|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Decision Tree and Random Forest Classifications - Modifications | | | | | | | |
|  | **Section** | **Change** | **Previous code** | **New code** | **Description** | **Level of Difficulty (1-10)** | **Time taken** |
| 1 |  | changed style | plt.style.use('seaborn-whitegrid') | plt.style.use('seaborn-v0\_8-whitegrid') | Code gave me an error because of 'plt.style.use('seaborn-whitegrid')' part. Then I checked available styles in pyplot and identified that package called 'seaborn-whitegrid' is not in the list. Instead there was 'seaborn-v0\_8-whitegrid'. Then I replaced previous code with this new package and made the code worked. | 4 | 15 minutes |
| 2 | Creating a Decision Tree | Spliting synthetic dataset into 3 parts |  | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42)  X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42) | Before training decision tree, dataset was splitted to 3 parts. Training - 60%, Validation - 20%, Testing - 20%. This step is essential to validate model performance. For this 'train\_test\_split' package was imported | 5 | 30 minutes |
| 3 | Creating a Decision Tree | Training the model only using the training dataset | tree = DecisionTreeClassifier().fit(X, y) | tree = DecisionTreeClassifier().fit(X\_train, y\_train) | Train the model only using the training set created in the previous step | 5 | 5 minutes |
| 4 | Creating a Decision Tree | Evaluate model on validation set |  | val\_score = tree.score(X\_val, y\_val) | Model was evaluated on validating set to check whether further adjustment is needed or not. Here accuraccy showed 95% and I decided not to do any other adjustments | 7 | 20 minutes |
| 5 | Creating a Decision Tree | Evaluate model on testing set |  | test\_score = tree.score(X\_test, y\_test) | Finally evaluated the model on testing set. Accuracy was 88.3% which is good | 5 | 5 minutes |
| 6 | Creating a Decision Tree | Stopped retraining model in 'visualize\_classifier' function | model.fit(X, y) | Removed | I removed 'model.fit(X, y)' part as the model gets retrained due to this line of code | 9 | 30 minutes |
| 7 | Creating a Decision Tree | Visualise data only related to the training set | visualize\_classifier(DecisionTreeClassifier(), X, y) | visualize\_classifier(tree, X\_train, y\_train) | Adjusted 'visualize\_classifier' according to the parameters I used. (only used training dataset) | 6 | 10 minutes |
| 8 | Creating a Decision Tree | Include modified helper script as a python code |  |  | Changed helper script into a python code and adjust it to use only training data. | 10 | 40 minutes |
| 9 | Creating a Decision Tree | Remove dependancy to the helper script | helpers\_05\_08.plot\_tree\_interactive(X, y) | plot\_tree\_interactive(X\_train, y\_train) | I changed this code line as I created a code to use instead of helper scripts. This visualises decision boundaries only considering training dataset. | 6 | 5 minutes |
| 10 | Decision Trees and Overfitting | Remove dependancy to the helper script | helpers\_05\_08.randomized\_tree\_interactive(X, y) | randomized\_tree\_interactive(X\_train, y\_train) | This part also executed without depending on helper script. This provides randomized subsets only using training dataset. | 6 | 5 minutes |
| 11 | Ensembles of Estimators: Random Forests | Fit and visualise BaggingClassifier only to training dataset | bag.fit(X, y)  visualize\_classifier(bag, X, y) | bag.fit(X\_train, y\_train)  visualize\_classifier(bag, X\_train, y\_train) | This keeps consistency across all models (Decision Tree, Bagging, and Random Forest all use the same splits) | 5 | 10 minutes |
| 12 | Ensembles of Estimators: Random Forests | Fitted traing dataset to Random Forest Classifier and visualised | visualize\_classifier(model, X, y) | model.fit(X\_train, y\_train)  visualize\_classifier(model, X\_train, y\_train) | Added 'model.fit(X\_train, y\_train)' to the code as it wasn't there in the original version. | 5 | 5 minutes |
| 13 | Example: Random Forest for Classifying Digits | Assigned values to X,y correctly |  | X, y = digits.data, digits.target | Code gave me an error when splitting it. Then I checked the shape of x,y and found out those were not balanced. So this line was added before splitting data in this section | 8 | 20 minutes |
| 14 | Example: Random Forest for Classifying Digits | Improved data split | Xtrain, Xtest, ytrain, ytest = train\_test\_split(digits.data, digits.target, random\_state=0) | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)  X\_valid, X\_test, y\_valid, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42, stratify=y\_temp) | Previous data split was a basic split and it will be randomized in each split as random\_state=0  New split divids dataset into 3 parts (70%, 15%, 15%) and division will be same in every execution as random\_state=42 | 7 | 15 minutes |
| 15 | Example: Random Forest for Classifying Digits | Improved Random Forest creation and fit the model | model = RandomForestClassifier(n\_estimators=1000)  model.fit(Xtrain, ytrain) | model = RandomForestClassifier(n\_estimators=200, max\_depth=10, random\_state=42)  model.fit(X\_train, y\_train) | Code was modified to avoid overfitting and improve the training speed. n\_estimators=200 was added to speed up the code, max\_depth=10 as added to overfitting, random\_state=42 was added to make training process is reproducible | 8 | 10 minutes |
| 16 | Example: Random Forest for Classifying Digits | Evaluate model for validation dataset | ypred = model.predict(Xtest) | y\_valid\_pred = model.predict(X\_valid) | Previous code is directly evaluates test dataset as it doesn't have validation set. Here I first evaluated validation set before evaluating my test dataset | 5 | 5 minutes |
| 17 | Example: Random Forest for Classifying Digits | Classification report for validation set | print(metrics.classification\_report(ypred, ytest)) | print(metrics.classification\_report(y\_valid, y\_valid\_pred)) | Classification report was generated using validation test instead of training set. Accuracy on validation set is 96% | 5 | 5 minutes |
| 18 | Example: Random Forest for Classifying Digits | Evaluate model on Testing dataset |  | y\_test\_pred = model.predict(X\_test)  print(metrics.classification\_report(y\_test, y\_test\_pred)) | Since the model is accurate on validation set, model was evaluated on test set. Then checked classification report and found that model is performing well | 5 | 5 Minutes |
| 19 | Example: Random Forest for Classifying Digits | Confusion matrix using testing dataset | mat = confusion\_matrix(ytest, ypred) | mat = confusion\_matrix(y\_test, y\_test\_pred) | Confusion matrix also generated to evaluate model performance usind test dataset | 5 | 5 minutes |

**My Example**

**Application of Decision Tree and Random Forest - Iris dataset**

The Iris dataset consists of flower measurements (sepal length, sepal width, petal length, petal width) and their corresponding species (Setosa, Versicolor, or Virginica). Here I'm trying to classify iris flowers into the correct species.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Purpose | Code | Description | Level of difficulty (1-10) | Time spent |
| 1 | Importing Libraries | from sklearn.datasets import load\_iris | For this ‘load\_iris’ was imported from ‘sklearn.datasets’. All libraries have included at the beginning of the notebook to avoid redundancy. | 3 | 5 minutes |
| 2 | Loading the dataset and making a data frame | iris = load\_iris()  df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)  df['target'] = iris.target | In this step dataset has loaded and made it into a data frame. Then creates a new column named "target" in the df and assigns the values from iris.target to it. | 6 | 10 minutes |
| 3 | Split into features (X) and target (y) | X = df.drop(columns=['target'])  y = df['target'] | Split into features and target. Features (x) contain all the variables except the column, which represents the species classification labels. Those data related to species was stored as y which is target. | 7 | 10 minutes |
| 4 | Train -test split | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)  X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42, stratify=y\_temp) | Dataset has been split into 3 parts. 70% training (X\_train, y\_train), 15% validation (X\_val, y\_val) and 15% testing (X\_test, y\_test). Here ‘random\_state=42’ used to ensure the reproducibility of the datasets in each execution. Stratify=y parameter ensures that the train and test datasets have the same class distribution as the original dataset. | 5 | 5 minutes |
| 5 | Train and predict the Decision Tree Classifier | dt\_model = DecisionTreeClassifier(random\_state=42)  dt\_model.fit(X\_train, y\_train)  y\_pred\_dt = dt\_model.predict(X\_val) | First, initialised a decision tree model for classification. Then trained the decision tree classifier using X\_train and Y\_train which is 70% from the full dataset. Here also ‘random\_state=42’ used to ensure the reproducibility of the datasets in each execution. Then predicted the trained model using X\_val. | 7 | 10 minutes |
| 6 | Train and predict the Random Forest Classifier | rf\_model = RandomForestClassifier(n\_estimators=500, random\_state=42)  rf\_model.fit(X\_train, y\_train)  y\_pred\_rf = rf\_model.predict(X\_val) | Similarly, Random Forest classifier also initiated and trained using X\_train and Y\_train. Here ‘n\_estimators=500’ has used to indicate number of decision trees in the random forest model. This means the model will create 500 individual decision trees. ‘random\_state=42’ has also been used here as well. | 6 | 10 minutes |
| 7 | Compare accuracy | dt\_accuracy = accuracy\_score(y\_val, y\_pred\_dt)  rf\_accuracy = accuracy\_score(y\_val, y\_pred\_rf)  print(f"Decision Tree Accuracy (Validation Set): {dt\_accuracy:.4f}")  print(f"Random Forest Accuracy (Validation Set): {rf\_accuracy:.4f}") | Accuracies of two models has been checked considering validation data set which is 15% from the total dataset. Found that accuracy of the decision tree is 95% and accuracy of random forest is 86%. For this ‘accuracy\_score’ was imported from sklearn.metrics. | 6 | 5 minutes |
| 8 | Hyperparameter Tuning - Define the parameter grid | param\_grid = {  'n\_estimators': [100, 300, 500],  'max\_depth': [None, 10, 20, 30],  'min\_samples\_split': [2, 5, 10],  'min\_samples\_leaf': [1, 2, 4],  'max\_features': ['sqrt', 'log2']  } | Decided to perform hyperparameter tuning for a Random Forest classifier as a optimization step. First, the parameter grid was defined. 'n\_estimators' is Number of trees. 'max\_depth' is Depth of each tree. 'min\_samples\_split' is Minimum samples to split a node. 'min\_samples\_leaf' is Minimum samples per leaf and 'max\_features' is Number of features per split. | 8 | 15 minutes |
| 9 | Perform Hyperparameter Tuning | rf\_model = RandomForestClassifier(random\_state=42)  grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, scoring='accuracy', cv=5, n\_jobs=-1, verbose=2)  grid\_search.fit(X\_train, y\_train) | First initialised Random Forest model and performed Grid Search with 5-fold cross-validation. This splits the dataset into 5 subsets and train the gridsearch using 4 subsets among them. This repeats 5 times changing training and testing data subsets. Then fit GridSearchCV to the training data. This part takes considerable time. | 9 | 20 minutes |
| 10 | Identifying the best Gridsearch parameters | best\_params = grid\_search.best\_params\_  print("Best Hyperparameters:", best\_params) | This code prints the best parameters obtained from the grid search cv as below:  Best Hyperparameters: {'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_leaf': 4, 'min\_samples\_split': 2, 'n\_estimators': 100} | 7 | 5 minutes |
| 11 | Train and predict the Random Forest model with optimized parameters | best\_rf = RandomForestClassifier(  n\_estimators=100,  max\_depth=None,  max\_features='sqrt',  min\_samples\_split=2,  min\_samples\_leaf=4,  random\_state=42  )  best\_rf.fit(X\_train, y\_train) | After getting optimal parameters using gridsearch method, Random Forest classification model can be trained again with optimal parameters using X\_train and y\_train. Then predicted the model using validation set and tested the accuracy. | 6 | 5 minutes |
| 12 | Predict on validation set | y\_pred\_best\_rf = best\_rf.predict(X\_val)  best\_rf\_accuracy = accuracy\_score(y\_val, y\_pred\_best\_rf)  print(f"Optimized Random Forest Accuracy (Validation Set): {best\_rf\_accuracy:.4f}") | Predicted the optimal model using X\_val dataset and evaluated accuracy score. | 6 | 5 minutes |
| 13 | Evaluate Random Forest and decision tree on Test Dataset | y\_test\_best\_rf = best\_rf.predict(X\_test)  best\_rf\_accuracy = accuracy\_score(y\_test, y\_test\_best\_rf)  print(f"Optimized Random Forest Accuracy (Validation Set): {best\_rf\_accuracy:.4f}")  dt\_test\_acc = accuracy\_score(y\_test, dt\_model.predict(X\_test))  print(f"Decision Tree Accuracy (Test Set): {dt\_test\_acc:.4f}") | Since validation set provided higher accuracy after hyperparameter tunning, Random Forest classification model was evaluated on test dataset. Also, decision tree model evaluated on test dataset to compare the performance. | 7 | 10 minutes |
| 14 | Combine split datasets | X\_final\_train = pd.concat([X\_train, X\_val])  y\_final\_train = pd.concat([y\_train, y\_val]) | This code combines all training, validation and testing datasets in order to train the final model. | 5 | 5 minutes |
| 15 | Train and predict the final model | final\_rf = RandomForestClassifier(  n\_estimators=100,  max\_depth=None,  max\_features='sqrt',  min\_samples\_split=2,  min\_samples\_leaf=4,  random\_state=42  )  final\_rf.fit(X\_final\_train, y\_final\_train)  y\_test\_pred = final\_rf.predict(X\_test) | Final Random Forest model was trained using optimal parameters. And predicted it using X\_test | 7 | 10 minutes |
| 16 | Calculate accuracy | final\_accuracy = accuracy\_score(y\_test, y\_test\_pred)  print(f"Final Random Forest Accuracy (Test Set): {final\_accuracy:.4f}")  print("Classification Report:")  print(classification\_report(y\_test, y\_test\_pred))  print("Confusion Matrix:")  print(confusion\_matrix(y\_test, y\_test\_pred)) | To evaluate the final model, Accuracy score, Classification report and Confusion matrix were generated. Accuracy of this final model is 96%. | 6 | 5 minutes |