|  |  |
| --- | --- |
| Name Of The Student | Himanshu |
| Internship Project Topic | TCS iON RIO-210: Build a Classification Model for Drug Trials Dataset |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Himdweep Walia |
| Name of the Institute | Amity University |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 06-06-2024 | Day-44 | 6 Hours |
| Activities done during the day:  **Project Hands-on – Hyper Parameter Tuning**  **Link of the google drive google Colab file :-**  <https://colab.research.google.com/drive/1VQRq0l6oc9Uj4cOOqiuhkfS1JmpKr3fU?usp=sharing>  **Hyper Parameter Tuning:**  Hyperparameter tuning for a classification model involves optimizing the hyperparameters of the chosen algorithm to enhance the model's performance in predicting categorical outcomes. Here's a step-by-step guide on how to perform hyperparameter tuning for a classification model:   * Choose a Classification Algorithm * Identify Hyperparameters * Define Search Space * Choose a Tuning Method   Grid Search  Random Search  Bayesian Optimization  Genetic Algorithms   * Split Data * Select Evaluation Metric * Perform Hyperparameter Tuning * Evaluate Results. * Refine and Iterate * Finalize Model     **HYPERPARAMETER TUNING**  Hyperparameter tuning played a pivotal role in refining the predictive capacity of the Random Forest Classifier and KNeighbours classifier. Employing RandomizedSearchCV, an extensive array of parameter combinations was explored, encompassing estimators, depth, sample splitting, leaf nodes, features, and bootstrapping. The optimal configuration emerged, featuring 200 estimators, minimum split of 2, leaf of 1, 'log2' features, unconstrained depth, and bootstrapping. This yielded an impressive 83.45% best score. Notably, this aligns with the model's initial performance, showcasing the meticulousness of the tuning process. This exercise demonstrates a nuanced understanding of the Random Forest algorithm, enhancing the model's predictive prowess for drug side effects. The fine-tuned model promises accurate predictions, bolstering the credibility of the predictive framework.  Utilizing the RandomizedSearchCV module and the K Neighbors Classifier from the scikit-learn library, a systematic exploration of hyperparameters was conducted. This involved varying factors such as the number of neighbors, distance weighting, algorithm type, and leaf size. Employing cross-validation for robustness, the optimal configuration was determined. The outcome revealed that the most effective setup consists of a 'distance' weight scheme, 9 neighbors, leaf size of 10, using the 'auto' algorithm, and a Euclidean distance metric (p=2). This configuration achieved a best score of 82.52%, enhancing the K Neighbors Classifier's performance for accurate classification of drug side effects.  **Code:**  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.neighbors import KNeighborsClassifier  # Define the parameter grid for K Neighbors Classifier  param\_grid = {      'n\_neighbors': [3, 5, 7, 9],      'weights': ['uniform', 'distance'],      'algorithm': ['auto', 'ball\_tree', 'kd\_tree', 'brute'],      'leaf\_size': [10, 20, 30, 40],      'p': [1, 2]}  # Create the K Neighbors Classifier model  model = KNeighborsClassifier()  # Create RandomizedSearchCV instance  random\_search = RandomizedSearchCV(model, param\_distributions=param\_grid, n\_iter=50, cv=5, random\_state=42, n\_jobs=-1)  # Fit the random search to the training data  random\_search.fit(X\_train, y\_train)  # Get the best parameters and best score  best\_params = random\_search.best\_params\_  best\_score = random\_search.best\_score\_  # Print the best parameters and best score  print("Best Parameters:", best\_params)  print("Best Score:", best\_score)  **OUTPUT:**  Best Parameters: {'weights': 'distance', 'p': 2, 'n\_neighbors': 9, 'leaf\_size': 10, 'algorithm': 'auto'}  Best Score: 0.825219678105918  **Second tunning with different parameters.**  param\_grid = {      'n\_estimators': np.arange(100, 501, 100),      'max\_depth': [None, 5, 10, 20],      'min\_samples\_split': np.arange(2, 11, 2),      'min\_samples\_leaf': np.arange(1, 11, 2),      'max\_features': ['auto', 'sqrt', 'log2'],      'bootstrap': [True, False]  }  # Create the model  model = RandomForestClassifier(random\_state=42)  # Create RandomizedSearchCV instance  random\_search = RandomizedSearchCV(model, param\_distributions=param\_grid, n\_iter=50, cv=5, random\_state=42, n\_jobs=-1)  # Fit the random search to the training data  random\_search.fit(X\_train, y\_train)  # Get the best parameters and best score  best\_params = random\_search.best\_params\_  best\_score = random\_search.best\_score\_  # Print the best parameters and best score  print("Best Parameters:", best\_params)  print("Best Score:", best\_score)  **OUTPUT:**  Best Parameters: {'n\_estimators': 200, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2', 'max\_depth': None, 'bootstrap': True}  Best Score: 0.83445625 | | |
|  | | |