Classification Using NNS Clustering Analysis

Viole, Fred
fred.viole@gmail.com

June 23, 2017

1 Introduction

NNS stands for Nonlinear Nonparametric Statistics, henceforth "NNS". What is NNS clustering analysis? NNS clustering is a method of partitioning the joint distribution into partial moment quadrants (clustering), and assigning identifiers to observations (classification). NNS clustering is very similar to k-means clustering and vector quantization, and we direct the reader to Vinod and Viole [1] for a proof and comparison between the methods. This article is intended to present working examples of several classification problems using NNS clustering analysis. NNS further offers: Numerical integration, Numerical differentiation, Clustering, Correlation, Dependence, Causal analysis, ANOVA, Regression, Classification, Seasonality, Autoregressive modelling, Normalization and Stochastic dominance with examples available at https://github.com/OVVO-Financial/NNS/tree/NNS-Beta-Version/examples.

We demonstrate how NNS clustering is quite effective, as well as an alternative method NNS employs for classification tasks.[1] We compare predictions of test sets with NNS, k-means using the cl.predict routine offered in R to "predict class ids or memberships from R objects representing partitions"[2], K nearest neighbors classification using the "knn" routine in R-package "class",[3] and a naive Bayes classification using the "e1071" package.[4]

2 NNS Classification Methods

NNS offers 2 methods of classification: a multivariate regression; and a reduced dimension regression. A *brief* description of each follows. The reader is strongly encouraged to visit the references for a thorough explanation of the NNS methodology and benefits.

[1]The following document was prepared using NNS v.0.3.4 available on GitHub, https://github.com/OVVO-Financial/NNS

[2]https://cran.r-project.org/web/packages/clue/clue.pdf

[3]http://stat.ethz.ch/R-manual/R-devel/library/class/html/knn.html

[4]https://cran.r-project.org/web/packages/e1071/e1071.pdf

2.1 NNS Multivariate Regression

Vinod and Viole [1] note the similarity between k-means clustering and `NNS` partitioning as originally put forth by Viole and Nawrocki [2]. `NNS` multivariate regression extends the clustering technique of the univariate regression to multiple regressors. A distance kernel[5] is used for classification, whereby the nearest (`"n.best"` parameter) regression points (cluster means) are weighted and averaged for a predicted value.

2.2 NNS Dimension Reduction Regression

Viole and Nawrocki [2] explain the dimension reduction technique. In short, they create a synthetic regressor (X*) by weighting the `NNS` nonlinear correlations of each regressor to the dependent variable. A simple `NNS` regression of Y on X* is then performed.

2.3 NNS Stacked

Stacking both `NNS` techniques provides even more robust classifications. The `NNS.stack` routine allows for two weighting schemes of base `NNS` predictions, either by mean squared error (MSE), or by number of features. Problems with a high number of features will benefit `NNS Dimension Reduction`. MSE weighting completely ignores the number of underlying features, and weights each model's output proportionate to its MSE (a lower MSE will generate a higher weight).

Typically, the base models predictions are used as features for a stacked model. Since `NNS` can fit any function, this technique would not be effectual for weighting models, resulting in an always equal weight of all features. Therefore, individual model criteria are used for prediction weightings.

[5] Observations are weighted by $\frac{1}{distance^2}$ where *distance* is measured from regression points, *not other observations.*

3 Iris

We begin with the ubiquitous Iris example. Fisher's Iris database (Fisher (1936)) is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

> "The use of this data set in cluster analysis however is not common, since the data set only contains two clusters with rather obvious separation. One of the clusters contains Iris 'setosa', while the other cluster contains both Iris 'virginica' and Iris 'versicolor' and is not separable without the species information Fisher used."
> `https://en.wikipedia.org/wiki/Iris_flower_data_set`

First we generate a random sample of Iris data as our testing hold out set, with the remaining observations as our training set. We test 3 different amounts of data to be used in our training set, 50% of the observations, 75% and 90%.

Here, we test each of the point estimates using `"iris.iv.train"` as our independent variables, and `"iris.dv.train"` as our dependent variable.

We round our predicted values to match the integer format of the Iris classes. When transforming the dependent variable into a numeric class value,

```
as.numeric(iris[,5])
```

R creates the following values for each Iris class:

1 - setosa
2 - versicolor
3 - virginica

The attached code creates the training and test sets, performs the `NNS` classification method and stores each method's output for a corresponding training set into an output matrix. We also combine the `NNS` techniques for an overall classification.[6] The remaining examples all follow this general procedure.

```
> require(devtools); install_github('OVVO-Financial/NNS',
>                                   ref = "NNS-Beta-Version")
> require(NNS)
> require(clue);require(class);require(e1071);require(OneR)
> new.iris=data.matrix(iris)
> errors=matrix(NA,nrow = 6,ncol=3)
> ### Create training and test sets
> test.sets=c(.5,.25,.1)
> l=length(iris[,1])
> for(test in 1:length(test.sets)){
 set.seed(123*test);test.set=sample(1:l,as.integer(test.sets[test]*l),
```

[6] The default weighting of the `NNS` techniques is based on the MSE of each model's output.

```
 replace = FALSE)


iris.iv.train=new.iris[c(-test.set),1:4]
iris.iv.test=new.iris[c(test.set),1:4]
iris.nb.iv.train=iris[c(-test.set),1:4]
iris.nb.iv.test=iris[c(test.set),1:4]

iris.dv.train=new.iris[c(-test.set),5]
iris.dv.test=new.iris[c(test.set),5]
iris.nb.dv.train=iris[c(-test.set),5]
iris.nb.dv.test=iris[c(test.set),5]

### Multivariate NNS Regression
nns=NNS.reg(iris.iv.train, iris.dv.train,order='max',
                          point.est = iris.iv.test,
                          plot=FALSE,type="CLASS")


errors[1,test]=sum(pmin(1,abs(as.numeric(iris.dv.test) -
                          round(nns$Point.est))))

### NNS Regression using dimension reduction technique

nns.class = NNS.reg(iris.iv.train,as.numeric(iris.dv.train),
                         dim.red = TRUE, point.est=iris.iv.test,
                         order=NULL,plot=FALSE)

errors[2,test] = sum(pmin(1,abs(as.numeric(iris.dv.test) -
                                  round(nns.class$Point.est))))


### K-means
o<- kmeans(cbind(iris.iv.train),3)

errors[3,test] = sum(pmin(1,abs(as.numeric(iris.dv.test)-
                  as.numeric(cl_predict(o, iris.iv.test)))))

### KNN
errors[4,test] = sum(pmin(1,abs(as.numeric(iris.dv.test)-
              as.numeric(knn(train = iris.iv.train,
                             test = iris.iv.test,
                             cl = iris.dv.train, k=1)))))

### NNS Stack
errors[5,test] =  sum(pmin(1,abs(as.numeric(iris.dv.test)-
```

```
                  round(NNS.stack(iris.iv.train,iris.dv.train,
                            iris.iv.test)$stack))))

 ###Naive Bayes
 nbmodel<- naiveBayes(iris.nb.iv.train, iris.nb.dv.train)

 errors[6,test] = length(iris.nb.dv.test)-
                   sum(predict(nbmodel, iris.nb.iv.test)==iris.nb.dv.test)


 }
> rownames(errors) = c("Multivariate Regression",
                       "Dimension Reduction Regression",
                       "k-means",
                       "KNN",
                       "NNS Stacked","Naive Bayes")
> colnames(errors) = c("50 % Test Set", "25 % Test Set", "10 % Test Set")
```

Figures 1 and 2 illustrate `NNS Multivariate Regression` and `NNS Dimension Reduction` fits respectively.

3.1 Iris Results

Calling our error output matrix, as expected, the larger the training set the more accurate our predictions. Table 1 is the raw number of mis-classifications.

```
> errors
```

	50 % Test Set	25 % Test Set	10 % Test Set
Multivariate Regression	4	2	0
Dimension Reduction Regression	8	4	1
k-means	9	24	14
KNN	4	2	0
NNS Stacked	2	3	0
Naive Bayes	3	2	1

```
> NNS.reg(iris[,1:4],iris[,5],order=NULL,
          location = 'topleft')
```

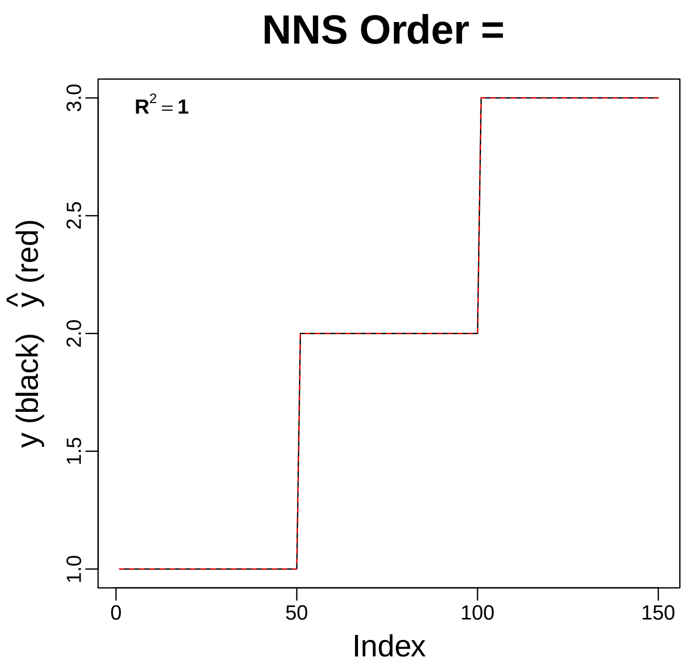

Figure 1: `NNS Multivariate Regression` perfect fit of Iris dataset.

```
> NNS.reg(iris[,1:4],iris[,5], dim.red=TRUE,
             location = 'topleft',order=NULL)
```

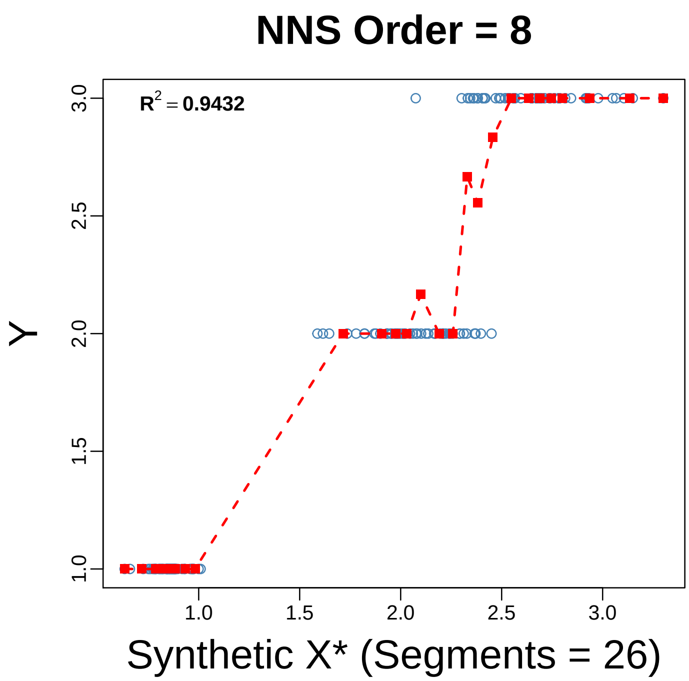

Figure 2: `NNS Dimension Reduction` fit of Iris dataset.

4 Wine Classification Example with 13 Attributes

Below is the code and output for the UCI wine example, expanding our regressors from 4 to 13 while still trying to classify 3 distinct classes.

```
> wine <- read.csv(url(
 "http://mlr.cs.umass.edu/ml/machine-learning-databases/wine/wine.data"))
> new.wine<- data.matrix(wine)
> wine.errors=matrix(NA,nrow = 6,ncol=3)
> ### Create training and test sets
> test.sets=c(.5,.25,.1)
> l=length(wine[,1])
> for(test in 1:length(test.sets)){
 set.seed(123*test);test.set=sample(1:l,
         as.integer(test.sets[test]*l),replace =FALSE)
 wine.iv.train=new.wine[c(-test.set),2:14]
 wine.iv.test=new.wine[c(test.set),2:14]
 wine.dv.train=new.wine[c(-test.set),1]
 wine.dv.test=new.wine[c(test.set),1]

 wine.nb.iv.train=wine[c(-test.set),2:14]
 wine.nb.iv.test=wine[c(test.set),2:14]
 wine.nb.dv.train=wine[c(-test.set),1]
 wine.nb.dv.test=wine[c(test.set),1]


 ### Multivariate NNS Regression
 nns.wine=NNS.reg(wine.iv.train,wine.dv.train,order='max',
                          point.est = wine.iv.test,
                          plot=FALSE,type="CLASS")$Point.est

 wine.errors[1,test]=sum(pmin(1,abs(as.numeric(wine.dv.test) -
                          round(nns.wine))))

 ### NNS Regression using dimension reduction technique

 wine.pred = NNS.reg(wine.iv.train,as.numeric(wine.dv.train),
                         dim.red=TRUE, point.est=wine.iv.test,
                         plot=FALSE)$Point.est

 wine.errors[2,test] = sum(pmin(1,abs(as.numeric(wine.dv.test)-
                          round(wine.pred))))


 ### K-means
 o<- kmeans(cbind(wine.iv.train),3)
```

```
wine.errors[3,test] = sum(pmin(1,abs(as.numeric(wine.dv.test)-
                                  as.numeric(cl_predict(o, wine.iv.test)))))

### KNN
wine.errors[4,test] = sum(pmin(1,abs(as.numeric(wine.dv.test)-
                      as.numeric(knn(train = wine.iv.train,
                      test = wine.iv.test,cl = wine.dv.train, k=1)))))

### NNS Stack
wine.errors[5,test]= sum(pmin(1,abs(as.numeric(wine.dv.test)-
                       round(NNS.stack(wine.iv.train,wine.dv.train,
                      wine.iv.test)$stack))))

###Naive Bayes
nbmodel<- naiveBayes(wine.nb.iv.train, wine.nb.dv.train)

wine.errors[6,test] = length(wine.nb.dv.test)-
                  sum(predict(nbmodel, wine.nb.iv.test)==wine.nb.dv.test)


}
> rownames(wine.errors) = c("Multivariate Regression",
                        "Dimension Reduction Regression",
                        "k-means",
                        "KNN",
                        "NNS Stacked","Naive Bayes")
> colnames(wine.errors) = c("50 % Test Set", "25 % Test Set", "10 % Test Set")
```

```
> NNS.reg(wine[,2:14],wine[,1],order=NULL,
           location = 'topleft',type="CLASS")
```

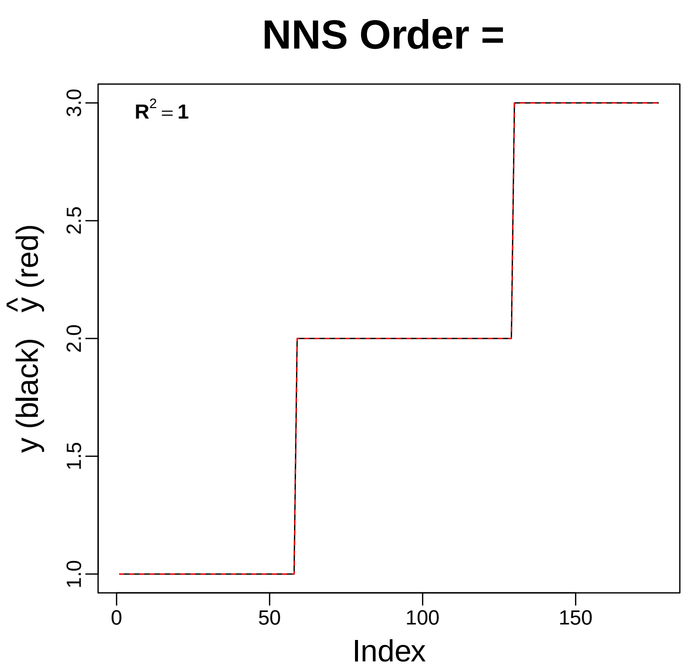

Figure 3: `NNS Multivariate Regression` perfect fit of Wine dataset.

```
> NNS.reg(wine[,2:14],wine[,1],order=NULL,
       location = 'topleft',dim.red=TRUE)
```

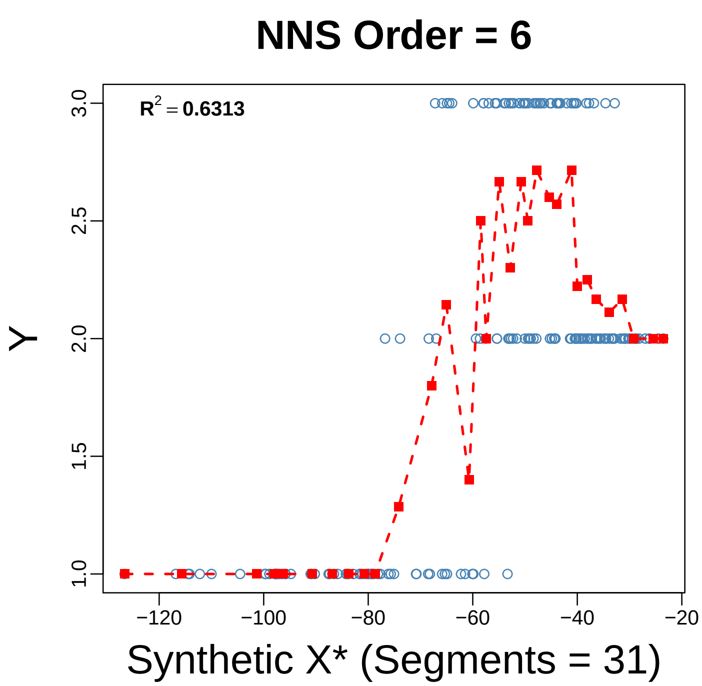

Figure 4: `NNS Dimension Reduction` fit of Wine dataset.

4.1 Wine Results

Calling our error output matrix, as expected, the larger the training set the more accurate our predictions. Table 2 is the raw number of mis-classifications.

```
> wine.errors
```

	50 % Test Set	25 % Test Set	10 % Test Set
Multivariate Regression	27	11	4
Dimension Reduction Regression	32	16	4
k-means	87	38	9
KNN	27	11	4
NNS Stacked	15	15	5
Naive Bayes	88	44	17

5 XOR Problem

The most classic example of linearly inseparable pattern is a logical exclusive-OR (XOR) function. We present an example used from: `https://pmirla.github.io/2016/09/10/Neural-Network-XOR-Problem.html`.

```
> x.or <- read.csv("https://goo.gl/4XOatp", sep = '\t',header =FALSE)
> x.or.errors = matrix(NA,nrow=5,ncol=3)
> test.sets=c(.5,.25,.1)
> l=length(x.or[,1])
> orig.x.or=x.or
> x.or=data.matrix(x.or)
> for(test in 1:length(test.sets)){
  set.seed(123*test);test.set=sample(1:l,
       as.integer(test.sets[test]*l),replace =FALSE)
  x.or.iv.train<- x.or[c(-test.set),1:2]
  x.or.iv.test<- x.or[c(test.set),1:2]
  x.or.y.train <- x.or[c(-test.set),3]; x.or.y.test <- x.or[c(test.set),3]
  A<- x.or.iv.train
  B<- x.or.iv.test


 nns.x.or=NNS.reg(A, x.or.y.train,order='max',
                      point.est = B, plot = FALSE,
                      type="CLASS")$Point.est

 x.or.errors[1,test] = sum(abs(x.or.y.test -
                      round(nns.x.or)))
```

```
 x.or.pred = NNS.reg(A, x.or.y.train, point.est= B,
                      plot = FALSE, dim.red=TRUE)$Point.est

 x.or.errors[2,test] = sum(abs(x.or.y.test - round(x.or.pred)))



 o<- kmeans(A,2)
 x.or.errors[3,test] = sum(pmin(1,abs(x.or.y.test-
                                  as.numeric(cl_predict(o, B)))))
 x.or.errors[4,test] =  sum(pmin(1,abs(as.numeric(x.or.y.test+1)-
                      as.numeric(knn(train = A, test = B,
                            cl = x.or.y.train+1, k=1)))))
 x.or.errors[5,test] = sum(pmin(1,abs(as.numeric(x.or.y.test)-
                      round(NNS.stack(A,x.or.y.train,
                      B)$stack))))

 }
> rownames(x.or.errors) = c("Multivariate Regression",
                      "Dimension Reduction Regression",
                      "k-means","KNN","NNS Stacked")
> colnames(x.or.errors) = c("50 % Test Set", "25 % Test Set", "10 % Test Set")
```

5.1 XOR Results

The XOR problem is well suited for the `NNS Multivariate Regression` technique, while the `NNS Dimension Reduction` did not fare as well. KNN also has difficulty with this dataset and required a (+1) to the classes to avoid zeros. Since there were only 2 features, a dimension reduction doesn't seem warranted. Furthermore, the dependent variable is likely better described with more features (and their probabilities) thus `NNS Dimension Reduction` should be reserved for problems with larger numbers of regressors. Note, the default MSE weighting in `NNS.stack` properly factors these considerations.

```
> x.or.errors
```

	50 % Test Set	25 % Test Set	10 % Test Set
Multivariate Regression	11	4	2
Dimension Reduction Regression	149	36	32
k-means	292	155	55
KNN	11	4	2
NNS Stacked	68	21	1

```
> NNS.reg(A,x.or.y.train,order=1,point.est = B,plot = TRUE)
```

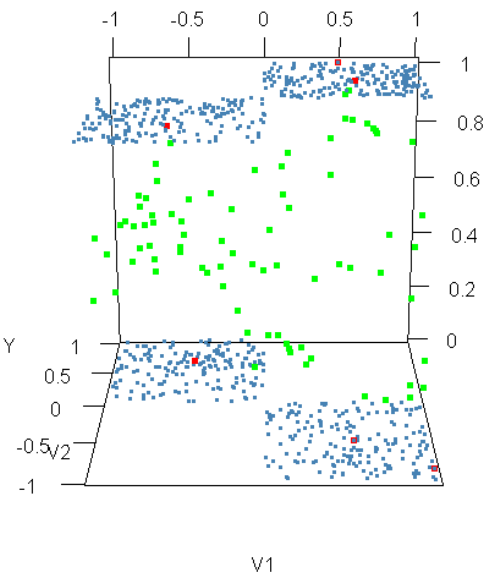

Figure 5: 3d representation of XOR problem. Data in blue, `NNS Multivariate Regression` predictions in green. `NNS` regression points as red squares. While a hyperplane may not be able to separate the data in 2 dimensions, it is abundantly clear a hyperplane separation exists in 3 dimensions along the vertical Y-axis with $Y = 0.5$ as the level of separation.

6 Cassini

The Cassini dataset is provided within the `"clue"` package in R, and they provide an example of k-means prediction using this dataset. Given the random starting assignment for k-means, we test these predictions over multiple seeds (100 used) and average the outcomes:

```
> data("Cassini")
> nr <- NROW(Cassini$x)
> tables=list();knntables=list()
> NNS.table=list()
> for(i in 1:100){
   set.seed(123*i)
 ind <- sample(nr, 0.9 * nr, replace = FALSE)
 party <- kmeans(Cassini$x[ind, ], 3,iter.max = 500)
 tables[[i]]=table(cl_predict(party, Cassini$x[-ind, ]),
       Cassini$classes[-ind])

 NNS.table[[i]]=table(round(NNS.reg(Cassini$x[ind,],
         as.numeric(Cassini$classes[ind]),
         point.est=Cassini$x[-ind,],
         plot=F,order='max')$Point.est),
         as.numeric(Cassini$classes[-ind]))


 knntables[[i]]=table(round(as.numeric(knn(train = Cassini$x[ind,],
              test = Cassini$x[-ind,], cl = Cassini$classes[ind],
              k=1))),
              as.numeric(Cassini$classes[-ind]))

 }
```

6.1 Simulation Results

The following tables show correct predictions along the main diagonal. `NNS Multivariate Regression` handily outpredicts k-means on this relatively simple dataset and produces the same output as KNN with $k = 1$.

k-means:

	1	2	3
1	13.90	12.47	6.99
2	11.62	14.53	6.65
3	14.57	12.19	7.08

NNS:

	1	2	3
1	40.09	0.00	0.00
2	0.00	39.19	0.00
3	0.00	0.00	20.72

KNN:

	1	2	3
1	40.09	0.00	0.00
2	0.00	39.19	0.00
3	0.00	0.00	20.72

The subtle differences in objectives between `NNS` and k-means, and the cluster number flexibility `NNS` enjoys over k-means translates to superior classification capabilities, on par with KNN.

```
> NNS.reg(Cassini$x[ind,],as.numeric(Cassini$classes[ind]),
    point.est = Cassini$x[-ind,],order='max')
```

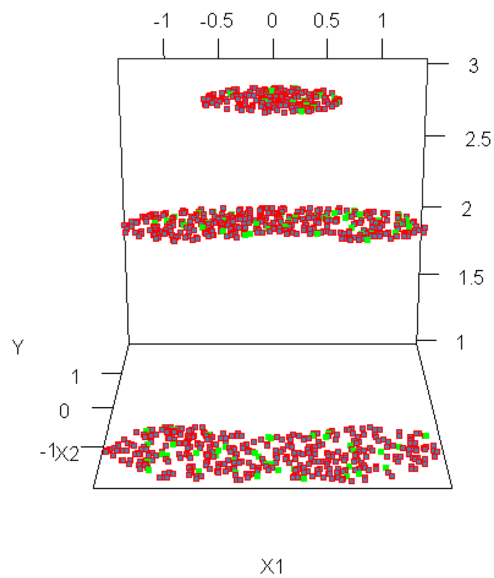

Figure 6: 3d representation of Cassini problem. Data in blue, `NNS Multivariate Regression` predictions in green. `NNS` regression points as red squares.

7 Text Classification

Text classification involves labelling data, generating a vocabulary, and creating a document term matrix (DTM).[7] It is the final step, creating the DTM that needs to be altered in order for `NNS Multivariate Regression` to work. Typically the DTM represents the alphabetical order of the vocabulary indexed, proceeded by a colon and the frequency it appears. We need to transform the DTM into a matrix whereby vocabulary are regressors with their frequencies as the associated elements.

The data is read directly from GitHub and is presented below:

```
> data <- read.csv('https://goo.gl/xuBhl9', sep = '\t', header = TRUE)
> data
```

```
                 Text IsSunny
1               sunny       1
2               rainy      -1
3         sunny sunny       1
4         sunny rainy      -1
5         rainy sunny      -1
6         rainy rainy      -1
7   sunny sunny sunny       1
8   sunny rainy sunny       1
9   sunny sunny rainy       1
10  rainy sunny sunny       1
11  rainy rainy sunny      -1
```

The out-of-sample text to predict is given by a list (or a data.frame):

```
> predictionData <- list("sunny sunny sunny rainy rainy",
                         "rainy sunny rainy rainy",
                         "hello",
                         "",
                         "this is another rainy world",
                         "sunny day we have here",
                         "sunny isn't it",
                         "sunny sunny rainy rainy")
```

The expected classifications (1 for sentences talking more about sunny weather, -1 for talking about rainy weather) for the out of sample text are: $(1, -1, 0, 0, -1, 1, 1, 0)$

Next we transform the in- and out-of-sample text into an `NNS term matrix`, creating the independent variables and dependent variables for classification via `NNS Multivariate Regression`.

```
> text=NNS.term.matrix(data,predictionData)
```

[7] Please see `http://www.svm-tutorial.com/2014/11/svm-classify-text-r/` and the links for a more thorough explanation.

7.1 Text Classification Results

We run the `NNS` classification techniques and compare to the expected classifications:

```
> Expected=c(1,-1,0,0,-1,1,1,0)
> NNS=NNS.reg(text$IV,text$DV, plot=FALSE,order=NULL,
       point.est = text$OOS,type="CLASS",n.best=2)$Point.est
> NNS.Dimension.Reduction=NNS.reg(text$IV,text$DV,
       plot=FALSE,point.est = text$OOS,dim.red=TRUE)$Point.est
> KNN = knn(train = text$IV,
               test = text$OOS, cl = text$DV, k=1)
> NNS.Stack=NNS.stack(text$IV,text$DV,
                       text$OOS,CV.size = .2)$stack
> cbind(Expected=Expected,NNS.Multivariate.Regression=NNS,
         NNS.Dimension.Reduction=NNS.Dimension.Reduction,
         KNN=KNN,NNS.Stack=NNS.Stack)
```

```
     Expected NNS.Multivariate.Regression NNS.Dimension.Reduction KNN NNS.Stack
[1,]        1                  1.00000000               0.5283019   2         1
[2,]       -1                 -1.00000000              -0.7538462   1        -1
[3,]        0                  0.09167309              -0.3846154   2         1
[4,]        0                  0.09167309              -0.3846154   1         1
[5,]       -1                 -1.00000000              -0.6923077   1        -1
[6,]        1                  1.00000000               0.1509434   2         1
[7,]        1                  1.00000000               0.1509434   2         1
[8,]        0                  0.20000000               0.1037736   2         1
```

Like the previous examples, the `NNS Multivariate Regression` technique is aptly suited for the task. It should be obvious, but for clarity, values less than $|0.5|$ round to 0 yielding a perfect classification for the baseline NNS technique. `NNS Dimension Reduction` is not suggested for use in text classification due to the sparse document term matrix.

8 Comments

`NNS` performs two steps, clustering the data (unsupervised learning) and then classifying the data (supervised classification). We demonstrate the NNS equivalence to KNN when both techniques are set to use the nearest observation. There are no odd number requirements or noticeable time deterioration from increasing the `NNS` `"n.best"` parameter (analogous k in KNN).

The methods and results presented immediately raise suspicions on the pervasive notion of dimension reduction given the consistent performance of the `NNS Multivariate Regression`. However, there is considerable additional research required within each technique and problem applicability in order to substantiate and generalize these observations.

This was not an exhaustive study, simulation, or description of techniques / examples. Our modest goal was to present base examples of how `NNS` can be used in similar types of problems, and specifically ***how cluster analysis can indeed be used, and integrated effectively, for classification purposes*** thanks to `NNS'` substitution of the k-means objective initial parameter as described in Vinod and Viole [1]. k-means fixed number of clusters equal to the number of DV classifications is too rigid of an assumption for effective out of sample classification.[8]

The other goal of this demonstration was to show the equivalence of KNN and `NNS` classification under this limit condition of maximum clustering. `NNS` enjoys a flexibility over KNN where `NNS` can automatically weight any number of selected nearest clusters and offer traditional multivariate regression analysis seemlessly.

The reader is strongly encouraged to visit all of the references and links provided, as well as experiment with other datasets using the combined efforts of `NNS` clustering and classification.

Time to run sweave:

```
  user  system elapsed 
398.60    7.85  431.56 
```

[8]The following discussion presents an excellent visualization to the underlying k-means assumptions. `http://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means`

References

[1] H D Vinod and F Viole. Clustering and curve fitting by line segments. *SSRN eLibrary*, 2016.

[2] F Viole and D Nawrocki. Deriving Nonlinear Correlation Coefficients from Partial Moments. *SSRN eLibrary*, 2012.