STAT 479: Machine Learning
Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2018/

Fall 2018

2 Nearest Neighbor Methods

2.1 Introduction

Nearest neighbor algorithms are among the "simplest" supervised machine learning algorithms and have been well studied in the field of pattern recognition over the last century. While nearest neighbor algorithms are not as popular as they once were, they are still widely used in practice, and I highly recommend that you are at least considering the k-Nearest Neighbor algorithm in classification projects as a predictive performance benchmark when you are trying to develop more sophisticated models.

In this lecture, we will primarily talk about two different algorithms, the Nearest Neighbor (NN) algorithm and the k-Nearest Neighbor (kNN) algorithm. NN is just a special case of kNN, where $k = 1$. To avoid making this text unnecessarily convoluted, we will only use the abbreviation NN if we talk about concepts that do not apply to kNN in general. Otherwise, we will use kNN to refer to nearest neighbor algorithms in general, regardless of the value of k.

2.1.1 Key concepts

While kNN is a universal function approximator under certain conditions, the underlying concept is relatively simple. kNN is an algorithm for supervised learning that simply stores the labeled training examples,

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n), \tag{1}$$

during the training phase. For this reason, kNN is also called a **lazy** learning algorithm.

What it means to be a *lazy* learning algorithm is that the processing of the training examples is postponed until making predictions [1] – again, the training consists literally of just storing the training data.

[1] When you are reading recent literature, note that the *prediction* step is now often called "inference" in the machine learning community

Then, to make a prediction (class label or continuous target), the *kNN* algorithms find the k nearest neighbors of a query point and compute the class label (classification) or continuous target (regression) based on the k nearest (most "similar") points. The exact mechanics will be explained in the next sections. However, the overall idea is that instead of approximating the target function $f(\mathbf{x}) = y$ globally, during each prediction, kNN approximates the target function locally. In practice, it is easier to learn to approximate a function locally than globally.

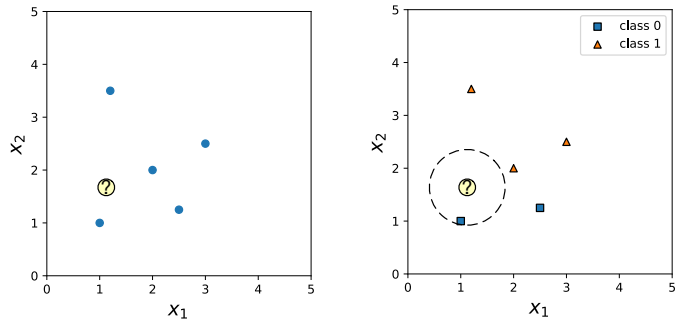

Figure 1: Illustration of the nearest neighbor classification algorithm in two dimensions (features x_1 and x_2). In the left subpanel, the training examples are shown as blue dots, and a query point that we want to classify is shown as a question mark. In the right subpanel, the class labels are, and the dashed line indicates the nearest neighbor of the query point, assuming a Euclidean distance metric. The predicted class label is the class label of the closest data point in the training set (here: class 0).

2.1.2 Nearest Neighbor Classification In Context

In the previous lecture, we learned about different kinds of categorization schemes, which may be helpful for understanding and distinguishing different types of machine learning algorithms.

To recap, the categories we discussed were

- eager vs lazy;
- batch vs online;
- parametric vs nonparametric;
- discriminative vs generative.

Since kNN does not have an explicit training step and defers all of the computation until prediction, we already determined that kNN is a *lazy* algorithm.

Further, instead of devising one global model or approximation of the target function, for each different data point, there is a different local approximation, which depends on the data point itself as well as the training data points. Since the prediction is based on a comparison of a query point with data points in the training set (rather than a global model), kNN is also categorized as **instance-based** (or "memory-based") method. While kNN is a lazy instance-based learning algorithm, an example of an eager instance-based learning algorithm would be the support vector machine, which will be covered later in this course.

Lastly, because we do not make any assumption about the functional form of the kNN algorithm, a kNN model is also considered a **nonparametric** model. However, categorizing

kNN as either discriminative or generative is not as straightforward as for other algorithms. Under certain assumptions, we can estimate the conditional probability that a given data point belongs to a given class as well as the marginal probability for a feature (more details are provided in the section on "kNN from a Bayesian Perspective" later) given a training dataset. However, since kNN does not explicitly try to model the data generating process but models the posterior probabilities ($p(\mathbf{x}|f(\mathbf{x}))$) directly, kNN is usually considered a discriminative model.

2.1.3 Common Use Cases of *k*NN

While neural networks are gaining popularity in the computer vision and pattern recognition field, one area where k-nearest neighbors models are still commonly and successfully being used is in the intersection between computer vision, pattern classification, and biometrics (e.g., to make predictions based on extracted geometrical features[2]).

Other common use cases include recommender systems (via collaborative filtering[3]) and outlier detection[4].

2.2 Nearest Neighbor Algorithm

After introducing the overall concept of the nearest neighbor algorithms, this section provides a more formal or technical description of the 1-nearest neighbor (NN) algorithm.

Training algorithm:

for $i = 1, ..., n$ in the n-dimensional training dataset $\mathcal{D}$ ($|\mathcal{D}| = n$):

- store training example $\langle \mathbf{x}^{[i]}, f\left(\mathbf{x}^{[i]}\right)\rangle$

Prediction algorithm [5]:

`closest_point := None`

`closest_distance` $:= \infty$

- `for` $i = 1, ..., n$:
  - `current_distance` $:= d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
  - `if current_distance` $<$ `closest_distance`:
    - `closest_distance` $:=$ `current_distance`
    - `closest_point` $:= \mathbf{x}^{[i]}$

prediction $h(\mathbf{x}^{[q]})$ is the target value of `closest_point`

Unless noted otherwise, the default distance metric (in the context of this lecture) of nearest neighbor algorithms is the Euclidean distance (also called L^2 distance), which computes the

[2] Asmaa Sabet Anwar, Kareem Kamal A Ghany, and Hesham Elmahdy. "Human ear recognition using geometrical features extraction". In: *Procedia Computer Science* 65 (2015), pp. 529–537.

[3] Youngki Park et al. "Reversed CF: A fast collaborative filtering algorithm using a k-nearest neighbor graph". In: *Expert Systems with Applications* 42.8 (2015), pp. 4022–4028.

[4] Guilherme O Campos et al. "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 891–927.

[5] We use ":=" as an assignment operator.

distance between two points, $\mathbf{x}^{[a]}$ and $\mathbf{x}^{[b]}$:

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} \left(x_j^{[a]} - x_j^{[b]}\right)^2}. \tag{2}$$

2.3 Nearest Neighbor Decision Boundary

In this section, we will build some intuition for the decision boundary of the NN classification model. Assuming a Euclidean distance metric, the decision boundary between any two training examples a and b is a straight line. If a query point is located on the decision boundary, this means its equidistant from both training example a and b.

While the decision boundary between a pair of points is a straight line, the decision boundary of the NN model on a global level, considering the whole training set, is a set of connected, convex polyhedra. All points within a polyhedron are closest to the training example inside, and all points outside the polyhedron are closer to a different training example.

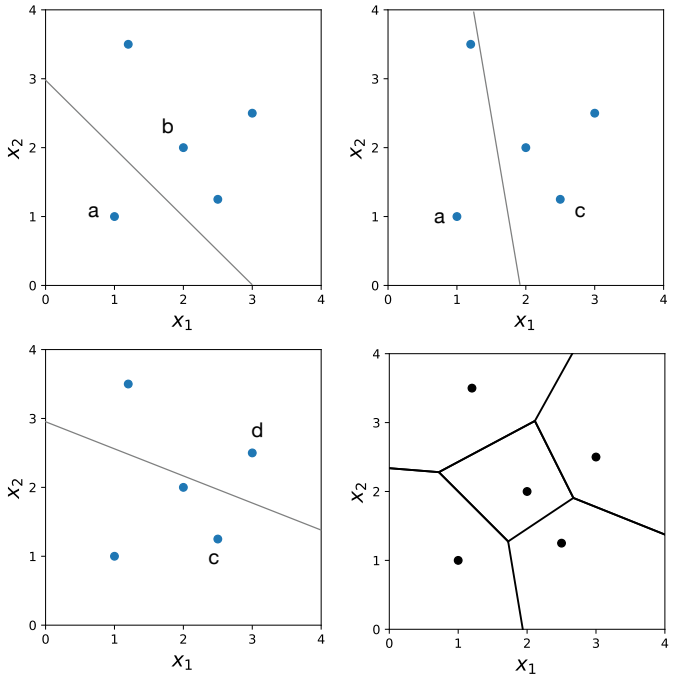

Figure 2: Illustration of the plane-partitioning of a two-dimensional dataset (features x_1 and x_2) via linear sigments between two training examples (a & b, a & c, and c & d) and the resulting Voronoi diagram (upper right corner)

This partitioning of regions on a plane in 2D is also called "Voronoi diagram" or Voronoi tessellation. (You may remember from geometry classes that given a discrete set of points, a Voronoi diagram can also be obtained by a process known as Delaunay triangulation[6] by connecting the centers of the circumcircles.)

While each linear segment is equidistant from two different training examples, a vertex (or node) in the Voronoi diagram is equidistant to three training examples. Then, to draw the decision boundary of a two-dimensional nearest neighbor classifier, we take the union of the pair-wise decision boundaries of instances of the same class.

[6] **https://en.wikipedia.org/wiki/Delaunay˙triangulation**.

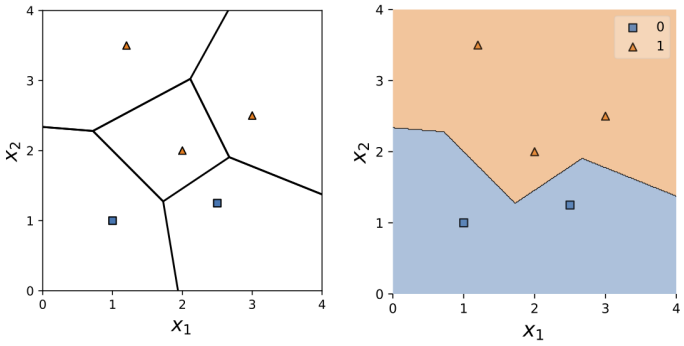

Figure 3: Illustration of the nearest neighbor decision boundary as the union of the polyhedra of training examples belonging to the same class.

2.4 *k*-Nearest Neighbor Classification and Regression

Previously, we described the NN algorithm, which makes a prediction by assigning the class label or continuous target value of the most similar training example to the query point (where similarity is typically measured using the Euclidean distance metric for continuous features).

Instead of basing the prediction of the single, most similar training example, kNN considers the k nearest neighbors when predicting a class label (in classification) or a continuous target value (in regression).

2.4.1 Classification

In the classification setting, the simplest incarnation of the kNN model is to predict the target class label as the class label that is most often represented among the k most similar training examples for a given query point. In other words, the class label can be considered as the "mode" of the k training labels or the outcome of a "plurality voting." Note that in literature, kNN classification is often described as a "majority voting." While the authors usually mean the right thing, the term "majority voting" is a bit unfortunate as it typically refers to a reference value of >50% for making a decision. In the case of binary predictions (classification problems with two classes), there is always a majority or a tie. Hence, a majority vote is also automatically a plurality vote. However, in multi-class settings, we do not require a majority to make a prediction via kNN. For example, in a three-class setting a frequency $> \frac{1}{3}$ (approx 33.3%) could already enough to assign a class label.

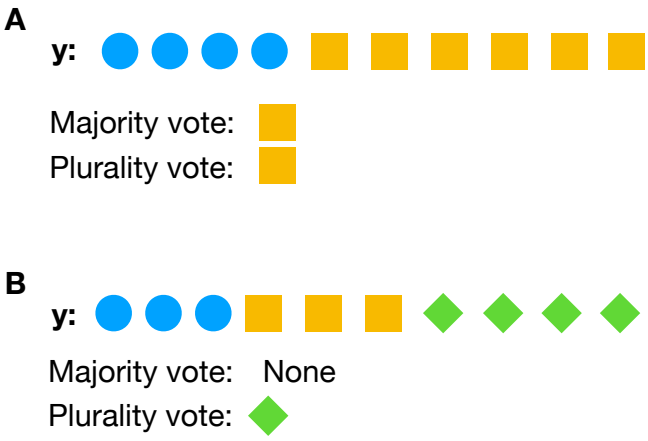

Figure 4: Illustration of plurality and majority voting.

Remember that the NN prediction rule (recall that we defined NN as the special case of kNN with $k = 1$) is the same for both classification or regression. However, in kNN we have two distinct prediction algorithms:

- Plurality voting among the k nearest neighbors for classification.
- Averaging the continuous target variables of the k nearest neighbors for regression.

More formally, assume we have a target function $f(\mathbf{x}) = y$ that assigns a class label $y \in \{1, \ldots, t\}$ to a training example,

$$f : \mathbb{R}^{\geqslant} \rightarrow \{1, ..., t\}. \tag{3}$$

(Usually, we use the letter k to denote the number of classes in this course, but in the context of k-NN, it would be too confusing.)

Assuming we identified the k nearest neighbors ($\mathcal{D}_{\|} \subseteq \mathcal{D}$) of a query point $\mathbf{x}^{[q]}$,

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f\big(\mathbf{x}^{[1]}\big)\rangle, \ldots, \langle \mathbf{x}^{[k]}, f\big(\mathbf{x}^{[k]}\big)\rangle\}, \tag{4}$$

we can define the kNN *hypothesis* as

$$h(\mathbf{x}^{[q]}) = \arg \max_{y \in \{1,\ldots,t\}} \sum_{i=1}^{k} \delta(y, f(\mathbf{x}^{[i]})). \tag{5}$$

Here, δ denotes the Kronecker Delta function

$$\delta(a, b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{if } a \neq b. \end{cases} \tag{6}$$

Or, in simpler notation, if you remember the "mode" from introductory statistics classes:

$$h(\mathbf{x}^{[t]}) = \text{mode}\big(\{f\big(\mathbf{x}^{[1]}\big), \ldots, f\big(\mathbf{x}^{[k]}\big)\}\big). \tag{7}$$

A common distance metric to identify the k nearest neighbors $\mathcal{D}_k$ is the Euclidean distance measure,

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} \left(x_j^{[a]} - x_j^{[b]}\right)^2}, \tag{8}$$

which is a pairwise distance metric that computes the distance between two data points $\mathbf{x}^{[a]}$ and $\mathbf{x}^{[b]}$ over the m input features.

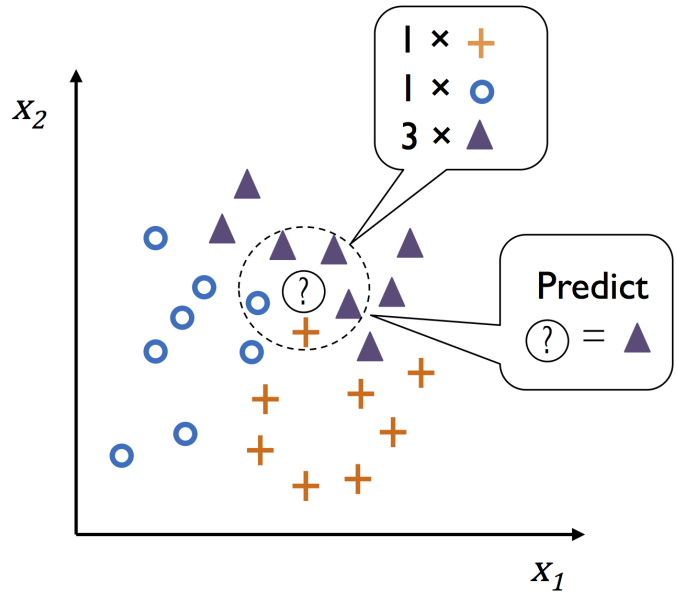

Figure 5: Illustration of kNN for a 3-class problem with k=5.

2.4.2 Regression

The general concept of kNN for regression is the same as for classification: first, we find the k nearest neighbors in the dataset; second, we make a prediction based on the labels of the k nearest neighbors. However, in regression, the target function is a real- instead of discrete-valued function,

$$f : \mathbb{R}^{\geq} \rightarrow \mathbb{R}. \tag{9}$$

A common approach for computing the continuous target is to compute the mean or average target value over the k nearest neighbors,

$$h(\mathbf{x}^{[t]}) = \frac{1}{k} \sum_{i=1}^{k} f(\mathbf{x}^{[i]}). \tag{10}$$

As an alternative to averaging the target values of the k nearest neighbors to predict the label of a query point, it is also not uncommon to use the median instead.

2.5 Curse of Dimensionality

The kNN algorithm is particularly susceptible to the *curse of dimensionality*[7]. In machine learning, the curse of dimensionality refers to scenarios with a fixed size of training examples but an increasing number of dimensions and range of feature values in each dimension in a high-dimensional feature space.

In kNN an increasing number of dimensions becomes increasingly problematic because the more dimensions we add, the larger the volume in the hyperspace needs to be to capture a

[7] David L Donoho et al. "High-dimensional data analysis: The curses and blessings of dimensionality". In: *AMS math challenges lecture* 1.2000 (2000), p. 32.