Multivariate multilabel classification with Logistic Regression

Introduction:

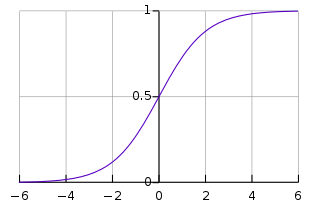
The goal of the blog post is show you how logistic regression can be applied to do multi class classification. We will mainly focus on learning to build a multivariate logistic regression model for doing a multi class classification. The data cleaning and preprocessing parts will be covered in detail in an upcoming post.

Logistic regression is one of the most fundamental and widely used Machine Learning Algorithms. Logistic regression is usually among the first few topics which people pick while learning predictive modeling. Logistic regression is not a regression algorithm but actually a probabilistic classification model.

Classification in Machine Learning is a technique of learning where a particular instance is mapped to one of the many labels. The machine learns patterns from data in such a way that the learned representation successfully maps the original dimension to the suggested label/class without any more intervention from a human expert.

Logistic regression has a sigmoidal curve.

Following is the graph for the sigmoidal function:



The equation for sigmoid function is:



It ensures that the generated number is always between 0 and 1 since the numerator is always smaller than the denominator by 1. See below,



The idea in logistic regression is to cast the problem in the form of generalized linear regression model.



where ŷ =predicted value, x= independent variables and the β are coefficients to be learnt.

This can be compactly expressed in vector form:



In its vanilla form logistic regression is used to do binary classification. Multiclass classification with logistic regression can be done either through one-vs-rest scheme or changing the loss function to cross- entropy loss. Multi class classification can be enabled/disabled by passing values to the argument called ‘‘multi\_class’ in the constructor of the algorithm. In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi\_class’ option is set to ‘ovr’, and uses the cross- entropy loss if the ‘multi\_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’ and ‘newton-cg’ solvers.) By default multi\_class is set to ’ovr’.

**Problem Statement :**

Classify a handwritten image of a digit into a label from 0-9. Use multi class logistic regression for this task.

# **About the Dataset :**

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

* train-images-idx3-ubyte.gz: training set images (9912422 bytes)
* train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
* t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
* t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

|  |  |
| --- | --- |
| **Parameters** | **Number** |
| Classes | 10 |
| Samples per class | ~7000 samples per class |
| Samples total | 70000 |
| Dimensionality | 784 |
| Features | integers values from 0 to 255 |

The MNIST database of handwritten digits is available on the following website: [MNIST Dataset](http://yann.lecun.com/exdb/mnist/)

Import libraries :

|  |
| --- |
| from sklearn.datasets import fetch\_mldata  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  import pandas as pd  import numpy as np |

Load data :

|  |
| --- |
| # You can add the parameter data\_home to wherever to where you want to download your data  mnist = fetch\_mldata('MNIST original') |

Check loaded data :

|  |
| --- |
| print(mnist.data.shape)  print(mnist.COL\_NAMES)  print(mnist.target.shape)  print(np.unique(mnist.target))  (70000, 784) ['label', 'data'] (70000,) [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.] |

Split data into train / test :

|  |
| --- |
| # test\_size: what proportion of original data is used for test set  train\_img, test\_img, train\_lbl, test\_lbl = train\_test\_split(  mnist.data, mnist.target, test\_size=1/7.0, random\_state=122) |

Standardize the data :

|  |
| --- |
| scaler = StandardScaler()  # Fit on training set only.  scaler.fit(train\_img)  # Apply transform to both the training set and the test set.  train\_img = scaler.transform(train\_img)  test\_img = scaler.transform(test\_img) |

Fit the model :

|  |
| --- |
| model = LogisticRegression(solver = 'lbfgs')  model.fit(train\_img, train\_lbl) |

Validate the fitting :

|  |
| --- |
| # use the model to make predictions with the test data  y\_pred = model.predict(test\_img)  # how did our model perform?  count\_misclassified = (test\_lbl != y\_pred).sum()  print('Misclassified samples: {}'.format(count\_misclassified))  accuracy = metrics.accuracy\_score(test\_lbl, y\_pred)  print('Accuracy: {:.2f}'.format(accuracy))  Misclassified samples: 829 Accuracy: 0.92 |

Way ahead :

The model actually has a 92% accuracy score, since this is a very simplistic data set with distinctly separable classes. But there you have it. That’s how to implement multi-variate multi-class with losgistic regression using scikit-learn. Load your favorite data set and give it a try!. From here, all you need is practice.