Machine Learning Assignment

Programming Assignment

Name: Amruth Karun M V
RollNo: 2020MCS120004
Date: 05-Nov-2021

Introduction

The intention of this assignment is to get familiarized with different techniques of ensemble learning and how we can obtain higher predictive performance with multiple Machine Learning algorithms. Here, we are presented with a binary classification problem to identify the people who are likely to survive the infamous Titanic crash using machine learning techniques. We use concepts like cross validation and hyper-parameter tuning to improve our machine learning models. This problem can be considered as a binary classification problem where the two classes are 'Survived' and 'Not Survived'.

Problem Statement

The Royal Mail Steamer (RMS) Titanic shipwreck resulted in 1502 deaths from among the 2224 passengers and crew. It was observed that some people have higher chances of survival than others. The task is to build a predictive model to identify people who are more likely to survive.

Dataset

The dataset used for this problem is the Kaggle dataset "**Titanic - Machine Learning from Disaster".** The dataset has been split into two parts: train.csv and test.csv. The training set which includes the ground truth as well can be used to build the model. This train.csv is further split into sets for training and testing. The dataset contains around 891 rows and 12 columns out of which only some are relevant for our analysis and some contains missing values. From the initial analysis, around 70 percent of women survives and only around 20 percent of men survive the shipwreck. Also, passengers in the lower age group have a higher probability of survival. We need to clean and prepare the data before we can build the model to learn other interesting relationship between the data.

Data Preparation

The data must be processed and prepared before building the models. As mentioned above, the initial data contains 12 columns and not all are relevant for our learning. First we load the train.csv into a pandas data frame and this is further split into train and test.

Challenge: The dataset contains missing or null values. Common practices involves replacing these values with mean and dropping the rows.

The 'Age' attribute contains around 20% of missing values and the 'Cabin' attribute contains 77% missing values. So, it contributes to a lot of noise in the data. We can drop this column to avoid this noisy data. And, the feature 'Embarked' is independent of whether the person survived or not. So, this column is also dropped. Other missing values are filled with median or filled by interpolation. Then we apply feature engineering to create new features from sibling-spouse and parents-child combination to form the 'FamilySize'. The final cleaned train dataset is created by dropping the other two redundant columns. Sibsp: 'Sibling-Spouse' and Parch: 'Parent-Child'.

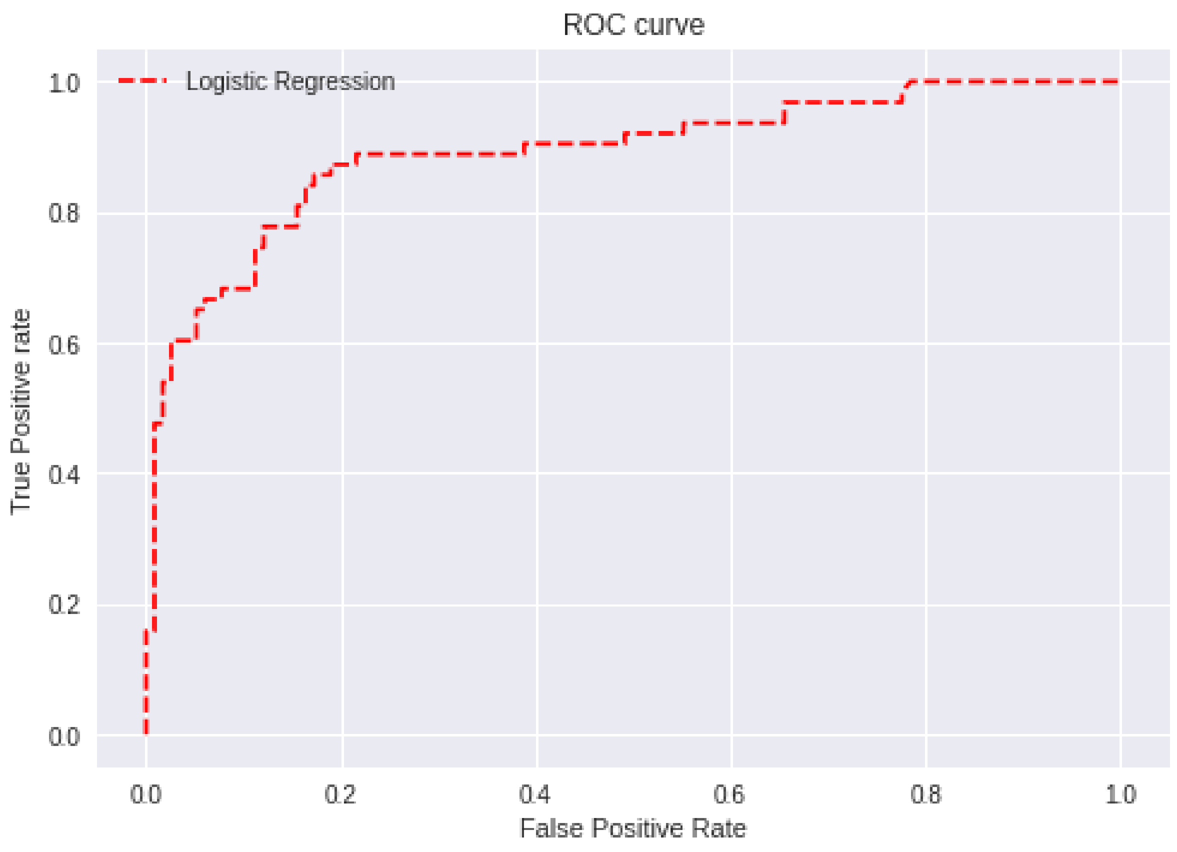
Build Logistic Regression Model

The logistic regression model is built from the preprocessed dataset using the following steps:

1. We have to divide the dataset into train and test so that we can evaluate the generalizing power of the model. Here we do not use the test set from the original dataset because it does not contain the target classes that is required for calculation of different metrics.
    - 20 % of the data is taken as test set.
    - Use a random seed value to obtain the same result every time.
2. We have to drop the column 'Survived' from the training set since this our target label and we cannot use this for training.
    - Drop the column 'Survived' and use it as the X_train.
    - Use the column 'Survived' as y_train (1 for survived and 0 for not survived).
    - Similarly, create X_test and y_test.
3. Create a Logistic Regression model using scikit-learn and keep max_iterations as 1000.
    - Fit the model using the train data.
    - Calculate the accuracy score for train and test data (approx. 80-83%).

Plot ROC Curve and Calculate AUC Score

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. Using the Logistic Regression model that we created in the previous step, we calculate the probability values for the test set, so that we can use this to plot the ROC curve with varying threshold values. Using *roc_curve(),* we calculate the false positive rate (FPR), false positive rate (TPR), and threshold. Then the ROC curve represented as the plot between FPR and TPR. pos_label is set to 1 to indicate the positive class (Survived). The AUC score is calculated as the area under the ROC curve. The closer the graph is to the top left portion, the better the AUC score and this indicates a good classifier. Here we obtained an AUC score of 0.891.



Perform K-Fold Cross Validation

Cross validation is a statistical method to measure the performance of machine learning models. Here, we divide the data into 10 folds and compute the average of the accuracy score. Cross validation protects the model from overfitting and it is also a good approach to evaluate the generalizing power of machine learning models. The data set is divided into k number of subsets and the usual train-test holdout method is repeated k number of times:

1. Randomly split the dataset into k number of folds.
2. Build the model using k–1 folds and test the model to check the effectiveness for kth fold.
3. Repeat until all k folds have has served as the test set.
4. Compute the cross validation accuracy by taking the average.
5. Compare it with the accuracy value obtained without using Cross Validation.

Inference: A mean cross validation accuracy of 0.849 is obtained which is greater than the initial test accuracy (0.83).

Hyper-parameter Tuning

Hyper-parameter are parameters that controls the learning process of a machine learning algorithm. Hyper-parameter tuning or optimization is the problem of identifying the optimal set of hyper-parameter for the learning algorithm. Here, Grid Search and Randomized Search is used to find the best model for finding the parameters for the best model obtained.

Grid Search: The grid search provided by **GridSearchCV** in sklearn exhaustively generates candidates from a grid of parameter values specified with the *param_grid* parameter. All the parameter combinations are evaluated and best combination is retained. The grid parameters are provided in the form of a dictionary below.

```
Grid = {
        'penalty':['l2'], 'tol':[0.0001, 0.0002, 0.0003],
        'max_iter':[1000, 2000, 3000],
        'C':[0.01, 0.1, 1, 10, 100],
        'intercept_scaling':[1, 2, 3, 4],
        'solver':['liblinear', 'lbfgs']
        }
```

Result:

```
Best Score = 0.783689549886733
Best Parameter = {
                  'C': 0.1, 'intercept_scaling': 1,
                  'max_iter': 1000, 'penalty': 'l2',
                  'solver': 'liblinear', 'tol': 0.0001
                  }
Test Accuracy using grid search = 0.8324022346368715
```

Note: The value of C should be a sequence and we cannot take it from a uniform distribution. So, for that we need to use Randomized Search.

Randomized Search: RandomizedSearchCV implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values. Here, we take the parameter C from a uniform distribution. We fit the models with the training data and predictions are made on the test data.

```
random_dist = {
                'penalty':['l2', 'l1'],
                'tol':[0.0001, 0.0002, 0.0003],
                'max_iter':[1000, 2000, 3000],
                'C':uniform(loc=0, scale=2),
                'intercept_scaling':[1, 2, 3, 4],
                'solver':['liblinear']
                }
```

Result:

```
Best Score = 0.7780655963754556
Best Parameter = {
                 'C': 0.5596, 'intercept_scaling': 2,
                 'max_iter': 3000, 'penalty': 'l1',
                 'solver': 'liblinear', 'tol': 0.0001
                 }
Test Accuracy using randomized search = 0.8324022346368715
```

Model Evaluation Metrics

Accuracy might not be always the best metrics that gives you a good idea about our classifier. Other metrics like Precision, Recall and F1-Score proves to be a better metrics in the case of problems like class imbalance. So, in order to measure the performance of our machine learning algorithms, we can use these metrics. These metrics are computed from the predicted values of the test set using each of the models below:

1. Logistic Regression without cross validation
2. Grid Search Method
3. Randomized Search Method

We can compute these metrics together or separately, Here I used precision_recall_fscore_support from sklearn to compute all three values. Other metrics, micro, macro and weighted precision, recall and fscore can be calculated by passing additional average parameters.

Logistic Regression	- Precision: 0.73, Recall 0.825, F1-Score: 0.77
Grid Search	- Precision: 0.746, Recall 0.79, F1-Score: 0.769
Randomized Search	- Precision: 0.739, Recall 0.809, F1-Score: 0.77

Macro Average: Macro average calculates the corresponding metric for each of the binary cases, and then averages the results together.

Logistic Regression	- Precision: 0.815, Recall 0.830, F1-Score: 0.821
Grid Search	- Precision: 0.815, Recall 0.823, F1-Score: 0.818
Randomized Search	- Precision: 0.815, Recall 0.8273, F1-Score: 0.819

Micro Average: Micro averaging treats the entire set of data as an aggregate result, and calculates 1 metric rather than k metrics that get averaged together.

Logistic Regression	- Precision: 0.832, Recall 0.832, F1-Score: 0.832
Grid Search	- Precision: 0.832, Recall 0.832, F1-Score: 0.832
Randomized Search	- Precision: 0.832, Recall 0.832, F1-Score: 0.832

Weighted Average: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label).

Logistic Regression	- Precision: 0.839, Recall 0.832, F1-Score: 0.834
Grid Search	- Precision: 0.835, Recall 0.832, F1-Score: 0.833
Randomized Search	- Precision: 0.837, Recall 0.832, F1-Score: 0.833

viii. Comment on the models built with and without hyper-parameter tuning based on the metric values.

Hyper-parameter tuning is used to find the best combination of parameters that would give us the best model. Here we built 3 models, one without using hyper-parameter tuning and other two using Grid Search and Randomized search respectively, for Logistic Regression.

So, from the above calculated parameters we can say that when using the model without hyper-parameter tuning, we get a train accuracy of around 0.78 and 0.83 test. We get similar result when using the cross validated data. So, The model we are using is a decent model for the given data. Precision, recall and F1-score are also in the range 0.81-0.84.

When using *randomized search*, we get around a 2% increase in these metrics value which gives us a better classifier. The quality of the model also depends upon the search space and the distribution we are taking. If we give a different random distribution to work on, the algorithm might end up in a global optimum and gives the maximum performance.