KNN Algorithms

Nearest Neighbor Analysis is a method for classifying cases based on their similarity to other cases. In machine learning, it was developed as a way to recognize patterns of data without requiring an exact match to any stored patterns, or cases. Similar cases are near each other and dissimilar cases are distant from each other. Thus, the distance between two cases is a measure of their dissimilarity.

Cases that are near each other are said to be "neighbors." When a new case (holdout) is presented, its distance from each of the cases in the model is computed. The classifications of the most similar cases – the nearest neighbors – are tallied and the new case is placed into the category that contains the greatest number of nearest neighbors.

You can specify the number of nearest neighbors to examine; this value is called *k*. The pictures show how a new case would be classified using two different values of *k*. When *k* = 5, the new case is placed in category *1* because a majority of the nearest neighbors belong to category *1*. However, when *k* = 9, the new case is placed in category *0* because a majority of the nearest neighbors belong to category *0*.

Nearest neighbor analysis can also be used to compute values for a continuous target. In this situation, the average or median target value of the nearest neighbors is used to obtain the predicted value for the new case.

Notation

The following notation is used throughout this chapter unless otherwise stated:

$\mathbf{Y}$	Optional $1\times N$ vector of responses with element y_n, where n=1,...,N indexes the cases.
$\mathbf{X}^0$	$P^0\times N$ matrix of features with element x^0_{pn}, where p=1,...,P^0 indexes the features and n=1,...,N indexes the cases.
$\mathbf{X}$	$P\times N$ matrix of encoded features with element x_{pn}, where p=1,...,P indexes the features and n=1,...,N indexes the cases.
P	Dimensionality of the feature space; the number of continuous features plus the number of categories across all categorical features.
N	Total number of cases.
$N_j, j = 1, 2, \cdots, J$	The number of cases with $Y = j$, where Y is a response variable with J categories
$\hat{N}_j$	The number of cases which belong to class j and are correctly classified as j.
$\hat{N}^*_j$	The total number of cases which are classified as j.

Preprocessing

Features are coded to account for differences in measurement scale.

Continuous

Continuous features are optionally coded using adjusted normalization:

$$x_{pn} = \frac{2\left(x_{pn}^0 - \min\left(x_p^0\right)\right)}{\max\left(x_p^0\right) - \min\left(x_p^0\right)} - 1$$

where x_{pn} is the normalized value of input feature *p* for case *n*, x_p^0 is the original value of the feature for case *n*, $\min\left(x_p^0\right)$ is the minimum value of the feature for all training cases, and $\max\left(x_p^0\right)$ is the maximum value for all training cases.

Categorical

Categorical features are always temporarily recoded using one-of-*c* coding. If a feature has *c* categories, then it is is stored as *c* vectors, with the first category denoted (1,0,...,0), the next category (0,1,0,...,0), ..., and the final category (0,0,...,0,1).

Training

Training a nearest neighbor model involves computing the distances between cases based upon their values in the feature set. The nearest neighbors to a given case have the smallest distances from that case. The distance metric, choice of number of nearest neighbors, and choice of the feature set have the following options.

Distance Metric

We use one of the following metrics to measure the similarity of query cases and their nearest neighbors.

Euclidean Distance. The distance between two cases is the square root of the sum, over all dimensions, of the weighted squared differences between the values for the cases.

$$Euclidean_{ih} = \sqrt{\sum_{p=1}^{P} w_{(p)}\left(x_{(p)i} - x_{(p)h}\right)^2}$$

City Block Distance. The distance between two cases is the sum, over all dimensions, of the weighted absolute differences between the values for the cases.

$$CityBlock_{ih} = \sum_{p=1}^{P} w_{(p)}\left|x_{(p)i} - x_{(p)h}\right|$$

The feature weight $w(p)$ is equal to 1 when feature importance is not used to weight distances; otherwise, it is equal to the normalized feature importance:

$$w_{(p)} = FI_{(p)} / \sum_{p=1}^{P} FI_{(p)}$$

See "Output Statistics " for the computation of feature importance $FI_{(p)}$.

Crossvalidation for Selection of k

Cross validation is used for automatic selection of the number of nearest neighbors, between a minimum $k_{\min}$ and maximum $k_{\max}$. Suppose that the training set has a cross validation variable with the integer values 1,2,..., *V*. Then the cross validation algorithm is as follows:

- For each $k \in [k_{\min}, k_{\max}]$, compute the average error rate or sum-of square error of *k*: $CV_k = \sum_{v=1}^{V} e_v / V$, where e_v is the error rate or sum-of square error when we apply the Nearest Neighbor model to make predictions on the cases with $X = v$; that is, when we use the other cases as the training dataset.
- Select the optimal *k* as: $\hat{k} = arg\{\min CV_k : k_{\min} \leq k \leq k_{\max}\}$.

Note: If multiple values of *k* are tied on the lowest average error, we select the smallest *k* among those that are tied.

Feature Selection

Feature selection is based on the wrapper approach of Cunningham and Delany (2007) and uses forward selection which starts from J_{Forced} features which are entered into the model. Further features are chosen sequentially; the chosen feature at each step is the one that causes the largest decrease in the error rate or sum-of squares error.

Let S_J represent the set of *J* features that are currently chosen to be included, S_J^c represents the set of remaining features and e_J represents the error rate or sum-of-squares error associated with the model based on S_J .

The algorithm is as follows:

- Start with $J = J_{Forced}$ features.
- For each feature in S_J^c , fit the *k* nearest neighbor model with this feature plus the existing features in S_J and calculate the error rate or sum-of square error for each model. The feature in S_J^c whose model has the smallest error rate or sum-of square error is the one to be added to create S_{J+1}.
- Check the selected stopping criterion. If satisfied, stop and report the chosen feature subset. Otherwise, *J*=*J*+1 and go back to the previous step.

Note: the set of encoded features associated with a categorical predictor are considered and added together as a set for the purpose of feature selection.

Stopping Criteria

One of two stopping criteria can be applied to the feature selection algorithm.

Fixed number of features. The algorithm adds a fixed number of features, J_{add}, in addition to those forced into the model. The final feature subset will have $J_{add} + J_{Forced}$ features. J_{add} may be user-specified or computed automatically; if computed automatically the value is

$$J_{add} = \max\left\{\min\left(20, P^0\right) - J_{Forced},\ \ 0\right\}$$

When this is the stopping criterion, the feature selection algorithm stops when J_{add} features have been added to the model; that is, when $J_{add} = J + 1$, stop and report S_{J+1} as the chosen feature subset.

Note: if $J_{add} = 0$, no features are added and S_J with $J = J_{Forced}$ is reported as the chosen feature subset.

Change in error rate or sum of squares error. The algorithm stops when the change in the absolute error ratio indicates that the model cannot be further improved by adding more features. Specifically, if $e_{J+1} = 0$ or $e_J \geq e_{J+1}$ and

$$\frac{|e_J - e_{J+1}|}{e_J} \leq \Delta_{\min}$$

where $\Delta_{\min}$ is the specified minimum change, stop and report S_{J+1} as the chosen feature subset.

If $e_J < e_{J+1}$ and

$$\frac{|e_J - e_{J+1}|}{e_J} > 2\Delta_{\min}$$

stop and report S_J as the chosen feature subset.

Note: if $e_J = 0$ for $J = J_{Forced}$, no features are added and S_J with $J = J_{Forced}$ is reported as the chosen feature subset.

Combined k and Feature Selection

The following method is used for combined neighbors and features selection.

1. For each *k*, use the forward selection method for feature selection.
2. Select the *k*, and accompanying feature set, with the lowest error rate or the lowest sum-of-squares error.

Blank Handling

All records with missing values for any input or output field are excluded from the estimation of the model.

Output Statistics

The following statistics are available.

Percent correct for class j

$$\frac{\hat{N}_j}{N_j} \times 100\%$$

Overall percent for class j

$$\frac{\hat{N}_j^*}{N} \times 100\%$$

Intersection of Overall percent and percent correct

$$\left(\sum_{j=1}^{J} \hat{N}_j / N \right) \times 100\%$$

Error rate of classification

$$\left(1 - \sum_{j=1}^{J} \hat{N}_j / N \right) \times 100\%$$

Sum-of-Square Error for continuous response

$$\sum_{n=1}^{N} (y_n - \hat{y}_n)^2$$

where $\hat{y}_n$ is the estimated value of y_n.

Feature Importance

Suppose there are $X_{(1)}, X_{(2)} \cdots X_{(m)}$ $(1 \leq m \leq P^0)$ in the model from the forward selection process with the error rate or sum-of-squares error *e*. The importance of feature $X_{(p)}$ in the model is computed by the following method.

- Delete the feature $X_{(p)}$ from the model, make predictions and evaluate the error rate or sum-of-squares error $e_{(p)}$ based on features $X_{(1)}, X_{(2)} \cdots X_{(p-1)}, X_{(p+1)}, \cdots, X_{(m)}$.
- Compute the error ratio $e_{(p)} + \frac{1}{m}$.

The feature importance of $X_{(p)}$ is $FI_{(p)} = e_{(p)} + \frac{1}{m}$

Scoring

After we find the *k* nearest neighbors of a case, we can classify it or predict its response value.

Categorical response

Classify each case by majority vote of its *k* nearest neighbors among the training cases.

- If multiple categories are tied on the highest predicted probability, then the tie should be broken by choosing the category with largest number of cases in training set.
- If multiple categories are tied on the largest number of cases in the training set, then choose the category with the smallest data value among the tied categories. In this case, categories are assumed to be in the ascending sort or lexical order of the data values.

We can also compute the predicted probability of each category. Suppose k_j is the number of cases of the *j*th category among the *k* nearest neighbors. Instead of simply estimating the predicted probability for the *j*th category by $\frac{k_j}{k}$, we apply a Laplace correction as follows:

$$\frac{k_j + 1}{k + J}$$

where *J* is the number of categories in the training data set.

The effect of the Laplace correction is to shrink the probability estimates towards to 1/*J* when the number of nearest neighbors is small. In addition, if a query case has *k* nearest neighbors with the same response value, the probability estimates are less than 1 and larger than 0, instead of 1 or 0.

Continuous response

Predict each case using the mean or median function.

Mean function. $\hat{y}_n = \sum_{m \in Nearest(n)} y_m / k$, where $Nearest(n)$ is the index set of those cases that are the nearest neighbors of case *n* and y_m is the value of the continuous response variable for case *m*.

Median function. Suppose that $y_m, m \in Nearest(n)$ are the values of the continuous response variable, and we arrange $y_m, m \in Nearest(n)$ from the lowest value to the highest value and denote them as $y_{(j_1)} \leq y_{(m_2)} \leq \cdots \leq y_{(m_k)}$, then the median is

$$\hat{y}_n = \begin{cases} y_{\left(\frac{k+1}{2}\right)} & k \text{ is odd} \\ \frac{y_{\left(\frac{k}{2}\right)} + y_{\left(\frac{k}{2}\right)+1}}{2} & k \text{ is even} \end{cases}$$

Blank Handling

Records with missing values for any input field cannot be scored and are assigned a predicted value and probability value(s) of $null$.

References

Arya, S., and D. M. Mount. 1993. Algorithms for fast vector quantization. In: *Proceedings of the Data Compression Conference 1993,* , 381–390.

Cunningham, P., and S. J. Delaney. 2007. k-Nearest Neighbor Classifiers. *Technical Report UCD-CSI-2007-4, School of Computer Science and Informatics, University College Dublin, Ireland*, , – .

Friedman, J. H., J. L. Bentley, and R. A. Finkel. 1977. An algorithm for finding best matches in logarithm expected time. *ACM Transactions on Mathematical Software*, 3, 209–226.