

Machine Learning using Logistic Regression in Python with Code

 becominghuman.ai/machine-learning-using-logistic-regression-in-python-with-code-ab3c7f5f3bed

Meet Nandu

January 26, 2019



Meet Nandu

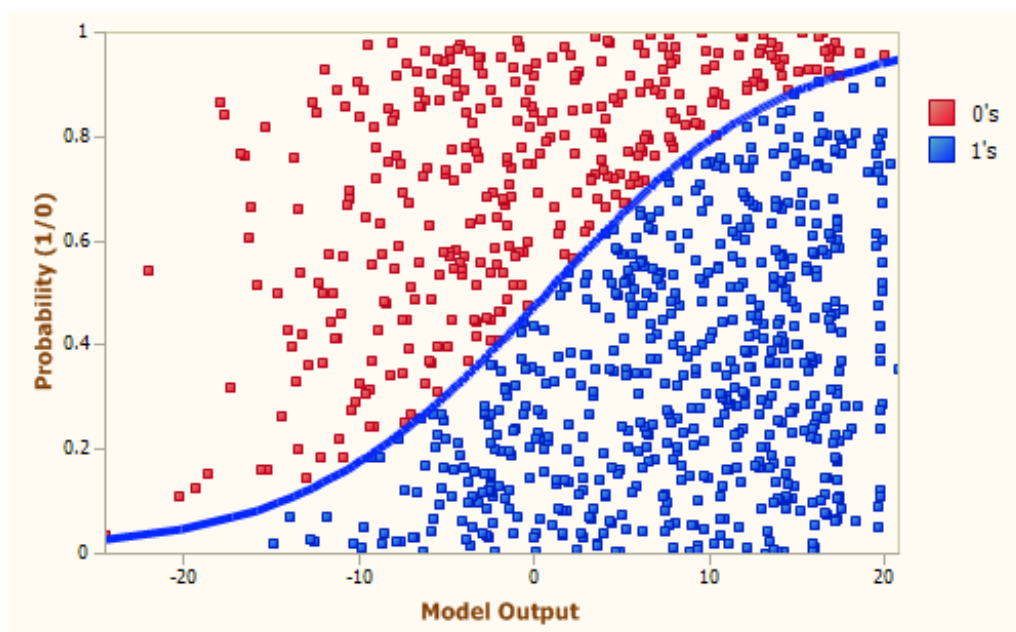
Jan 25

Classification techniques are an essential part of machine learning and data mining applications. Approximately 70% of problems in Data Science are classification problems. There are lots of classification problems that are available, but the logistics regression is common and is a useful regression method for solving the binary classification problem.



Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. The real life example of classification example would be, to categorize the mail as spam or not spam, to categorize the tumor as malignant or benign and to categorize the transaction as fraudulent or genuine. All these problem's answers are in categorical form i.e. Yes or No. and that is why they are two class classification problems.



Today, we will be working with the famous [Titanic Data Set from Kaggle](#). This is a very famous data set and very often is a student's first step in machine learning! We'll be trying to predict a classification- survival or deceased. Let's begin our understanding of implementing Logistic Regression in Python for classification. We'll use a "semi-cleaned" version of the titanic data set, if you use the data set hosted directly on Kaggle, you may need to do some additional cleaning not shown in this article. Also, I have made two separate files for training and testing. The training data can be found by clicking [here](#) and the test data can be found by clicking [here](#).

Trending AI Articles:

- | [1. Making a Simple Neural Network](#)
- | [2. Google will beat Apple at its own game with superior AI](#)
- | [3. The AI Job Wars: Episode I](#)
- | [4. Introducing Open Mined: Decentralised AI](#)

Imports

Let us first start by importing some common libraries of Python.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

These libraries help us with importing data, structuring the data, visualizing the data and then finally build our model on the data.

Get the Data

After importing these libraries, let us now **import our data** using the pandas libraries.

```
train = pd.read_csv('titanic_train.csv')
test = pd.read_csv('titanic_test.csv')
```

Let's look at how this data looks.

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

This is how our data will look.

Exploratory Data Analysis

Let's begin some exploratory data analysis! We'll start by checking out missing data!

Missing Data

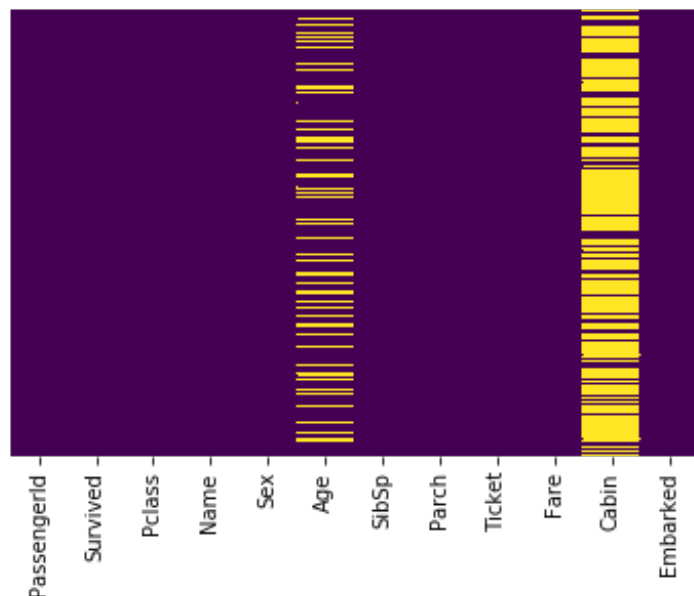
We can use seaborn to create a simple heatmap to see where we are missing data!

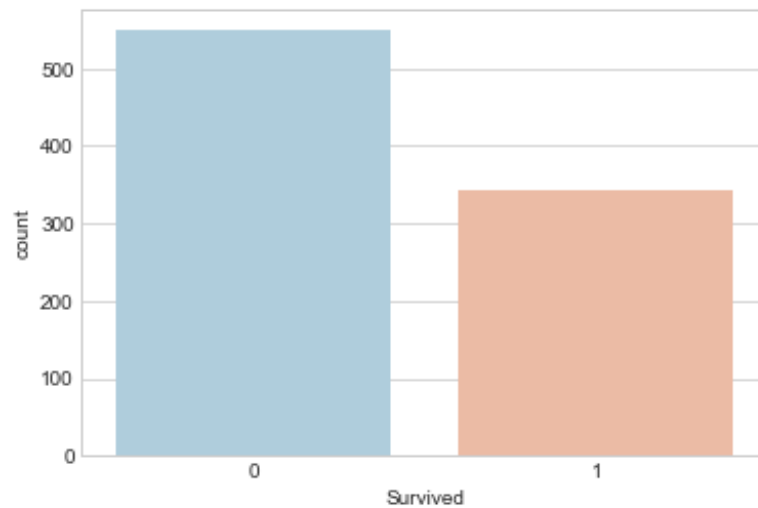
```
sns.heatmap(test.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

Let's continue on by visualizing some more of the data!

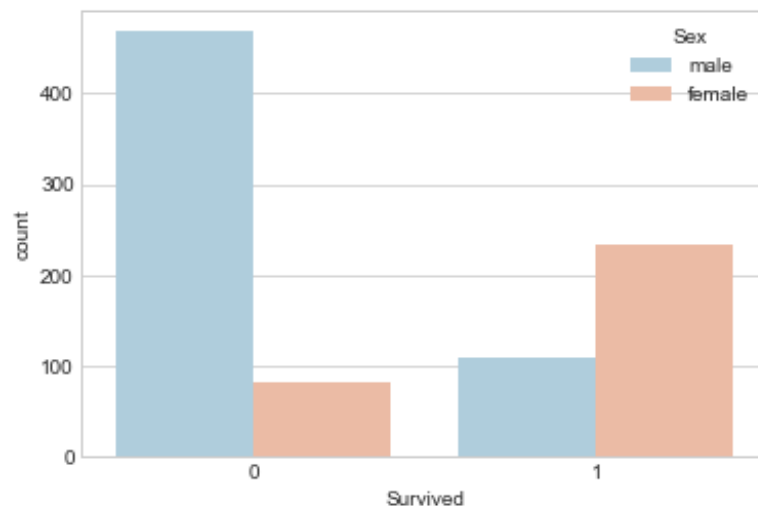
```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```





This shows the count of people who survived.

```
sns.set_style('whitegrid')  
sns.countplot(x='Survived', hue='Sex', data=train, palette='RdBu_r')
```

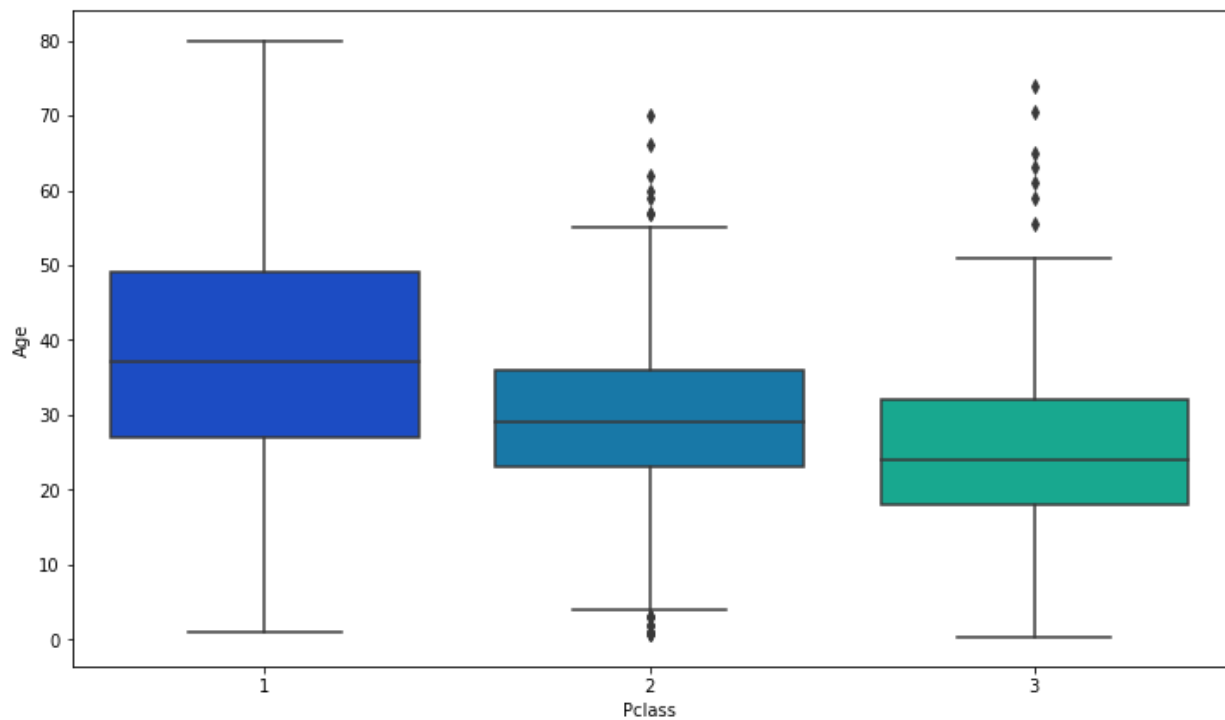


This shows the count of males and females survived.

Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

```
plt.figure(figsize=(12, 7))  
sns.boxplot(x='Pclass', y='Age', data=train, palette='winter')
```



This shows the average age of people belonging to different class in the ship.

We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

Now apply that function!

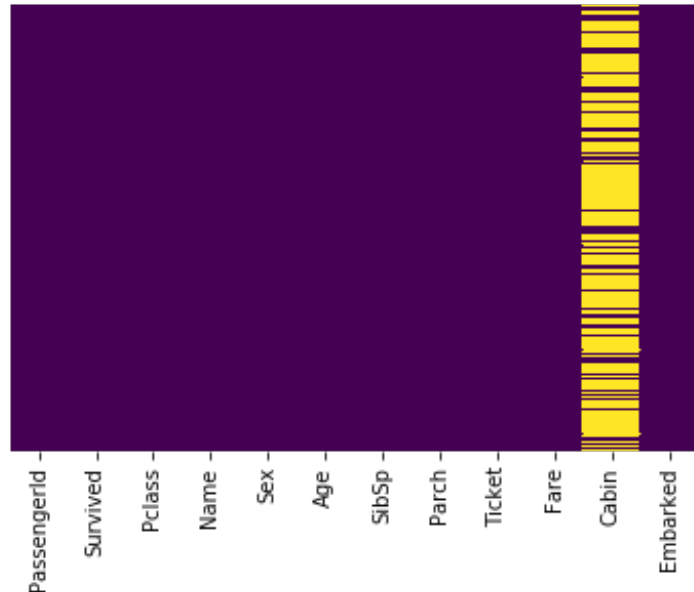
```
train['Age'] = train[['Age', 'Pclass']].apply(impute_age,axis=1)
test['Age'] = test[['Age', 'Pclass']].apply(impute_age,axis=1)
```

Now let's check that heat map again!

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

We can now see there are no null values in the Age column.

Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.



```
train.drop('Cabin',axis=1,inplace=True)
test.drop('Cabin',axis=1,inplace=True)
train.dropna(inplace=True)
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

After dropping the Cabin column, our data will look like this.

Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

```
train.info()
```

Output of train.info() should look something like this -

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.6+ KB
```

As we can see, there are 4 categorical columns namely Name, Sex, Ticket and Embarked. Out of these 4, the Name and Ticket column have no relationship with whether the person survived or not. So we drop these 2 columns and we convert the other two columns into numerical values.

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)

train.drop(['Sex', 'Embarked', 'Name', 'Ticket'],axis=1,inplace=True)

train = pd.concat([train,sex,embark],axis=1)

train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Cabin	male	Q	S
0	1	0	3	22.0	1	0	7.2500	NaN	1	0	1
1	2	1	1	38.0	1	0	71.2833	C85	0	0	0
2	3	1	3	26.0	0	0	7.9250	NaN	0	0	1
3	4	1	1	35.0	1	0	53.1000	C123	0	0	1
4	5	0	3	35.0	0	0	8.0500	NaN	1	0	1

Great! Our data is ready for our model!

Building a Logistic Regression model

Let's start by splitting our data into a training set and test set (there is another test.csv file that you can play around with in case you want to use all this data for training).

Train Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(train.drop('Survived',axis=1),train['Survived'],
test_size=0.30,random_state=101)
```

Training and Predicting

```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)

predictions = logmodel.predict(X_test)
```

Let's move on to evaluate our model!

Evaluation

We can check precision, recall, f1-score using classification report!

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, predictions))
print("Accuracy:", metrics.accuracy_score(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.81	0.93	0.86	163
1	0.85	0.65	0.74	104
avg / total	0.82	0.82	0.81	267

Accuracy: 0.7903225806451613

Well, you got a classification rate of 79%, considered as good accuracy.

Precision: Precision is about being precise, i.e., how accurate your model is. In other words, you can say, when a model makes a prediction, how often it is correct.

Advantages

Because of its efficient and straightforward nature, doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist. Also, it doesn't require scaling of features. Logistic regression provides a probability score for observations.

Disadvantages

Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to overfitting. Also, can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

Conclusion


Logistic regression has an array of applications. Here are a few applications used in real-world situations.

Marketing: A marketing consultant wants to predict if the subsidiary of his company will make profit, loss or just break even depending on the characteristic of the subsidiary operations.

Human Resources: The HR manager of a company wants to predict the absenteeism pattern of his employees based on their individual characteristic.

Finance: A bank wants to predict if his customers would default based on the previous transactions and history.

Thanks for reading the article! If you liked it, a clap would be appreciated. :)

Don't forget to give us your  !



0

