Hyperparameter tuning is the process of selecting the best hyperparameters for a machine learning algorithm to improve its performance.

In the context of machine learning algorithms, the parameters 'learning\_rate' and 'n\_estimators' control the behavior and performance of the models.

The learning rate determines the step size.  smaller learning rate makes the model converge slower but can potentially lead to better generalization. Conversely, a larger learning rate allows the model to converge faster but may result in overfitting. In the given example, the learning rate is set to a list of values: [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]. This means that the algorithm will be run multiple times, each time with a different learning rate, and the performance of the model will be evaluated for each value.

The number of estimators refers to the number of individual models (e.g., decision trees) in the ensemble. Each estimator contributes to the final prediction. Increasing the number of estimators generally improves the model's performance, but at the cost of increased computation time. In the given example, the number of estimators is set to a list of values: [50, 75, 100, 150, 200, 300]. Similar to the learning rate, the algorithm will be executed multiple times, each time with a different number of estimators, and the performance will be evaluated for each value.

from sklearn.ensemble import GradientBoostingClassifier

param\_grid = {

    'learning\_rate' : [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1],

    'n\_estimators': [50, 75,100, 150, 200,300],

}

gbc=RandomizedSearchCV(GradientBoostingClassifier(random\_state=42),param\_grid,cv=5)

gbc.fit(X\_train,y\_train)

y\_pred\_gbc=gbc.predict(X\_test)

confusion\_gbc=confusion\_matrix(y\_test,y\_pred\_gbc)

plt.figure(figsize=(8,8))

sns.heatmap(confusion\_gbc,annot=True)

plt.xlabel("Predicted")

plt.ylabel("Actual")

print(classification\_report(y\_test,y\_pred\_gbc))

print(gbc.best\_params\_)

Precision is the ratio of true positive predictions to the total number of positive predictions. For class 0 (negative class), the precision is 0.95, indicating that 95% of the predicted negative instances are actually negative. For class 1 (positive class), the precision is 0.33, suggesting that only 33% of the predicted positive instances are actually positive. These values indicate that the model performs well in predicting the negative class.Recall is the ratio of true positive predictions to the total number of actual positive instances. For class 0, the recall is 1.00, indicating that 100% of the actual negative instances are correctly identified. The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. The F1-score for class 0 is 0.97, indicating a good balance between precision and recall for the negative class.

param\_grid = {

'n\_estimators': [50, 75,100, 150, 200,300],

}

rcv=RandomizedSearchCV(RandomForestClassifier(random\_state=42),param\_grid,cv=5)

rcv.fit(X\_train,y\_train)

y\_pred\_rcv=rcv.predict(X\_test)

confusion\_rcv=confusion\_matrix(y\_test,rcv.predict(X\_test))

plt.figure(figsize=(8,8))

sns.heatmap(confusion\_rcv,annot=True)

plt.xlabel("Predicted")

plt.ylabel("Actual")

print(classification\_report(y\_test,y\_pred\_rcv))

from sklearn.model\_selection import KFold #for K-fold cross validation

from sklearn.model\_selection import cross\_val\_score #score evaluation

from sklearn.model\_selection import cross\_val\_predict #prediction

kfold = KFold(n\_splits=10, random\_state=2020, shuffle=True) # k=10, split the data into 10 equal parts

xyz=[]

accuracy=[]

std=[]

classifiers=['Logistic Regression',

             'SVC',

             'Random Forest',

             'XGB',

             'LGBM',

             'KNeighbors',

            'AdaBoost']

models=[LogisticRegression(),

        SVC(),

        RandomForestClassifier(),

        XGBClassifier(),

        LGBMClassifier(),

        KNeighborsClassifier(),

       AdaBoostClassifier()]

for i in models:

    model = i

    cv\_result = cross\_val\_score(model,X,y, cv = kfold,scoring = "accuracy")

    cv\_result=cv\_result

    xyz.append(cv\_result.mean())

    std.append(cv\_result.std())

    accuracy.append(cv\_result)

new\_models\_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)

new\_models\_dataframe2

plt.subplots(figsize=(18,10))

box=pd.DataFrame(accuracy,index=[classifiers])

box.T.boxplot();