-Project-
Titanic Prediction

Objective:

To make a machine learning model to predict the survivability of a set of passengers on board the Titanic, trained using another set of passengers.

Methodology:

To make our model we must do the following -

a) Choose a statistical model and tidy the training data accordingly

b) Train said model using the tidied training data

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X. We see that survivability is a binary result hence we choose to use a logistic regression model for making our ML model where 1 denotes surviving the sinking. This allows us to make a model which is relatively accurately. We also must include only independent meaningful values for training our model.

We import our datasets using the pandas library and tidy it as we require. We tidy the training and test data in the same way. Firstly we normalize the data by removing all forms of not applicable or missing data. We remove Cabin values as they are not required for our model. We also remove rows with missing Embarked and Fare values. Finally we must improve the Age data. We do this by using the pd.fillna() function with method='pad' which fills missing values with the previous valid age value. Now we map the data and get dummy values to get numerical values for non numeric types Sex,Embarked and Pclass which can be put into a single equation of the logistical model. We then drop the columns which are not required. Hence we know have Pclass, Sex, Age, Sibsp, Parch, Fare and Embarked which now all have numerical values or equivalent dummy values.

Now that we have our datasets ready we must make use of our model to predict the result. We use LogisticRegression class from the sklearn library which composes of multiple regression and ML models and classifications. The liblinear algorithm is used here for the model. Since this is a multivariable model the training algorithm uses the one vs rest scheme where the one represents our output i.e. survivability. We train the model by simply using the fit() function of LogisticRegression which allows us to input our training dataset(without survivability) as the training vector and the survivability column of the training dataset as the target vector relative to the training vector. We get these vectors by using the train_test_split() function also from the sklearn library which splits datasets into random train subsets. Finally we use the predict() function from LogisticRegression class which puts the data into the equation to get the probability passing our test data as the input to get the survivability column for our test data, which we add to a duplicate of the test data and finally print the resulting dataset.

-Project-
Titanic Prediction

Documentation:

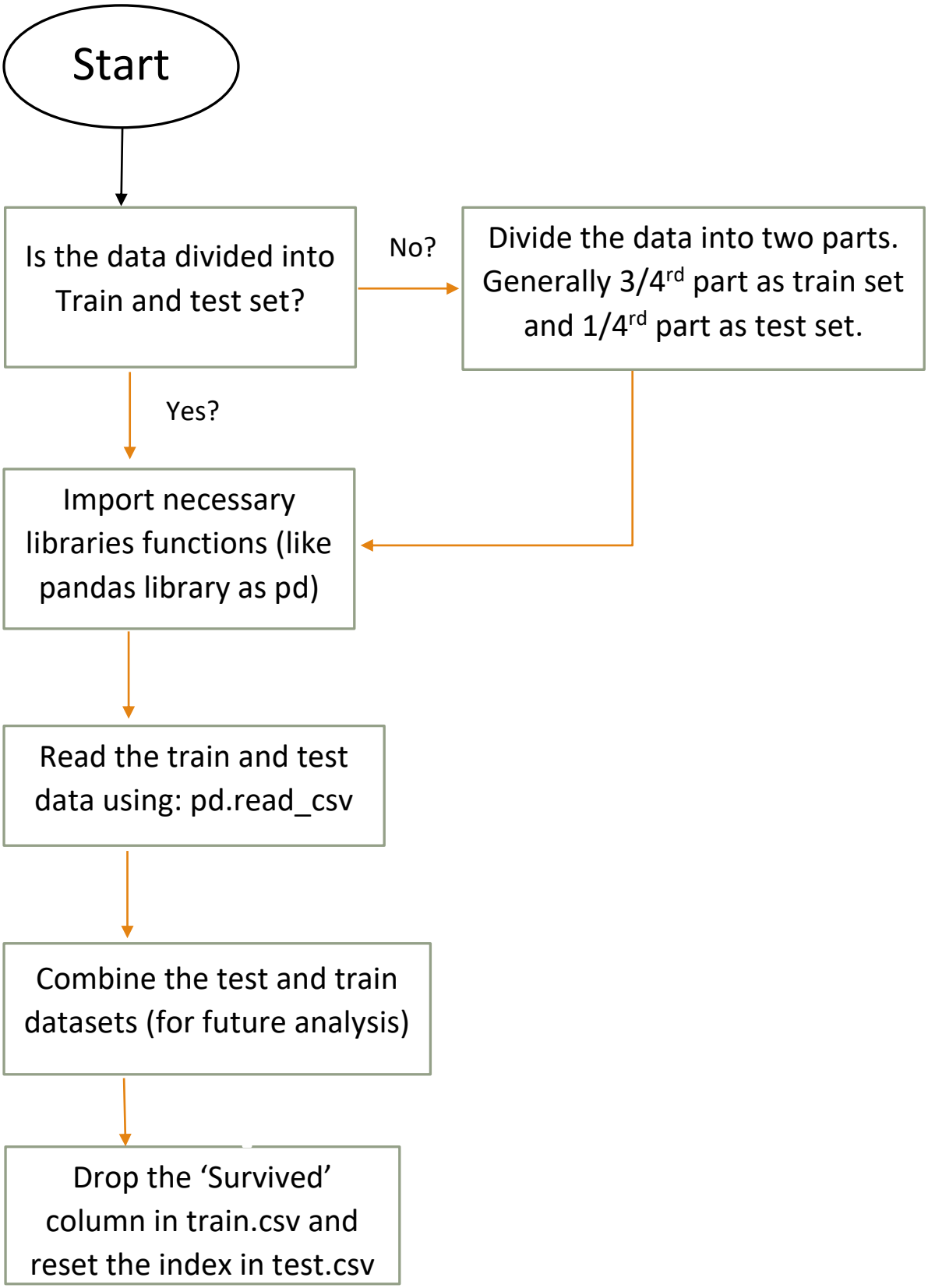
- **Variables used –**
  - train : a DataFrame object containing data used for training the ML model
  - test : a DataFrame object containing data for prediction of survivability
  - dupe : a copy of the test variable
  - predictor : an object of class LogisticRegression from the sklearn library

- **Classes used –**
  - LogisticRegression - This class implements regularized logistic regression. It can handle both dense and sparse input. Contains the required functions for training and predicting.

- **Functions used –**
  - train_test_split() – function of model_selection class which splits matrices into random train and test subsets. Takes arrays or dataframes as input.
  - get_dummies() – function of pandas library used to get dummy variables according to certain columns. Used to get numeric/indicator values of a column.
  - fit() – function of LogisticRegression class which fits the training model according to the training vector and corresponding target vector. Sets appropriate cutoffs for probabilities and such. Takes a DataFrame or Series as input.
  - predict() – function of LogisticRegression class which solves for the output variable according to the data fitted in using fit().Takes one input i.e. data for making prediction.

-Project-
Titanic Prediction

Flowchart:



Data is currently raw and needs to be tidied before use. Working on test.csv now

Drop ‘Cabin’ as ‘Cabin’ is not relevant in predicting ‘Survived’.

Embarked and Fare have some missing values (<1% of table)

Remove rows of ‘Embarked’ and ‘Fare’ which have missing values

Age needs to be improved. To do so, average age of the remaining is taken.

Fill the missing ages using previously valid age values. Also define dupe=test (for future priniting).

Data is now tidy. However, to use logistic regression, one will need numerical values for all columns. This will be done by mapping. Also, dummy columns will be made so that only one equation is needed for any given prediction.

Use: get_dummies function for ‘Sex’, ‘Embarked’ and ‘Pclass’ columns.

Drop all other values that are not needed for predictions, eg: "PassengerId", "Pclass", "Name", "Ticket", “Embarked".

-Project-
Titanic Prediction

Drop “PassengerId”, "Pclass", "Name", "Ticket", "Embarked" columns from

1) Drop ‘Cabin’ as ‘Cabin’ is not relevant in predicting ‘Survived’
2) Remove rows of ‘Embarked’ and ‘Fare’ which have missing values
3) Fill the missing ages using previously valid age values
4) Use: get_dummies function for ‘Sex’, ‘Embarked’ and ‘Pclass’ columns.
5) Drop “PassengerId”, "Pclass", "Name", "Ticket", "Embarked" columns from test.csv

Data is now sufficiently tidy and usable. Prediction needs to be made.

Save ‘Survived’ and the rest of columns of train.csv, separately, in 2 variables: ‘feat’ and ‘tar’ respectively.

Store ‘feat’ and ‘tar’ in 4 variables (train_feat, test_feat, train_tar, test_tar) using: train_test_split() function

Use LogisticRegression() function on the variables: “train_feat, train_tar” using ‘.fit’ for

To check metrics on accuracy

Making the predictions and assigning a column in the dataset.

From sklearn.metrics import the function classification_report(), and apply it on 'test_tar' and 'predictions'.

Print: classification_report(test_tar, predictions)

Save 'predictor.predict(test)' into a variable: 'predictions'

Save the 'predictions' values in another column of 'dupe' (defined earlier as equal to test) named: 'Survived'

Print: dupe

Enter any key to terminate the process

Input any key

Exit

Discussion:

The accuracy of this model is dependent on the data as well as the type of data. Hence by using different methods of filling missing data we can get more accurate results. For example, filling of age could be done using the mean value and plotting a distribution curve. For the sake of simplicity we have avoided doing so.

Conclusion:

Hence we can see that our model can give us predictions which can be considered to be quite accurate and that applying ML to a dataset can be a very useful tool which can also be done simplistically and effectively with some minimal effort.

Program Listing:

```
'''
Predict whether or not a passenger survives the Titanic.
datasets =  test - for tesing.
        train - has survival values.

Planning to use multiple linear regression.
'''

# Importing the necessary libraries and functions.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


################################################################################


# Loading the necessary files.
test = pd.read_csv('titanic/test.csv')
train = pd.read_csv('titanic/train.csv')


# Combining the test and train datasets for future analysis. Removing Survived
# and resetting index as well.
temp = train.drop(['Survived'],axis=1)
test = temp.append(test).reset_index()
test = test.drop(['index'],axis=1)
```

```
###############################################################################
# Data is currently raw and needs to be tidied up before use.

# Very few valid Cabin values (<25% of the table). Also, Cabin is not relevant
# to predicting Survived. Dropping Cabin.
print(test.count())
test = test.drop(['Cabin'],axis=1)
print(test)
print('\n\n')


# Embarked and Fare have some missing values(<1% of table), but not too many.
# Acceptable.Those rows will simply be removed.
test.dropna(subset=['Fare','Embarked'],axis=0,inplace=True)
print(test)
print(test.count())
print('\n\n')


# Age needs to be improved.
# To do so, average age of the remaining is taken. The missing ages are filled
# using previous valid age values.
test.fillna(inplace=True,method='pad')
print(test)
print(test.count())
dupe = test


###############################################################################
# Data is now tidy. However, to use logistic regression, one will need
# numerical values for all columns. This will be done by mapping. Also, dummy
# columns will be made so that only one equation is needed for any given
# prediction.


x = pd.get_dummies(test['Sex'],columns='Male',drop_first=True)
test['Sex'] = x

x = pd.get_dummies(test['Embarked'],drop_first=True)
test = pd.concat([test,x],axis=1)

x = pd.get_dummies(test['Pclass'],drop_first=True)
test = pd.concat([test,x],axis=1)
```

```
# Also, going to drop all values not needed for predictions.
test.drop(["PassengerId","Pclass","Name","Ticket","Embarked"],axis=1,inplace=True)
print(test)

##############################################################################
# All the previous steps are going to be repeated for the training set of data.

train = train.drop(['Cabin'],axis=1)
train.dropna(subset=['Fare','Embarked'],axis=0,inplace=True)
train = train.reset_index()
train.drop(['index'],axis=1,inplace=True)
train.fillna(inplace=True,method='pad')
x = pd.get_dummies(train['Sex'],columns='Male',drop_first=True)
train['Sex'] = x
x = pd.get_dummies(train['Embarked'],drop_first=True)
train = pd.concat([train,x],axis=1)
x = pd.get_dummies(train['Pclass'],drop_first=True)
train = pd.concat([train,x],axis=1)
train.drop(["PassengerId","Pclass","Name","Ticket","Embarked"],axis=1,inplace=True)

##############################################################################
# Data is now sufficiently tidy and usable. Prediction needs to be made.
# Reminder - train is to be used for training, and has the 'Survived' values.
# test needs to be used for final predictions.

feat = train.drop('Survived',axis=1)
tar = train['Survived']

train_feat, test_feat, train_tar, test_tar = train_test_split(feat, tar)

predictor = LogisticRegression()
predictor.fit(train_feat,train_tar)

# to check metrics on accuracy
#predictions = predictor.predict(test_feat)
#from sklearn.metrics import classification_report
#print(classification_report(test_tar, predictions))

##############################################################################
# Actually making the predicitons and assigning a column in the dataset.
predictions = predictor.predict(test)
dupe['Survived'] = predictions
print(dupe)
input('Press key to continue.')
```