Paper SD-09

KNN Classification and Regression using SAS®

Liang Xie, The Travelers Companies, Inc.

ABSTRACT

K-Nearest Neighbor (KNN) classification and regression are two widely used analytic methods in predictive modeling and data mining fields. They provide a way to model highly nonlinear decision boundaries, and to fulfill many other analytical tasks such as missing value imputation, local smoothing, etc.

In this paper, we discuss ways in SAS® to conduct KNN classification and KNN Regression. Specifically, `PROC DISCRIM` is used to build multi-class KNN classification and `PROC KRIGE2D` is used for KNN regression tasks. Technical details such as tuning parameter selection, etc are discussed. We also discuss tips and tricks in using these two procedures for KNN classification and regression.

Examples are presented to demonstrate full process flow in applying KNN classification and regression in real world business projects.

INTRODUCTION

kNN stands for k Nearest Neighbor. In data mining and predictive modeling, it refers to a memory-based (or instance-based) algorithm for classification and regression problems. It is a widely used algorithm with many successfully applications in medical research, business applications, etc. In fact, according to Google Analytics, it is the second most viewed article on my SAS programming blog of all time, with more than 2200 views a year.

In classification problems, the label of potential objects is determined by the labels of closest training data points in the feature space. The determination process is either through ”majority voting” or ”averaging”. In ”majority voting”, the label of object is assigned to be the label which most frequent among the k closest training examples. In ”averaging”, the object is not assigned a label, but instead, the ratio of each class among the k closest training data points.

In Regression problems, the property of object is obtained via a similar ”averaging” process, where the value of the object is the average value of the k closest training points.

In practice, both the ”majority voting” and ”averaging” process can be refined by adding weights to the k closest training points, where the weights are proportional to the distance between object and the training point. In this way, the closest points will have the biggest influence on the final results. In fact, the SAS implementation of kNN classification has the averaging process be weighted by volume size. Interested readers are encouraged to read the manual for details.

CHARACTERISTICS OF KNN-BASED ALGORITHMS

kNN algorithm has several characteristics that worth addressing. Understanding its advantages and disadvantages in theory and in practice will help practicioners better leverage this tool. Generally, KNN algorithm has 4 advantages worth noting.

1. It is simple to implement. Theoretically, kNN algorithm is very simple to implement. The naive version of the algorithm is easy to implement. For every data point in the test sample, directly computing the desired distances to all stored vectors, and choose those shortest k examples among stored vectors. It is, however, computationally intensive, especially when the size of the training set grows. Over the years, many nearest neighbor search algorithms have been proposed, seeking to reduce the number of distance evaluations actually performed. Using an appropriate nearest neighbor search algorithm makes k-NN computationally tractable even for large data sets. In SAS, the k-d Tree data structure and associated search algorithm of Friedman et. al. is implemented.
2. It is analytically tractable.
3. It is highly adaptive to local information. kNN algorithm uses the closest data points for estimation, therefore it is able to take full advantage of local information and form highly nonlinear, highly adaptive decision boundaries for each data point.

4. It is easily implemented in parallel. Because it is instance based, for each data point to be scored, the algorithm check against the training table for the k nearest neighbor. Since each data point is independent of the others, the execution of search and score can be conducted in parallel.

On the other hand, kNN algorithm has 3 disadvantages, which will also be addressed in details in section *PRACTICAL ISSUES ON IMPLEMENTATION*.

1. It is computationally very intensive, because for each records to be classified and/or scored, the program has to scan through available data points to find the nearest neighbors and then determine which ones to use for further calculation.
2. It is demanding in storage, because all records in training sample have to be stored and used whenever scoring is necessary.
3. It is highly susceptible to curse of dimensionality, because as dimension increases, the distance calculation and finding the appropriate space to contain at least one relevant record is becoming difficult.

KNN USING SAS

In order to conduct either KNN Classification or KNN regression in SAS, there are two basic approaches. The first one utilize special options in certain SAS Procedures and conduct the KNN analysis directly. I call it "Direct Approach"; while the second method will first find the nearest data points explicitly leveraging search capability in some SAS procedures and manually calculate the final response values, which is simple anyway. I call it "Manual Approach".

To take the manual approach, there are three PROCs that are capable to find nearest neighbor points in efficient way, namely `PROC FASTCLUS`, `PROC KRIGE2D` and `PROC MODECLUS`. Each has its pro and con and all need data preparation. The discussion of this issue will beyond the scope of this paper.

To take the direct approach, `PROC DISCRIM` can be directly used for KNN Classification while `PROC KRIGE2D` can be directly used for KNN regression, even though the functionality SAS provides is only basic. For kNN-based classification, `PROC DISCRIM` offers intuitive programming interface and rich options. For kNN-based regression, even though there is not dedicated procedure available, `PROC KRIGE2D` can be used to fulfill this task, with some tweaks. In this section, we will demonstrate how to conduct these two data mining tasks in SAS and address some closely related issues. For kNN-based regression, we will discuss the details of how to tweak `PROC KRIGE2D`.

KNN CLASSIFICATION USING SAS

SAS offers kNN classification capability via `PROC DISCRIM`, by invoking the following procedure options:

```
METHOD=NPAR  K=
```

The 'METHOD=NPAR' option asks SAS to use non-parametric discrimination function, together with 'K=' option `PROC DISCRIM` will use kNN classification, where 'K=' tells SAS how many neighbors to use in determining the . A typical SAS program for kNN-based classification looks like the follows:

```
proc discrim data=train          .........................[1]
             method=npar  k=5    .........................[2]
             testdata=toscore    .........................[3]
             testout=toscore_out.........................[4]
             ;
     class y;                    .........................[5]
     var   x1-x10;               .........................[6]
run;
```

We explain the function of each statement below.

1. Statement [1] tells SAS that `PROC DISCRIM` should be called to process the data named *train*.
2. Statement [2] tells SAS to apply kNN Classification method using 5 nearest neighbors.
3. Statement [3] tells SAS that the classification rule be applied to a test data called 'toscore'.
4. Statement [4] tells SAS to output classification result for the test data and to name the output data as *toscore-out*.
5. Statement [5] defines Y as the dependent variable to use in `PROC DISCRIM`.
6. Statement [6] tells SAS to use $x1$ to $x10$ as input features in calculating the distance to neighbors.

kNN classification is computationally intensive. Based on SAS manual [4], for the specific tree search algorithm implemented in `PROC DISCRIM`, the time usage, excluding I/O time, is roughly proportional to $\log(N) * (N * P)$, where N is the number of observations and P is the number of variables used.

kNN is a memory-based method, when an analyst wants to score the test data or new data in production, the whole raw table will be loaded to memory in the scoring process. In computing, `PROC DISCRIM` uses memory approximately proportional to the second order of number of variables, i.e. proportional to P^2.

Therefore, it would be interesting to see how the cost of computing, in terms of memory usage and time consumption, increases as the number of observations in training data and the number of features are increasing.

KNN REGRESSION USING SAS

kNN technique can be applied to regression problems, too, but the coding in SAS is not as straightforward as in a classification problem. kNN regression uses the average value of dependent variable over the selected nearest neighbors to generate predicted value for scoring data point. In SAS, we can use `PROC KRIGE2D` to conduct this analysis, but it needs more cautions and tweaks.

The following example code instructs SAS to do a kNN regression on variable Y using 5 nearest neighbors and 2 features: X1 and X2.

```
proc krige2d data=train outest=pred  outn=NNlist; ..............[1]
     coordinate  xc=X1  yc=X2;                    ..............[2]
     predict     var=Y  NP=5;                     ..............[3]
     model       scale=10  range=1E-7  form=exp;  ..............[4]
     grid        griddata=toscore xc=X1  yc=X2;   ..............[5]
run;
```

1. Statement [1] invokes `KRIGE2D` procedure and ask SAS to operate on data named *train*, and output the prediction to data named *pred* via option OUTEST=. The option OUTN= ask SAS to output information about the selected nearest neighbors of each data points in the data specified in GRID statement. KRIGE2D will directly output the prediction of dependent variable in the data from 'OUTEST= ', but only output the corresponding value of dependent variable in the selected closest data points as well as the coordinates. This is a disadvantage in the sense that when cross validation is conducted, the data from 'OUTN=' does not contain the actual value of dependent variable and this data set has to be merged back to the validation part to obtain actual values so that errors can be computed.
2. Statement [2] tells `KRIGE2D` that the names of the 2 coordinates to be used, or in the language of data mining, the 2 features to be used. In this example, it is X1 and X2.
3. Statement [3] tells `KRIGE2D` to use 5 nearest neighbors to predicts dependent variables's value at current grid position. As to be discussed, the number of nearest neighbors is a tuning parameter in this algorithm, and should be determined using data driven approaches, such as Cross Validation.
4. Statement [4] tells `KRIGE2D` what model to be used in kriging using 'FORM=' option and the associated parameters, scale and range. Note that, in order to conduct a kNN regression, it is required to specify a large scale value and a small range value. For example, in example above, scale is set to 10 while range is set to

1E-7. Typically, a scale value larger than 1 and a range value smaller than 1E-7 work well. The type of model is irrelevant. The reason why this works is beyond the scope of this paper and we will not discuss on this issue here.

5. Statement [5] tells `KRIGE2D` that the name of the data to be scored is *tosore* in option `GRIDDATA=` and the corresponding names of features (coordinates) to be used for scoring.

`PROC KRIGE2D` was designed to perform ordinary kriging in two dimensions, therefore it is not fully functional in the standard kNN regression practice. There are two major issues with `PROC KRIGE2D` when it is applied in a kNN regression analysis. The first problem is that `PROC KRIGE2D` accept two and only two inputs as features because it treat kNN regression as a two dimensional kriging. The second problem is due to the nature of semivariogram model used in kriging. In cases where the data are dense and highly overlapped, `PROC KRIGE2D` will report singularity in kriging system. However, these issues should not bother analysis in practice and there are good reasons which will be detailed below.

Overcome Fixed Number of Features in KRIGE2D In a real kNN regression application, the number of features ranges from 1 to hundreds if necessary, but `PROC KRIGE2D` requires exactly two features as inputs. When only 1 feature is used, or when more than 2 features are required, the analysts can take a detour by pre-processing the data up front. We will discuss these two cases separately below.

1. Only one feature. When only one feature is used, we can set up a dummy input, which is a constant number, in both the training data and scoring data. The dummy variable will cancel out in calculating, thus it has no effect virtually. The code below shows a demonstration:

```
data trainv/view=trainv;
     set train;
     dummy=1;
run;
data toscorev/view=toscorev;
     set toscore;
     dummy=1;
run;
proc krige2d data=trainv  outest=pred outn=knn;
     coordinate  xc=dummy  yc=feature1;
     predict     var=Y     NP=5;
     model       scale=10 range=1E-7  form=exp;
     grid        gdata=toscorev  xc=dummy  yc=feature1;
run;
```

   In the example code above, we used SAS data view to avoid re-write the data sets and save computing resources. This technique is explained in [10].

2. More than two features. When more than two features are used, there are two approaches to handle this problem. The first approach conduct dimension reduction upfront and `PROC KRIGE2D` take two derived new dimensions as features to perform kNN regression; the second approach follows the ensemble principle [5], and randomly select two features out of the pool of available features. The final kNN regression prediction is an average of predictions from each of the ensemble regressions. The first approach can also use the ensemble principle, too, in practice. There are many dimension reduction techniques, and among all, the most widely used is Singular Value Decomposition (SVD), or its equivalent counterpart Principal Components Analysis (PCA). Details of SVD or PCA is beyond the scope of this paper and interested readers can refer to [7], [4]. The example code in Appendix 1 demonstrates the first method where we conduct dimension reduction using PCA upfront and use the leading two orthogonal bases for kNN regression.

   Note that in the example code, we first combine both train and score data sets together because PCA is sample sensitive [7], and using records from both training data and scoring data will stabilize the generated orthogonal bases.

The second method is demonstrated below. It is worth noting that when the number of features is small, say 10, it is a good practice to select all pairwise combination as input features and ensemble all results at the final stage. On the other hand, when the number of such combinations blows as the number of features increases, a random sample of pairwise combination works fine, just like in a RandomForest algorithm. In the code below, we assume 100 features which are stored in a table called FEATURES and the ensemble iterates 50 times, each with a pair of randomly selected features.

```
%macro knnreg(train_dsn, score_dsn,
              feature1 , feature2 ,
              loopid);
   proc krige2d  data=&train_dsn
                 outest=pred&loopid
                 outn=nn&loopid;
         coord   xc=&feature1  yc=&feature2;
         pred    var=Y   NP=10;
         model   scale=2  range=1E-8 form=exp;
         grid    gdata=&score_dsn  xc=&feature1  yc=&feature2;
   run;
%mend;
%let nloops=50;
proc surveyselect data=features   out=feature2
                  reps=&nloops    sampsize=2   method=srs;
run;
data _null_;
     set feature2; by replicate;
     retain train_dsn 'train'   score_dsn 'toscore';
     retain _id 0;
     array _f{2} $ _temporary_;
     if first.replicate then do;
        call missing(of _f[*]);
         _id=1;
     end;
     _f[_id]=varname;
     _id+1;

     if last.replicate then do;
        call execute('%knnreg('
                     || compress(train_dsn) || ','
                     || compress(score_dsn) || ','
                     || compress(_f[1])     || ','
                     || compress(_f[2])     || ')'
                   );
      end;
run;
```

The 'call execute' statement calls the pre-defined macro %kddreg within a data step. For details about 'call execute' command, consult [9].

Singularity in kriging process When the scoring point is extremely close to at least one of the training record, local ordinary kriging process will be singular and the estimation for the scoring point will be skipped. One solution is to set a large number for scale and small number for range in MODEL statement, for example:

```
MODEL SCALE=10 RANGE=1e-8 FORM=EXP;
```

In the extreme cases, while inconvenient, it is necessary to manually process the data containing nearest neighboring points from `OUTN=` to generate prediction. It turns out to be fairly simple, because ordinary kNN regres-

sion only requires average every k observations. This technique is demonstrated in Appendix 2. In such case, it is unnecessary to output predictions from KRIGE2D directly, and the output can be suppressed by specifying `OUTEST=_NULL_`.

PRACTICAL ISSUES ON IMPLEMENTATION

kNN is a proven technique in many classification and regression problems. To fully appreciate its power in real analysis, there are a couple of issues need to be addressed appropriately. These problems touch the pain points of efficiency, accuracy, robustness and more. In the sections below, we will first discuss efficiency around storage and computation issues and see how parallelization can improve the efficiency. Then we will discuss model selection that balances accuracy and robustness. Lastly, we will discuss other miscellaneous issues when applying KNN in practice.

OPTIMIZE STORAGE AND COMPUTATION

kNN algorithm is expensive in both storage and computation. Therefore, it is important to the factors that contribute to complexity in time and space of this algorithm, thus optimize the implementation in SAS.

Optimize Storage kNN classification requires a lot of storage because this is a in-memory algorithm, and all the training information to score a new data will be load into the memory to build a kd-Tree when conduct scoring. Therefore, when the size of the training data is large, and when the number of variables used is large, the required memory storage will be non-trivial. According to SAS/STAT Manual, the required storage for a kNN classification is at least $c(32v+3l+128)+8v^2+104v+4l$ bytes, where v is the number of variables, c is the number of classes levels of the dependent variable, and l is the length of $CLASS$ variable. Apparently, the storage requirement is linear in the latter two factors which are given by the problem at hand, while quadratic in the number of variables. To optimize storage, it is necessary to reduce the number of variables involved in the analysis.

1. Only choose the necessary variables in kNN analysis. In many cases, 2 or 3 features is enough, actually. On the other hand, like in RandomForest algorithm, multiple runs of kNN algorithm were conducted, each with a small subset of variables, and these classifiers or regression results will be ensembeled to be the final solution.
2. When selected number of variables is still relatively large, one commonly used technique for dimension reduction is Singular Value Decomposition (SVD), or equivalently, the Principal Component Analysis (PCA). Because kNN algorithm is highly adaptive to local information, SVD or PCA won't suffer from common critics that the information in dependent variable is not used when variable reduction is used. This technique is demonstrated in the Digits Recognition example below.

kNN algorithm used in classification and regression suffers a lot from curse of dimensionality, therefore, choosing only necessary and more useful features for the problem in hand is critical for the success of application and implementation.

Optimize Computation kNN classification is also computationally intensive. Unlike many classification methods, such as logistic regression, discriminant analysis and other parametric regressions, kNN has no condense mathematical form to represent the information in training data, and lack of explicit model training. When scoring is needed, the training data is simply used to populate the sample search space with known results, and scoring is no different from a searching and comparison process, which is very time consuming. In the most naive way of searching, the algorithm compares scoring current data point to all data in the training sample, calculate desired distance metrics and select the top k nearest neighbors for their response information. This is computationally very redundant and intensive and doesn't leverage the distribution information in training sample well. Modern algorithm relies on heuristic searching algorithms and one of the most significant one is kd-Tree [2]. kd-Tree stands for k-dimensional tree, and it is a space-partitioning data structure to organize data points in k-dimensional space for efficient search. In SAS, `PROC DISCRIM` stores data in kd-Tree structure. The time required to organize the observations into this specific tree structure is proportional to $vn\ln(n)$. The time for performing each tree search is proportional to $\ln(n)$. Apparently, the way to reduce computational time is reduce the number of data points n. On the other hand, it is advised to keep enough data points in training sample. This is because a k-d trees are not suitable for efficiently finding the nearest neighbor in high dimensional spaces. As a general rule, if the dimensionality is k, the number of

points in the data, N, should be $N >> 2k$. Otherwise, when k-d trees are used with high-dimensional data, most of the points in the tree will be evaluated and the efficiency is no better than exhaustive search, and approximate nearest-neighbor methods are used instead [6].

SELECT THE OPTIMAL VALUES FOR TUNING PARAMETERS

kNN algorithms have two basic tuning parameters, the number of nearest neighbors k, and the dimension of feature space d. The typical way to find the right values for these two parameters is brutal force computation, where program searches through a grid of different combinations of these two parameters and decide the best set based on cross validation results. The k in kNN refers to the number of closest data points, which is an important tuning parameter in kNN algorithm. There is no set rule for choosing the right k value, but usually when the data is dense, k is set to be small while when data is sparse, a relatively higher k should be used to reduce the variance. In many cases, the dimension is pre-determined. However, kNN is sensitive to curse of dimensionality and more often than not, the features are derived values, such as Principle Component Scores. The analysts have to find which subset of features is enough for the problem at hand. According to Friedman, et al, an objective data driven approach to choose the right value for k is the way to go. Typically, a Cross-Validation (CV) based approach is used, usually 5-fold CV or 10-fold CV. When using an M-fold CV to choose the tuning parameter k and d, it is customized to check the mean value and standard deviation of certain model accuracy measurement. For example, for Classification problem, both error rate and AUC can be used; for regression problems, RMSE is a good candidate measurement. The code in the two examples below illustrate these techniques.

PARALLELIZATION

kNN classification and regression can be easily parallelized. The most significant applications are in Cross-validation based tuning parameter evaluation and scoring.

In Cross-Validation process, the analyst is able to open M concurrent sessions, each overs mutually exclusive set of tuning parameters. For example, the simplest case is issuing 2 concurrent SAS sessions, one examines $k = 1, \cdots, 5$ while the other one examines $k = 6, \cdots, 11$. The actual implementation follows the idea of [1].

In scoring, because each data point is assumed to be independent from the others, the scoring program can split the scoring data set into several pieces, depending on the number of available CPUs and the master control module will simultaneously call the scoring module which only operates on each piece at a time.

KNN IN ACTION

In this section, we demonstrate kNN algorithms in real applications. Both examples will go through steps such as data introduction, data analysis and modeling using kNN algorithm, as well as selecting the tuning parameter k. The first example examines handwritten digits recognition, a 10-label classification problem. The second example examines a regression problem where the researcher wanted to estimate the phenolic records.

KNN CLASSIFICATION IN OPTICAL DIGITS

One successful application of kNN classifiers is in digits recognition projects, see [8] for examples. In this example, I will demonstrate a complete data analysis process using kNN classifier to identity hand written digits from Optical Digits images. The data is from the pre-processed Optical Digits images available at UC Irving Machine Learning data set repository and can be downloaded at :
`http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`.
The data is extracted using the NIST preprocessing routines [3] from normalized bitmaps of handwritten digits from a preprinted form. The original bitmaps are 32X32 but divided into non-overlapping blocks of 4X4 and the the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality to 64 instead of 1024, and gives invariance to small distortions. In this project, the following steps are performed to fine tune the kNN classifier.

1. First, the training and testing data in CSV format are imported at *Step 1*.
2. Then a baseline 1-NN classifier using all 64 features is built for comparison purpose at *Step 2*. This baseline classifier return an error rate of 3.73%.
3. In *Step 3*, a fine tuned kNN classifier is built, where the tuning parameter k, number of closest points, and d, number features, are selected based on 10-fold Cross Validation. Besides, instead of using the original 64

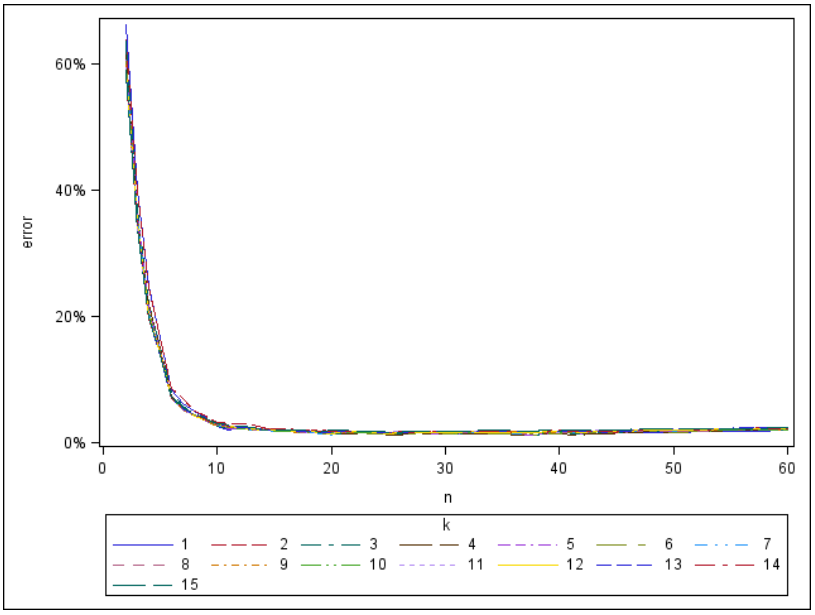

Figure 1: Classification Error Rate as a function of K and D

features, a Singular Value Decomposition (SVD) was conducted on the training and testing data sets combined to obtain derived features. This data pre-processing technique brings in several advantages. First, the derived features are ranked by how much variance in the feature space each explained, so that the selection of features becomes simpler using CV. Second, SVD changes original sparse feature space into a dense one, and in general fewer closest data points are required to obtain optimal performance in this case.

4. In *Step 4*, the selected kNN classifier, with $k = 4$ closest data points and $d = 36$ derived features, is applied to the testing data set, obtaining an error rate of 1.89%.

5. As an alternate to 10-fold CV, in *Step 5*, Leave-One-Out CV was used to fine tune the kNN classifier, resulting a set of parameters of $k = 6$ and $d = 4$.

6. , In *Step 6*, the new classifier was applied to the testing data set, resulting in an error rate of 1.4%.

The 10-fold Cross Validation process is pretty computationally intensive even for such a small data set. Scanning across 900 different combinations of k, d pairs took almost 25 minutes on a main stream PC. The pay off worth it, however, as the error rate drops to 1.89% from 3.73%, a 49% improvement. One the other hand, using Leave-One-Out Cross Validation process took 22 minutes on the same PC, and the selected parameter pair is $k = 6, d = 38$, with an error rate of 2.06% from the same scoring data set, which is not as good as the result from 10-fold CV but still way better than baseline result. The 10-fold CV is preferred to Leave-One-Out cross validation from pure computational perspective, as the computing time for the latter one increases quadratically and will become infeasible when the number of observation is more than 100K.

Figure 1 shows the error rate as a function of k and d, while figure 2 zooms in for a closer look at area where number of features is between 20 and 40 and closest data points is smaller than 5. We can see that globally, the error rate is a relatively smooth function of number of features over all and the number of neighbors does not influence the error rate as much as the number of features does, while locally there is significant variation as shown in figure 2. This implies that one heuristic way to scan for the best possible numbers of features is bi-section searching.

The complete SAS code that conducts brute force searching for tuniing parameters can be found in Appendix 1.

KNN REGRESSION EXAMPLE

As a demonstration of kNN regression, we used the *Galaxy* data set from the book *Elements of Statistical Learning 2*, which is available at [11]. The galaxy data was used to demonstrate smoothing techniques in the book and was visualized in figure 6.8, where local linear regression with nearest neighbor window of 15% of the data in R^2

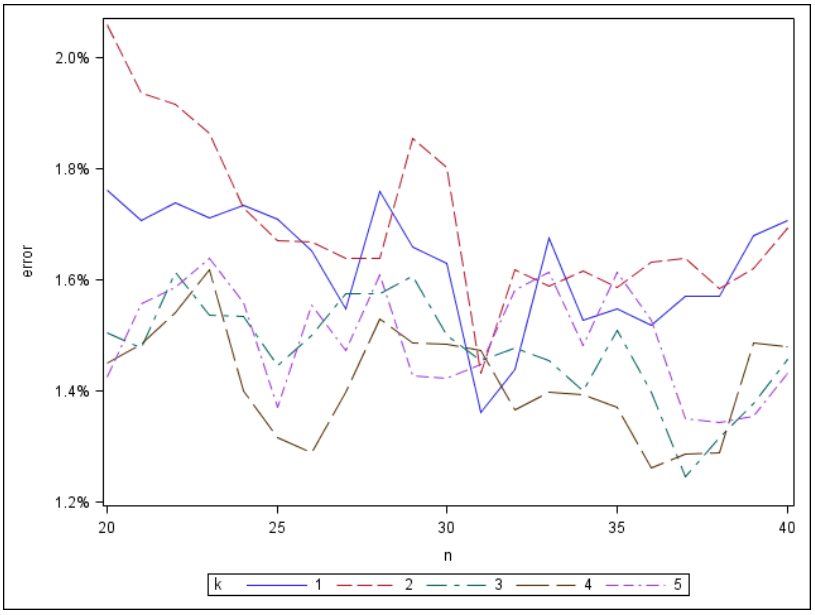

Figure 2: Closer Look at Classification Error Rate as a function of K and D

was applied to predict velocity data of galaxies. In our example, a k-Nearest Neighbor prediction is applied, where two values of k are used: $k = 5$ and $k = 48$. $k = 48$ is equivalent to using 15% of the data as neighbor while $k = 5$ is arbitrarily choosen to demonstrate the fact that smaller number of neighbors will produce more accurate but higher variance prediction while larger value of k will product less accurate but smaller variance hence smoother prediction. Combine the two predictions in many cases may product better results among all. In figure 3, 3-D views of actual data and smoothed surface using different smoothing parameters are shown. The upper left panel shows the raw data. Obviously, the galaxy has a star like spatial shape with a total of about 10 arms extending outside from the center. The upper right panel shows smoothed 3-D surface using 5-nearest neighbors while the lower left panel shows smoothed 3-D surface using 48-nearest neighbors. It is noticeable that using less neighbors produce a rougher surface but adaptive better to local where there are more data, such as along the galaxy arms, but in more empty spaces, the prediction is wilder. On the other hand, using 48 neighbors produces much smoother surface yet the galaxy arms are less visible. The lower right panel shows the average of these two surface maps. This averaged out map keeps the galaxy arms clearly visible yet smooth out empty space pretty well. The complete SAS code is in Appendix 2.

CONCLUSION

kNN algorithm is an old but very powerful technique if used appropriately. It has been shown that both kNN Classification and kNN regression can be carried out within SAS/STAT very easily. The functions are basic but more than enough to handle many real world problems. Both the theory and practice are discussed, with example SAS code demonstrated.

The discussion here, however, is only the tip of iceberg. There are many advanced topics that can't be fitted into this short article. These topics include but not limited to: customized distance function, tangent invariance, etc. Interested readers are encouraged to read the related books and papers.

REFERENCES

[1] Steven First. A generic method of parallel processing in base sas ®8 and 9. In *Proceedings of the SAS Global Forum*, 2007.

[2] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

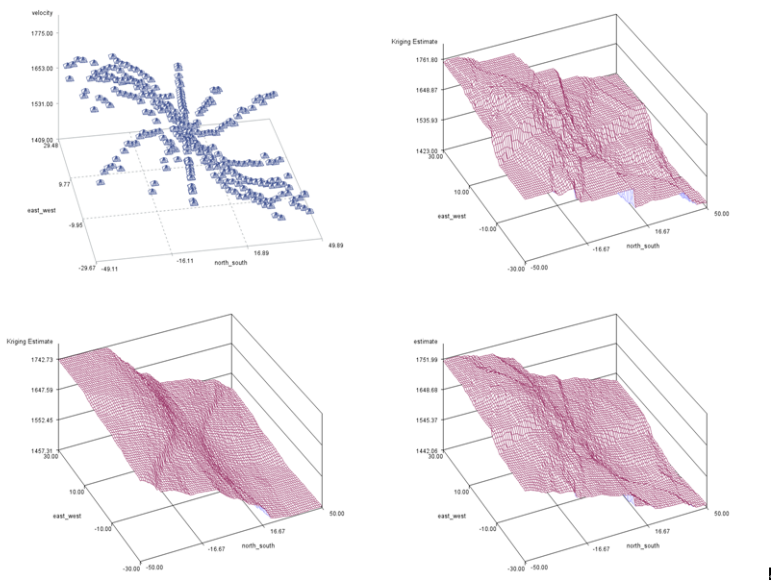

Figure 3: Galaxy Data shown in raw and smoothed surfaces using different smoothing parameters

[3] Michael D. Garris, James L. Blue, Gerald T. Candela, Gerald T. C, Darrin L. Dimmick, Jon Geist, Patrick J. Grother, Stanley A. Janet, and Charles L. Wilson. Nist form-based handprint recognition system. Technical report, Technical Report NISTIR 5469 and CD-ROM, National Institute of Standards and Technology, 1994.

[4] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[5] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2003.

[6] SAS Institute Inc. *SAS/STAT 9.2 Users Guide*. Cary, NC, 2008.

[7] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, October 2002.

[8] Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[9] Denis Michel. Call execute: A powerful data management tool. In *Proceedings of the SAS User Group International*, volume 30, 2005.

[10] Robert Ray. Save time today using sas®views. In *Proceedings of the SAS User Group International*, volume 27, 2002.

[11] Robert Tibshirani Trevor Hastie and Jerome Friedman. Elements of statistical learning website, 2010.

ACKNOWLEDGMENTS

This paper is written using LaTeXwith the SUGCONF class contributed by Ronald J. Fehd. The author would like to thank Jason Grove and James Landgrebe for their support. The views and opinions expressed in this article are those of the author and do not necessarily reflect the official policy, position or views of The Travelers Companies, Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author(s): Name Liang Xie
E-mail: mailto:xie1978@gmail.com
Web: sas-programming.blogspot.com

APPENDIX 1: SAS CODE FOR OPTICAL DIGITS CLASSIFICATION

```
/*@@@@ --- Step 1 --- @@@@*/
libname digits "C:\_Data\Digits";

data digits.train;
     infile "C:\_Data\Digits\optdigits.tra"  lrecl=512  dlm=',';
     input  x1-x64 y;
run;

data digits.test;
     infile "C:\_Data\Digits\optdigits.tes"  lrecl=512  dlm=',';
   input  x1-x64 y;
run;

/*@@@@ --- Step 2 --- @@@@*/
proc discrim data=digits.train   testdata=digits.test
       testout=testout
       method=npar  k=1 noprint;
 class y;
 var   x1-x64;
run;

proc freq data=testout;
     table y*_INTO_;
run;

/*@@@@ --- Step 3 --- @@@@*/
data all;
     set digits.train(in=_1)     digits.test(in=_2);
     if _1 then Group='T';
     else Group='S';
run;

proc princomp data=all  out=pca noprint noint cov;
     var x1-x64;
run;
proc standard data=pca  out=pcaout  mean=0  std=1;
     var Prin1-Prin62;
run;

/*@@@@@ 10-fold CV approach @@@@@@@@@@*/
data pcaout;
     set pcaout;
   array _xp{10} p1-p10 (10*0.1);
   cv=rantbl(2342342, of _xp[*]);
run;
proc freq data=pcaout;    table cv;  run;
```

```
%macro loopy;
ods select none;
options nonotes nosource;
%do k=1 %to 15;
   %do n=2 %to 60;
   %put k=&k   n=&n;
   %do cv=1 %to 10;
       proc discrim data=pcaout(where=(group='T' & cv^=&cv))
                    testdata=pcaout(where=(group='T' & cv=&cv))
              testout=testout
              method=npar  k=&k noprint;
          class y;
          var   Prin1 - Prin&n;
       run;
       proc freq data=testout;
            table y*_INTO_ /out=result;
       run;
       proc sql;
            create table _tmp&cv as
          select sum(COUNT) as total,
                 sum((y=_INTO_)*COUNT) as correct,
             &k as k, &n as n,
             1-calculated correct / calculated total as error format percent7.3
          from   result
     ;
       quit;
  %end;
data _tmp;
    set %do cv=1 %to 10;   _tmp&cv      %end;;
run;
proc means data=_tmp noprint;
     var  error;
       output  out=_tmp  mean(error)=error  std(error)=error_std;
run;
data _tmp;
     length k n 8;
       set _tmp;
       k=&k; n=&n;  drop _TYPE_;
run;
  %if %eval(&k+&n=3) %then %do;
      data iter; set _tmp; run;
  %end;
  %else %do;
      proc append base=iter data=_tmp  force; run;
   %end;

%end;
%end;
ods select all; options notes source;
%mend;

dm log 'clear';
%let time0=%sysfunc(time());
%loopy;
%let time1=%sysfunc(time());
```

```
data _null_;
     dt=(&time1-&time0)/60;  put dt= 'Min';
run;

data iter;
     set iter;  format error percent7.4;
run;

proc transpose data=iter out=itert name=n prefix=n;
     by k;
     var error;
     id n;
run;

data itert;
     set itert;
     array _n{*} n2-n60;
     min=min(of _n[*]);  max=max(of _n[*]);  format min max percent7.4;
run;

/*@@@@ --- Step 4 --- @@@@*/
/*@@@@ use selected parameter to score the testing data @@@@*/
proc discrim data=pcaout(where=(group='T'))   testdata=pcaout(where=(group='S'))
 testout=testout(drop=x: Prin:)
 method=npar  k=4 noprint;
 class y;
 var  Prin1-Prin36;
run;
proc freq data=testout;
     table y*_INTO_ /out=result;
run;
proc sql;
     create table temp as
   select sum(COUNT) as total,
          sum((y=_INTO_)*COUNT) as correct,
      4 as k, 36 as n,
      1-calculated correct / calculated total as error format percent7.3
   from   result
   ;
quit;


/*@@@@ --- Step 5 --- @@@@*/
/*@@@@@ Use leave-one-out cross-validation approach @@@@@@*/
%macro loop;
%do k=1 %to 15;
%do i=1 %to 62;
proc discrim data=pcaout(where=(group='T'))
 method=npar  k=&k   noprint  crossvalidate
 outcross=outcross   out=out;
 class   y;
 var     Prin1-Prin&i;
run;

proc freq data=outcross noprint;
```

```
     table y*_INTO_ /out=table;
run;

proc sql;
     create table _tmp as
     select &k as K, &i as I, sum(PERCENT) as correct
 from   table
 where  y=_INTO_
 ;
quit;
%if %eval(&i+&k)=2 %then %do;
    data result; set _tmp; run;
%end;
%else %do;
    proc append base=result data=_tmp; run;
%end;
%end;
%end;
%mend;

options nonotes nosource;
%let time0=%sysfunc(time());
dm log 'clear';
%loop;
options notes source;
%let time1=%sysfunc(time());
data _null_;
     dt=(&time1-&time0)/60;
 put dt= 'Min';
run;

proc transpose data=result out=result_t name=i prefix=n;
     by k;
     var correct;
 id i;
run;

data result_t;
     set result_t;
   array _n{*} n2-n62;
   min=min(of _n[*]);    max=max(of _n[*]);
   loc_min=whichN(min, of _n[*]);  loc_max=whichN(max, of _n[*]);
   format min max best.;
run;


proc discrim data=pcaout(where=(group='T'))
             testdata=pcaout(where=(group='S'))
 testout=testout(drop=x: Prin:)
 method=npar  k=6 noprint;
 class y;
 var  Prin1-Prin38;
run;
proc freq data=testout;
     table y*_INTO_ /out=result;
run;
```

```
proc sql;
     create table temp as
 select sum(COUNT) as total,
        sum((y=_INTO_)*COUNT) as correct,
6 as k, 38 as n,
1-calculated correct / calculated total as error format percent7.3
 from   result
 ;
quit;
```

APPENDIX 2: SAS CODE FOR REGRESSION EXAMPLE

```
proc import datafile="c:\galaxy.csv" dbms=csv  out=galaxy replace; run;
run;

proc g3d data=galaxy;
   scatter east_west*north_south=velocity /rotate=25  size=1 noneedle  tilt=45;
   format _numeric_ 8.2;
run;
quit;

%macro knnreg(k);
ods select none;
proc krige2D data=galaxy  outest=knn_pred&k  outn=neighbor;
     coordinates xc=east_west  yc=north_south;
 predict     var=velocity  NP=&k;
 model       scale=10      range=1E-8 form=exp;
 grid        gdata=a(drop=velocity)   xc=east_west   yc=north_south ;
run;
ods select all;

proc g3d data=knn_pred;
      plot gxc*gyc=estimate /  grid  rotate=25 tilt=45
             ctop=cx993366
             cbottom=cx9999ff
             caxis=black;
 label   gxc="east_west"
         gyc="north_south";
 format _numeric_ 8.2;
run;

%mend;
%knnreg(5);
%knnreg(48);

data knn_pred;
     merge knn_pred5(rename=(estimate=estimate5))
       knn_pred48(keep=Estimate  rename=(estimate=estimate48));
 estimate=(estimate5+estimate48)*0.5;
run;

proc g3d data=knn_pred;
      plot gxc*gyc=estimate /  grid  rotate=25 tilt=45
             ctop=cx993366
             cbottom=cx9999ff
             caxis=black;
```

```
 label   gxc="east_west"
         gyc="north_south";
 format _numeric_ 8.2;
run;
```