Sparse Canonical Correlation Analysis (SCCA)

Farida Fakhri - 0801CS171024

Contents

Chapter 1 Introduction

1.1 Multiview Learning and CCA

Multi-view learning is an emerging direction in machine learning which considers learning with multiple views to improve the gener- alization performance. Multi-view learning is also known as data fusion or data integration from multiple feature sets.Canonical correlation analysis (CCA) is a learning method to find linear relationships between two groups of multidimensional variables. The goal of CCA is to seek two bases which would maximize the correlation of data by projecting two-view data obtained from various information sources, e.g. sound and image. In the past decades, CCA and its variants have been successfully applied to many fields such as image processing, pattern recognition, medical image analysis and data regression analysis.

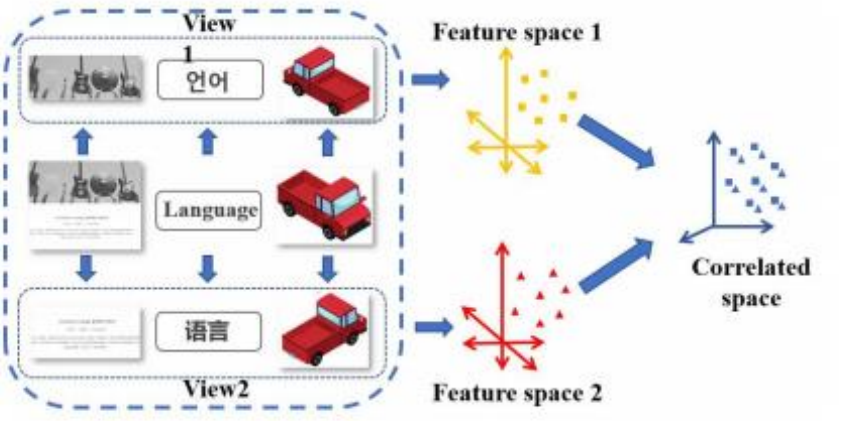


Fig. 1. Multi-view data and CCA-based subspace learning.

1.2 SCCA

Canonical correlation analysis proposed by Hotelling measures linear relationship between two multidimensional variables. In high dimensional setting, the classical canonical correlation analysis breaks down.
Canonical matrices calculated by CCA is dense. When **number of features >> no of rows,** the CCA method is not feasible and the canonical vectors cannot be estimated accurately.Hence, we might want to seek a **sparse** representation of features in X matrix and features in Y matrix so that we can get interpretability of the data. We will purposely introduce sparsity in our canonical vector by adding **l1 constraints** so that the extra un needed dataset features‘s contribution is neglected(weight corresponding to that feature is set 0) in computing our new feature .We also reduce our problem from nonvex to biconvex using SCCA.

Chapter 2

Mathematical Formulation

2.1 Formulation

Canonical correlation studies correlation between two multidimensional random variables. Let x ∈ R^p and y ∈ R^q be random variables, and let Σx, Σy be covariance of x and y respectively, and their covariance matrix be Σxy. In simple words, it seeks linear combinations of x and y such that the resulting values are **mostly correlated**. The mathematical definition is.

$$\underset{u\in\mathbb{R}^p, v\in\mathbb{R}^q}{\text{maximize}} \quad \frac{u^T\Sigma_{xy}v}{\sqrt{u'\Sigma_x u}\sqrt{v'\Sigma_y v)}}.$$

When no of features >> no of data instances we might want to seek a **sparse** representation of features in x and features in y.
Let X ∈ $R^{(n\times p)}$, Y ∈ $R^{(n\times q)}$ be the data matrix. **Regularized version** of the Problem is considered and the canonical vectors are found such that the resulting linear combination of X and Y is mostly correlated.

$$\begin{aligned}\underset{u,v}{\text{minimize}} \quad & -\text{Cov}(Xu, Yv) + \tau_1|u|_1 + \tau_2|v|_1 \\ \text{subject to} \quad & \text{Var}(Xu) = 1; \text{Var}(Yv) = 1,\end{aligned}$$

Since the constraints of minimization problem are not convex,we further relax it as

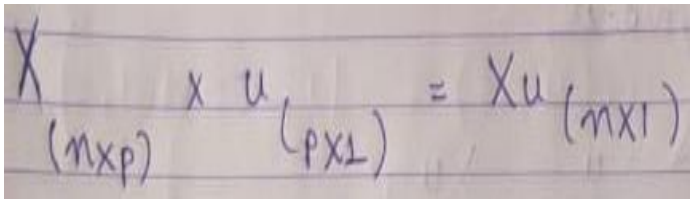
$$\begin{aligned}\text{minimize} \quad & -\text{Cov}(Xu, Yv) + \tau_1|u|_1 + \tau_2|v|_1 \\ \text{subject to} \quad & \text{Var}(Xu) \leq 1; \text{Var}(Yv) \leq 1.\end{aligned}$$

Note that resulting problem is still nonconvex, however, it is a **biconvex**.
u and v are the canonical vector to be calculated for matrix X and Y respectively.

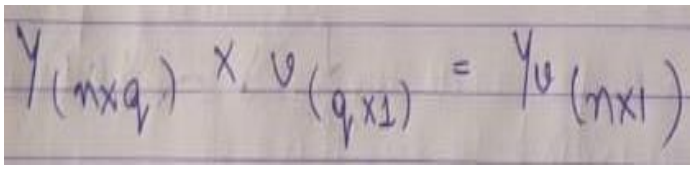
$\tau_1|u|_1 + \tau_2|v|_1$ terms are added to introduce **sparsity**.
The above minimization is done and value of u,v pairs are found using dense neural network.

X : dataset matrix	dimension : n x p
Y : dataset matrix	dimension : n x q
u : canonical vector	dimension : p x 1
v : canonical vector	dimension : q x 1
Xu : projected matrix	dimension : n x 1
Yv : projected matrix	dimension : n x 1

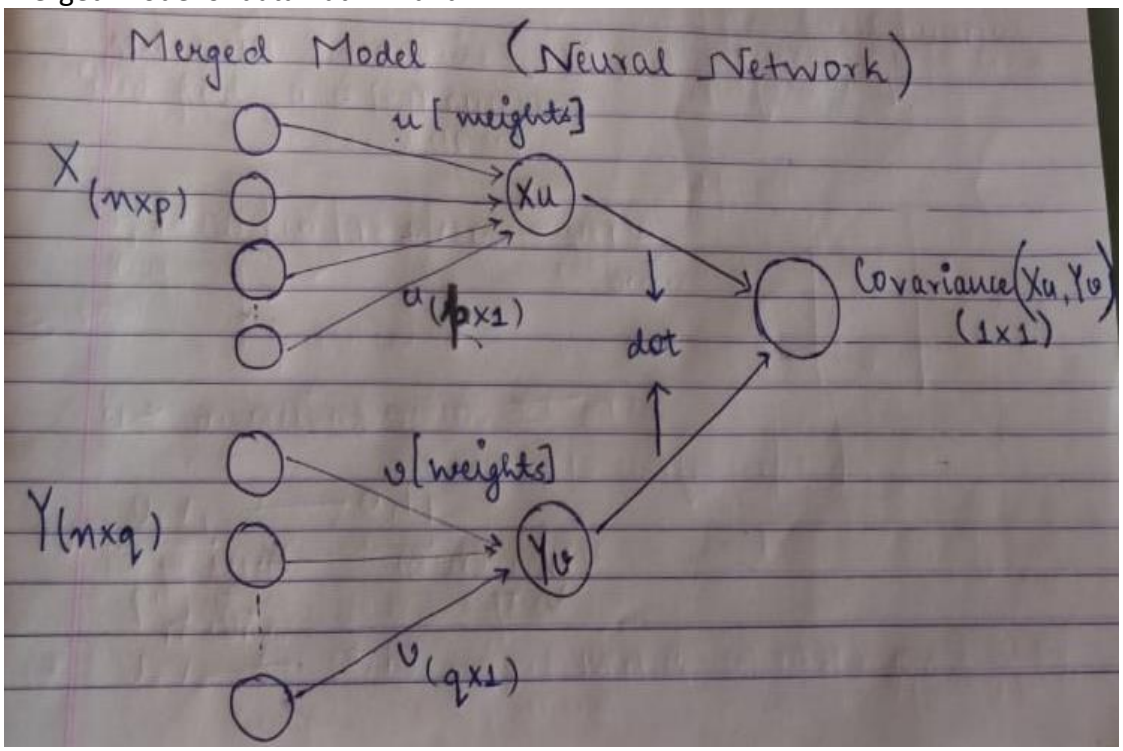
For Datamatrix X :



For Datamatrix Y:



Merged Model of datamatrix X and Y :



A pair of canonical vector is found by this neural network. Subsequent canonical vectors are then calculated by **deflating the data matrices** with the current pair of canonical vectors. This converts the data matrices so that the eigen values of the current pair of vectors become 0 in the new data matrices. This allows us to apply the same optimization technique done to find the first pair onto the new matrices, to find the next best pair of canonical vectors.

Chapter 3

Algorithm

3.1 SCCA algorithm

Algorithm for sparse canonical correlation analysis:
Input: Training sets X,Y represents the data matrices for two views.
Output: Projection Vectors u,v and the projected matrix Xu,Yv.

1. Construct a dense neural network whose input layer has p nodes, matching the dimensions of X(n x p). This input layer is then connected to a single node output layer of dimension (n x 1). The weight vector (of dimension (p x 1)) between the two layers will represent the canonical vector(u). The output of this network will contain the variable Xu. The weights of this network will be constrained to Unit vectors to ensure that $||u||_2$ is = 1. An L1 regularizer is then added to the network to introduce sparsity in the weights. We shall call this network modelX.
2. A similar network is created for the data matrix Y. The weights of which will be canonical vector(v), and its output will represent Yv. This will be called modelY.
3. A merged network is formed by taking the dot product of the outputs of modelX and modelY. The output of this merged vector represents Cov(Xu, Yv).
4. The merged network is then trained with a loss function of -Cov(Xu, Yv) and trained to minimize loss. This results in maximized Covariance.
5. The weights of the trained network are extracted to obtain canonical vectors u, v.
6. The matrix X is deflated with the vector u and similarly Y with v. Pair u,v is added to list of canonical vectors.
7. Steps 1 to 6 are repeated k times to obtain k canonical vectors.
8. The obtained canonical matrices of dimension (p x k) and (q x k) are used to project X and Y respectively onto space (n x 2k).

Chapter 4

Documentation of API

4.1 Package organization

Required Module : NumPy,Keras

Package Name : scca

4.2 Methods

1. **Deflate_inputs(current_inputs, original_inputs,u,v)**

   Used to deflate the data matrix with the current pair of canonical vectors.

   Parameters:

   current_inputs : Deflated data matrix of X and Y
   original_inputs : Original data matrix of X and Y
   u : canonical vector of X dataset
   v : canonical vector of Y dataset

   Returns:

   Deflated data matrix of X and Y

2. **Build_scca_model(params)**

   Creates the Xmodel and Ymodel and merges the two models to create a dense neural network.

   Parameters:

   params['shape_X'] = Shape of X
   params['shape_Y'] = Shape of Y
   params['sparsity_X'] = sparsity in X
   params['sparsity_Y'] = sparsity in Y
   params['algo'] = algorithm of the optimizer
   params['its'] = no of iteration
   params['keras_verbose'] = level of verbosity.
   params['nonneg'] = nonnegative weights
   params['nvecs'] = reduced dimension.

   Returns:

   Returns merged model.

3. **fit(current_inputs,params)**
   Fit model to data. Extracts the weights from the build model and builds a canonical matrix of weights by performing deflation on the current_inputs repeatedly.
   Paramters :
   current_inputs : Deflated dataset of X and Y.
   params['shape_X'] = Shape of X
   params['shape_Y'] = Shape of Y
   params['sparsity_X'] = sparsity in X
   params['sparsity_Y'] = sparsity in Y
   params['algo'] = algorithm of the optimizer
   params['its'] = iteration
   params['keras_verbose'] = level of verbosity.
   params['nonneg'] = nonnegative weights
   params['nvecs'] = reduced dimension.
   Return:
   Canonical matrices u and v.

4. **Transform(orig_inputs,u_comp,v_comp)**
   To get the reduced data, with the given weights.
   Parameters:
   orig_inputs : Original X Dataset and Y dataset.
   u_comp : Canonical matrix u.
   v_comp : Canonical matrix v.
   Returns :
   reduced projected dataset of X and Y.
5. **Fit_transform(inmats, nvecs, sparsity, its, algo, verbose, nonneg)**
   Learn and apply the dimension reduction on the train data. Uses weights obtained from the training and returns the reduced/expected data.
   Parameters:
   inmats : a tuple or list (dataset1,dataset2)
   nvecs : The number of dimensions to project onto.
   sparsity: list of L1 penalty for dataset 1 and dataset2.
   its: No of epochs for training.
   algo: which gradient descent algorithm to use.
   verbose: -1(print nothing),0(print correlations after each run),+1(print full keras training status).
   nonneg: Boolean(whether the weights of components can be non- negative.

   Returns :
   canonical matrix u,canonical matrix v,reduced projected matrix X,reduced projected matrix Y.

Chapter 5

Example

5.1 Example 1

Code:

```
N = 10 # number of instances
NVECS = 2 # number of reduced dimensions
# Generate random data
X = np.random.rand(N, 3)
Y = np.random.rand(N, 4)
```

```
[15] print(X)
     print(Y)
```

```
[[ 0.19611704 -0.11134492 -0.33601391]
 [ 0.17701444  0.0071424  -0.09864914]
 [-0.17783916 -0.15231251 -0.24169747]
 [-0.22114023  0.5737552   0.16319148]
 [-0.18819093  0.1478955   0.18101603]
 [-0.1266482  -0.19508349  0.05042631]
 [ 0.15825013  0.324674    0.15779097]
 [-0.1511678  -0.0188489   0.16059388]
 [ 0.5094792  -0.20016002  0.44770278]
 [-0.17587449 -0.37571727 -0.48436092]]
[[ 1.49759248e-01 -4.04847780e-01 -2.03044454e-02  3.65088412e-01]
 [ 1.82945153e-01  1.54272735e-01 -4.52041017e-01  1.76722321e-01]
 [ 2.37194441e-01  3.61463359e-01  5.46500868e-01  1.87310224e-01]
 [ 2.27718532e-01 -4.11276106e-01  6.96651414e-03 -4.12846248e-01]
 [-7.77341239e-02 -2.23206909e-01 -1.58556406e-01  4.34644463e-01]
 [ 1.79401054e-01  1.05659273e-01  1.61633589e-01  3.40719524e-02]
 [-1.42943548e-01 -1.37047790e-01  3.55257101e-01 -8.15713572e-02]
 [-4.59793751e-02  8.60748203e-02 -3.11741347e-01  3.90652755e-04]
 [-4.28822182e-01 -3.86934488e-02 -1.41456215e-01 -4.45226999e-01]
 [-2.81539200e-01  5.07601846e-01  1.37413592e-02 -2.58583422e-01]]
```

```
# Center X to it's mean
X_mean = X.mean(axis = 0)
X = X - X_mean
# Center Y to it's mean
Y_mean = Y.mean(axis = 0)
Y = Y - Y_mean
(u_comp, v_comp),(x_proj,y_proj = fit_transform([X,Y]
                                  Nvecs = NVECS
                                  Verbose = 1)
```

Output:

```
[ ]  print(u_comp)
     print(v_comp)
```

```
[[-0.34914488 -0.9320403   0.09694686]
 [-0.23017113  0.10298563  0.96768534]]
[[ 0.5728054  -0.0893603  -0.22511925  0.7830898 ]
 [-0.7129282   0.17911138  0.6729026   0.08279005]]
```

```
[ ]  print(x_proj)
     print(y_proj)
```

```
[[ 0.31086409 -0.23446581]
 [ 0.02874904 -0.25423301]
 [ 0.34264812 -0.04215254]
 [-0.08563392 -0.06332965]
 [-0.34439422  0.45420294]
 [ 0.22139881  0.38559536]
 [-0.50689822 -0.34190337]
 [-0.06755955  0.08438034]
 [-0.31157786  0.05602656]
 [ 0.41240372 -0.04412082]]
[[ 0.50587053 -0.29254251]
 [-0.20501331 -0.31907042]
 [-0.09953609 -0.34461807]
 [-0.32847772  0.03204389]
 [ 0.03743188  0.37462904]
 [ 0.04208439  0.32027059]
 [-0.45063054  0.19292063]
 [ 0.37014267 -0.30488982]
 [-0.30311687  0.4166703 ]
 [ 0.43124507 -0.07541364]]
```

5.2 Example 2

Code:

```
N = 10 # number of instances
NVECS = 2 # number of reduced dimensions
# Generate random data
X = np.random.rand(N, 3)
Y = np.random.rand(N, 4)
```

```
print(X)
print(Y)
```

```
[[ 0.04016548 -0.22336117 -0.04847886]
 [ 0.14957587  0.18340814  0.3015352 ]
 [ 0.09501305 -0.41523437 -0.18614446]
 [-0.2945709  -0.45044734 -0.18285987]
 [-0.27596717  0.27453561 -0.19010269]
 [ 0.35963409  0.06277439  0.26217337]
 [ 0.27668742  0.45731846  0.45795336]
 [ 0.04132625  0.25764999 -0.09553801]
 [-0.57038842 -0.32483692  0.15312735]
 [ 0.17852431  0.17819322 -0.4716654 ]]
[[-0.15056553 -0.05961438  0.19645011 -0.46235477]
 [ 0.01885437 -0.27703332  0.13390541  0.16314935]
 [-0.45866123  0.37464344 -0.07259652 -0.01183282]
 [ 0.17437046  0.17493037 -0.04437864  0.13177228]
 [ 0.07736537 -0.34230004  0.3341019   0.21477854]
 [-0.24984965  0.09154892 -0.04782916  0.23162099]
 [ 0.07629388  0.23610805 -0.41775885 -0.18950143]
 [ 0.39409955  0.09140308  0.48187359  0.11790942]
 [-0.41398879  0.04337645 -0.16397172  0.05570606]
 [ 0.53208158 -0.33306257 -0.39979612 -0.25124762]]
```

```
# Center X to it's mean
X_mean = X.mean(axis = 0)
X = X - X_mean
# Center Y to it's mean
Y_mean = Y.mean(axis = 0)
Y = Y - Y_mean
(u_comp, v_comp),(x_proj,y_proj = fit_transform([X,Y]
                                  Nvecs = NVECS
                                  Verbose = 1)
```

Output:

```
[17] print(u_comp)
     print(v_comp)
```

```
[[-0.56763095 -0.80200666  0.18595748]
 [-0.77166414  0.41932675 -0.47822523]]
[[-0.55280066  0.10778094  0.25692865  0.78535473]
 [ 0.3618098  -0.6168689   0.6843872   0.14205755]]
```

```
[18] print(x_proj)
     print(y_proj)
```

```
[[ 0.14732297 -0.10147176]
 [-0.17592572 -0.18271614]
 [ 0.24447343 -0.15841807]
 [ 0.49446517  0.12587339]
 [-0.0988829   0.418986  ]
 [-0.20573182 -0.37657167]
 [-0.43866894 -0.24074875]
 [-0.24786107  0.12183824]
 [ 0.61276667  0.23070612]
 [-0.33195779  0.16252267]]
[[-0.23583141  0.05106513]
 [ 0.12225263  0.29253469]
 [ 0.26598256 -0.44841908]
 [ 0.01454789 -0.0564731 ]
 [ 0.17485664  0.49831178]
 [ 0.31760024 -0.14670189]
 [-0.27288742 -0.43087277]
 [ 0.00840088  0.4327434 ]
 [ 0.23514841 -0.28084946]
 [-0.63007043  0.0886613 ]]
```

Chapter 6

Learning Outcome

6.1 Importance of covariance and basics of dimensionality reduction .
6.2 Usage of multiple views associated to data objects.
6.3 Dimension Reduction using SCCA when number of features are much greater than dataset instances.
6.4 Importance of canonical vectors and using neural networks for minimization for solving optimization problems.
6.5 Analysed and evaluated higher mathematical concepts with the ability to clearly implement and present the conclusions and the knowledge behind it.

Appendix A

References

1. Canonical Correlation Analysis (CCA) Based Multi-View Learning: An Overview Chenfeng Guo and Dongrui Wu.

2. Sparse canonical correlation analysis Xiaotong Suo , Victor Minden , Bradley Nelson , Robert Tibshirani , Michael Saunders June 6, 2017