## Master in Computer Science : Unsupervised Machine Learning

## Goal :

Apply Unsupervised Algorithm on the datasets to understand how clustering techniques and feature selection works.

Git Repo :

### Dataset : marketing\_survey.csv

Here we assume a large product brand ( say Adidas ) runs the following survey !

Users are interviewed to provide opinion about purchase behavior.

(1 = Strongly Agree, 2 = Agree, 3 = Neither Agree nor Disagree, 4 = Disagree, 5

= Strongly Disagree)

All the following features are assigned the user feedback as integer value (1…5)

prefer\_emails, quality\_products\_priced\_high , consult\_multiple\_sites\_before\_buy

, prefer\_television\_ads , prefer\_mobile\_ads , watching\_ads\_makes\_me\_buy ,

prefer\_social\_recommendations , women\_main\_decision\_maker , prefer\_online\_discount\_coupons , prefer\_big\_brands , frequently\_visit\_malls , regularly\_bid\_cheaper\_items, prefer\_mail\_coupons, kids\_items\_buy\_online,

#### Run KMeans Clusterer

As we know KMeans provide iterative distance-based disjoint sets of clusters .

In Weka, we select the marketing survey file and try to generate 6 clusters (k=6)

Scheme:weka.clusterers.SimpleKMeans -N 6 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: marketing\_survey

Time taken to build model (full training data) : 0.06 seconds

Cluster centroids:

Attribute Full Data 0 1 2 3 4 5

(1436) (656) (229) (153) (166) (121) (111)

=====================================================================

v1 1.9735 1.9771 1.9432 1.915 2.0602 1.9917 1.9459

v2 2.1079 2.0076 2.1092 2.7386 2.1205 2.0579 1.8649

v3 2.2632 1.9284 2.0568 4.6144 2.0241 2.0826 1.982

v4 2.2765 1.9101 1.9301 3.6667 2.0723 2.1653 3.6667

v5 2.3628 1.8933 2.0306 2.5752 1.994 3.5702 4.7658

v6 2.2166 1.8826 2.0611 1.9085 2.1506 4.9339 2.0721

v7 2.0954 1.8384 2.2751 2.3399 2.0241 2.843 2.1982

v8 2.3085 1.9055 3.4672 2.3725 2.1325 2.6529 2.0991

v9 2.3245 1.8323 3.7467 2.0915 2.3434 2.5041 2.3964

v10 2.2472 1.875 2.9301 2.3725 2.8795 2.0331 2.1532

v11 2.2974 1.8476 2.1834 2.1895 4.4157 2.3554 2.1081

v12 2.2006 1.9024 2.0393 2.2484 3.3614 2.3388 2.3423

v13 2.1936 1.9345 2.2183 2.2026 2.8193 2.5785 2.3063

v14 2.0884 2.0198 2.1878 2.1176 2.1084 2.1405 2.1622

v15 1.9666 1.968 1.9956 1.8889 1.8976 1.8678 2.2162

We clearly see Cluster 0 offers one of the best clusters which helps us find which customers provided highest rating in most of the required questions in the survey .

We can change the seed value to generate different set of clusters and see which one minimizes the total squared distance from instances to their local cluster centers !

Next we visualize the clusters in Weka.



#### Run Expectation Maximization Clusterer

It finds clusters with prior probabilities. It takes considerably longer time to find the clusters.

It can calculate the cluster membership prob of any instance.

In Weka, we run EM against the same dataset.

Scheme:weka.clusterers.EM -I 50 -N -1 -M 1.0E-6 -S 10

Relation: marketing\_survey

Time taken to build model (full training data) : 29.65 seconds

Log likelihood: -20.13928 ( overall quality measure)

Clustered Instances

0 138 ( 10%)

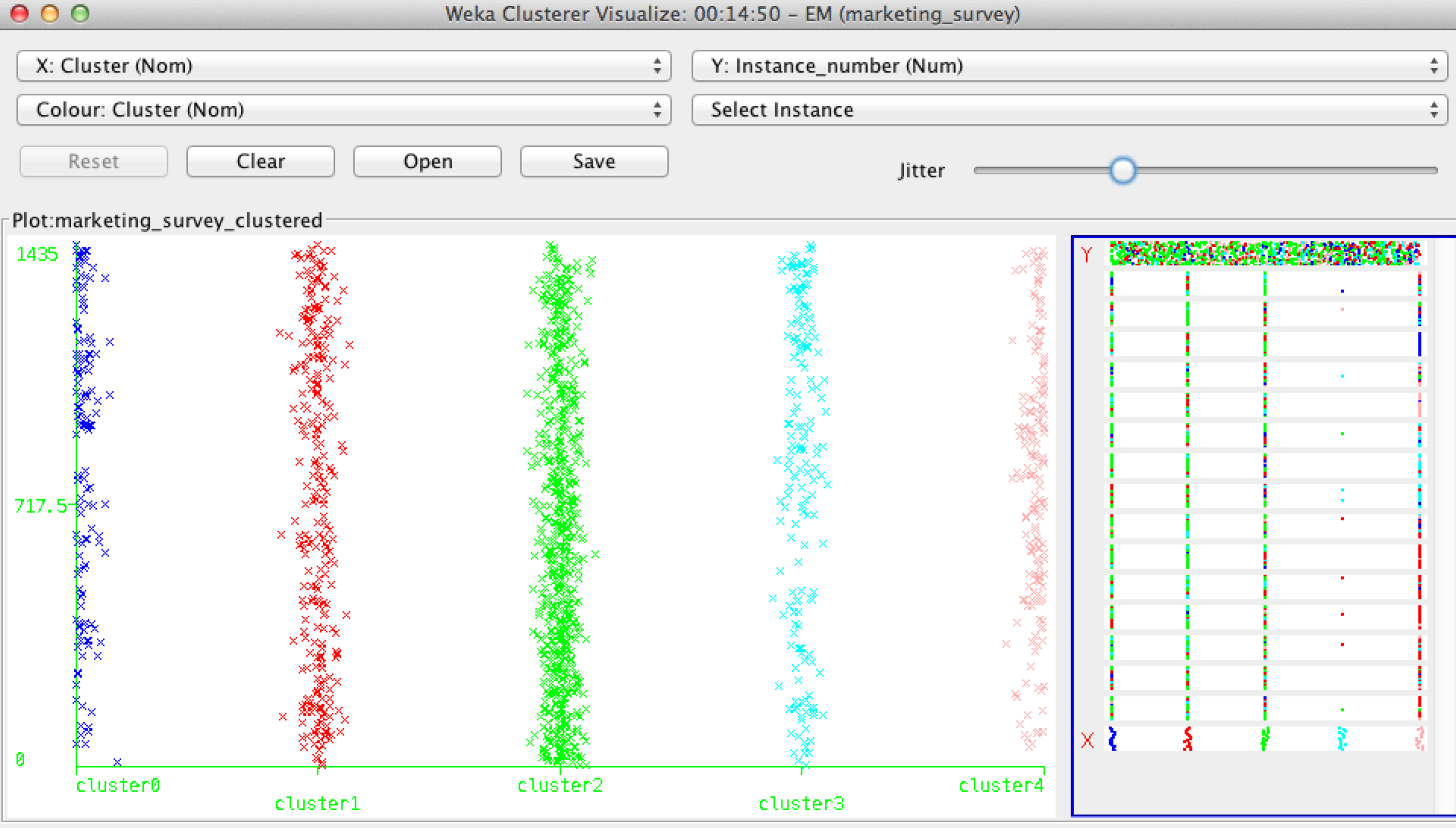
1 274 ( 19%)

**2 675 ( 47%)**

3 178 ( 12%)

4 171 ( 12%)

As wee see in the graph, clearly **Cluster 2** offers the best market segment that the product agency should spend money!



#### Apply the dimensionality reduction algorithms

##### **Principal Component Analysis**

Dimensionality Reduction is achieved by choosing enough Eigen vectors to account for 95% of variance of original data.

Evaluator: weka.attributeSelection.PrincipalComponents -C -R 0.95 -A 5

Search:weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1

Relation: marketing\_survey

Instances: 1436

Attributes: 15

Feature 5 is positively correlated with Feature 4

Feature 5 and 6 each completely uncorrelated with Feature 1 as each of the pair’s covariance is 0.

Covariance matrix

0.69 0.15 -0.01 -0.02 0 0 0.01 0.01 -0.01 -0.02 0.04 0 0 0.02

0.15 1.02 0.29 -0.04 -0.02 -0.05 0.06 -0.01 -0.01 0.03 0.01 0.03 0.06 -0.01

-0.01 0.29 1.27 0.39 0.14 -0.01 0.05 0.08 -0.03 0.06 0.02 0 0.03 0.01

-0.02 -0.04 0.39 1.44 **0.34** 0.01 0.03 -0.03 0.07 0.05 0.02 0.09 0.01 0.04

0 -0.02 0.14 0.34 1.52 **0.35** 0.04 0.06 0.04 0.02 0.02 0.05 0.07 0.02

0 -0.05 -0.01 0.01 0.35 1.24 0.2 0.09 0.1 -0.03 0.11 0.06 0.13 0.05

0.01 0.06 0.05 0.03 0.04 0.2 1.03 0.19 0.04 -0.04 0.05 0.02 0.03 -0.01

….. omitted rest of Matrix for the sake of brevity !

PCA answers the questions - which of these M parameters explain a significant amount of variation contained within the data set?

Top 5 Ranked attributes:

0.864 1 0.443**v5**+0.377**v4**+0.354**v11**+0.327**v9**+0.298v6...

0.7509 2 -0.461**v4**-0.422**v5**+0.382**v9**+0.377**v11**+0.339v10...

0.6513 3 0.519v8-0.505v12-0.415v11+0.388v9-0.294v13...

0.5564 4 0.529v6-0.473v3+0.361v5-0.327v10-0.318v4...

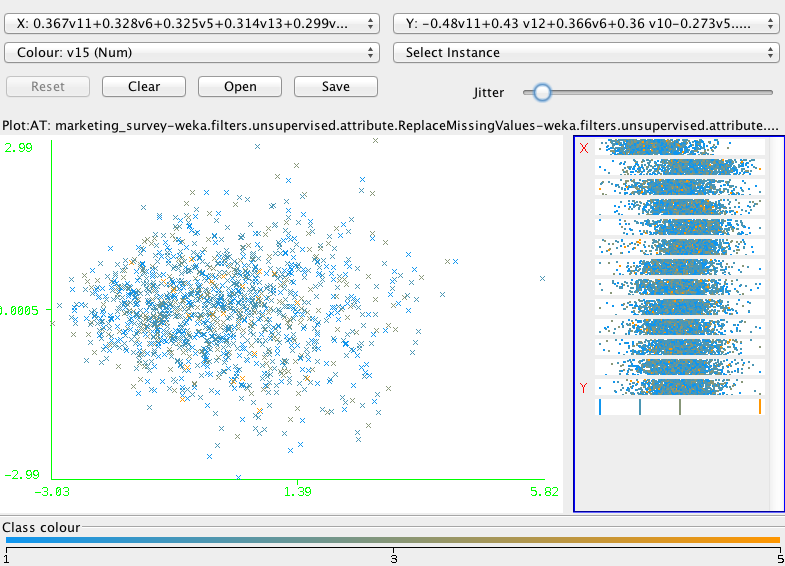
0.4744 5 -0.446v2-0.373v8-0.362v7-0.36v3+0.326v10...

Useful features should have high variance. When features varies then we can use other classifiers like KNN / DT.

If we specify the ‘no of attributes to retain’ in Ranker configuration to 5, then we find exactly top 5 attributes.

I ran the algo few times to generate various combinations of the features.

Here goes one curve that shows pair wise correlation between 2 set of attributes !



**Apply KMeans Clustering on the selected features**

Scheme:weka.clusterers.SimpleKMeans -N 5 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Instances: 1436

Attributes: 13

Time taken to build model (full training data) : 0.13 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 218 ( 15%)

**1 676 ( 47%)**

2 252 ( 18%)

3 152 ( 11%)

4 138 ( 10%)

##### **Independent Component Analysis**

**Run the algo by using the app bundled with the assignment.**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.IndepenentComponentAnalysisTest

It finds best 8 features !

After ICA

Max : 2.854673, 4.733342, 4.023659, 1.913115, 2.575106, 3.897878, 3.574437, 3.953243

Min : -3.930278, -1.991571, -2.217109, -3.912960, -4.241246, -2.396595, -2.387973, -3.622240

0.436075, -1.155682, 0.590050, 0.444946, 0.974476, 1.155473, -1.653201, -0.238860

1.016673, -0.095530, 0.221859, 0.175713, 0.120442, 0.773767, -0.545583, -0.258282

……….. and so on

It generates enough Eigen Vectors in the newly generated feature set to provide an idea about features with higher variations that can be selected for further classifications.

##### **Run KMeans on Independent Component Analysis**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.KMeansWithIndependentComponentAnalysisTest

We perform ICA to find eigen vectors of all the features and then feed them to KMeans Clusterer.

After KMeans

k = 3

-0.684807, -0.033531, 0.357355, -0.462641, -0.054510, -0.196218, -0.050365, 0.201007, -0.058557, -0.007099, -0.143259, 0.701118, 0.029675, 0.127806, 0.159852

0.277581, 0.092826, -0.201979, 0.239617, 0.077794, 0.095226, 0.045132, 0.265410, 0.010526, 0.033431, 0.051937, -0.338022, 0.037585, 0.027126, -0.274720

0.452543, -0.377877, 0.052271, 0.042742, -0.245188, 0.050450, -0.091506, -1.884184, 0.105389, -0.149566, 0.125629, -0.191549, -0.270098, -0.482962, 0.954233

##### **Random Projections**

Next we apply the Random Projection on the same raw dataset.

Using Weka apply RandomProjection filter on the marketing survey dataset.

Reduces the dimensionality of the data by projecting it onto a lower dimensional subspace using a random matrix with columns of unit length (i.e. It will reduce the number of attributes in the data while preserving much of its variation like PCA, but at a much less computational cost).

It first applies the NominalToBinary filter to convert all attributes to numeric before reducing the dimension.

We select numOfAttributes = 15, Sparse distribution model.

Using Weka , we find the Top 6 attributes in accordance with higher variances ( say above 3.5 )

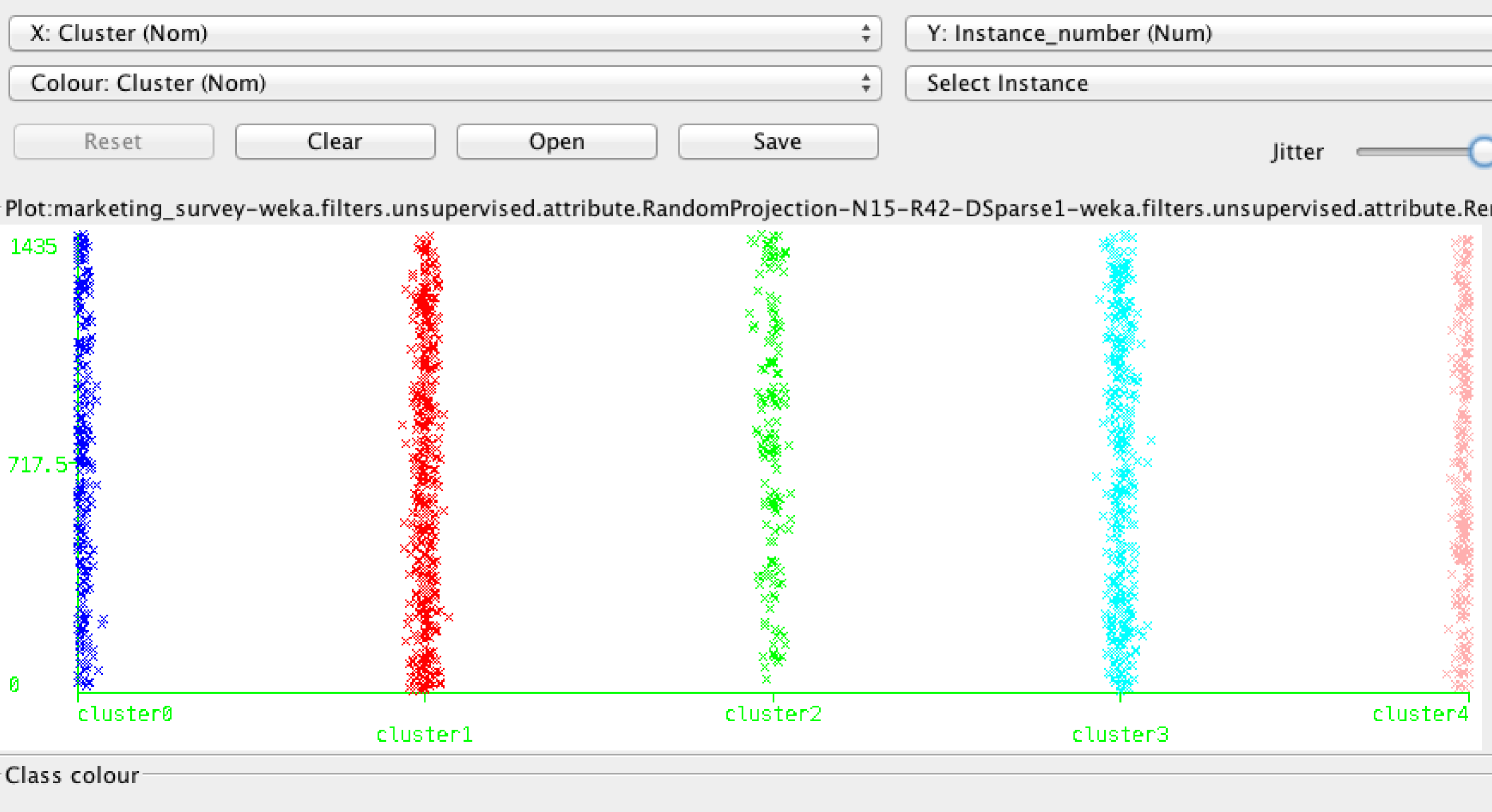
--- K7 (5.2), K5 (4.447), K11 (4.76), K4 (3.9), K1 (3.8), K15 (3.549).

##### **Run KMeans Clusterer on the output of Random Projections (using Weka)**

Scheme:weka.clusterers.SimpleKMeans -N 5 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Instances: 1436

Attributes: 6



We can pick up **Cluster 1** and **Cluster 3** .

\*\* Weka allows us to drill down into data points and save the cluster members.

Time taken to build model (full training data) : 0.05 seconds

Clustered Instances

0 232 ( 16%)

1 428 ( 30%)

2 156 ( 11%)

3 376 ( 26%)

4 244 ( 17%)

##### **Run EM on PCA**

Select marketing\_survey.csv

In the Preprocess step, filter 10 most important attributes

Then in the Cluster step, run EM

Scheme:weka.clusterers.EM -I 50 -N -1 -M 1.0E-6 -S 100

Instances: 1436

Attributes: 10

0.367v11+0.328v6+0.325v5+0.314v13+0.299v12...

-0.555v3-0.448v4+0.316v11-0.313v2-0.283v5...

-0.428v12+0.411v8-0.331v13-0.31v2+0.308v9...

Number of clusters selected by cross validation: 11

Cluster

Attribute 0 1 2 3 4 5 6 7 8 9 10

(0.04) (0.08) (0.17) (0.08) (0.06) (0.05) (0.04) (0.07) (0.03) (0.04) (0.34)

==================================================================================

0.367v11+0.328v6+0.325v5+0.314v13+0.299v12...

mean 0.9637 0.8818 -0.9107 0.733 1.2083 1.0164 0.8749 0.6796 0.9207 1.344 -0.877

std. dev. 0.8655 0.9136 0.7237 1.2939 0.7783 1.0249 0.9326 1.215 1.1168 1.4123 0.8174

………………………………………….. and so on …

Time taken to build model (full training data) : 143.33 seconds

=== Model and evaluation on training set ===

**Log likelihood: -13.61062**

Clustered Instances

0 54 ( 4%)

1 116 ( 8%)

2 208 ( 14%)

3 114 ( 8%)

4 89 ( 6%)

5 68 ( 5%)

6 56 ( 4%)

7 97 ( 7%)

8 46 ( 3%)

9 53 ( 4%)

10 535 ( 37%)

##### **Run EM on PCA**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.EM\_ICATest

After KMeans

Types : {CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS, CONTINUOUS}

Max : 2.026884, 3.456089, 4.366822, 3.609669, 4.678442, 2.887937, 4.568612, 4.521904, 3.286862, 2.362177, 3.948787, 1.902298, 2.346718, 2.552849, 2.700824

Min : -3.758354, -4.239894, -2.148862, -2.258217, -2.494637, -4.051538, -1.899990, -4.100186, -4.069454, -3.952051, -4.165837, -3.863814, -3.744402, -3.675157, -3.899749

{5.3520299E-01, 4.6479701E-01}

mean =

-0.226593, 0.110715, 0.109642, 0.193024, 0.097445, -0.182343, 0.070370, 0.023911, -0.134811, -0.227934, 0.053843, -0.242654, -0.227665, -0.264775, -0.194916

covariance matrix =

1.422561 -0.005264 0.046285 0.081712 -0.006913 -0.088709 0.035489 0.047778 -0.044760 -0.096964 0.043750 -0.102766 -0.083044 -0.139553 -0.100899

### Dataset : CreditRating-train.arff

#### Apply Principal Component Analysis

Ranked attributes:

**attribute** captures the maximum possible **variance** of the data

**0.962** 1 -0.5310.521v21-0.319v18+0.277v14-0.256v13+0.234v7...+0.4620.577v23-0.43v22+0.311v28-0.306v30-0.271v13……

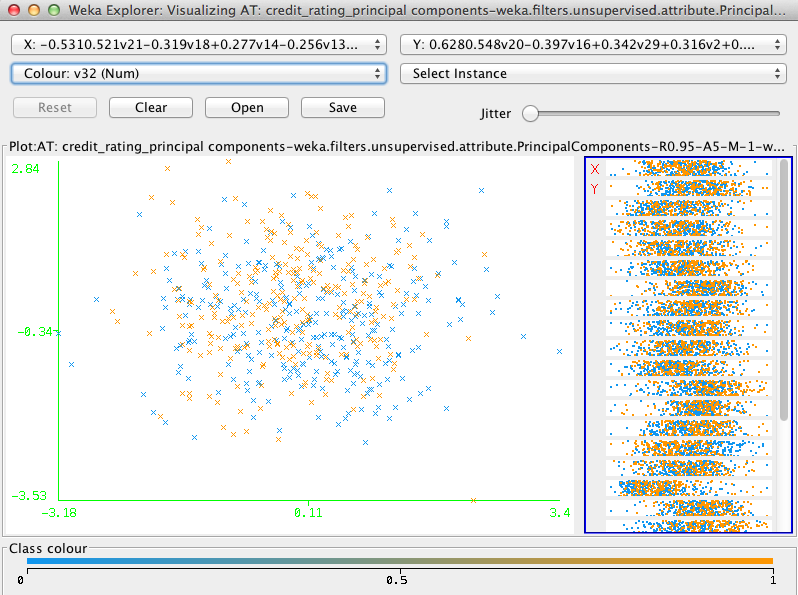
**0.923** 2 0.6280.548v20-0.397v16+0.342v29+0.316v2+0.208v18...+0.4940.398v4-0.391v28+0.342v30-0.329v21-0.322v22……..

**0.885** 3 -0.37-0.549v1-0.479v6-0.316v21+0.248v17+0.225v29...-0.3260.45 v17+0.397v18-0.377v19+0.292v6-0.287v30……..

**0.846** 4 -0.4190.521v9+0.347v31-0.345v6+0.331v18+0.326v19...+0.3650.428v15-0.35v20+0.341v31+0.34 v28+0.279v29…….

**0.808** 5 -0.411-0.446v29-0.344v13+0.338v31-0.306v12-0.262v1........

In general, using one variable as a concomitant for another retains high efficiency only if there is a very high correlation between the variables.



Here we save the output of PCA in the file credit\_rating\_selected\_features.arff

**Run Neural NW Learner**

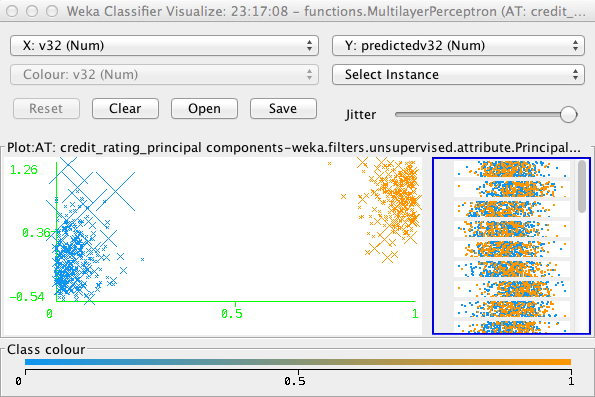
Scheme:weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

=== Evaluation on training set ===

Time taken to build model: 1.45 seconds

Correlation coefficient 0.9665 (correlation between actual and predicated values)

As we see in the chart, NN predicts really well by using the given feature set.



inst#, actual, predicted, error

1 0 0.038 0.038

2 0 -0.181 -0.181

3 0 -0.301 -0.301

4 0 -0.143 -0.143

5 0 -0.395 -0.395

6 0 0.088 0.088

7 0 -0.172 -0.172

8 0 -0.245 -0.245

9 0 -0.136 -0.136

10 0 -0.102 -0.102

11 0 0.089 0.089

12 0 -0.135 -0.135

13 0 -0.032 -0.032

14 0 -0.131 -0.131

…………. ……………. ……….. …………

565 1 0.876 -0.124

566 1 0.967 -0.033

567 1 1.017 0.017

568 1 0.906 -0.094

569 1 0.963 -0.037

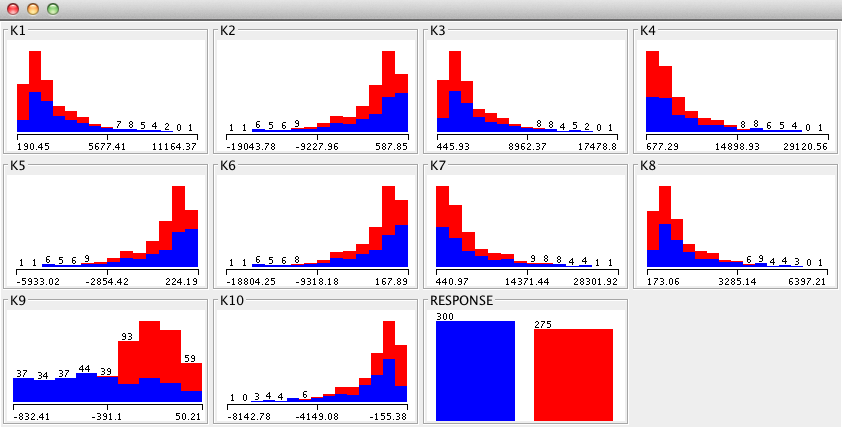
570 1 0.957 -0.043

**Run Random Projections**

Select credit\_rating\_training.arff and apply the RandomProjection Filter in the Preprocess step.

It uses Gaussian distribution to create the random matrix.

A Random set of 10 projections are created with high variances .



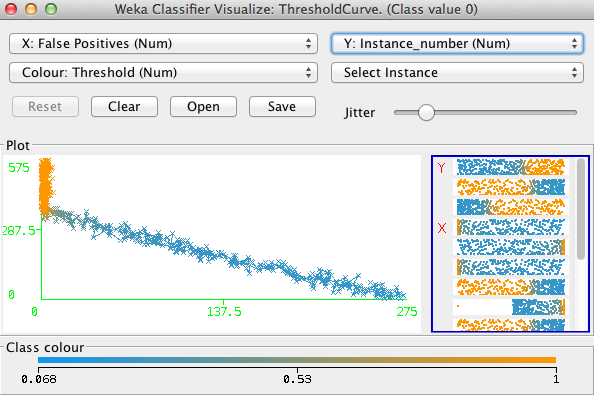
**Run Neural NW Learner**

Scheme:weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Correctly Classified Instances 480 83.4783 %

Incorrectly Classified Instances 95 16.5217 %

As wee very small number of False Positives and False Negatives !



Multiple sets of Random Projections were generated ( e.g. … ) and MLP Learner was applied !

Looks like its best classification rate is 84%

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.707 0.025 0.968 0.707 0.817 0.863 0

0.975 0.293 0.753 0.975 0.849 0.863 1

Weighted Avg. 0.835 0.154 0.865 0.835 0.832 0.863

=== Confusion Matrix ===

a b <-- classified as

212 88 | a = 0

7 268 | b = 1

Performance Analysis

Precision (P) = tp/(tp+fp) Recall (R) = tp/(tp+fn) Accuracy (A) = (tp+tn)/Total # samples

**Apply ICA on Credit Rating dataset and Run Neural Network Classifier :**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.NNWithIndependentComponentAnalysisTest

Label Description: Types : {BINARY}, Max : 1.000000, Min : 0.000000

After ICA

Max : 1.000000

Min : 0.000000

-2490.588268, -372.481954, 20.922994, -15.565248, -1.264527, 0.666688, -0.099465, 0.319200, 0.615871, -0.912240, -0.256310, 0.174429, -0.657022, -0.567082, 0.148123 : 0.000000

-1409.574754, -363.080409, -15.380724, -4.914063, -2.058954, -0.476745, -0.762651, 0.868919, -1.019714, 0.455058, -0.287246, -0.507575, 0.814973, 0.580407, 0.029908 : 0.000000

……

Convergence in 500 iterations

~~

0.604827

~~

0.022379

**Run KMeans Clusterer and then apply NN on the Cluster points :**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.NNWithKMeansClusterTest

Convergence in 500 iterations

True Positives % 84.0

**Run EM Cluster and then apply NN on the Cluster points :**

java -cp unsupervised\_algo.jar:lib/ABAGAIL.jar com.ml.unsupervised.tests.NNWithEMClustererTest

Convergence in 500 iterations

True Positives % 82.0