Regression Analysis of Boston Housing Data

Google Colab Link

In my exploration of regression algorithms using the Boston Housing dataset, I tried several models to predict housing prices.

Gradient Boost Tree Regressor

this model stands out due to its impressive performance metrics. It achieves a Root Mean Square Error (RMSE) of 3.67409 on the test data, coupled with an R-squared value of 0.84279, indicating a strong fit compared to the basic linear regression models.

```
9]: from pyspark.ml.evaluation import RegressionEvaluator
    gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
                    labelCol="medv",metricName="r2")
    print("R Squared (R2) on test data = %g" % gbt_evaluator.evaluate(gbt_predictions))

    R Squared (R2) on test data = 0.84279

0]: gbt_evaluator = RegressionEvaluator(predictionCol="prediction", \
                    labelCol="medv",metricName="rmse")

1]: print("RMSE on test data = %g" % gbt_evaluator.evaluate(gbt_predictions))

    RMSE on test data = 3.67409
```

Random Forest Regressor

This model displays robustness similar to the Gradient Boost Tree, with a comparable RMSE and an R-squared value of 0.786508. It offers a reliable alternative with its ensemble approach.

```
]: from pyspark.ml.evaluation import RegressionEvaluator
   rf_evaluator = RegressionEvaluator(predictionCol="prediction", \
                  labelCol="medv",metricName="r2")
   print("R Squared (R2) on test data = %g" % rf_evaluator.evaluate(rf_predictions))

   R Squared (R2) on test data = 0.786508

]: rf_evaluator = RegressionEvaluator(predictionCol="prediction", \
                  labelCol="medv",metricName="rmse")

]: print("RMSE on test data = %g" % gbt_evaluator.evaluate(gbt_predictions))

   RMSE on test data = 3.67409
```

Decision Tree Regressor

Although slightly less effective than the Random Forest, the Decision Tree Regressor still shows substantial predictive capability with an R-squared value of 0.727668, illustrating its utility in scenarios where model interpretability is key.

```
:
trainingSummary = lr_model.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)

RMSE: 4.774136
r2: 0.727668
```

Classification for SMS Spam Detection

Google Colab Link

The Logistic Regression

Excelling in predictive performance, this model leads with the highest scores across the board: an Area Under the Curve (AUC) of 0.94755, an F1-score of 0.98489, and an accuracy of 98.52%. It is the recommended choice for spam detection tasks due to its precision and efficiency.

AUC: 0.9475499400198194
F1: 0.9848942383213879
Accuracy: 0.9852173913043478

The Naive Bayes classifier

This classifier, while showing a slightly superior AUC of 0.94986, falls short in F1-score and accuracy at 0.93722 and 93.22% respectively, compared to Logistic Regression. It remains a strong contender, especially in scenarios favoring faster model training times.

AUC: 0.949864392635477
F1 Score: 0.9372189798267707
Accuracy: 0.9321739130434783

The Random Forest

With an AUC of 0.5, this model does not demonstrate effectiveness in spam detection, suggesting limitations in handling this specific classification task.

The Gradient Boosting Tree

Though not surpassing Logistic Regression, this model achieves a commendable AUC of 0.87683, an F1-score of 0.95194, and an accuracy of 95.30%. It is a viable option when additional model robustness is required.

AUC: 0.8768254837531946

F1 Score: 0.9519440759890753
Accuracy: 0.9530434782608695