LAB_8_Regression_Logistic_Reg

February 22, 2021

1 CSB352: Data Mining LAB

Instructor : [Dr. Chandra Prakash]

- For more information visit the class website.
- DATE: 22-Feb-2021

2 LAB 8: Regression and Logistic Regression

- Regression
  - Linear Regression
  - Multiple Regression
- Logistic Regression
  - Multiclass classification Logistic Regression

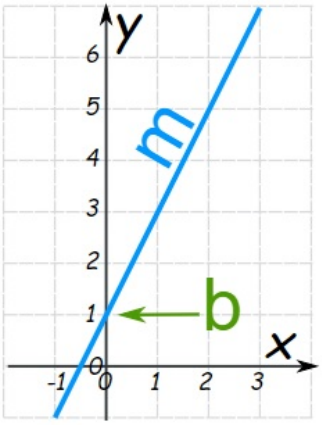
2.0.1 Building Linear Regression in Sklearn

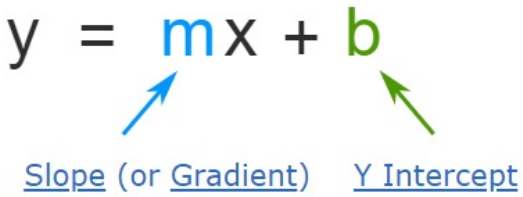
https://scikit-learn.org/stable/index.html

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

2.0.2 Regression

Slope-Intercept Form The most common form is the slope-intercept equation of a straight line:





y=mx+b

Equation of a Straight Line y=mx+b Slope (or Gradient) Y Intercept

Example:

$y = 2x + 1$

Slope: $m = 2$

Intercept: $b = 1$

Explore more about lines here

Straight Line : https://www.mathsisfun.com/data/straight_line_graph.html

Gradient : https://www.mathsisfun.com/gradient.html

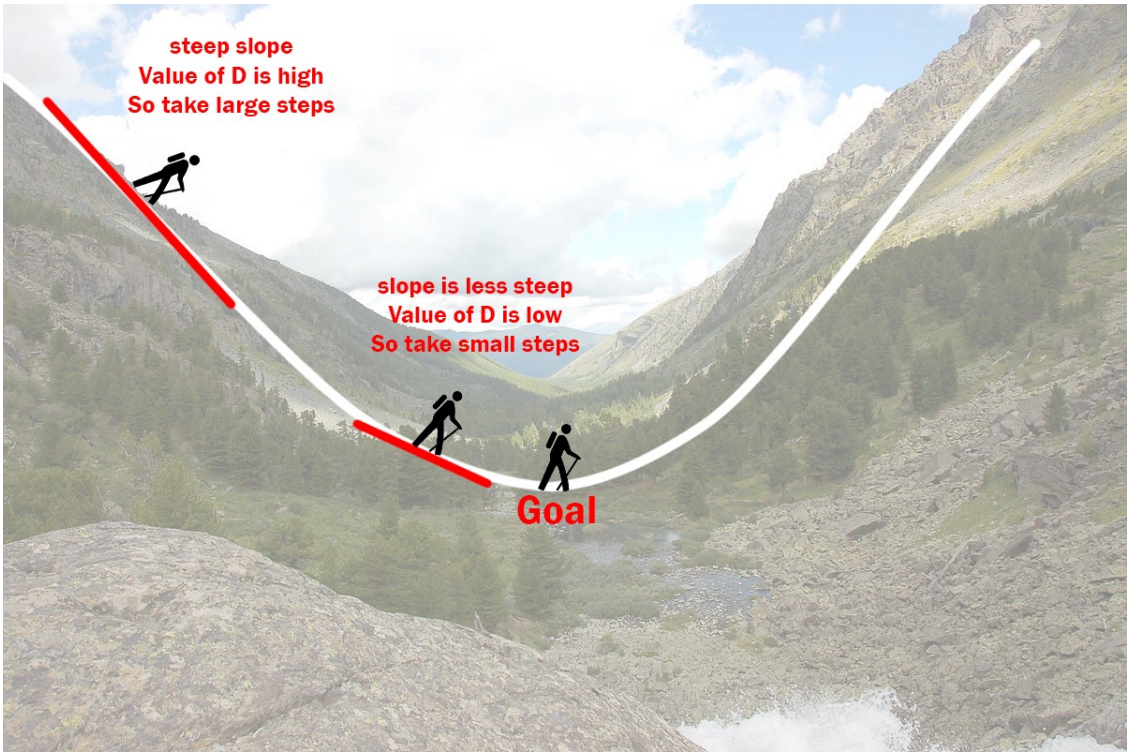
Derivaties : https://www.mathsisfun.com/calculus/derivatives-introduction.html

Play more with derivatives https://www.mathsisfun.com/calculus/derivative-plotter.html

Linear Regression using Gradient Descent :

https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

Illustration of how the gradient descent algorithm works



More Discussion on the Regression - Linear

Linear regression is a method used to find a relationship between a dependent variable and a set of independent variables. In its simplest form it consist of fitting a function y=w.x+b to observed data, where y is the dependent variable, x the independent, w the weight matrix and b the bias.

Illustratively, performing linear regression is the same as fitting a scatter plot to a line. As can be seen for instance in Fig

3 Error Function

Squared Loss, which measures the average of the squared difference between an estimation and the ground-truth value.

MSE - Mean Squared Error

he squared loss function can be seen in Eq. (1), where M is the number of training points, y is the estimated value and y$\widehat{\ }$ is the ground-truth value.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - y_i)^2 \tag{1}$$

e can further expand Eq. (1) in order to incorporate our model. Doing so we obtain Eq. (2).

$$\mathcal{L}(y, x, w) = \frac{1}{M} \sum_{i=1}^{M} \big(\hat{y}_i - (w^T x_i + b)\big)^2 \tag{2}$$

Naturally, we want a model with the smallest possible MSE, therefore we're left with the task of minimizing Eq. (2).

4 Gradient Descent

to minimize Eq. (2) is Gradient Descent, which is based on using gradients to update the model parameters (w and b in our case) until a minimum is found and the gradient becomes zero.

Taking the gradients of Eq. (2) (the derivatives with respect to w and b) yields Eqs. (3) and (4).

$$\frac{\partial \mathcal{L}(y, x, w)}{\partial w} = -\frac{1}{M} \sum_{i=1}^{M} 2x_i \big(\hat{y}_i - (w^T x_i + b)\big) \tag{3}$$

$$\frac{\partial \mathcal{L}(y, x, w)}{\partial b} = -\frac{1}{M} \sum_{i=1}^{M} 2\big(\hat{y}_i - (w^T x_i + b)\big) \tag{4}$$

$$\tag{5}$$

Remember from calculus that the gradient points in the direction of steepest ascent, but since we want our cost to decrease we invert its symbol, therefore getting the Eqs. 5 and 6:

$$w = w - \alpha \frac{\partial \mathcal{L}(y, x, w)}{\partial w} \tag{6}$$

$$b = b - \alpha \frac{\partial \mathcal{L}(y, x, w)}{\partial b} \tag{7}$$

Where is called learning rate and relates to much we trust the gradient at a given point, it is usually the case that 0< <1. Setting the learning rate too high might lead to divergence since it risks overshooting the minimum

Regression Summary :-

- **Hypothesis $h(x)$:**

$$h(x) = \theta_0 + \theta_1(x)$$

- **Cost $J(\theta, \theta_1)$:**

$$J(\theta, \theta_1) = \frac{1}{2}\sum_{i=1}^{m}(h(x)^{(i)} - (y)^{(i)})^2$$

- **Slope (θ_1) :**

$$\theta_1 := \theta_1 - \alpha.\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})\ x_j{}^{(i)}$$

- **Intercept (θ_0):**

$$\theta_0 := \theta_0 - \alpha.\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})$$

5 Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

[ ]:
```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False

# Print your name and Roll No.
# Print the curent time
```

5.1 PART 8.1: Linear Regression using SKlearn

CASE STUDY 1: Student Score

[ ]:
```
import pandas as pd
```

[ ]:
```
dataset=pd.read_csv('student_scores.csv')
```

[ ]:
```
dataset.shape
```

[ ]:
```
dataset.head()
```

[ ]:
```
dataset.describe()
```

[ ]:
```
import matplotlib.pyplot as plt
#matlplotlib inline
```

[ ]:
```
#write your code
dataset.plot(x='Hours',y='Scores',style='*')
plt.title('Hours vs Precentage')
plt.xlabel('Hour Studied')
plt.ylabel('Percentage Score')
plt.show()
```

[ ]:
```
# method to retrieve rows from a Data frame
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

[ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split



```

[ ]:
```
?train_test_split
```

[ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=0)

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[ ]:
```
len(x_train),len(y_train)
```

[ ]:
```
len(x_test), len(y_test)
```

[ ]:
```
x_train
```

[ ]:
```
# import Linear Regression function from sklearn
from sklearn.linear_model import LinearRegression
```

[ ]:
```
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

[ ]:
```
regressor.intercept_
```

```
[ ]: regressor.coef_
```

```
[ ]: # For Testing
y_pred=regressor.predict(x_test)
```

```
[ ]: df=pd.DataFrame({'Actual':y_test, 'Predicited':y_pred})
df
```

```
[ ]: #to see the relationship between the training data values
plt.scatter(x_train, y_train, c='red')
plt.show()
```

```
[ ]: #to see the relationship between the predicted
#brain weight values using scattered graph
plt.plot(x_test,y_pred)
plt.scatter(x_test,y_test,c='red')
plt.xlabel('actual values')
plt.ylabel('predicted values')
```

```
[ ]: # To Evalaute the model
```

```
[ ]: from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))
```

5.2 PART 8.2: Multiple Regression using SKlearn

when we have more than one vaiable/features in input

CASE STUDY 2: Petrol_consumption

Use Multiple Linear Regression to predict the consumption of petrol given relevant variables are the petrol tax, the per capita, income, the number of miles of paved highway, and the proportion of the population with driver's licenses.

Dataset There are 48 rows of data. The data include:

I, the index;

A1, the petrol tax;

A2, the per capita income;

A3, the number of miles of paved highway;

A4, the proportion of drivers;

B, the consumption of petrol.

[ ]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


dataset=pd.read_csv('petrol_consumption.csv')
dataset.shape
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

[ ]:
```
dataset.head(5)
```

[ ]:
```
dataset.describe()
```

[ ]:
```
dataset.corr()
```

[ ]:
```
# install if you are using heatmapz for the first time
# !pip install heatmapz
```

[ ]:
```
# Import the two methods from heatmap library
from heatmap import heatmap, corrplot

plt.figure(figsize=(8, 8))
corrplot(dataset.corr(), size_scale=300);

# Blue means positive, red means negative. The stronger the color, the larger␣
 ↪the correlation magnitude.
```

[ ]:
```
# load the data to input and output vairiable

X = dataset[['Petrol_tax', 'Average_income',␣
 ↪'Paved_Highways','Population_Driver_licence(%)']]
y = dataset['Petrol_Consumption']
```

[ ]:
```
# Split data into train, test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
```

[ ]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[ ]:
```
X_train
```

[ ]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

[ ]:
```
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

[ ]:
```
y_pred = regressor.predict(X_test)
```

[ ]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

[ ]:
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))
```

5.3 PART 8.3: Logistic_Regression_using_Sklearn

[ ]:
```
# Run this cell if you are using scikit-learn for the first time
# ! pip install -U  scikit-learn
```

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import plot_confusion_matrix
```

[ ]:
```
dataset= pd.read_csv('sample_dataset.csv')  # Dataset of diabities
```

[ ]:
```
dataset.head(5)
```

[ ]:
```
dataset.columns
```

[ ]:
```
independent_variables=['Pregnancies', 'Glucose', 'BloodPressure',␣
 ↪'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']
```

[ ]:
```
data= dataset[independent_variables] # Features
label= dataset.Outcome # Target variable
```

[ ]:
```
data.head(3)
```

[ ]:
```
label.head(3)
```

[ ]:
```
# split X and y into training and testing sets
train_data,test_data,␣
↪train_label,test_label=train_test_split(data,label,test_size=0.20)
```

[ ]:
```
train_data.shape, test_data.shape, train_label.shape, test_label.shape
```

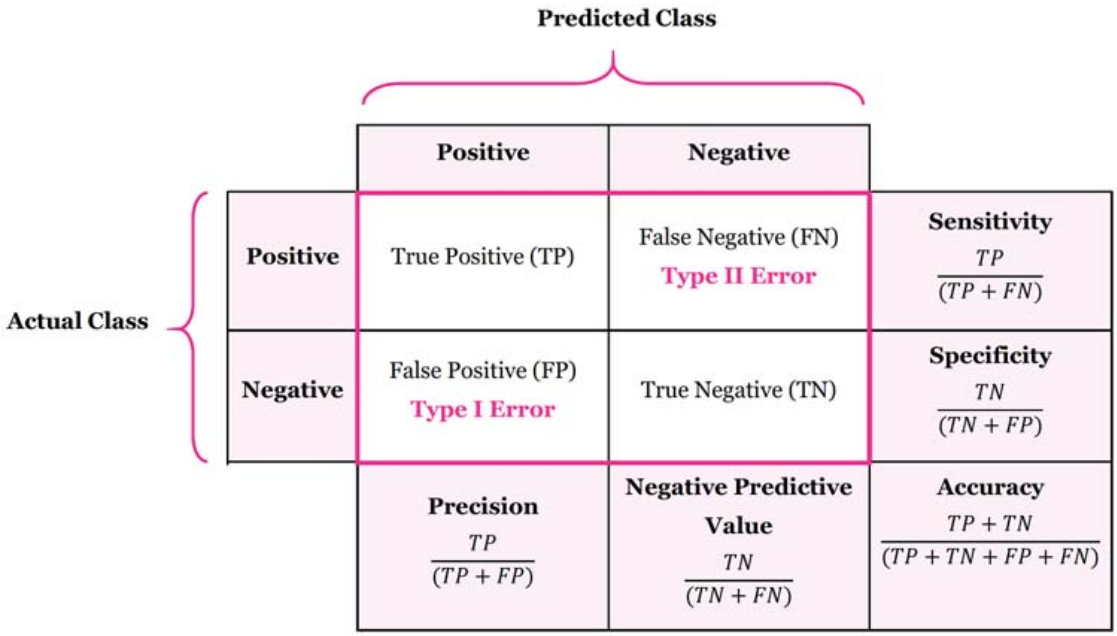
[ ]:
```
# instantiate the model (using the default parameters)
regressor= LogisticRegression()

# fit the model with data
regressor.fit(train_data, train_label)

# predicting model's performancce on test data
predicted_test_label=regressor.predict(test_data)
```

[ ]:
```
confusion_matrix= metrics.confusion_matrix(test_label, predicted_test_label)
confusion_matrix
```

Evaluation Paramters : Confusion Matrix



[ ]:
```
plot_confusion_matrix(regressor, test_data, test_label)

# TruePositive , True Negative, False Positive, False Negative
```

[ ]:
```
print("Accuracy:",metrics.accuracy_score(test_label, predicted_test_label))
print("Precision:",metrics.precision_score(test_label, predicted_test_label))
print("Recall:",metrics.recall_score(test_label, predicted_test_label))
```

[ ]:
```
test_predictions= regressor.predict_proba(test_data)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_label,  test_predictions)
auc = metrics.roc_auc_score(test_label,  test_predictions)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

[ ]:
```

```

Refernce:

1. https://jinglescode.github.io/2019/05/07/why-linear-regression-is-not-suitable-for-classification/
2. https://towardsdatascience.com/logistic-regression-from-scratch-with-numpy-da4cc3121ece
3. https://towardsdatascience.com/using-logistic-regression-to-create-a-binary-and-multiclass-classifier-from-basics-26f5e1e92777

[ ]:
```

```

[ ]:
```

```