**Random Forest Classifier Documentation**

**1. Overview**

**This project aims to build a Breast Cancer Detection Model using a Random Forest Classifier. The model predicts whether a tumor is Benign or Malignant based on various diagnostic features.**

**2. Objective**

**To develop an accurate and robust machine learning model that can classify breast cancer tumors using diagnostic features such as:**

* **Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Symmetry, and Fractal Dimension (Mean, Standard Error, and Worst values for each feature).**

**3. Why Random Forest?**

* **Ensemble Learning: Combines multiple decision trees to improve accuracy and robustness.**
* **Reduces Overfitting: Averages multiple trees, reducing the variance.**
* **Feature Importance: Provides insight into the most influential features.**
* **High Accuracy: Generally performs well on complex classification tasks.**

**4. Dataset Information**

* **Source: Breast Cancer Wisconsin (Diagnostic) Dataset**
* **Target Variable: diagnosis (0 = Benign, 1 = Malignant)**

**Features: 30 numeric features derived from digitized images of breast mass.**

**5. Data Preprocessing**

1. **Dropping Unnecessary Columns:**

**data = data.drop(columns=['id', 'Unnamed: 32'])**

* + **Removed id (unique identifier) and Unnamed: 32 (empty column) to avoid noise.**

1. **Label Encoding:**

**data['diagnosis'] = LabelEncoder().fit\_transform(data['diagnosis'])**

* + **Encoded diagnosis from categorical (B, M) to numerical (0, 1).**

1. **Feature Scaling:**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

* + **Standardized features for better performance of the model.**

1. **Train-Test Split:**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)**

* + **Split the data into 80% training and 20% testing sets.**
  + **random\_state=42 ensures reproducibility.**

**6. Model Training**

**Random Forest Classifier Initialization**

**rf\_model = RandomForestClassifier(**

**n\_estimators=200,**

**max\_depth=10,**

**min\_samples\_split=5,**

**min\_samples\_leaf=2,**

**random\_state=42**

**)**

* **n\_estimators=200: Number of trees in the forest.**
* **max\_depth=10: Limits the depth of each tree to avoid overfitting.**
* **min\_samples\_split=5: Minimum samples required to split an internal node.**
* **min\_samples\_leaf=2: Minimum samples required to be at a leaf node.**
* **random\_state=42: Ensures consistent results.**

**Training the Model**

**rf\_model.fit(X\_train, y\_train)**

* **The model is trained on the training set using the selected hyperparameters.**

**7. Model Evaluation**

**Performance Metrics**

* **Classification Report: Displays Precision, Recall, F1-Score, and Support.**
* **Accuracy Score: Percentage of correctly predicted instances.**

**y\_pred\_rf = rf\_model.predict(X\_test)**

**print("\nRandom Forest Model Results:")**

**print(classification\_report(y\_test, y\_pred\_rf))**

**print("Accuracy:", accuracy\_score(y\_test, y\_pred\_rf))**

**ROC Curve and AUC Score**

* **ROC Curve: Plots True Positive Rate (TPR) vs False Positive Rate (FPR).**
* **AUC Score: Measures the area under the ROC curve, indicating model performance.**

**def plot\_roc\_curve(model, model\_name, X\_test, y\_test):**

**y\_pred\_prob = model.predict\_proba(X\_test)[:, 1]**

**fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_prob)**

**roc\_auc = auc(fpr, tpr)**

**plt.plot(fpr, tpr, label=f'{model\_name} (AUC = {roc\_auc:.2f})')**

**8. Hyperparameter Tuning**

**Purpose:**

* **To optimize model performance and reduce overfitting by selecting the best hyperparameters.**

**Grid Search Implementation**

**rf\_param\_grid = {**

**'n\_estimators': [100, 200, 300],**

**'max\_depth': [5, 10, 20],**

**'min\_samples\_split': [2, 5, 10],**

**'min\_samples\_leaf': [1, 2, 4]**

**}**

**grid\_search = GridSearchCV(**

**RandomForestClassifier(random\_state=42),**

**rf\_param\_grid,**

**cv=5,**

**scoring='accuracy',**

**n\_jobs=-1**

**)**

**grid\_search.fit(X\_train, y\_train)**

**best\_model = grid\_search.best\_estimator\_**

**Explanation:**

* **cv=5: 5-fold cross-validation to ensure robustness.**
* **scoring='accuracy': Optimizes for maximum accuracy.**
* **n\_jobs=-1: Utilizes all CPU cores for faster computation.**
* **best\_estimator\_: Returns the model with the best hyperparameters.**

**9. Feature Importance**

* **Evaluates the contribution of each feature in making predictions.**

**importances = rf\_model.feature\_importances\_**

**feature\_names = X.columns**

**feature\_importance\_df = pd.DataFrame({'Feature': feature\_names, 'Importance': importances})**

**feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)**

**print(feature\_importance\_df.head(10))**

* **Helps in understanding which features are most influential.**

**10. Model Deployment**

* **Model Saving:**
* **joblib.dump(rf\_model, 'breast\_cancer\_rf\_model.pkl')**
  + **Saves the trained model for future use.**
* **Model Loading and Prediction:**
* **loaded\_model = joblib.load('breast\_cancer\_rf\_model.pkl')**
* **prediction = loaded\_model.predict(new\_data)**
  + **Enables predictions on new, unseen data.**