# LMLVQ Documentation

***Release 1.0***

**Avinash Maheshwari**

**Sep 24, 2020**

# CONTENTS:

CHAPTER

ONE

# ALGORITHM DESCRIPTION

Learning Vector Quantization(LVQ) is well-known for Supervised Vector Quantization. Large margin LVQ is to maximize the distance of sample margin or to maximize the distance between decision hyperplane and datapoints.

## 1.1 Pseudo-code

1. Get data with labels.
2. Initialize prototypes with labels.
3. Calculate Euclidean distance between datapoints and prototypes.
4. Extract minimum distance from datapoints to each prototypes with same label by using Euclidean distance.
5. Extract distances between datapoints and prototypes with different label by using Euclidean distance.
6. Compute the cost function:

$$E = \overbrace{\sum_{i=1}^{m} d_i^+}^{pull} + \frac{1}{2C} \cdot \overbrace{\sum_{i=1}^{m} \sum_{l \in I_i} ReLU(d_i^+ - d_{i,l} + \gamma)^2}^{push}$$

7. Finally updates the prototypes:

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w(t)}$$

CHAPTER

TWO

# INSTALLATION REQUIREMENTS

Following are the basic requirements to run this program:

1. python with minimum version 3.8.
2. numpy with minimum version 1.19.0.
3. matplotlib.
4. Scikit Learn .
5. Ubuntu required version 20.04.

## 2.1 Execuation

The execuation of the program required following steps:

1. Copy both files in a folder.
2. Go to folder and run the **python3 lmlvq_call.py** command.

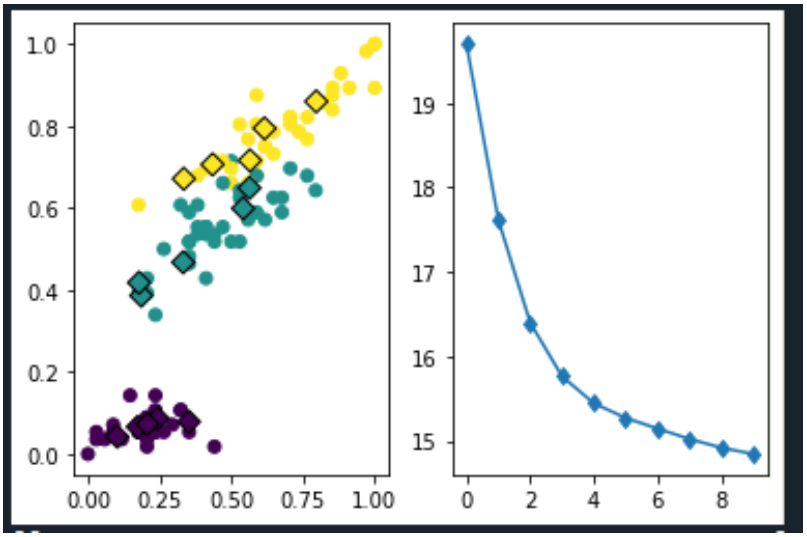
CHAPTER

THREE

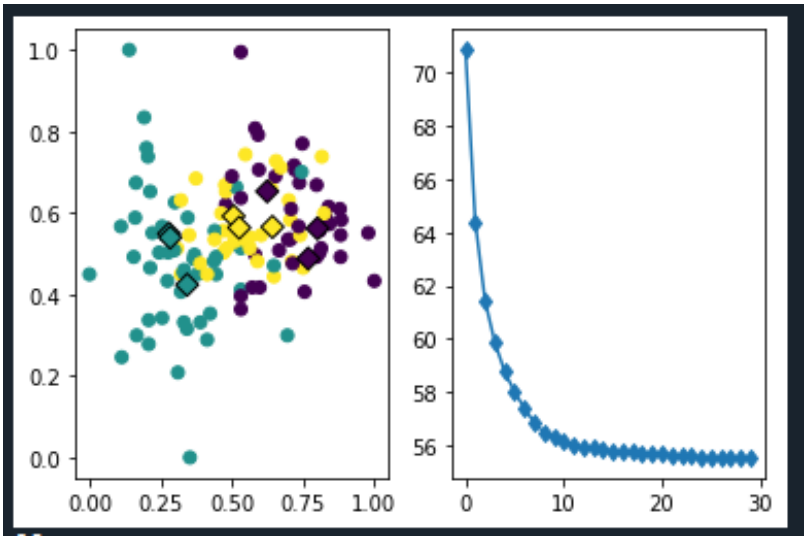
# EXAMPLES

These graph represents the training of model. On the left side it shows the datapoints with different classes and these are represented by set of prototypes with different classes. On the right side it shows the error which is minimizing on every iteration.

## 3.1 Iris Dataset



## 3.2 Wine Recognition Dataset



CHAPTER

# FOUR

# CLASSES AND FUNCTIONS

*class*lmlvq_numpy.**LMLVQ**(*prototype_per_class*)

Large margin LVQ is to maximize the distance of sample margin or to maximize the distance between decision hyperplane and data point.

**Attributes:**

prototype_per_class:

The number of prototypes per class to be learned.

**normalization**(*input_value*)

Normalize the data between range 0 and 1.

**Parameters:**

input_value:

A n x m matrix of input data.

**Return:**

normalized_data:

A n x m matrix with values between 0 and 1.

**prt**(*input_data*, *data_labels*, *prototype_per_class*)

Calculate prototypes with labels either at mean or randomly depends on prototypes per class.

**Parameters:**

input_value:

A n x m matrix of datapoints.

data_labels:

A n-dimensional vector containing the labels for each datapoint.

prototypes per class:

The number of prototypes per class to be learned. If it is equal to 1 then prototypes assigned at mean position else it assigns randolmy.

**Return:**

prototype_labels:

A n-dimensional vector containing the labels for each prototype.

prototypes:

A n x m matrix of prototyes.for training.

**euclidean_dist**(*input_data*, *prototypes*)

Calculate squared Euclidean distance between datapoints and prototypes.

**Parameters:**

input_data:

A n x m matrix of datapoints.

prototpes:

A n x m matrix of prototyes of each class.

**Return:**

A n x m matrix with Euclidean distance between datapoints and prototypes.

**distance_plus**(*data_labels*, *prototype_labels*, *prototypes*, *eu_dist*)

Calculate squared Euclidean distance between datapoints and prototypes with same labels.

**Parameters:**

data_labels:

A n-dimensional vector containing the labels for each datapoint.

prototype_labels:

A n-dimensional vector containing the labels for each prototype.

prototpes:

A n x m matrix of prototyes of each class.

eu_dist:

A n x m matrix with Euclidean distance between datapoints and prototypes.

**Return:**

d_plus:

A n-dimensional vector containing distance between datapoints and prototypes with same label.

w_plus:

A m x n matrix of nearest correct matching prototypes.

w_plus_index:

A n-dimensional vector containing the indices for nearest prototypes to datapoints with same label.

**d_il**(*data_labels*, *prototype_labels*, *prototypes*, *eu_dist*)

Calculate squared Euclidean distance between datapoints and prototypes with different labels.

**Parameters:**

data_labels:

A n-dimensional vector containing the labels for each datapoint.

prototype_labels:

A n-dimensional vector containing the labels for each prototype.

prototpes:

A n x m matrix of prototyes of each class.

eu_dist:

A n x m matrix with Euclidean distance between datapoints and prototypes.

**Return:**

A n x m matrix with Euclidean distance between datapoints and prototypes of different labels.

**cost_function**(*dplus*, *in_dist*, *constant*, *margin*)

Calculate cost function of LMLVQ.

**Parameters:**

**d_plus:** A n-dimensional vector containing distance between datapoints and prototypes with same label.

**in_dist:** A n x m matrix with Euclidean distance between datapoints and prototypes of different labels.

**Constant:** The regularization constant.

**margin:** The margin parameter.

**Return:**

normalized_data:

The value which is the sum of all local errors.

**w_update**(*input_data*, *data_labels*, *prototypes*, *w_plus_index*, *dplus*, *in_dist*, *constant*, *margin*, *_lr*)

Calculate the update of prototypes.

**Parameters:**

input_data:

A n x m matrix of datapoints.

data_labels:

A n-dimensional vector containing the labels for each datapoint.

prototpes:

A n x m matrix of prototyes of each class.

w_plus_index:

A n-dimensional vector containing the indices for nearest prototypes to datapoints with same label.

d_plus:

A n-dimensional vector containing distance between datapoints and prototypes with same label.

in_dist:

A n x m matrix with Euclidean distance between datapoints and prototypes of different labels.

Constant:

The regularization constant.

margin:

The margin parameter.

_lr:

Learning rate also called step size.

**Return:**

The result of updated prototypes after calculated the update of prototypes with same and different labels.

**relu**(*x_value*)

An activation function.

**Parameters:**

x_value:

A n x m matrix of datapoints or any value.

**Return:**

It return the value if it is greater than 0 otherwise returns 0.

**plot**(*input_data*, *data_labels*, *prototypes*, *prototype_labels*)

Scatter plot of data and prototypes.

**Parameters:**

**input_data:** A n x m matrix of datapoints.

**data_labels:** A n-dimensional vector containing the labels for each datapoint.

**prototpes:** A n x m matrix of prototyes of each class.

**prototype_labels:** A n-dimensional vector containing the labels for each prototype.

**fit**(*input_data*, *data_labels*, *learning_rate*, *epochs*, *margin*, *constant*)

TODO.

**Parameters:**

**input_data:** A m x n matrix of distances. Note that we have no preconditions for this matrix.

**data_labels:** A m dimensional label vector for the data points.

**learning_rate:** The step size.

**epochs:** The maximum number of optimization iterations.

**margin:** The margin parameter.

**Constant:** The regularization constant.

**predict**(*input_value*)

redicts the labels for the data represented by the given test-to-training distance matrix. Each 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.

**Return:**

ylabel: A n-dimensional vector containing the predicted labels for each datapoint.