GLVQ with Renyi Divergence

Release 1.0

Foqrul Islam

Jul 27, 2021

CONTENTS:

CHAPTER

ONE

RENYI

1.1 Renyi_Classes and Functions

class Renyi_final.**GLVQ**(*class_prototype*)

Bases: object

Generlaized Learning Vector Quantization (GLVQ)

Attributes:

class_prototype: The number of prototype of each class to be learned

Renyi_Divergence(*data_in*, *prototypes*)

Calculate the Renyi divergence between datapoints and prototypes.

Parameters

- **data_in** – A n x m matrix of datapoints.
- **prototypes** – A n x m matrix of prototyes of each class

Returns A n x m matrix with Renyi diverence between datapoints and prototypes

calculate_d_minus(*data_labels*, *prot_labels*, *pro_types*, *Renyi_div*)

Calculate the distance between data points and prototypes

Parameters

- **data_labels** – A n-dimensional vector containing the labels for each datapoint.
- **prot_labels** – A n-dimensional vector containing the labels for each prototype.
- **pro_types** – A n x m matrix of prototyes of each class.
- **Renyi_div** – A n x m matrix with Renyi divergence between datapoints and prototypes.

Returns d_minus: A n-dimensional vector having distance between datapoints and prototypes with different label. w_minus: A m x n matrix of nearest incorrect matching prototypes. w_minus_index: A n-dimensional vector having the indices for nearest prototype to datapoints with different label.

calculate_d_plus(*data_labels*, *prot_labels*, *pro_types*, *Renyi_div*)

Calculate the distance between data points and prototypes

Parameters

- **data_labels** – A n-dimensional vector containing the labels for each datapoint.
- **prot_labels** – A n-dimensional vector containing the labels for each prototype.
- **pro_types** – A n x m matrix of prototyes of each class.

- **Renyi_div** – A n x m matrix with Renyi divergence between datapoints and prototypes.

Returns d_plus: A n-dimensional vector having distance between datapoints and prototypes with the same label. w_plus: A m x n matrix of nearest correct matching prototypes. w_plus_index: A n-dimensional vector having the indices for nearest prototype to datapoints with the same label.

change_in_w_minus(*data_in*, *learning_rate*, *classifier*, *w_minus*, *w_minus_index*, *d_plus*, *d_minus*)
Calculate the update of prototypes

Parameters

- **data_in** – A n x m matrix of datapoints.
- **learning_rate** – Learning rate (step size)
- **classifier** – Classify the vector as a winner or runner up prototype.
- **w_minus** – A m x n matrix of nearest incorrect matching prototypes.
- **w_minus_index** – A n-dimensional vector having the indices for nearest prototype to datapoints with different label.
- **d_plus** – A n-dimesional vector having the distance between datapoints and prototypes with the same label.
- **d_minus** – A n-dimesional vector having the distance between datapoints and prototypes with different label.

Returns The result of the updated prototype after calculating the update of prototypes with the same or different label.

change_in_w_plus(*data_in*, *learning_rate*, *classifier*, *w_plus*, *w_plus_index*, *d_plus*, *d_minus*)
Calculate the update of prototypes

Parameters

- **data_in** – A n x m matrix of datapoints.
- **learning_rate** – Learning rate (step size)
- **classifier** – Classify the vector as a winner or runner up prototype.
- **w_plus** – A m x n matrix of nearest correct matching prototypes.
- **w_plus_index** – A n-dimensional vector having the indices for nearest prototype to datapoints with the same label.
- **d_plus** – A n-dimesional vector having the distance between datapoints and prototypes with the same label.
- **d_minus** – A n-dimesional vector having the distance between datapoints and prototypes with different label.

Returns The result of the updated prototype after calculating the update of prototypes with the same or different label.

create_data_prototype(*data_in*, *data_labels*, *class_prototype*)
Calculate prototypes with labels. The calculation is based on the mean or at random based on the prototype in each class.

Parameters

- **data_in** – A n x m matrix of datapoints
- **data_labels** – A n-dimensional vector containing the labels for each datapoint

- **class_prototype** – The number of prototype in each class to be learned. If the number of prototypes

in each class in 1, the prototypes aer assigned in the mean position, otherwise it is assigned at random.

Returns

pro_labels: A n-dimensional vector having the labels for every prototype

Prototypes: A n x m matrix of prototypes for the training purpose.

data_predictor(*input_value*)

The prediction of the labels for the data. The data are represented by the test-to-training distance matrix. Every datapoint will be assigned to the closest prototype.

Parameters **input_value** – A n x m matrix of distances from the test to the training datapoints.

Returns y_label - A n-dimensional vector having the predicted labels for each and every datapoint.

do_processing(*data_in*, *data_labels*, *learning_rate*, *epochs*)

Main function to train the algorithm

Parameters

- **data_in** – A m x n matrix of distance.
- **data_labels** – A m-dimensional vector containing the labels for each datapoint.
- **learning_rate** – Learning rate (step size).
- **epochs** – The maximum number of optimization iterations.

Returns A n-dimensional updated prototype vector.

new_proto_types = array([], dtype=float64)

normalize_data(*data_in*)

Normalizing the data so that the data is between the range 0 and 1 and also we can further the data as a probability distribution.

Parameters **data_in** – A nx m matrix of our input data.

Returns Data_normalized: A n x m matrix with values between 0 and 1

plot(*data_in*, *data_labels*, *prot_types*, *prot_labels*)

Visualizing the data and prototypes in a scatter plot.

Parameters

- **data_in** – A n x m matrix of datapoints.
- **data_labels** – A n-dimensional vector containing the labels for each datapoint.
- **prot_types** – A n x m matrix of prototyes of each class.
- **prot_labels** – A n-dimensional vector containing the labels for each prototype.

prototype_data_labels = array([], dtype=float64)

sigmoid_calc(*j*, *beta=10*)

Calculate the sigmoid activation function.

Parameters

- **j** – A n x m matrix of datapoints or any value
- **beta** – The value of the parameter is taken as 10

Returns It returns values in the range 0 to 1

CHAPTER

TWO

INSTALLATION GUIDE

The following requirements need to be fulfilled to run the program:

- python with minimum version 2.8
- numpy with minimum version 1.19.0
- matplotlib
- Scikit Learn.

2.1 Execution

- Copy the file in a folder
- Install the required packages mentioned in the ‘Installation guide’
- Run the program

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX