Machine Learning : Using Scikit - Learn

Objective

The Titanic dataset is starter dataset on Kaggle. The objective of this exercise is to apply Scikit Learn to build a basic predictive model for the titanic dataset. Once you have built your model you can upload the final predicted results to Kaggle in order to measure your achieved accuracy.

The following is a list of the features in the dataset. The class we are trying to predict is Survived.

VARIABLE DESCRIPTIONS:

Survived	Survival (0 = No; 1 = Yes)
Pclass	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
Name	Name
Sex	Sex
Age	Age
SibSp	Number of Siblings/Spouses Aboard
Parch	Number of Parents/Children Aboard
Ticket	Ticket Number
Fare	Passenger Fare
Cabin	Cabin
Embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Part A: Generating a Baseline Result

1. You will notice that the exercise folder on Blackboard contains two files. The first is train.csv and the second is test.csv. The files contain different instances of data. The main difference is that the data in train.csv is labelled (that is, it includes the 'Survived' class) for all data instances. However, the test.csv data does not include the label for each instance (this is your unseen data). Your objective is build a model using the data in train.csv and then use it to make predictions from train.csv.

2. In order to submit you results you will need to create an account at Kaggle.

3. A template code file is provided for you. This file reads in the two datasets (test and train) and merges them together so that any pre-processing you undertake will be performed on all the data at once.

4. [**Remove Features**] Some of the features may not contribute to the prediction of the final class.

   a. For the purposes of this exercise we will remove the Name, Cabin and Ticket features (even through it is possible to transform some of these features into more meaningful features with a little work).

5. [**Missing Values**] You will notice that in the Titanic dataset there are a number of missing values for some features. Determine the number of missing value in each column.

   a. You will notice the **Age** column has the highest number of missing values in both the train and test dataset. Use Scikit's SimpleImputer class to impute the missing value using a strategy of mean.
   b. The **Fare** feature in the test dataset is missing one value. You should again impute for this value using mean.
   c. The **Embarked** feature in the train dataset is missing two values. Impute the most frequent value for these missing values.

6. **[Encoding Categorical Variables]** Because Scikit-learn classification and regression algorithms accept as input a NumPy array containing continuous values we must convert the category based features to numeric values.
   a. Use the map function in Pandas to map the values for Sex (female and male) to numerical values 0 and 1.
   b. Use the map function to map the values for the Embarked feature to numeric values ({'S': 0, 'C': 1, 'Q': 2}).

7. With the pre-processing complete, we now want to separate the training and test data. The training data contains -1 values only in the survived column. Use this information to separate the train and test data.

8. Next separate the training data into feature training data and class label data (the Survived column).

9. Drop the passenger ID from both the test and training dataset. However, you should store the passenger ID from the test dataset as a series object as you will need this data for submitting your model (step 11).

10. Next you should use Scikit Learn's Nearest Neighbour class (KNeighborsClassifier) to predict the Survived value for test dataset.

11. Finally, before submitting your results you should merge the results from step 10 with the PassengerID feature from step 8 into a single dataframe (see code below). Write the result of the merge to a CSV file. Your merged CSV file should consist of two columns, the passenger ID and the prediction from your model (did the passenger survive or not).

    Once you have this done you can submit the results online at Kaggle and see the accuracy of your model.

```
# The variable, results is the NumPy array of predicted results returned from our classifier

resultSeries = pd.Series(data = results, name = 'Survived', dtype='int64')



# create a data frame with just the PassengerID feature from the test dataset and the predicted results

df = pd.DataFrame({"PassengerId":passengerIDSeries, "Survived":resultSeries})



# write the results to a CSV file (you should then upload this file)

 df.to_csv("submission.csv", index=False, header=True)
```

Part B: Normalize the Feature Vectors

1. **[Normalize Data].** Use the MinMaxScaler class to standardize the following features: "Age", "SibSp", "Parch", "Fare", Pclass. You would typically normalize the data after you have encoded the categorical variables (after Step 5 in part 1).

   Please note that if you apply MinMaxScaler to a dataframe containing multiple features with different data types it will generate a warning. It still performs the normalization. To view the datatypes of each column in a dataframe you can use df.info(). If you then wish to change the datatype of any column in a dataframe you can use pd.astype().

2. The lecture notes demonstrated how to encode nominal features using one-hot-encoding. Rather than using the map function to encode the Embarked column, instead use Scikit's OneHotEncoder class.

   Remember when you run the OneHotEncoder on a feature that contains x unique values, it will return a NumPy array containing X Boolean columns. You will need to append the one-hot-encoded NumPy array with your original dataframe.

   One way to do this is to create a DataFrame from the NumPy array (pd.DataFrame(oneHotEncodedArray)). You can then concatenate the resulting dataframe with your original titanic data frame using pd.concat (pd.concat([sourceData, newData], axis=1)).

   (Also don't forget to remove the original Embarked column).