87.86127167630057 %
```

Confusion Matrix and classification report

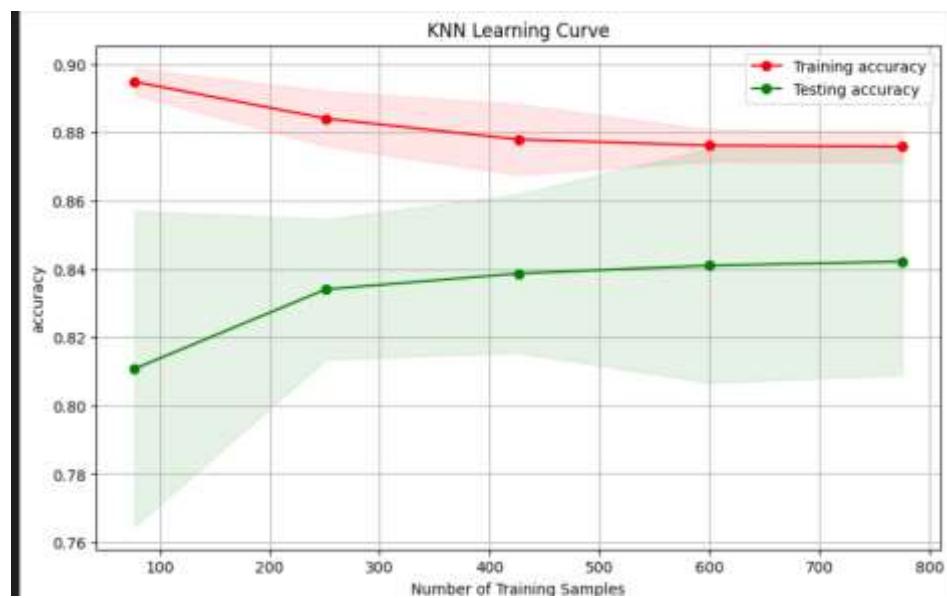
True Positive: 63
True Negative: 89
False Positive: 11
False Negative: 10

Accuracy Data Test: 0.8786127167630058
Precision Data Test: 0.8513513513513513
Recall Data Test: 0.863013698630137
F1-Score Data Test: 0.8571428571428572

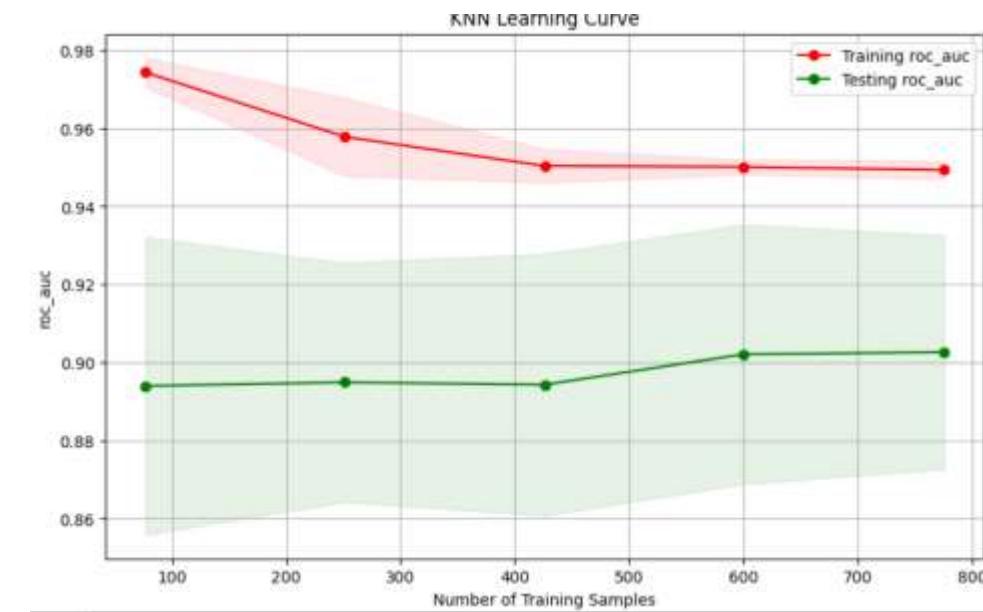
```
optimum n_neighbor: 7
/usr/local/lib/python3.10/dist-packages/skopt/optimizer/optimize
  warnings.warn("The objective has been evaluated "
```

ROC-AUC and Accuracy Graphs for KNN

Accuracy



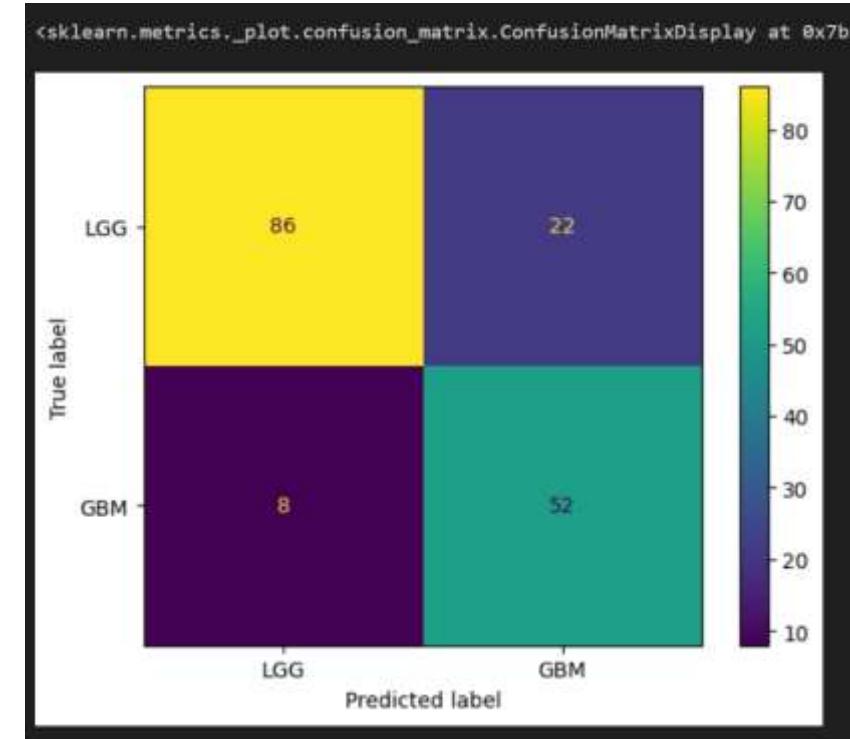
AUC



Random Forest Classifier

Code:

Confusion Matrix:



Accuracy

```
roc_auc = roc_auc_score(y_test,rfc_pred)  
  
print(roc_auc)  
  
0.8314814814814815
```

"Hyperparameter Tuning of Random Forest"

- we used ***GridSearchCV*** to tune the number of trees and depth in Random Forest. After tuning, the model improved by 2% in accuracy, which is significant for a classification task in a medical context.

Code:

```
from sklearn.model_selection import GridSearchCV

cv = GridSearchCV(rfc,parameters, cv=5)
cv.fit(X_train, y_train.values.ravel())

def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean, std, params in zip(mean_score, std_score, params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')

display(cv)
```

Output::

```
0.847 + or -0.023 for the {'max_depth': 2, 'n_estimators': 5}
0.856 + or -0.031 for the {'max_depth': 2, 'n_estimators': 10}
0.875 + or -0.034 for the {'max_depth': 2, 'n_estimators': 50}
0.87 + or -0.028 for the {'max_depth': 2, 'n_estimators': 100}
0.873 + or -0.029 for the {'max_depth': 2, 'n_estimators': 250}
0.855 + or -0.019 for the {'max_depth': 4, 'n_estimators': 5}
0.872 + or -0.03 for the {'max_depth': 4, 'n_estimators': 10}
0.876 + or -0.03 for the {'max_depth': 4, 'n_estimators': 50}
0.875 + or -0.026 for the {'max_depth': 4, 'n_estimators': 100}
0.87 + or -0.027 for the {'max_depth': 4, 'n_estimators': 250}
0.87 + or -0.027 for the {'max_depth': 8, 'n_estimators': 5}
0.86 + or -0.027 for the {'max_depth': 8, 'n_estimators': 10}
0.873 + or -0.028 for the {'max_depth': 8, 'n_estimators': 50}
0.873 + or -0.024 for the {'max_depth': 8, 'n_estimators': 100}
0.875 + or -0.026 for the {'max_depth': 8, 'n_estimators': 250}
0.85 + or -0.021 for the {'max_depth': 16, 'n_estimators': 5}
0.863 + or -0.02 for the {'max_depth': 16, 'n_estimators': 10}
0.861 + or -0.034 for the {'max_depth': 16, 'n_estimators': 50}
0.86 + or -0.033 for the {'max_depth': 16, 'n_estimators': 100}
0.864 + or -0.033 for the {'max_depth': 16, 'n_estimators': 250}
0.823 + or -0.036 for the {'max_depth': 32, 'n_estimators': 5}
0.83 + or -0.016 for the {'max_depth': 32, 'n_estimators': 10}
0.845 + or -0.028 for the {'max_depth': 32, 'n_estimators': 50}
0.855 + or -0.026 for the {'max_depth': 32, 'n_estimators': 100}
0.845 + or -0.026 for the {'max_depth': 32, 'n_estimators': 250}
0.836 + or -0.033 for the {'max_depth': None, 'n_estimators': 5}
0.85 + or -0.026 for the {'max_depth': None, 'n_estimators': 10}
0.849 + or -0.027 for the {'max_depth': None, 'n_estimators': 50}
0.845 + or -0.028 for the {'max_depth': None, 'n_estimators': 100}
0.842 + or -0.033 for the {'max_depth': None, 'n_estimators': 250}
```

We evaluated different combinations of *n_estimators* and *max_depth*. The best mean accuracy of 87.5% was achieved using 100 trees and a depth of 4. So we selected that parameter.

Accuracy after fine tuning:

```
roc_auc = roc_auc_score(y_test, rfc_pred)

print(roc_auc)
```

```
0.8518518518518517
```

Cat Boost -Results

CatBoost Library

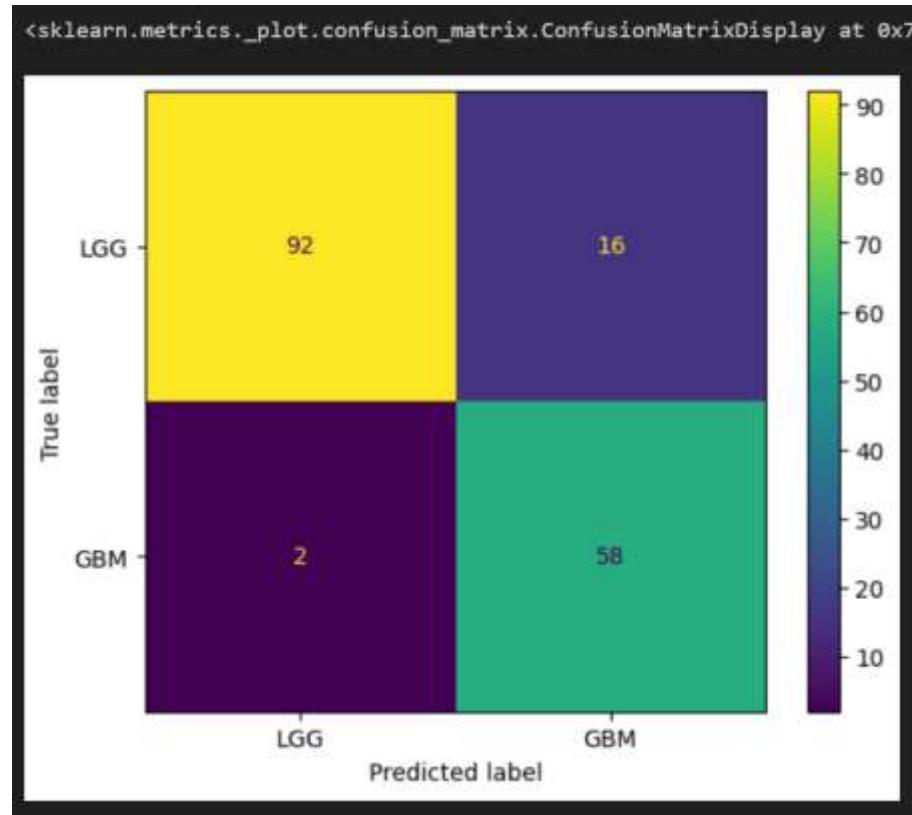
```
from catboost import CatBoostClassifier  
  
cbc = CatBoostClassifier()  
cbc.fit(X, y, verbose=10)
```

Accuracy :

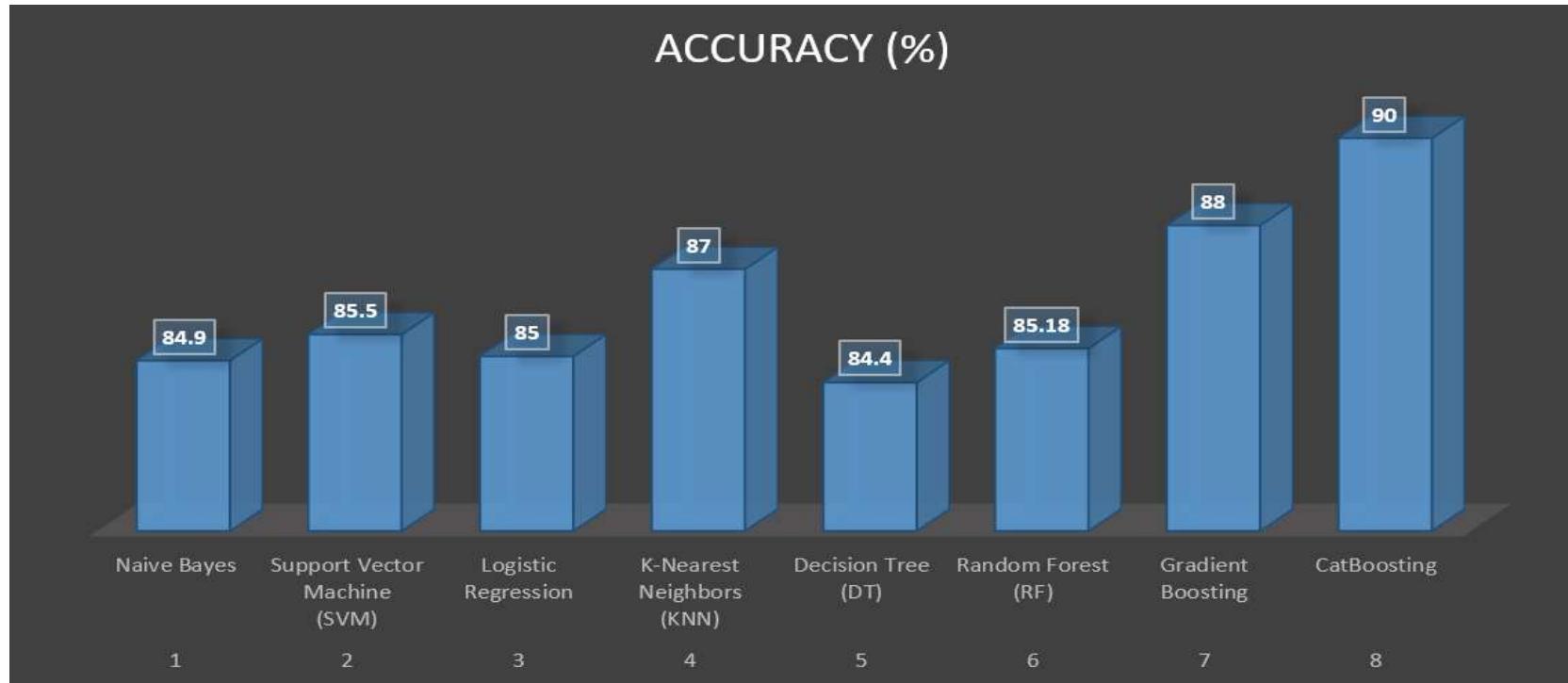
```
roc_auc = roc_auc_score(y_test, cbc_pred)  
  
print(roc_auc)
```

0.9092592592592593

Confusion Matrix



Final Result for each classifier



Among all classifiers, CatBoost performed very well in terms of accuracy!

Voting Classifier Model

What is voting classifier?

- ❖ A Voting Classifier in machine learning is an ensemble learning technique that combines the predictions of multiple different models to make a final prediction.
- ❖ It's like a team of experts voting to reach a consensus, with the goal of improving accuracy and robustness compared to using a single model alone.

Why combine classifiers and Why ensemble methods can outperform single models?

- ❖ Combining multiple classifiers, especially through ensemble methods, significantly improves model performance and robustness compared to using a single model alone.
- ❖ Ensembles leverage the collective strengths of diverse models to reduce errors, improve accuracy, and prevent overfitting.

How we achieved this task?

Code

```
seed = 1
lr = LogisticRegression(random_state=seed)
svm = SVC(random_state=seed, probability=True)
knn = KNeighborsClassifier()
rf = RandomForestClassifier(random_state=seed)
adaboost = AdaBoostClassifier(random_state=seed)

classifiers = [('lr', lr), ('svm', svm), ('knn', knn), ('rf', rf), ('adaboost', adaboost)]

voting_classifiers = []
for n in range(3, 6):
    for subset in combinations(classifiers, n):
        voting_classifier = VotingClassifier(estimators=list(subset), voting='soft')
        voting_classifiers.append(voting_classifier)

results_df = pd.DataFrame(columns=['Classifier', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
results_dff = pd.DataFrame(columns=['Classifier', 'TP', 'FP', 'FN', 'TN'])

precs = []
recs = []

for vc in voting_classifiers:
    vc.fit(X_train, y_train)
    y_pred = vc.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    prec, rec, _ = precision_recall_curve(y_test, y_pred)
    precs.append(prec)
    recs.append(recs)
    f1 = f1_score(y_test, y_pred)
    clf_names = [name for name, _ in vc.estimators]
    clf_names_str = ', '.join(clf_names)
    cm = confusion_matrix(y_test, y_pred)
    tp, fp, fn, tn = cm.ravel()
    results_df = results_df.append({'Classifier': clf_names_str, 'Accuracy': accuracy,
                                    'Precision': precision, 'Recall': recall, 'F1 Score': f1,
                                    ignore_index=True})
results_dff = results_dff.append({'Classifier': clf_names_str, 'TP': tp, 'FP': fp, 'FN': fn, 'TN': tn}, ignore_index=True)
```

Important Libraries

```
from itertools import combinations

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
                             roc_curve, roc_auc_score, confusion_matrix, precision_recall_curve)
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

Model Evaluation Results

Output:

```
results_df.style.format(precision=3)
```

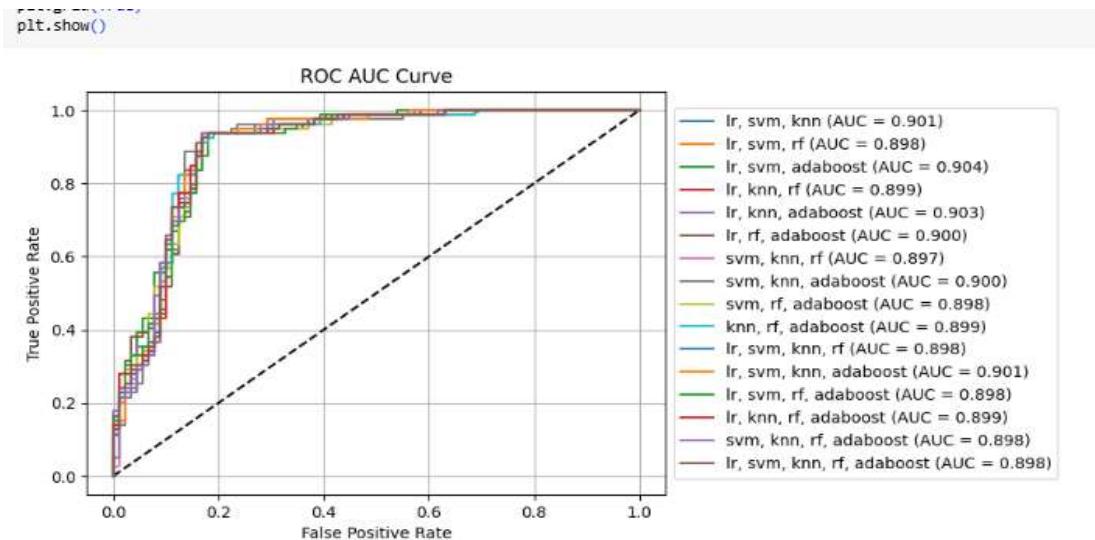
	Classifier	Accuracy	Precision	Recall	F1 Score
0	lr, svm, knn	0.869	0.820	0.924	0.869
1	lr, svm, rf	0.875	0.830	0.924	0.874
2	lr, svm, adaboost	0.869	0.820	0.924	0.869
3	lr, knn, rf	0.875	0.830	0.924	0.874
4	lr, knn, adaboost	0.875	0.822	0.937	0.876
5	lr, rf, adaboost	0.875	0.830	0.924	0.874
6	svm, knn, rf	0.875	0.830	0.924	0.874
7	svm, knn, adaboost	0.869	0.820	0.924	0.869
8	svm, rf, adaboost	0.863	0.826	0.899	0.861
9	knn, rf, adaboost	0.869	0.828	0.911	0.867
10	lr, svm, knn, rf	0.875	0.830	0.924	0.874
11	lr, svm, knn, adaboost	0.869	0.820	0.924	0.869
12	lr, svm, rf, adaboost	0.875	0.830	0.924	0.874
13	lr, knn, rf, adaboost	0.875	0.830	0.924	0.874
14	svm, knn, rf, adaboost	0.875	0.830	0.924	0.874
15	lr, svm, knn, rf, adaboost	0.875	0.830	0.924	0.874

Metrics	Formula
Precision	How many predicted positives were actually positive. = $TP / (TP + FP)$
Recall	How many actual positives were correctly predicted. = $TP / (TP + FN)$
F1 Score	Harmonic mean of Precision and Recall. = $2 * (P * R) / (P + R)$
Accuracy	Fraction of total predictions that were correct. = $(TP + TN) / \text{Total}$

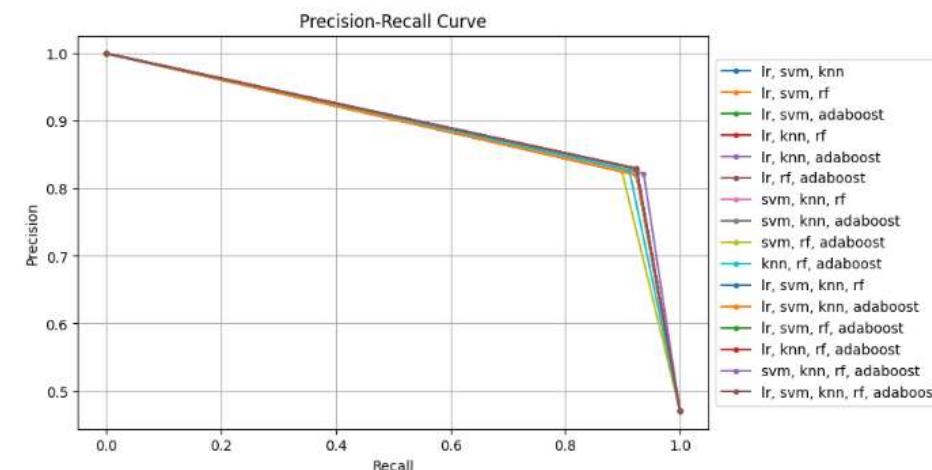
Confusion Matrix

```
results_dff
```

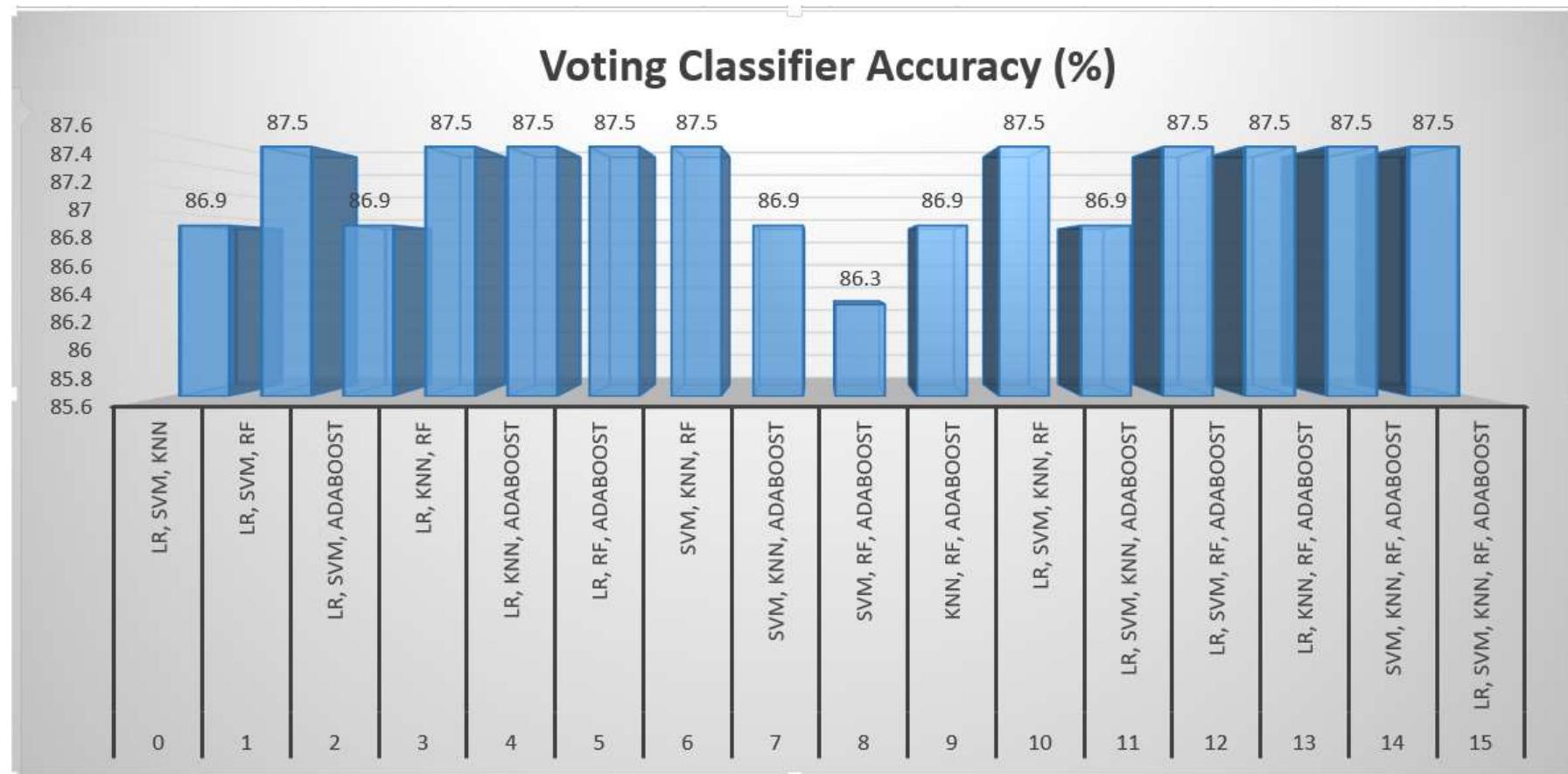
	Classifier	TP	FP	FN	TN
0	lr, svm, knn	73	16	6	73
1	lr, svm, rf	74	15	6	73
2	lr, svm, adaboost	73	16	6	73
3	lr, knn, rf	74	15	6	73
4	lr, knn, adaboost	73	16	5	74
5	lr, rf, adaboost	74	15	6	73
6	svm, knn, rf	74	15	6	73
7	svm, knn, adaboost	73	16	6	73
8	svm, rf, adaboost	74	15	8	71
9	knn, rf, adaboost	74	15	7	72
10	lr, svm, knn, rf	74	15	6	73
11	lr, svm, knn, adaboost	73	16	6	73
12	lr, svm, rf, adaboost	74	15	6	73
13	lr, knn, rf, adaboost	74	15	6	73
14	svm, knn, rf, adaboost	74	15	6	73
15	lr, svm, knn, rf, adaboost	74	15	6	73



```
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
plt.show()
```



Final Result for voting classifier



Comparison With literature

Name of classifier	Accuracy mentioned in paper	Accuracy achieved by our project
Random Forest	83%	85%
Naïve Bayes	72%	84%
Gradient Boosting	71%	88%
Support Vector Machine	82%	85%
Logistic Regression	90 %	85%
Decision Tree	88%	84%
KNN	86.5%	87%
CatBoosting	86%	90%

Resources used for Comparison

<https://PMC10305272>

<https://peerj.com/articles/5982/> (LR)

https://www.researchgate.net/publication/360123293_Machine_Learning_Models_for_Classifying_High_and_Low-Grade_Gliomas_A_Systematic_Review_and_Quality_of_Reportin Analysis (DT)

<https://www.ijitee.org/wp-content/uploads/papers/v8i8/H7543068819.pdf> (KNN)

https://PMC11282236/?utm_source=chatgpt.com
(CatBoosting)

Thank You