Codes and other relevant explanations for supervised learning (Part 2)

Decision Tree and Random Forest:

GitHub link for the code:

https://github.com/Sabyasachi123276/ML-using-Python/blob/main/DecisionTree_RandomForest_SupervisedLearning_(Part_2).ipynb

Iris Dataset Description:

The Iris dataset was used in R.A. Fisher's classic 1936 paper. It can also be found in the UCI Machine Learning Repository. It includes three iris species with 50 samples each: Iris setosa, Iris versicolor, and Iris virginica, as well as some properties of each flower. Iris Setosa has index values of 0-49, Iris Versicolor has index values of 50-99, and Iris Virginica has index values of 100-149.

NumPy :

- It is a numeric Python module that provides fast math functions for calculations.
- It is used to read data in NumPy arrays and for manipulation purposes.

Pandas :

- It is used to read and write different files.
- Data manipulation can be done easily with data frames.

scikit-learn / sklearn:

- In Python, sklearn is a machine learning package that includes a lot of ML algorithms.
- Here, we are using some of its modules like train_test_split, DecisionTreeClassifier, and accuracy_score.

Matplotlib:

Matplotlib is a visualization library in Python. It is built on NumPy arrays and consists of several plots like line, bar, scatter, histogram, etc.

Pyplot:

Pyplot makes Matplotlib work like Matlab.

Seaborn:

Seaborn is a library mainly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.

random_state=0 : With random_state=0, we get the same train and test sets across different executions.

DecisionTreeClassifier: A Decision Tree Classifier is a type of algorithm that uses a tree-like structure to classify instances based on their feature values.

RandomForestClassifier: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

n_estimators : The number of trees in the forest.

classifier.fit(): The learning algorithm / estimator instance is first fitted to the model, i.e., it must *learn* from the model. This is done by passing our training set to the fit method.

classifier.predict(): The predict() will perform a prediction for each test instance, and it usually accepts only a single input.

confusion_matrix: Confusion matrix represents the prediction summary in matrix form. It shows how many predictions are correct and incorrect per class.

seaborn.heatmap() / sns.heatmap(): Plot rectangular data as a color-encoded matrix.

Parameters

(i) data: Here, the data input is the 'result'.

(ii) annot: bool or rectangular dataset, optional.

If True, write the data value in each cell. If an array-like object has the same shape as data, then use this to annotate the heatmap instead of the data. Note that DataFrames will match on position, not index.

In this example, in each confusion matrix, correctly and incorrectly classified test sample numbers are shown.

(iii) fmt: string formatting code to use when adding annotations.

G or g (General Format) suffix in Python: It can be interpreted as the total number of digits before the decimal; otherwise, it represents the total number of zeros after the decimal point but before the significant digits.

In this example, fmt = 'g' is used.

(iv) xticklabels, yticklabels: Here, sequentially, three classes, 'Setosa', 'Versicolor', and 'Virginica' are marked.

classification_report: Build a text report showing the main classification metrics.

(i) Precision: TP / (TP + FP)

(ii) Recall / Sensitivity: TP / (TP + FN)

(iii) F1 Score = (2 * Precision * Recall) / (Precision + Recall)

True Positive (TP): A true positive is an outcome where the model correctly predicts the positive class.

True Negative (TN): A true negative is an outcome where the model correctly predicts the negative class.

False Positive (FP): A false positive is an outcome where the model incorrectly predicts the positive class.

False Negative (FN): A false negative is an outcome where the model *incorrectly* predicts the *negative* class.

Support: Support is a measure of the number of times an item set appears in a dataset. Here, the item set is a test data sample.

Macro Average: Average the precision, recall, and F1 scores across all classes to get the final macro-averaged precision, recall, and F1 scores, respectively. You may do the hand calculations and cross-check the output you get.

Weighted Average:

Weighted Average of Precision = [Sum(Individual Class Precision * Individual Class Support)] / Total Support.

In a similar way, you can do it for the rest of the recall and F1 score. You may do the hand calculations and cross-check the output you get.

accuracy_score: In Python, the accuracy_score function of the sklearn.metrics package calculates the accuracy score for a set of predicted labels against the true labels.

Accuracy: (TP + TN) / (TP + TN + FP + FN)

The IRIS data used in the example has a total of 150 data samples that comprise 50 from each class of IRIS Setosa, IRIS Virginica, and IRIS Versicolor.

During the training and test split, 70% was used for training and 30% for testing purposes.

30% of 150 is 45. That is the reason the total support count is 45.

For instance, depending on the output, among the 45, there can be 11 Iris Setosa, 17 Iris Virginica, and 17 Iris Versicolor. The same number will reflect during the prediction evaluations in the confusion matrix.

Linear Regression:

GitHub link for the code:

https://github.com/Sabyasachi123276/ML-using-Python/blob/main/Linear_Regression.ipynb

- **`linear_model`**: `linear_model` is a class of the sklearn module that contains different functions for performing machine learning with linear models. The term linear model implies that the model is specified as a linear combination of features.
- **`linear_model.LinearRegression()`**: LinearRegression fits a linear model with coefficients `w = (w1, …, wp)` to minimize the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation.
- **`regr.fit(X,y)`:** Linear regression is fitted over the predictor variable (`X`) and response variable (`y`).
- **`regr.predict`(numerical value(s) of predictor variable(s)):** Make predictions depending on the predictor variables.
- **`regr.coef_ :`** Prints the values of the linear regression coefficients.

Logistic Regression:

GitHub link for the code:

https://github.com/Sabyasachi123276/ML-using-Python/blob/main/Logistic_Regression.ipynb

- **`np.array()`**: This function is used to create an array in NumPy.
- **`reshape(-1, 1):`** `reshape(-1, 1)` results in an array with a single column and multiple rows (a column vector).
- **`linear_model.LogisticRegression():`** LogisticRegression() method for creating the logistic regression object from the sklearn module.
- **`logr.fit(X,y):`** Logistic regression is fitted over the predictor variable (`X`) and response variable (`y`).
- **`logr.predict(column vector):`** Make predictions depending on the predictor variables given in a form of column vector.
- **`logr.coef_ :`** Prints the values of the logistic regression coefficients.
- **`log_odds :`** In logistic regression, the coefficient is the expected change in log odds of having the outcome per unit change in X.
- **`numpy.exp() : numpy.exp`**`()` is a function in the Python NumPy library that calculates the exponential value of an input array.