Deploying K Nearest Neighbor Modeling Methodologies for Real World Problems

White Paper

AXTRIA
INGENIOUS INSIGHTS

Introduction

There has been an immense improvement of theory and algorithms in recent times leading to enhancement in computation power. Also, online data availability was never easier than now. As a result, machine learning techniques have become more popular in solving real world problems. There are a number of machine learning techniques available now for continuous as well as categorical target events. K Nearest Neighbor (KNN), Decision Trees, Random Forest, Support Vector Machine (SVM) are some of the popular techniques that have emerged in the recent past. In this paper, we will focus on elaborating the deployment of one such technique - KNN.

Before we go into the details of KNN, let us first understand the emergence of KNN as an alternative to traditional models. The traditional models (e.g. Linear Regression, Logistic Regression etc.) are based on some assumption about the structure/ distribution of the problem. Once the model is developed, the equation, input variables, parameter coefficients etc. are all fixed. The same equation is then applied to new data samples until redevelopment of the model. There is a significant cost involved with maintaining these models, regularly monitoring and addressing the issues pertaining to specific segments where the traditional model might not be working well.

On the other hand, KNN is a non-parametric method which does not assume anything about the underlying distribution. KNN models can be quickly refreshed/redeveloped using the most recent data and incorporating the most recent trends. As a result, KNN has become quite popular in recent times.

To understand the nuts and bolts of KNN, let us pick "recommender system" as the appropriate use case. Assume we have a set of items I to be recommended to a set of users J. Let p be a utility function which measures the usefulness of item i ($\in$ I) to user j ($\in$ J), i.e. p: I $\times$ J $\rightarrow$ R, where R is a totally ordered set (for example, non-negative integers or real numbers in a range). The purpose is to first learn the utility function p based on the past data and then use p to predict the utility value of each item i ($\in$I) to each user j ($\in$J). Typically, there are two approaches adopted to learn and predict the utility value of each item:

- Content-based recommendations - The users are recommended items similar to the ones the user preferred in the past. This is also termed as user based method. "Recommend items that are similar to those the user liked in the past".
- Collaborative filtering (or collaborative recommendations) - The user is recommended items that people with similar tastes and preferences liked in the past. This is also termed as item based method. "Recommend items that similar users liked".

KNN is an example of the hybrid approach which deploys both user-based and item-based methods to make the predictions. With this backdrop, let's now go into details of KNN, to understand how it uses user-based and item-based methods for predictions.

Methodology

The basic methodology of KNN is to find k most similar labeled points (closest neighbors) among available sample points in a cell of volume V and assign the most frequent class among those neighbors to our query(or unlabeled) point x.

The generic expression for density estimation is $p(x) \approx (k/n)/V$. In this expression, k represents the number of points inside the cell (space) with volume V. In KNN, k is determined by the user; and then it determines the appropriate volume V of the cell which contains k points inside. In this kind of estimation, two possibilities can occur, as shown below:

- Density is high near a point x; therefore, the cell having volume V will be small (but adequate) which provides a good resolution.
- Density is low near a point x; in such a situation, the cell will grow larger in volume V until higher density is obtained.

Once the cell with volume V has been determined, the class selected for the unlabeled point x is the one that has the majority in a number of points in the cell. In mathematical terms, the conditional probability of a class being assigned to the unlabeled point x is nothing but $p(c_i|x) \approx k_i/k$ where k_i points (among k points) belong to class c_i in the cell.

While the logic of KNN looks pretty straightforward, it has to be understood that the performance of nearest neighbor model depends on multiple parameters. What makes it trickier is that there is no hard and fast rule for parameter selection. One can only follow some guidelines (in addition to one's judgment) while choosing a parameter value. The table below highlights the parameters involved in KNN modeling.

Table 2.1: Parameters of KNN modeling

Parameter	Description
Features	The variables based on which similarity between two points is calculated
Distance function	Distance metric to be used for computing similarity between points
Neighborhood (k)	Number of neighbors to search for
Scoring function	The function which combines the labels of the neighbors to form a single score for the query point

Let us look at each of these parameters in more details.

Features

Features are pretty much dependent on the particular target we want to model using KNN. E.g. the items purchased by users could be one of the variables on which similarity is to be assessed between two points. Similarly, at a user level, the items purchased by the user in the past could be another variable for assessing similarity. This is how KNN is able to deploy hybrid approach of user level as well as item level similarity assessment. The important aspect of feature selection is the quality of underlying data. It is critical that prior to KNN model development, appropriate data processing steps have been undertaken, as shown below:

Missing Value Treatment

Variables with missing values should be treated before they are thrown into the model. If 'Missing' has a special significance, the variable can be retained and missing may be replaced with meaningful values, as illustrated in the table below. Inappropriate treatment of missing values would result in the sub-optimal selection of neighbors. Hence, even though this step sounds trivial, it is super critical in KNN modeling. In our experience, the lot of modelers tend to ignore this step in KNN modeling.

Table 2.2: List of commonly used missing value treatments

% Missing	Treatments
< 1 %	Delete Observations Mean Imputation
1 – 10 %	Mean Imputation
10 – 50 %	Regression Imputation
> 50 %	Drop Variable

Special Value Treatment (if applicable)

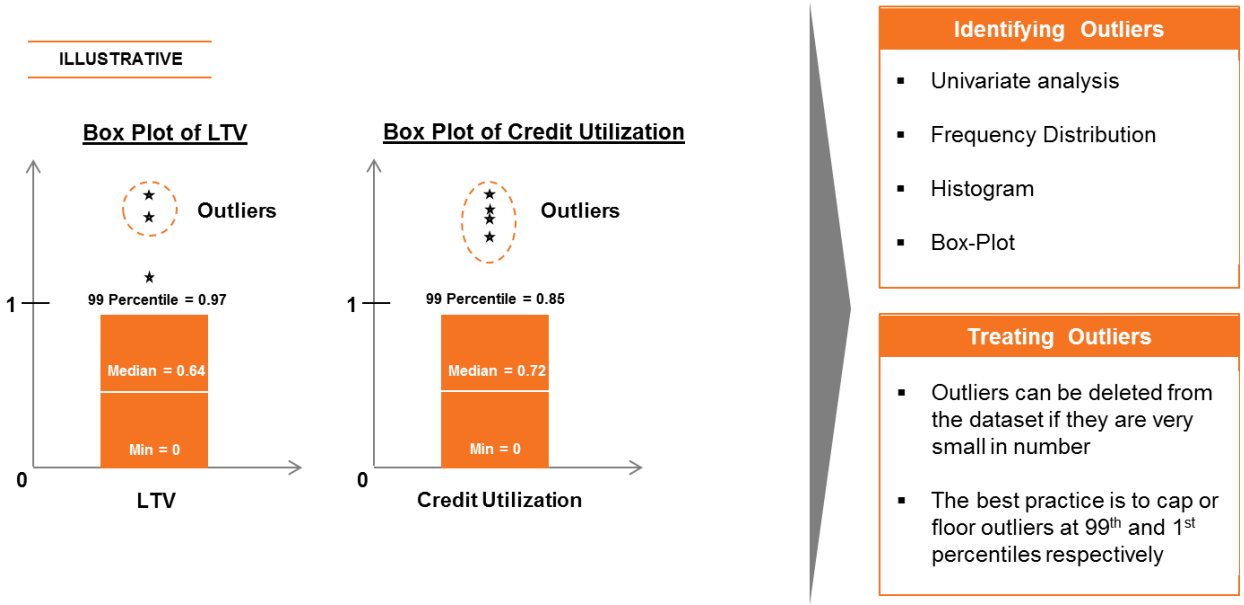
On the lines of our observations above, special value treatment also needs attention before KNN modeling is done to identify nearest neighbors. As an illustration, sometimes, negative values or large positive numbers may denote default values. These need to be treated (or removed) before throwing into the KNN scoring algorithms.

Table 2.3: List of commonly used special value treatments

Treatments
Special values may be combined with other values of the variable on the basis of target rates
A default value flag can be created and introduced as a variable

Outlier Treatment

An observation of a variable is said to be an outlier if it is significantly outside the distribution of the remaining observations. Outliers have to be treated just as carefully as the missing and special values are treated before modeling. The graphic below represents select ways of identifying and treating the outliers.



Standardization

It is recommended that each feature is then standardized to a mean of zero 0 and standard deviation of 1 .Thus for each feature vector with mean μ_i and standard deviation σ_i, the new feature ends up being $\frac{(x_i-\mu_i)}{\sigma_i}$

There are few other standardization/normalization techniques available too. These are as shown below:

- Min-max normalization: $\frac{(x_i-\min(x_i))}{\max(x_i)-\min(x_i)}$
- Percentile normalization: $\frac{(x_i-p1(x_i))}{p99(x_i)-p1(x_i)}$

Where p1 is the 1st percentile and p99 is the 99th percentile. In our experience, the standardization to a mean of zero and standard deviation of 1 is the most prevalent method.

Distance Function

Once features have been selected and appropriate variable treatment and transformation / standardization has been done, suitable distance function is applied to calculate the distance between two points. There are multiple options available for distance functions, as shown below:

Euclidian Distance: $d(x,y) = \sum_{i=1}^{n}(x_i - y_i)^2$

This is most commonly used distance function in KNN. However, it has some drawbacks. Euclidian distance treats each feature as equally important. It is however possible that some features are much more discriminative than others (dimensions). In case there are a lot of irrelevant features that do not discriminate adequately, it would result in Euclidean distance being dominated by noise – and thus could cause wrong identification of nearest neighbors. The same is shown mathematically below:

$$\sum_{i=1}^{n}(x_i - y_i)^2 = \sum_{i=1}^{n1}(x_i - y_i)^2 + \sum_{i=1}^{n2}(x_i - y_i)^2$$

Where n1 is the number of discriminative features; n2 is the number of noisy features and n1 +n2=n. If the number of discriminative features (n1) is smaller than the number of noisy features (n2), the Euclidean distance can be dominated by noise.

There are several other distance functions available in KNN that can be deployed too. They are as follows:

- Canberra Distance: $d(x,y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$
- Manhattan Distance: $d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$
- Chebysev Distance: $d(x,y) = \overset{Max}{i} \ |x_i - y_i|$
- Minkowski Distance $d(x,y) = \sqrt[p]{\sum_{i=1}^{n} (x_i - y_i)^p}$:
- Mahalanobis Distance: $d(x,y) = \sqrt{(x-y)^T S^{-1}}(x-y)^T$

All the distance functions mentioned above are for numerical variables only. In presence of categorical features, the following distance function should be deployed:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} {\delta_i}^2}$$

Where $\delta_i = \begin{cases} 1 \ if \ x_i = y_i \ , \ i \in \text{categorical} \\ 0 \ if \ x_i \neq y_i \ , \ i \in \text{categorical} \\ x_i - y_i \ if \ i \in \text{numerical} \end{cases}$

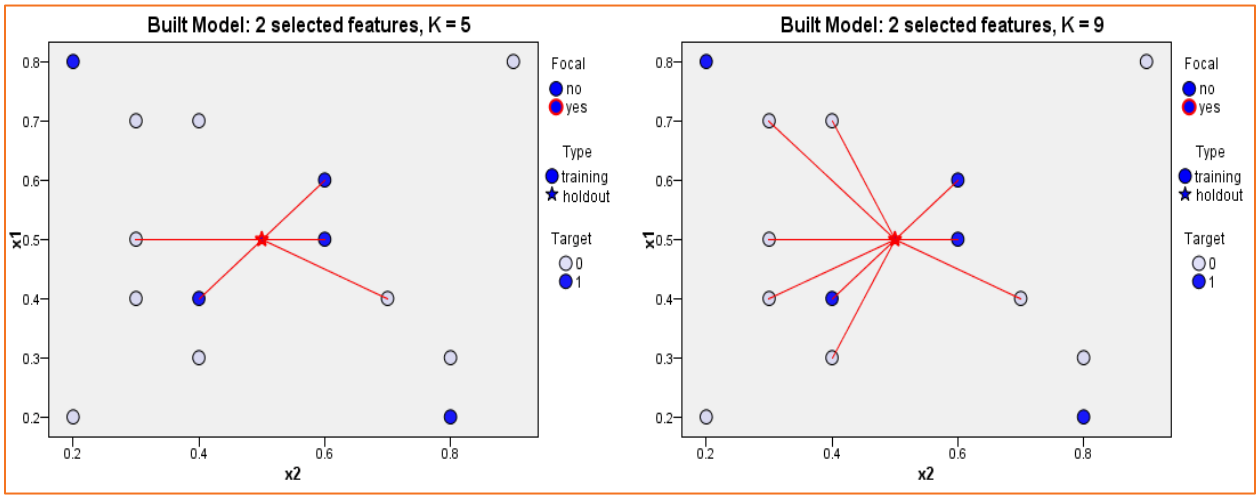
To address the issue of giving higher weightage to features that are more discriminating, the feature can be scaled /weighted for better classification. The weight w_i can learn from the validation data. The weights can be increased or decreased until classification improves. The weighted distance function would thus be formulated as follows:

- Weighted Euclidian Distance: $d(x,y) = \sum_{i=1}^{n} w_i (x_i - y_i)^2$
- Weighted Manhattan Distance: $d(x,y) = \sum_{i=1}^{n} w_i |x_i - y_i|$
- Weighted Chebysev Distance: $d(x,y) = \overset{Max}{i} \ w_i |x_i - y_i|$
- Weighted Canberra Distance: $d(x,y) = \sum_{i=1}^{n} \frac{w_i |x_i - y_i|}{|x_i| + |y_i|}$

Neighborhood (K)

The neighborhood parameter k indicates the number of neighbors in the neighborhood of the query point. Therefore, it plays a crucial role on the performance of nearest neighbor classifier by controlling the volume of neighborhood and the smoothness of the density estimates (refer **Figure 2.2**). The big question is: how to choose optimum value for k?

Figure 2.2: Influence of 'k' on model outcome



Source:https://www.ibm.com/support/knowledgecenter/SS3RA7_15.0.0/com.ibm.spss.modeler.help/images/out_knn_overview.gif

In theory, when the infinite number of samples is available, the larger the k, the better is the classification but the caveat is that all k neighbors need to be close to x .In practice ,k should be large enough so that error rate is minimized .Too small k will lead to noisy decision boundaries. On the other hand, k should be small enough so that only nearby samples are included. Too large k will lead to over-smoothed boundaries. Balancing 1 and 2 is not a simple ask.

In a particular application of KNN in a hybrid recommender system, it has been observed that the prediction accuracy (measured by Root Mean Squared Error) is pretty much dependent on the number of neighbors taken into account. Relatively small neighborhood size produces imprecise estimates. On the other hand, in a case of a comparatively large neighborhood, it might happen that too many users or items with very low similarities are taken into account leading to poor predictions. Also, the past experiments exhibit that the item based implementation produces better prediction results than the user based counterpart (See Hydra: A Hybrid Recommender System, Spiegel, Kunegis, Li)

In our experience, for a large sample, if we use Euclidean distance for classification, k should be varied with n such that $k \to \infty$ and $k/n \to 0$ as $n \to \infty$. A good "rule of thumb" is $k = \sqrt{n}$. Having said that, we believe there is no strict rule or theoretical guideline for choosing the optimum value of k for small or moderately large sample sizes. We have to estimate the optimum value using the available training data. Therefore, the optimum value of k can vary a lot from one data to another.

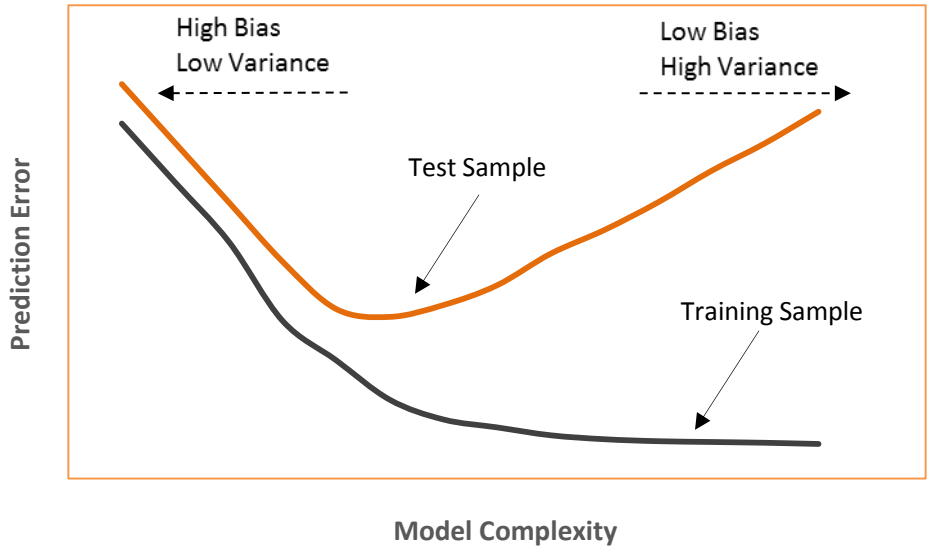
Significant research has been done in this area. Several techniques have been applied to find optimum value of k. Some of the techniques are as follows:

- Likelihood cross-validation (see Silverman, 1986): The optimum value of k is estimated by maximizing the log likelihood score.
- A slightly different version of this likelihood criterion has been used by Holmes and Adams (2002, 2003) for aggregating the results of different nearest neighbor classifiers.
- Cross-validation methods (see Lachenbruch and Mickey, 1968; Stone, 1977) to estimate the misclassification rate for different values of k and chooses that one which leads to the lowest estimate of misclassification rate.
- A smooth estimate (see Ghosh and Chaudhuri, 2004) for the misclassification probability function for finding the optimal bandwidth parameter
- Bayesian method (see Anil K. Ghosh, 2005) for selecting the optimum k

KNN algorithm can be subject to over-fit with the developmental data. When 'k' is low, the variance of the model with respect to the training data is high, resulting in very good model performance within the training data, while causing poor performance on other datasets (i.e. Over-fitting).

On the other hand, when 'k' is high, the model becomes biased, resulting in poor model performance and test data. This trade off requires careful tuning to find an optimal parameter 'k' for the KNN algorithm to produce a robust, well performing model.

Figure 2.3: Training and Test error as a function of complexity



Scoring Function

Once k nearest neighbors are identified, the behaviors of lookalikes are then used as inputs to a scoring function which predicts the behavior of the query point. The commonly used scoring function is as follows:

Standard: predicted label = $$\frac{\sum_{i=1}^{k} l_i(\frac{1}{d_i^2})}{\sum_{i=1}^{k}(\frac{1}{d_i^2})}$$

Where l_i is the label of ith neighbor; d_i is the distance to ith neighbor. As is evident from the mathematical formulation, the above scoring function imposes higher importance to the neighbors close to the query point.

In case, observation level weights are present, the predicted label = $$\frac{\sum_{i=1}^{k} w_i l_i(\frac{1}{d_i^2})}{\sum_{i=1}^{k} w_i(\frac{1}{d_i^2})}$$

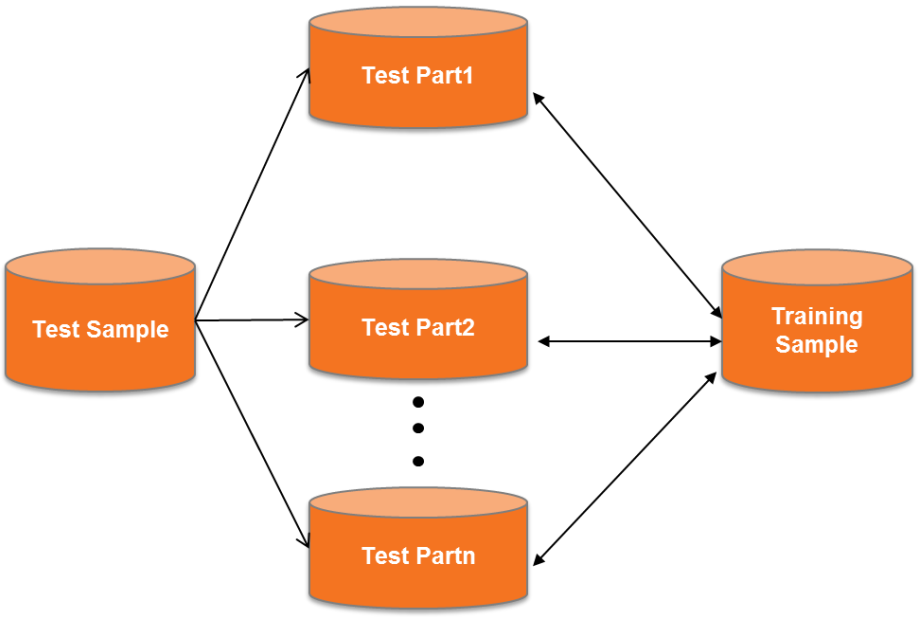
KNN Algorithm

An important element of deploying KNN algorithm is how the point of test dataset is mapped on the training dataset. The algorithm by which we map a point of test dataset to a point in training dataset and identify the neighbors to our query point is important, since the time-efficiency of the model depends on it. There are several ways to map test dataset to the training dataset. The two more prevalent approaches are described below:

Exhaustive Search

This approach is preferred when the training dataset is small. In this approach, the entire training dataset is used as is. The test dataset is split across various mappers on the Hadoop cluster doing exhaustive computation at each node. This is shown in the figure below.

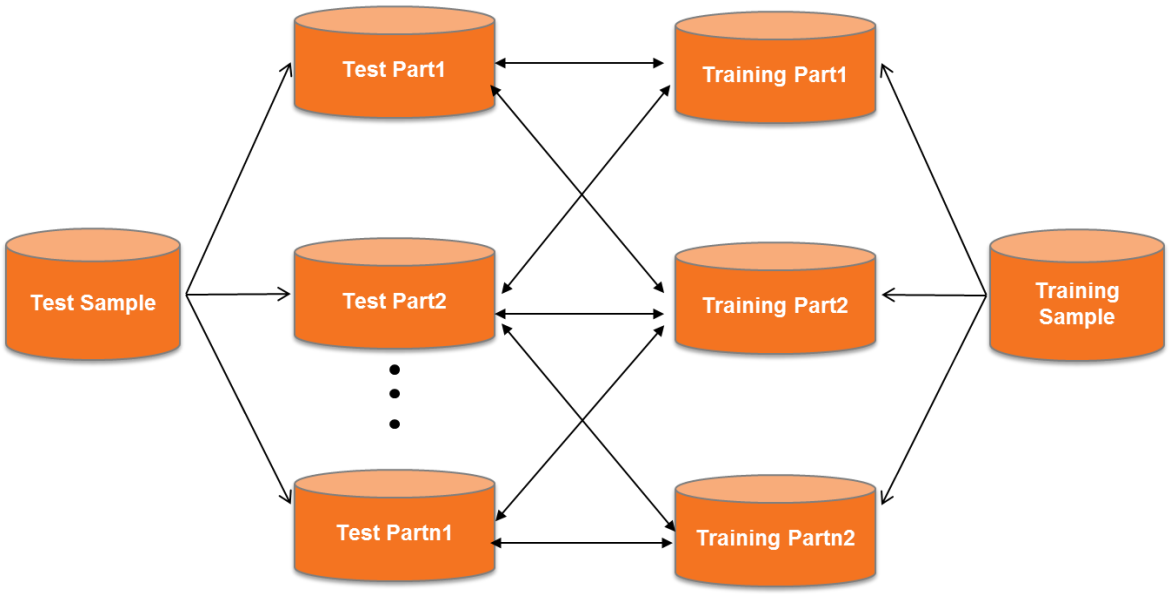
Figure 3.1: Mapping Approach -Exhaustive Search



Exhaustive Large Search

This approach is preferred when there is moderate or large training dataset. In this approach, both training and test datasets are split and distributed across the cluster. As an illustration, if the test data is split into $n1$ parts and training data is split into $n2$ parts; there are $n1.n2$ possible combinations. Hadoop works on each set separately and performs map-reduce to combine the results from all the sets pertaining to each observation. The same is highlighted graphically in the figure below.

Figure 3.2: Mapping Approach - Exhaustive Large Search



Voronoi Partitioning:

A more sophisticated method for mapping test data with training data is Voronoi Partitioning. This is based on a mathematical technique called Voronoi diagram. The diagram is created by taking pairs of points that are close together and drawing a line that is equidistant between them and perpendicular to the line connecting them. Therefore, all points on the lines in the diagram are equidistant to the nearest two (or more) source points. A Voronoi cell contains all neighboring points that are nearest to each sample.

Let there are *n* input samples $x_1, x_2, \dots, x_n$.Each input sample has *p* number of features (eg. Monthly Spend, Daily Balance etc.). $d(x_i, x_m)$ is the Euclidian distance between two samples x_i and x_m based on their p features. Then a Voronoi cell is defined as: $R_i = \{y \in R_p : d(y, x_i) \leq d(y, x_m), \forall\, i \neq m\}$

Where R_i is the Voronoi cell for sample x_i,

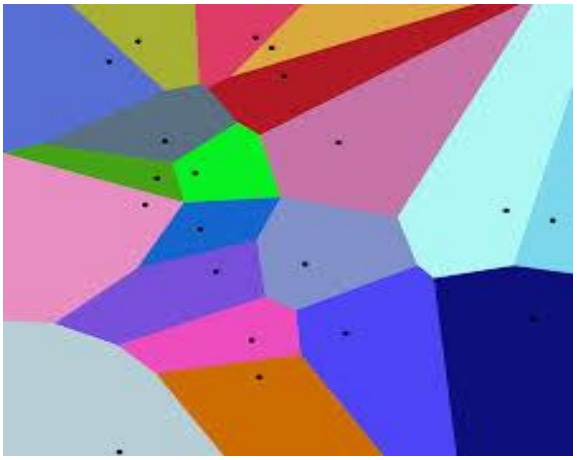
x_m is an input sample other than x_i and
y represents all possible points within Voronoi cell R_i.

The above reflects two things:

- All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample, and
- For any sample, the nearest sample is determined by the closest Voronoi cell edge. The idea is to partition space recursively and search for nearest neighbors only close to the test point.

As a result, finding the k-nearest neighbors is not required for the entire space if nearby partitions are known and this partition contains k points. In order to achieve this, the distances between partitions and number of points in each partition have to be constantly tracked. There are many variations present to improve this further e.g. Imposing a stopping criteria on k, region size, minimum and maximum distances within a partition. The technique is depicted graphically in the figure below.

Figure 3.3: Voronoi Partitioning

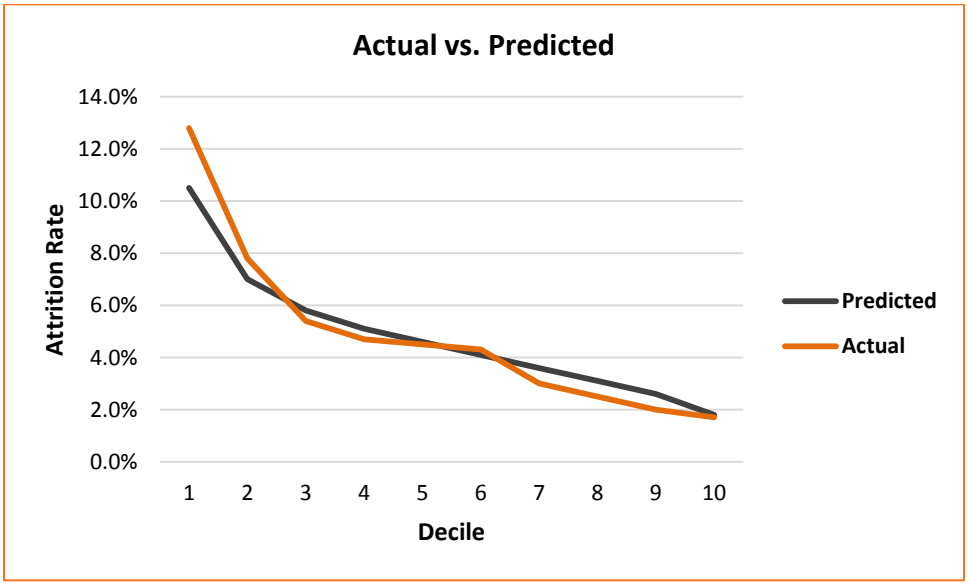


KNN Model Performance

KNN methodologies have been applied for both binary target events (e.g. attrition modeling, response modeling) as well as continuous target events (Monthly spend, Daily Balance etc.).

In our experience, if the necessary precautions (as described in earlier sections of this document) are taken by the modeler, the model performance turns out to be quite strong. As an illustration, the figure below highlights the actual versus predicted values of attrition rates across the deciles of a large credit card portfolio. As can be observed, the accuracy of KNN is pretty accurate across the deciles.

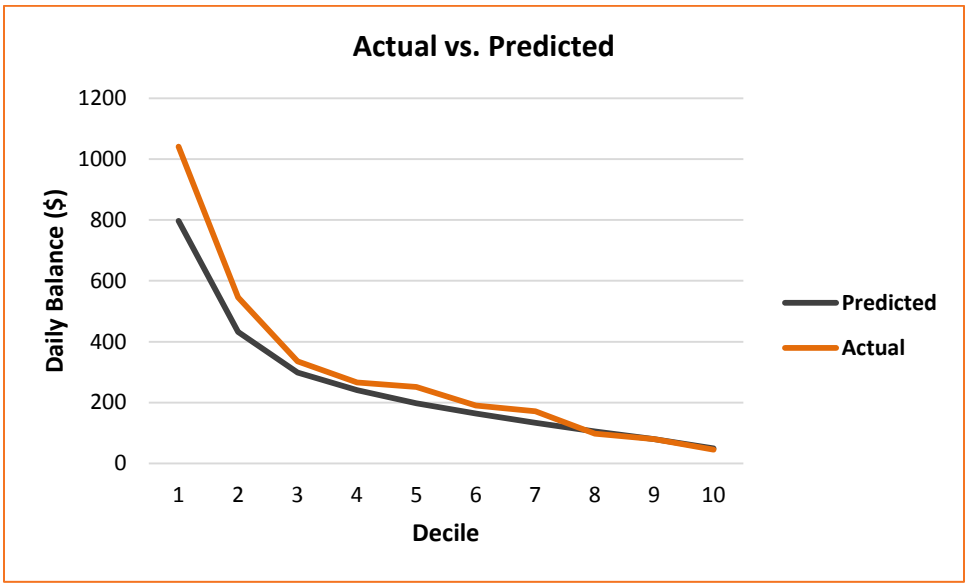
Figure 4.1: Attrition Model Performance[1]



On similar lines, the performance of the model on a categorical variable (Daily Balance) is also pretty accurate across the deciles as can be observed in the figure below.

[1] *The figures are transformed / disguised to protect the confidentiality of this modeling exercise.*

Figure 4.2: Daily Balance Model Performance[2]



However, if the treatment of feature variable (missing values, special values, outliers) is not done appropriately or if k is not selected intelligently, it can cause serious prediction errors in KNN modeling.

Hence, it is very critical to deploy the steps highlighted in this document with utmost care and appropriate diligence.

[2] *The figures are transformed / disguised to protect the confidentiality of this modeling exercise.*

Pros and Cons of KNN Modeling

As the reader would have observed across the earlier sections of this document, the pros of KNN modeling are as follows:

- Very easy to interpret
- Robust to noisy training data
- For large samples, the KNN accuracy is high
- It is a non-parametric method. So, it can be applied to the data from any distribution

The cons of KNN modeling are as follows:

- Choosing best k is difficult
- Distance based learning is confusing at times which type of distance to use and which attribute (or attributes) to use to produce best results.
- Computation cost is quite high because we need to compute the distance of each query instance to all training samples. Some indexing may reduce this computational cost.

At Axtria, we believe KNN methodology will become more and more popular in the times to come. This would be aided by the higher computational power of the technology infrastructure present in organizations. Further, over time, it would stop being a pure black box exercise, as it is being deployed by modelers today. Modelers would develop the "art" of determining appropriate thresholds of k. It would enable them to become more nuanced in modeling through KNN technique. We can't wait to see the power of techniques like KNN unleash in the real world!

Bibliography

- Anil K. Ghosh, On optimum choice of k in nearest neighbor classification, July 2005
- Leif E. Peterson (2009), Scholarpedia, 4(2):1883.,
- Lectures of The Department of Computer Science at the University of Haifa, http://www.cs.haifa.ac.il/~rita/ml_course/lectures_old/KNN.pdf

About the Authors:

Amanjeet Saluja – Principal
amanjeet.saluja@axtria.com

Amanjeet Saluja *is a Principal with Axtria. He has over 17 years of experience across analytics and management consulting. His specific area of focus has been Risk, Marketing and Operations Analytics for Banking and Financial Services clients. Most recently, he was President and Angel Investor at CoCubes Technologies Pvt. Ltd, an online / mobile startup. He scaled the business to 100+ team with 5X growth in revenue over 2 years.*

Prior to that, he was with Ocwen Financial, as Head of Quantitative Analytics. There he ran a $20 Mn servicing technology and loss mitigation product lines.

Sangita Samanta - Senior Manager
sangita.samanta@axtria.com

Sangita Samanta *is a Senior Manager with Axtria. She has around 8 years of experience across various functions in banking industry. Worked extensively on developing new credit risk scorecards, monitoring performance, underwriting risk modeling, devising portfolio management, segmentation strategies, independent model validation etc.*

Axtria Inc.: *Axtria is an advanced analytics company that combines industry knowledge, analytics and technology to deliver solutions that help companies make better data-driven sales and marketing decisions, with measurable results.*

For more information:
Please reach out to **Amanjeet Saluja** :
Email: amanjeet.saluja@axtria.com ; phone: +919582221678

Axtria, Inc. | Ingenious Insights; 400 Connell Drive, Suite 1300, Berkeley Heights, NJ 07922, USA
Email: info@axtria.com; phone: +1-877-929-8742
www.axtria.com