Regularised Generalised Canonical Correlation Analysis (RGCCA)

Submitted By

Anay Gupta - 0801CS171010
Aatmik Jain - 0801CS171003
Shashwat Jain - 0801CS171073

Contents

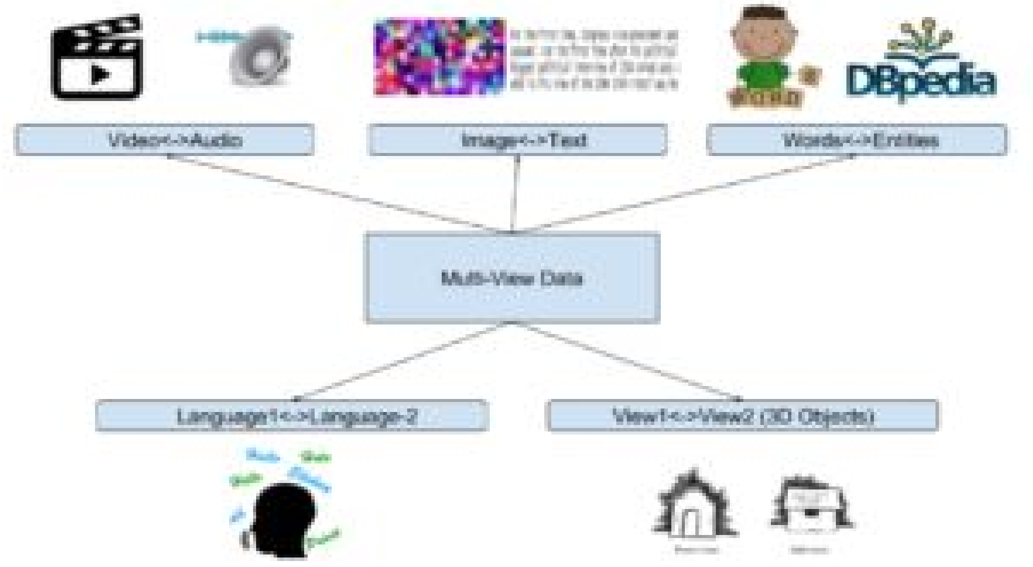
Chapter 1

Introduction

1.1 Multi View Learning

Multi-View Learning (MVL) is a machine learning framework where data is represented by multiple distinct feature groups, and each feature group is referred to as a particular view. It’s aim is to improve the generalised performance and is also referred to as data fusion or data integration.

A multi view can be understood as



MVL approaches can be divided into three major categories

1) Co-training : which exchanges discriminative information between two views by training the two models alternately.

2) Multi-kernel learning : which maps data to different feature spaces with different kernels, and then combines those projected features from all spaces.

3) Subspace learning which assumes all views are generated from a latent common space where shared information of all views can be exploited.

We will be focusing on subspace learning which includes Regularised Canonical Correlation Analysis (RCCA), Regularised Generalised Canonical Correlation Analysis (RGCCA) and Multi View Canonical Correlation Analysis.

1.2 RGCCA

CCA is a method for finding linear correlational relationships between two or more multidimensional datasets. CCA finds a canonical coordinate space that maximizes correlations between projections of the datasets into that space.

Regularized Generalized Canonical Correlation Analysis (RGCCA) is a generalization of regularized canonical correlation analysis to three or more sets of variables. It constitutes a general framework for many multi-block data analysis methods. It combines the power of multi-block data analysis methods (maximization of well identified criteria) and the flexibility of PLS path modeling (the researcher decides which blocks are connected and which are not).

Chapter 2

Mathematical Formula

The second generation RGCCA (Tenenhaus, Tenenhaus, and Groenen 2017) subsumes fifty years of multiblock component methods. It provides important improvements to the initial version of RGCCA (Tenenhaus and Tenenhaus 2011) and is defined as the following optimization problem:

$$\underset{\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_J}{\text{maximize}} \sum_{j,k=1}^{J} c_{jk} g(\text{cov}(\mathbf{X}_j\mathbf{a}_j, \mathbf{X}_k\mathbf{a}_k)) \text{ s.t. } (1-\tau_j)\text{var}(\mathbf{X}_j\mathbf{a}_j) + \tau_j\|\mathbf{a}_j\|^2 = 1, j = 1, \ldots, J$$

Where:

- The scheme function g is any continuous convex function and allows us to consider different optimization criteria. Typical choices of g are the identity (horst scheme, leading to maximizing the sum of covariances between block components), the absolute value (centroid scheme, yielding maximization of the sum of the absolute values of the covariances), the square function (factorial scheme, thereby maximizing the sum of squared covariances).
- The design matrix C is a symmetric J × J matrix of nonnegative elements describing the network of connections between blocks that the user wants to take into account.
- • The τj are called shrinkage parameters ranging from 0 to 1 and interpolate smoothly between maximizing the covariance and maximizing the correlation.

Chapter 3

Algorithm

Input: list of two views of training dataset, X1 ϵ $\mathbb{R}^{d1xn}$, X2 ϵ $\mathbb{R}^{d2xn}$,.. where d_1 and d_2 are the dimensionality of X1 and X2 views and so on.
Output: weights ($[w_1,....w_m]^T$) and correlation between variates

1. Compute $C_{11},...C_{1m}.......\ C_{m1},..\ C_{mm}$
2. $$\begin{bmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} = \lambda \begin{bmatrix} C_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_{mm} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$
    a. Compute left : $\begin{bmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{bmatrix}$
    b. Compute right: $\begin{bmatrix} C_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_{mm} \end{bmatrix}$
3. Make left and right matrices symmetric
4. Solve the generalized eigenproblem in (2)
5. Obtain Eigenvalue: λ and Eigenvector: $[w_1,.....\ w_m]^T$

Chapter 4
Documentation of API

4.1 Package Organization

class rgcca.RGCCA(n_comp=2,reg_param=0.5)

Parameters:

n_comp: the number of components
reg_param: regularization parameter

4.2 Methods

__init__(self,n_comp,reg_param):

To Initialize class
Where self represents the class object itself.
Parameters:

n_comp: the number of components (Default=2)
reg_param: regularization parameter (Default=0.5)

fit(self,data):

To fit the standardized data to RGCCA so as to calculate weights and correlation of variates.

Parameters:

data: datasets in the form of list of length m.

transform(self,data):

To get the reduced data with the weights associated with it by returning dot product of the standardized data and weights.

Parameters:

data: datasets in the form of list of length m.

fit_transform(self,data):

To get the combined result of fit and transform method as reduced data.

Parameters:

data: datasets in the form of list of length m.

Chapter 5

Example 1

Input:

```
X1 = [[ 0.71148954  0.70984814 -0.47941678  1.62813469 -0.99945694]
 [-0.85113942 -0.35030194  0.13286426 -1.00277376  1.58184675]
 [ 1.78327763 -0.34343359 -0.7517771   0.63533286  0.77313637]
 [ 0.96277529 -1.00570691  0.14761596 -1.28274249  0.16195089]]

X2 = [[-0.40699888 -0.81536496 -1.15196797 -1.74384642 -2.29406891 -1.37015486]
 [-0.03635971 -0.19504217  0.87291085  0.28782127 -0.71806635 -0.65524729]
 [ 0.23519534 -0.93199933  0.24420401  2.01186368  0.21149187 -0.8255969 ]
 [-0.77031922  2.49660776  0.14150121  0.65962846  0.24311093  0.83754923]]

X3 = [[-1.33166689 -0.59712277  0.99219584  1.03144507  0.31504197]
 [-0.40241474  0.29632499 -1.04719211  2.13342824  0.44572166]
 [-0.66580483 -0.58220267 -0.50674727  2.23304512  0.38570124]
 [ 0.12736867  1.91923305  0.07612161 -0.20495636  0.45918552]]

X4 = [[ 1.30908113 -0.01670904 -0.42420412 -0.14619312  0.60426733  0.96386107]
 [-1.23830359 -1.69121885 -0.062739   -0.51786337 -0.28521203  0.40653342]
 [ 0.08670155  0.18712859 -1.07533869 -2.06124316  1.0540786   0.66970649]
 [-0.55107394 -0.10192984 -0.77607929  0.88255119  1.21669727 -0.66047047]]

X5 = [[ 0.50494432  0.70942513  1.04529557 -0.37889066  1.46602863 -1.74505813 -0.88456685]
 [ 2.31135103 -0.91306521  2.76092413  0.02172482  0.53206772 -0.14543445 -0.95257197]
 [-0.85118246  0.2435092   1.03624959  0.61809385  1.29543571  1.53533629 -1.07851393]
 [ 1.54363049  0.68479411 -0.25872714 -0.46493316  0.83702913  0.33996983 -1.40869282]]
```

Output:

```
Reduced X1 = [[ 0.14950952, -0.1074133 ],
     [-0.04470615,  0.12327028],
     [ 0.04606077,  0.09328597],
     [-0.15086414, -0.10914295]]

Reduced X2 = [[ 0.14907465, -0.10835852],
     [-0.04043505,  0.1218042 ],
     [ 0.04437258,  0.09886075],
     [-0.15301218, -0.11230643]]
```

```
Reduced X3 = [[ 0.14906902, -0.1081322 ],
       [-0.04427942,  0.12609298],
       [ 0.04756174,  0.09491065],
       [-0.15235135, -0.11287143]]

Reduced X4 = [[ 0.14808853, -0.10711574],
       [-0.04313605,  0.12512864],
       [ 0.04691578,  0.09456834],
       [-0.15186827, -0.11258125]]

Reduced X5 = [[ 0.14864827, -0.10884373],
       [-0.0432544 ,  0.12575197],
       [ 0.0464251 ,  0.09620148],
       [-0.15181898, -0.11310971]]


Weights = [array([[-0.01614776, -0.02513033],
        [ 0.05104157,  0.00838837],
        [-0.03622109, -0.05425096],
        [ 0.04241094,  0.01007883],
        [ 0.00644322,  0.11362143]]),
 array([[ 0.01720324,  0.04269189],
        [-0.03512208, -0.02100869],
        [-0.05660039,  0.07827358],
        [ 0.0054506 ,  0.00216585],
        [-0.0092697 , -0.00432828],
        [-0.03310545, -0.01623868]]),
 array([[-0.03879329,  0.00854463],
        [-0.03023582, -0.02540276],
        [ 0.01593554, -0.0538948 ],
        [ 0.01572327,  0.05523099],
        [-0.03180961,  0.01492771]]),
 array([[ 0.05417325, -0.05179841],
        [ 0.01399727, -0.02841112],
        [ 0.01441902, -0.01809822],
        [-0.01505233, -0.07346871],
        [-0.01274477, -0.02067382],
        [ 0.05643238,  0.0132042 ]]),
 array([[-0.02822243, -0.00697477],
        [ 0.00835558, -0.03476316],
        [ 0.01723545,  0.03010489],
        [ 0.00918298,  0.04073902],
        [ 0.04249867, -0.01462678],
        [-0.02978637,  0.02976326],
        [ 0.04902356,  0.01242482]])]
```

Chapter 6

Learning Outcomes

- Successfully analysed and implemented the Concepts of RGCCA using two methods of fit and transform.
- Realised the use of RGCCA and it’s concepts in calculation and normalisation of eigenvalues and subsequent weights.
- Used the methods in the package on our locally generated data and got satisfactory results as expected from the analysis of the mathematical equations.

References:

1. Pyrcca: Regularized Kernel Canonical Correlation Analysis in Python and Its Applications to Neuroimaging, Natalia Y. Bilenko1 and Jack L. Gallant 1, 2 * , doi: 10.3389/fninf.2016.00049
2. Canonical Correlation Analysis (CCA) Based Multi-View Learning: An Overview by Chenfeng Guo, Dongrui Wu, https://arxiv.org/abs/1907.01693
3. On the regularization of canonical correlation analysis, T. D. Bie and B. D. Moor, https://www.researchgate.net/publication/229057909_On_the_Regularization_of_Canonical_Correlation_Analysis