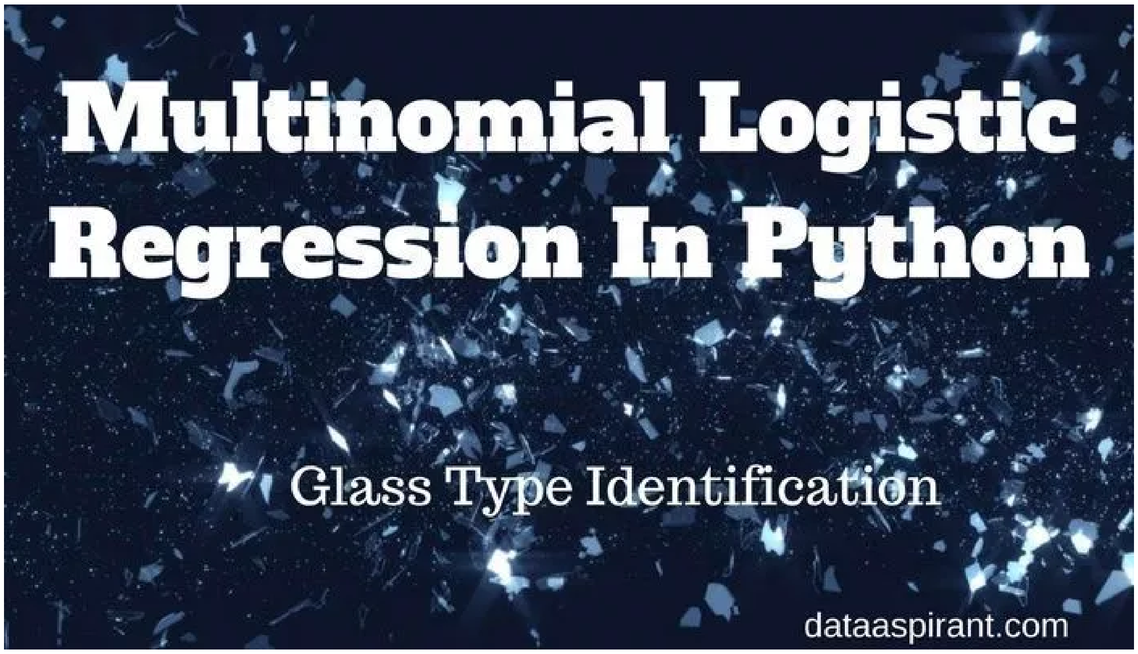
2 WAYS TO IMPLEMENT MULTINOMIAL LOGISTIC REGRESSION IN PYTHON

May 15, 2017 Saimadhu Polamuri 2 Comments Machine Learning

5
Shares


Multinomial Logistic Regression In Python
Glass Type Identification
dataaspirant.com

Multinomial Logistic Regression Python

Implementing Multinomial Logistic Regression in Python

Logistic regression is one of the most popular supervised classification algorithm. This classification algorithm mostly used for solving binary classification problems. People follow the myth that logistic regression is only useful for the binary classification problems.

Which is not true. Logistic regression algorithm can also use to solve the multi-classification problems. So in this article, your are going to implement the logistic regression model in python for the multi-classification problem in 2 different ways.

In machine learning way of saying implementing **multinomial logistic regression model in python**.

Implementing multinomial logistic regression model in python.

CLICK TO TWEET

5
Shares

Table of contents:

The difference between binary classification and multi-classification

The name itself signifies the key differences between binary and multi-classification. Below examples will give you the clear understanding about these two kinds of classification. Let's first look at the **binary classification** problem example. Later we will look at the **multi-classification** problems.

Binary Classification:

- Given the subject and the email text predicting, Email Spam or not.
- Sunny or rainy day prediction, using the weather information.
- Based on the bank customer history, Predicting whether to give the loan or not.

Multi-Classification:

- Given the dimensional information of the object, Identifying the shape of the object.
- Identifying the different kinds of vehicles.
- Based on the color intensities, Predicting the color type.

I hope the above examples given you the clear understanding about these two kinds of classification problems. In case you miss that, Below is the explanation about the two kinds of classification problems in detail.

Binary Classification Explanation:

In the binary classification task. The idea is to use the training data set and come up with any classification algorithm. In the later phase use the trained classifier to predict the target for the given features. The possible outcome for the target is one of the **two different** target classes.

If you see the above binary classification problem examples, In all the examples the predicting target is having **only 2** possible outcomes. For email spam or not prediction, the possible 2 outcome for the target is email is spam or not spam.

On a final note, binary classification is the task of predicting the target class from two possible outcomes.

Multi-classification Explanation:

In the multi-classification problem, the idea is to use the training dataset to come up with any classification algorithm. Later use the trained classifier to predict the target out of **more than 2** possible outcomes.

If you see the above multi-classification problem examples. In all the examples the predicting target is having **more than 2** possible outcomes. For identifying the objects, the target object could be triangle, rectangle, square or any other shape. Likewise other examples too.

On a final note, multi-classification is the task of predicting the target class from more two possible outcomes.

I hope you are having the clear idea about the binary and multi-classification. Now let's move on the Multinomial logistic regression.

Introduction to Multinomial Logistic regression

Multinomial logistic regression is the generalization of logistic regression algorithm. If the logistic regression algorithm used for the multi-classification task, then the same logistic regression algorithm called as the **multinomial logistic regression**.

5
Shares

The difference in the normal logistic regression algorithm and the multinomial logistic regression in not only about using for different tasks like binary classification or multi-classification task. In much deeper It's all about using the **different functions.**

In the logistic regression, the black function which takes the input features and calculates the probabilities of the possible two outcomes is the **Sigmoid Function**. Later the high probabilities target class is the final predicted class from the logistic regression classifier.

When it comes to the multinomial logistic regression the function is the **Softmax Function**. I am not going to much details about the properties of sigmoid and softmax functions and how the multinomial logistic regression algorithms work. As we are already discussed these topics in details in our earlier articles.

Before you drive further I recommend you, spend some time on understanding the below concepts.

- How logistic regression algorithm works in machine learning
- Softmax Vs Sigmoid function
- How Multinomial logistic regression classifier work in machine learning
- Logistic regression model implementation in Python

I hope you clear with the above-mentioned concepts. Now let's start the most interesting part. Building the multinomial logistic regression model.

You are going to build the multinomial logistic regression in **2 different ways**.

- Using the same python scikit-learn binary logistic regression classifier.
- Tuning the python scikit-learn logistic regression classifier to model for the multinomial logistic regression model.

Glass Identification Dataset Description

The classification model we are going build using the multinomial logistic regression algorithm is **glass Identification**. The Identification task is so interesting as using different glass mixture features we are going to create a classification model to predict what kind of glass it could be.

5
Shares

We will look into, what are those glass types in the coming paragraph. Before that let's quickly look into the key observation about the glass identification dataset.

Title	Glass Identification Dataset
Dataset Associated Task	Classification
Number of Observations	214
Number of features	10
Missing Values	No
Target	Glass Type

Features and Target Information

From the above table, you know that we are having 10 features and 1 target for the glass identification dataset, Let's look into the details about the features and target.

Features:

1. Id number: 1 to 214
2. RI: refractive index

3. Na: Sodium (unit measurement: weight percent in the corresponding oxide, as attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron

5
Shares

Target: Type of glass

The glass identification dataset having 7 different glass types for the target. These different glass types differ from the usage.

1. building_windows_float_processed
2. building_windows_non_float_processed
3. vehicle_windows_float_processed
4. vehicle_windows_non_float_processed (none in this database)
5. containers
6. tableware
7. headlamps

Multinomial Logistic regression implementation in Python

Below is the workflow to build the multinomial logistic regression.

- Required python packages
- Load the input dataset
- Visualizing the dataset
- Split the dataset into training and test dataset
- Building the logistic regression for multi-classification
- Implementing the multinomial logistic regression
- Comparing the accuracies

Let’s begin with importing the required python packages.

Required Python Packages

Below are the general python machine learning libraries. If you haven't setup python machine learning libraries setup. Python machine learning setup will help in installing most of the python machine learning libraries.

```
# Required Python Packages
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn import metrics
from sklearn.cross_validation import train_test_split

import plotly.graph_objs as go
import plotly.plotly as py
from plotly.graph_objs import *
py.sign_in('Your_ploly_username', 'API_key')
```

5
Shares

- **Pandas:** Pandas is for data analysis, In our case the tabular data analysis.
- **Numpy:** Numpy for performing the numerical calculation.
- **Sklearn:** Sklearn is the python machine learning algorithm toolkit.
  - **linear_model:** Is for modeling the logistic regression model
  - **metrics:** Is for calculating the accuracies of the trained logistic regression model.
  - **train_test_split:** As the name suggest, it's used for splitting the dataset into training and test dataset.
- **Plotly:** Plotly is for visualizing the data.

Now let's load the dataset into the pandas dataframe.

Dataset Path

You can download the dataset from UCI Machine learning Repository or you can clone the complete code for dataaspirant GitHub account.

```
# Dataset Path
DATASET_PATH = "../Inputs/glass.txt"
```

Loading dataset

```
def main():
    # Glass dataset headers
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",
    # Loading the Glass dataset in to Pandas dataframe
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    print "Number of observations :: ", len(glass_data.index)
    print "Number of columns :: ", len(glass_data.columns)
    print "Headers :: ", glass_data.columns.values

if __name__ == "__main__":
    main()
```

- The downloaded dataset is not having the header, So we created the **glass_data_headres**.
- We are loading the dataset into pandas dataframe by passing the **dataset location** and the **headers.**
- Next printing the loaded dataframe observations, columns and the **headers name**.

Script Output

5
Shares

```
Number of observations ::  214
Number of columns ::  11
Headers ::  ['Id' 'RI' 'Na' 'Mg' 'Al' 'Si' 'K' 'Ca' 'Ba' 'Fe' 'glass-type']
```

Before we implement the multinomial logistic regression in 2 different ways. Let's understand about the dataset.

To understand the behavior of each feature with the target (Glass type). We are going to create a **density graph**. The density graph will visualize to show the relationship between single feature with all the targets types.

Not getting what I am talking about the density graph. Just wait for a moment in the next section we are going to visualize the density graph for example. Then you will get to know, What I mean by the density graph.

Now let's create a function to create the density graph and stores in our local systems.

```
def scatter_with_color_dimension_graph(feature, target, layout_labels):
    """
    Scatter with color dimension graph to visualize the density of the
    Given feature with target
    :param feature:
    :param target:
    :param layout_labels:
    :return:
    """
    trace1 = go.Scatter(
        y=feature,
        mode='markers',
        marker=dict(
            size='16',
            color=target,
            colorscale='Viridis',
            showscale=True
        )
    )
    layout = go.Layout(
        title=layout_labels[2],
        xaxis=dict(title=layout_labels[0]), yaxis=dict(title=layout_labels[1]))
    data = [trace1]
    fig = Figure(data=data, layout=layout)
    # plot_url = py.plot(fig)
    py.image.save_as(fig, filename=layout_labels[1] + '_Density.png')
```

- The function **scatter_with_color_dimension_graph** takes the feature, target, and the laytout_labels as inputs and creates the density graph I am talking about.
- Later saves the created density graph in our local system.
- The above code is just the template of the plotly graphs, All we need to know is the replacing the template inputs with our input parameters.

5
Shares

Now let's call the above function with the dummy feature and target.

```
def main():
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    print "glass_data_RI :: ", list(glass_data["RI"][:10])
    print "glass_data_target :: ", np.array([1, 1, 1, 2, 2, 3, 4, 5, 6, 7])

if __name__ == "__main__":
    main()
```

Script Output:

```
glass_data_RI ::  [1.52101, 1.5176100000000001, 1.5161799999999999, 1.51766, 1.5
glass_data_target ::  [1 1 1 2 2 3 4 5 6 7]
```

The above are the dummy feature and the target.

- **glass_data_RI:** Is the feature and the values of this feature are the refractive index. These are the first 10 values from the glass identification dataset.
- **glass_data_target:** Is the target and the values are the different glass types. In fact, I covered all the glass types (7 types.)

Now let's use the above dummy data for visualization

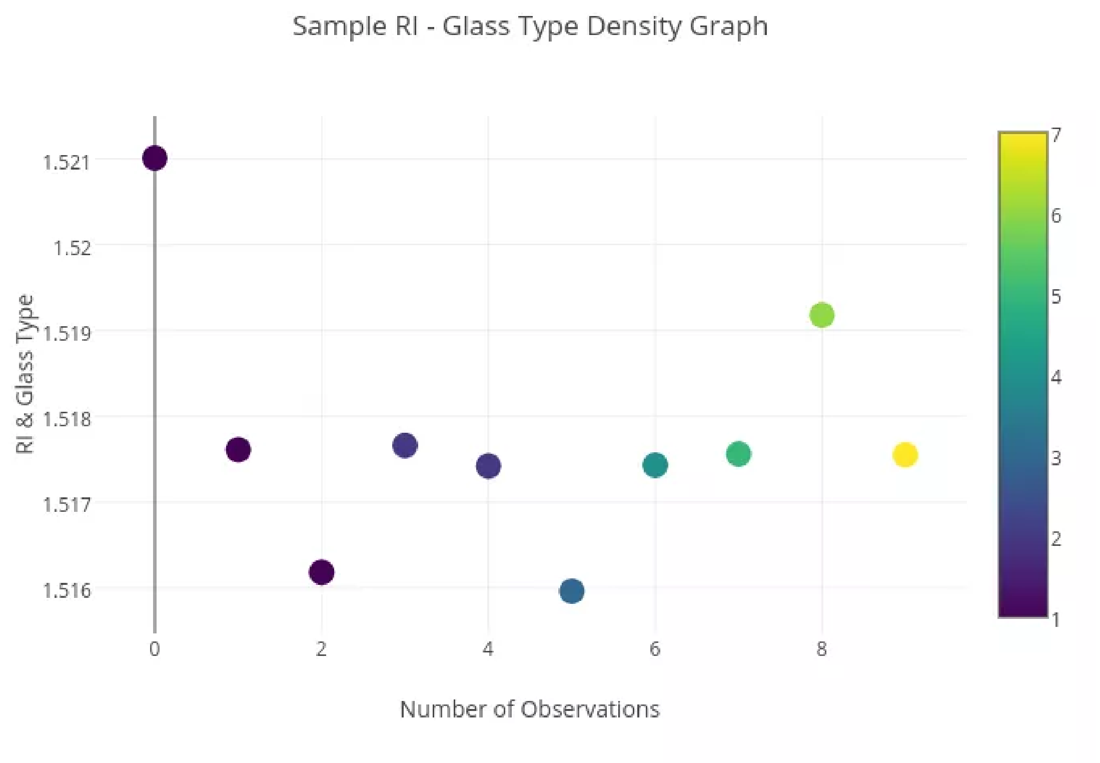
```
def main():
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    print "glass_data_RI :: ", list(glass_data["RI"][:10])
    print "glass_data_target :: ", np.array([1, 1, 1, 2, 2, 3, 4, 5, 6, 7])
    # Graph Labels
    graph_labels = ["Number of Observations", "RI & Glass Type", "Sample RI - G

    scatter_with_color_dimension_graph(list(glass_data["RI"][:10]),
                                       np.array([1, 1, 1, 2, 2, 3, 4, 5, 6, 7])

if __name__ == "__main__":
    main()
```

Calling the **scatter_with_color_dimension_graph** with dummy feature and the target. Below is the density graph for dummy feature and the target.



sample RI & Glass Type_Density

- The above graph helps to visualize the relationship between the feature and the target (7 glass types)
- The Yellow circle is for glass type 7.
- The **right sidebar** will help to know the circle type (target glass type) by its color and the **left side** values are the corresponding feature values.
- If we plot more number of observations we can visualize for what values of the features the target will be the glass type 7, likewise for all another target(glass type)

Now let’s create a function which creates the density graph and the saves the above kind of graphs for all the features.

```
def create_density_graph(dataset, features_header, target_header):
    """
    Create density graph for each feature with target
    :param dataset:
    :param features_header:
    :param target_header:
    :return:
```

```
    """
    for feature_header in features_header:
        print "Creating density graph for feature:: {} ".format(feature_header)
        layout_headers = ["Number of Observation", feature_header + " & " + tar
                          feature_header + " & " + target_header + " Density Gr
        scatter_with_color_dimension_graph(dataset[feature_header], dataset[tar
```

- The function **create_density_graph** takes the dataset, features_header and target_headers as input parameters.
- Inside the function, we are considering each feature_header in the features_header and calling the function scatter_with_clolor_dimenstion_graph.

5
Shares

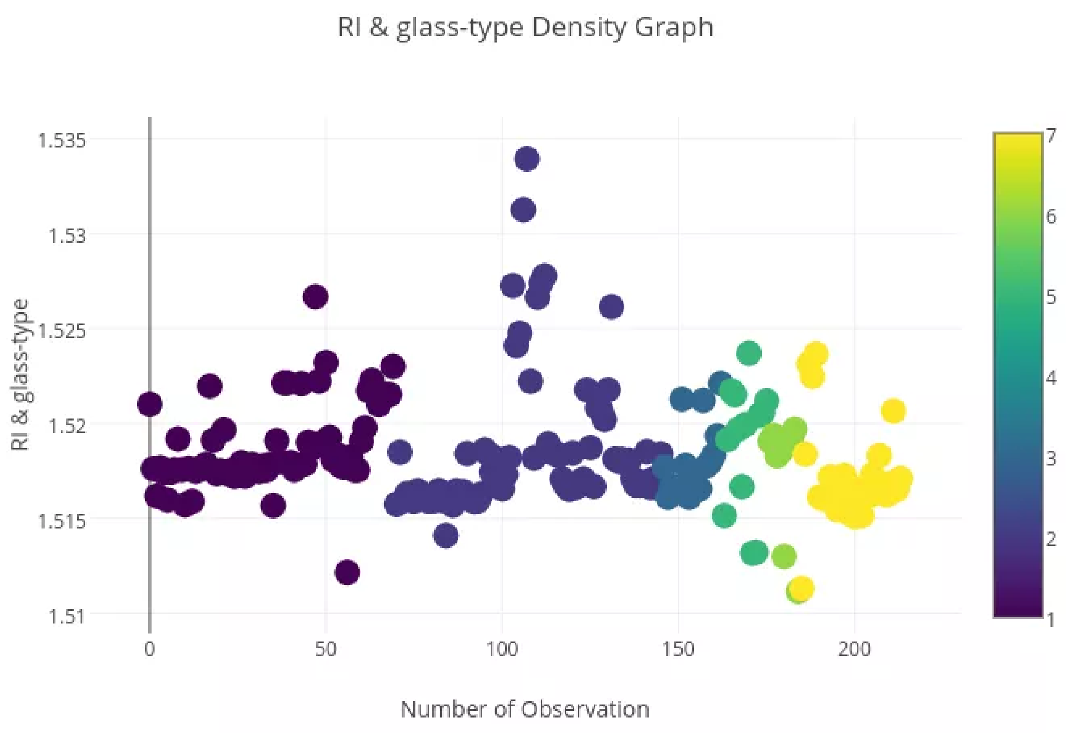
Now let's call the above function inside the main function.

```
def main():
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "
    create_density_graph(glass_data, glass_data_headers[1:-1], glass_data_header

if __name__ == "__main__":
    main()
```

The above code saves the below graphs, Each graph gives the relationship between the feature and the target.

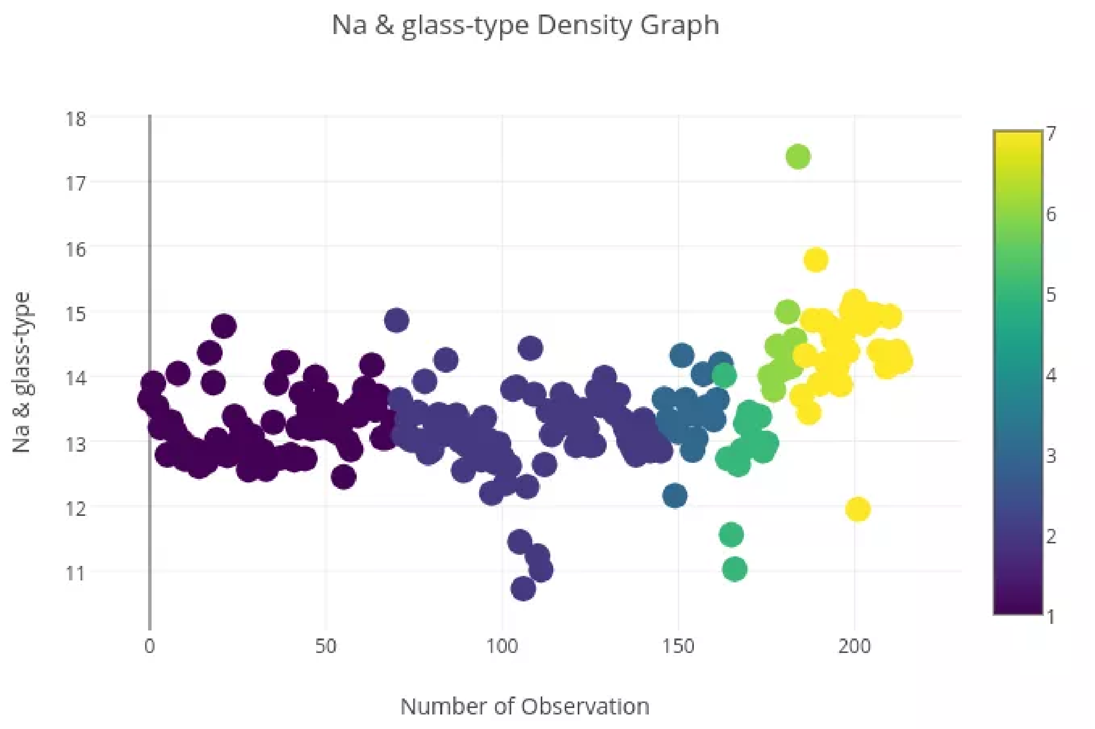
Density graph of Ri and glass type



RI & glass-type_Density

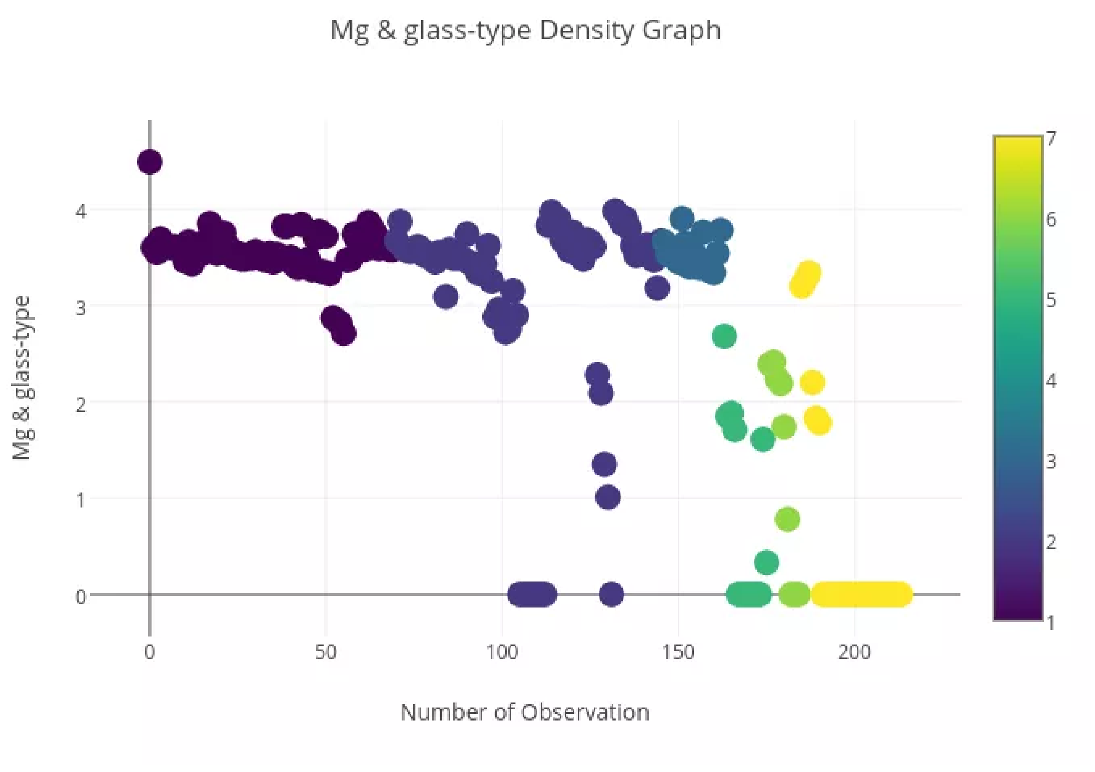
5
Shares

Density graph of Na and glass type



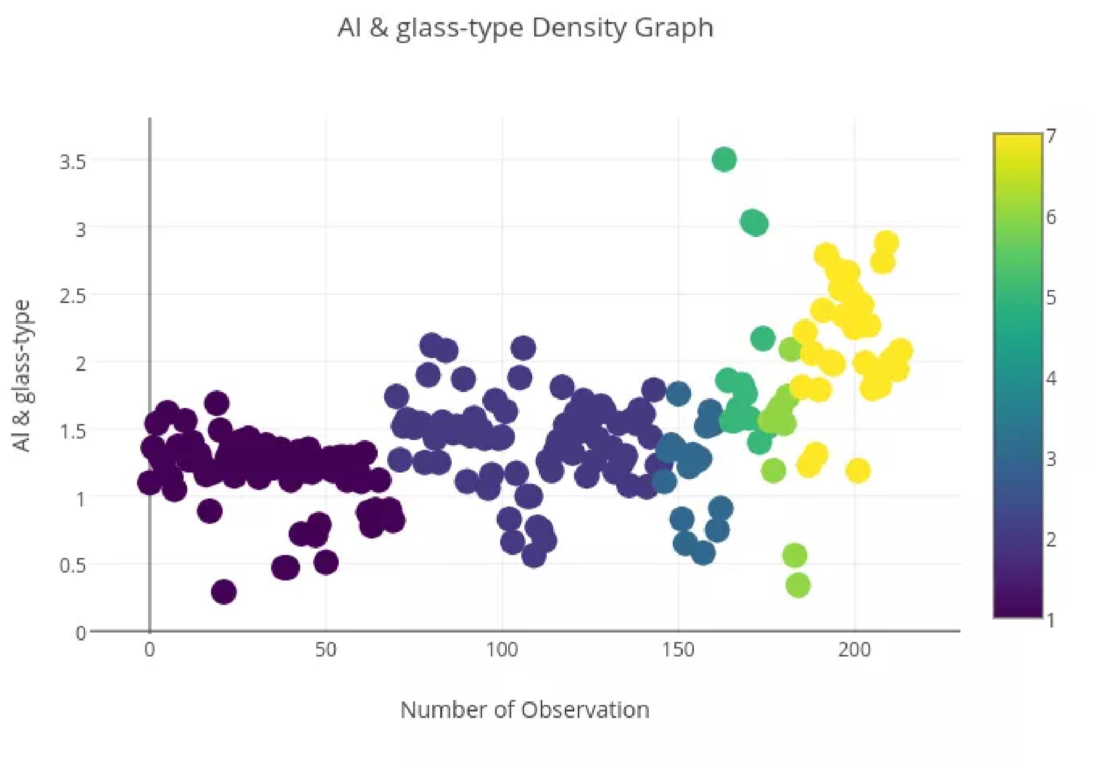
Na & glass-type_Density

Density graph of Mg and glass type



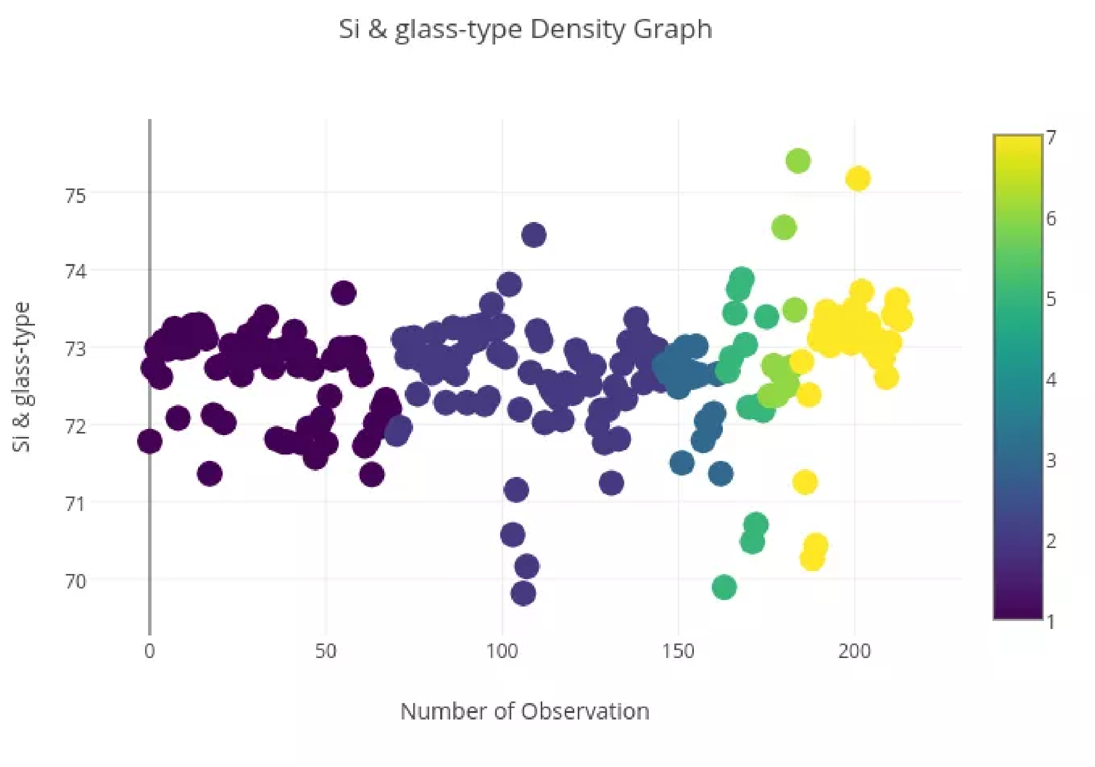
Mg & glass-type_Density

Density graph of Al and glass type



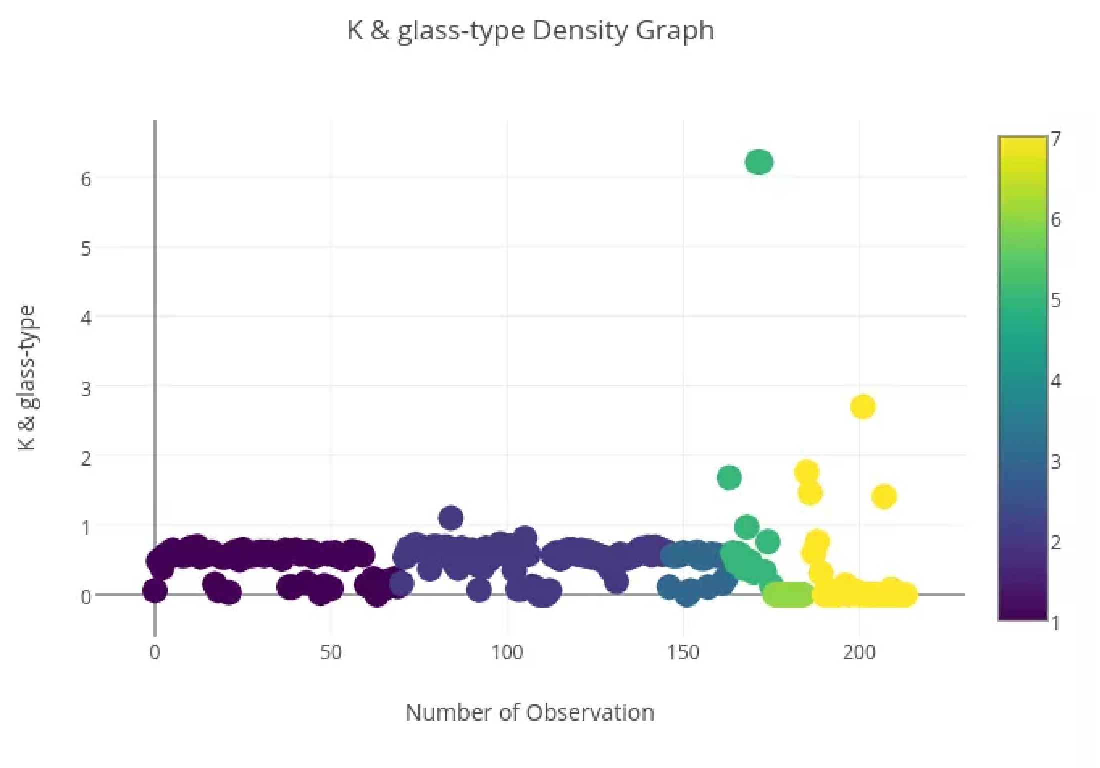
Al & glass-type_Density

Density graph of Si and glass type



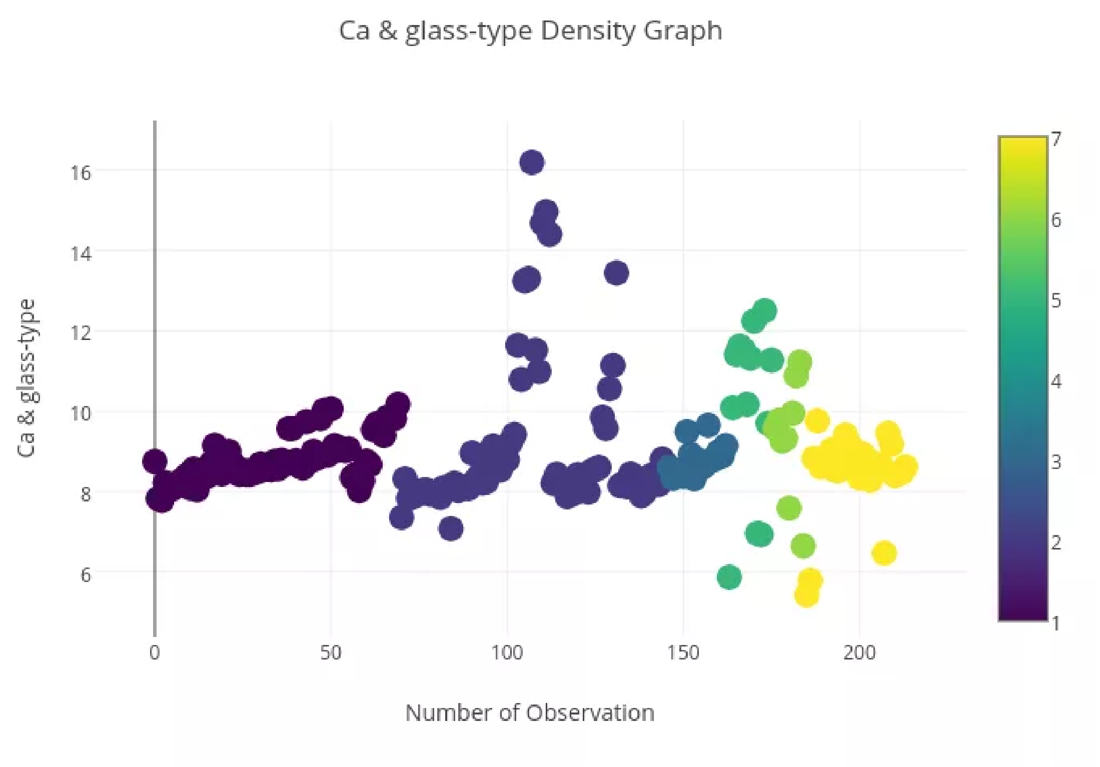
Si & glass-type_Density

Density graph of K and glass type



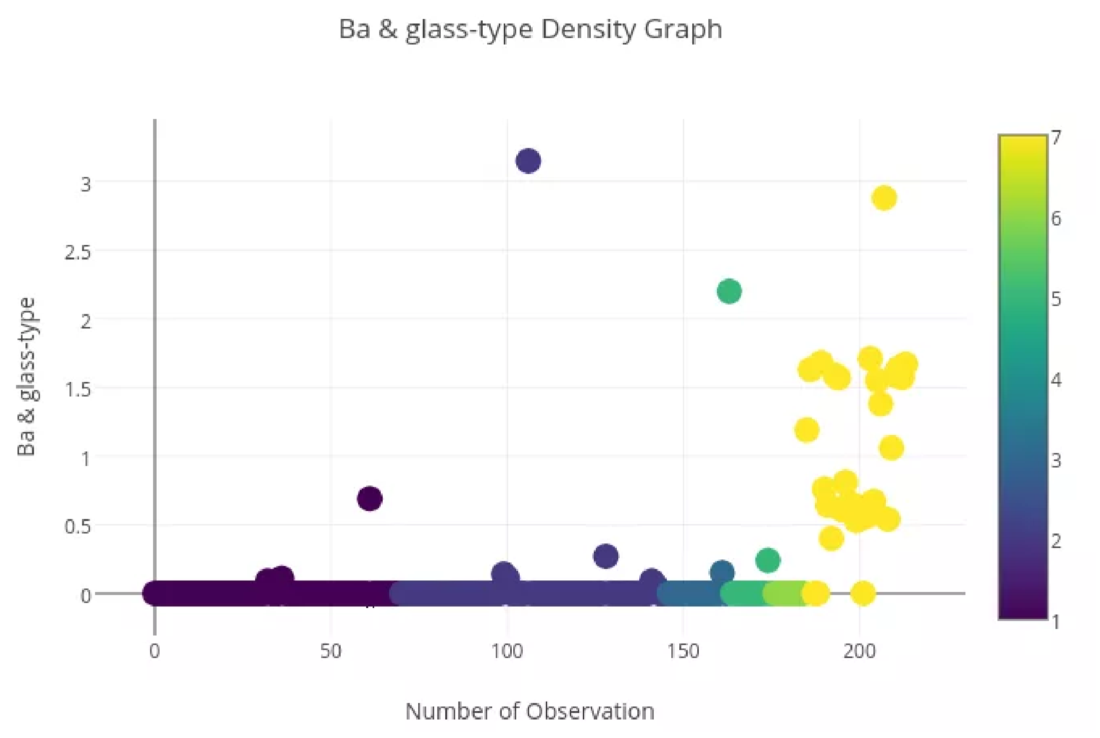
K & glass-type_Density

Density graph of Ca and glass type



Ca & glass-type_Density

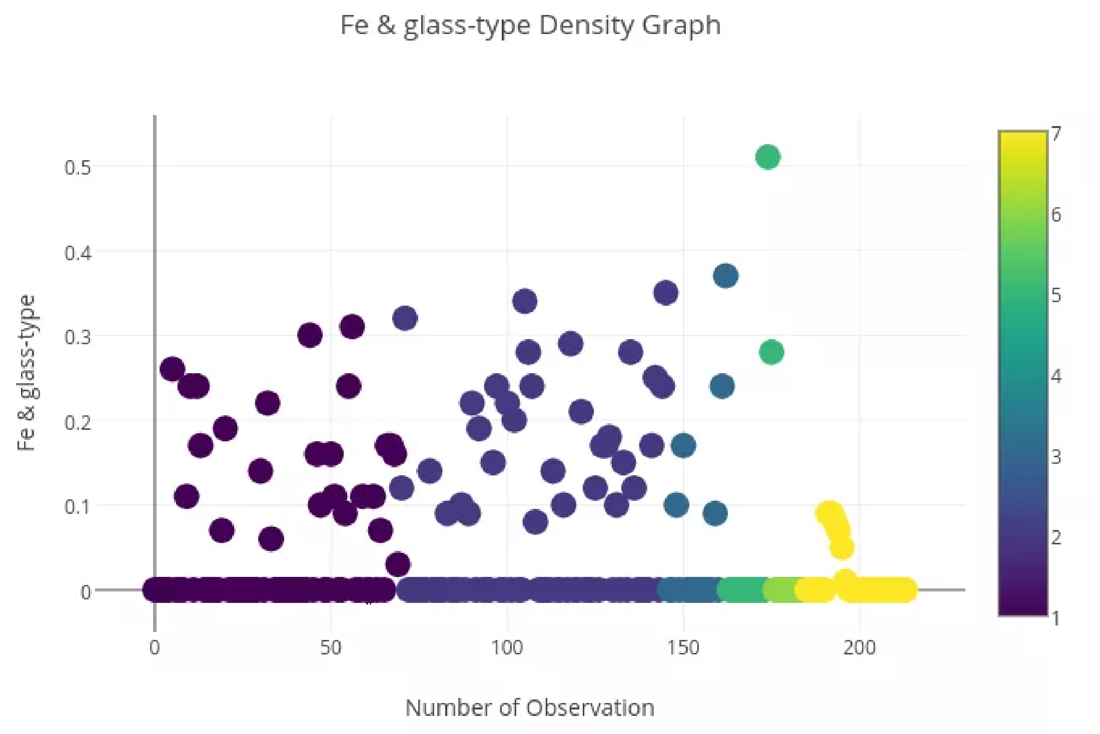
Density graph of Ba and glass type



Ba & glass-type_Density

5
Shares

Density graph of Fe and glass type



Fe & glass-type_Density

5
Shares

Please spend some time on understanding each graph to know which features and the target having the good relationship. So we can use those features to build the multinomial logistic regression model.

To build the multinomial logistic regression I am using all the features in the Glass identification dataset. You use the most suitable features you think from the above graphs and use only those features to model the multinomial logistic regression.

Training the multinomial logistic regression model requires the features and the corresponding targets. For this, we are going to split the dataset into four datasets. Which are

- ***train_x***
- ***test_x***
- ***train_y***
- ***test_y***

We are going to use the ***train_x*** and ***train_y*** for modeling the multinomial logistic regression model and use the ***test_x*** and ***test_y*** for calculating the accuracy of our trained multinomial logistic regression model.

Now let's split the loaded glass dataset into four different datasets.

Split the dataset into training and test dataset

```
def main():
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    train_x, test_x, train_y, test_y = train_test_split(glass_data[glass_data_he
    glass_data[glass_data_headers[-1]], train_size=0.7)

if __name__ == "__main__":
    main()
```

5
Shares

- We are using the scikit-learn **train_test_split** method to split the glass dataset.
- As we are passing **0.7** as the train_size value, The train_test_split method will split the glass dataset randomly **70%** for training and remaining **30%** for testing.

Building the logistic regression for multi-classification

In the first approach, we are going use the scikit learn logistic regression classifier to build the multi-classification classifier.

```
def main():
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    train_x, test_x, train_y, test_y = train_test_split(glass_data[glass_data_h
    glass_data[glass_data_headers[-1]], train_size=0.7)
    # Train multi-class logistic regression model
    lr = linear_model.LogisticRegression()
    lr.fit(train_x, train_y)

if __name__ == "__main__":
    main()
```

- Using the function LogisticRegression in scikit learn linear_model method to create the logistic regression model instance.
- Next using the fit method with the ***train_x*** and ***train_y*** to fit the logistic regression model for the glass identification training dataset.

Implementing the multinomial logistic regression

In the second approach, we are going pass the **multinomial** parameter before we fit the model with ***train_x, test_x***

```
def main():
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    train_x, test_x, train_y, test_y = train_test_split(glass_data[glass_data_h
    glass_data[glass_data_headers[-1]], train_size=0.7)

    # Train multinomial logistic regression
    mul_lr = linear_model.LogisticRegression(multi_class='multinomial', solver=


if __name__ == "__main__":
    main()
```

5
Shares

Comparing the accuracies

No compare the train and test accuracies of both the models.

```
def main():
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    train_x, test_x, train_y, test_y = train_test_split(glass_data[glass_data_h
    glass_data[glass_data_headers[-1]], train_size=0.7)

    # Train multi-classification model with logistic regression
    lr = linear_model.LogisticRegression()
    lr.fit(train_x, train_y)

    # Train multinomial logistic regression model
    mul_lr = linear_model.LogisticRegression(multi_class='multinomial', solver=

    print "Logistic regression Train Accuracy :: ", metrics.accuracy_score(trai
    print "Logistic regression Test Accuracy :: ", metrics.accuracy_score(test_

    print "Multinomial Logistic regression Train Accuracy :: ", metrics.accurac
    print "Multinomial Logistic regression Test Accuracy :: ", metrics.accuracy

if __name__ == "__main__":
    main()
```

- To calculate the accuracy of the trained multinomial logistic regression models we are using the scikit learn ***metrics*** method.
- We are calling the ***metrics*** method ***accuracy_score*** function with actual targets and the predicted targets.

Multinomial logistic regression model Accuracy

```
Logistic regression Train Accuracy ::  0.885906040268
Logistic regression Test Accuracy ::  0.830769230769
Multinomial Logistic regression Train Accuracy ::  1.0
Multinomial Logistic regression Test Accuracy ::  1.0
```

From the result, we can say that using the direct scikit-learn logistic regression is getting **less accuracy** than the multinomial logistic regression model. Now you use the code and play around with.

Multinomial logistic regression Complete Code

```
#!/usr/bin/env python
# multinomial_logistic_regression.py
# Author : Saimadhu Polamuri
# Date: 05-May-2017
# About: Multinomial logistic regression model implementation

import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn import metrics
from sklearn.cross_validation import train_test_split

import plotly.graph_objs as go
import plotly.plotly as py
from plotly.graph_objs import *
py.sign_in('Your_ployly_username', 'API_key')

# Dataset Path
DATASET_PATH = "../Inputs/glass.txt"


def scatter_with_color_dimension_graph(feature, target, layout_labels):
    """
    Scatter with color dimension graph to visualize the density of the
    Given feature with target
    :param feature:
    :param target:
    :param layout_labels:
    :return:
    """
    trace1 = go.Scatter(
        y=feature,
        mode='markers',
        marker=dict(
            size='16',
            color=target,
            colorscale='Viridis',
            showscale=True
        )
    )
    layout = go.Layout(
        title=layout_labels[2],
        xaxis=dict(title=layout_labels[0]), yaxis=dict(title=layout_labels[1])
    data = [trace1]
    fig = Figure(data=data, layout=layout)
    # plot_url = py.plot(fig)
    py.image.save_as(fig, filename=layout_labels[1] + '_Density.png')


def create_density_graph(dataset, features_header, target_header):
    """
    Create density graph for each feature with target
    :param dataset:
    :param features_header:
    :param target_header:
```

```
    :return:
    """
    for feature_header in features_header:
        print "Creating density graph for feature:: {} ".format(feature_header
        layout_headers = ["Number of Observation", feature_header + " & " + ta
                          feature_header + " & " + target_header + " Density G
        scatter_with_color_dimension_graph(dataset[feature_header], dataset[ta


def main():
    glass_data_headers = ["Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",
    glass_data = pd.read_csv(DATASET_PATH, names=glass_data_headers)

    print "Number of observations :: ", len(glass_data.index)
    print "Number of columns :: ", len(glass_data.columns)
    print "Headers :: ", glass_data.columns.values
    print "Target :: ", glass_data[glass_data_headers[-1]]
    Train , Test data split

    print "glass_data_RI :: ", list(glass_data["RI"][:10])
    print "glass_data_target :: ", np.array([1, 1, 1, 2, 2, 3, 4, 5, 6, 7])
    graph_labels = ["Number of Observations", "RI & Glass Type", "Sample RI -
    # scatter_with_color_dimension_graph(list(glass_data["RI"][:10]),
    #                                    np.array([1, 1, 1, 2, 2, 3, 4, 5, 6,

    # print "glass_data_headers[:-1] :: ", glass_data_headers[:-1]
    # print "glass_data_headers[-1] :: ", glass_data_headers[-1]
    # create_density_graph(glass_data, glass_data_headers[1:-1], glass_data_he

    train_x, test_x, train_y, test_y = train_test_split(glass_data[glass_data_
                                                        glass_data[glass_data_
    # Train multi-classification model with logistic regression
    lr = linear_model.LogisticRegression()
    lr.fit(train_x, train_y)

    # Train multinomial logistic regression model
    mul_lr = linear_model.LogisticRegression(multi_class='multinomial', solver

    print "Logistic regression Train Accuracy :: ", metrics.accuracy_score(tra
    print "Logistic regression Test Accuracy :: ", metrics.accuracy_score(test

    print "Multinomial Logistic regression Train Accuracy :: ", metrics.accura
    print "Multinomial Logistic regression Test Accuracy :: ", metrics.accurac


if __name__ == "__main__":
    main()
```

5
Shares

You can fork the complete code at dataaspirant GitHub account

Conclusion

In this article, you learn about

- The key differences between binary and multi-class classification.
- Introduced to the concept of multinomial logistic regression.
- Finally, you learned two different ways to multinomial logistic regression in python with Scikit-learn.

Follow us:

FACEBOOK| QUORA |TWITTER| GOOGLE+ | LINKEDIN| REDDIT | FLIPBOARD | MEDIUM | GI

I hope you like this post. If you have any questions, then feel free to comment below. If you want me to write on one particular topic, then do tell it to me in the comments below.

Related Data Science Courses

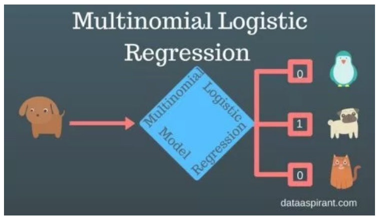
5
Shares

- Implementing supervised learning algorithms with Scikit-learn
- Machine learning classification concepts for beginners.
- Logistic regression model implementation with Python.
- Applying machine learning classification techniques case studies.

Share this:

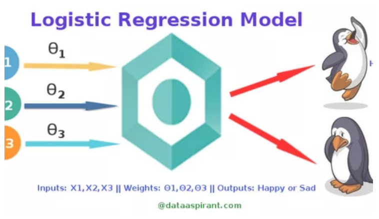
5

Related


Multinomial Logistic Regression
Multinomial Logistic Model Regression
0
1
0
dataaspirant.com

How Multinomial Logistic Regression Model Works In Machine Learning
March 14, 2017
In "Machine Learning"

How to implement logistic regression model in python for binary classification
April 15, 2017
In "Machine Learning"


Logistic Regression Model
θ1
θ2
θ3
1
2
3
Inputs: X1,X2,X3 || Weights: θ1,θ2,θ3 || Outputs: Happy or Sad
@dataaspirant.com

How the logistic regression model works
March 2, 2017
In "Machine Learning"

classification algorithms   logistic regression   multinomial logistic regression