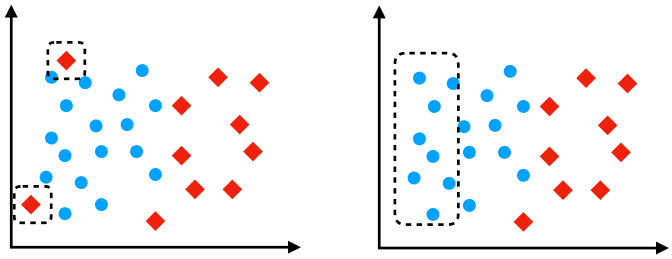

**Figure 7:** Illustration of $k$NN editing, where we can remove points from the training set that do not influence the predictions. For example, consider a 3-NN model. On the left, the two points enclosed in dashed lines would not affect the decision boundary as "outliers." Similarly, points of the "right" class that are very far away from the decision boundary, as shown in the right subpanel, do not influence the decision boundary and hence could be removed for efficiency concerning data storage or the number of distance computations.

#### Prototypes

Another strategy (somewhat related to KMeans, a clustering algorithm that we will cover towards the end of this course), is to replace selected data points by prototypes that summarize multiple data points in dense regions.

### 2.7.6 Parallelizing $k$NN

$k$NN is one of these algorithms that are very easy to *parallelize*. There are many different ways to do that. For instance, we could use distributed approaches like map-reduce and place subsets of the training datasets on different machines for the distance computations. Further, the distance computations themselves can be carried out using parallel computations on multiple processors via CPUs or GPUs.

## 2.8 Distance measures

There are many distance metrics or measures we can use to select $k$ nearest neighbors. There is no "best" distance measure, and the choice is highly context- or problem-dependent.

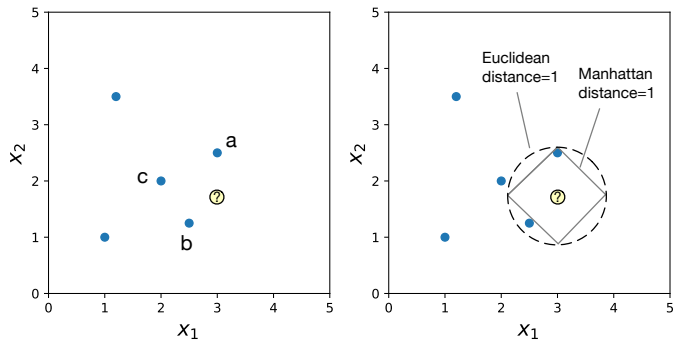


**Figure 8:** The phrase "nearest" is ambiguous and depends on the distance metric we use.

For continuous features, the probably most common distance metric is the Euclidean dis-

tance. Another popular choice is the Manhattan distance,

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sum_{j=1}^{m} \left| x^{[a]} - x^{[b]} \right|, \tag{14}$$

which emphasizes differences between "distant" feature vectors or outliers less than the Euclidean distance.

A generalization of the Euclidean or Manhattan distance is the so-called Minkowski distance,

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \left[ \sum_{j=1}^{m} \left( \left| x^{[a]} - x^{[b]} \right| \right)^p \right]^{\frac{1}{p}}, \tag{15}$$

which is equal to the Euclidean distance if $p = 2$ and equal to the Manhattan distanced if $p = 1$.

The Mahalanobis distance would be another good choice for a distance metric as it considers the variance of the different features as well as the covariance among them. However, one downside of using more "sophisticated" distance metrics is that it also typically negatively impacts computational efficiency. For instance, the Mahalanobis distance is substantially more challenging to implement efficiently, for example, if we consider running $k$NN in a distributed-fashion, as it requires the covariances as a scaling term.

### 2.8.1 Discrete Features

Minkowski-based distance metrics can be used for discrete and continuous features. One example would be the so-called Hamming distance, which really is just the Manhattan distance applied to binary feature vectors:

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sum_{j=1}^{m} \left| x^{[a]} - x^{[b]} \right|. \tag{16}$$

The Hamming distance is also called overlap metric, because it essentially measures how many positions in two vectors are different. For instance, considering two vectors

$$\mathbf{a} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{17}$$

The Hamming distance is one, because the vectors only differ in one position.

As we discussed in class, If we are working with vectors containing word counts of documents and we want to measure the similarity (or distance) between two documents, cosine similarity could be a metric that is more appropriate than, for example, the Euclidean distance. The cosine similarity[13] is defined as the the dot-product between two vectors normalized by their magnitude:

$$\cos(\theta) = \frac{\mathbf{a}^T \cdot \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||} \tag{18}$$

To provide some intuition for why this measure could be more indicative of the similarity of two document vectors, consider a document $\mathbf{a}$ in which we duplicated every sentence $\mathbf{a}'$.

[13] Please note that while cosine similarity can be useful to measure distances in *k*NN, it is not a proper distance metric as it violates the triangle-inequality, $d(\mathbf{a}, \mathbf{c}) \leq d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c})$.

Then, the cosine similarity to another document vector **b** would be the same for **a** and **a**′, which is not true for, for example, the dot product.

While document-word counts were used as an illustrative example above, please do not about the details at this point as we will discover text analysis in a separate lecture in this course.

Also, an important consideration when we are talking about discrete or categorical features is whether the features "categories" are on an ordinal or nominal scale. We will discuss this in detail in a feature lecture on "Data Preprocessing."

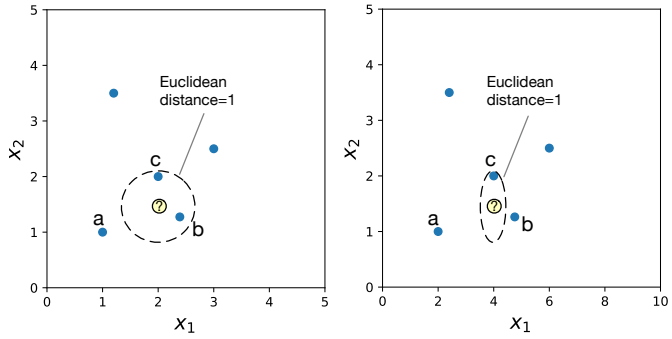


**Figure 9:** Next, to choosing an appropriate distance metric, feature scaling is another important consideration, which is illustrated in this figure. The right subpanel illustrates the effect of scaling the x_1 axis by a factor of 2 on finding the nearest neighbor via Euclidean distance.

### 2.8.2 Feature Weighting

Further, we can modify distances metrics by adding a weight to each feature dimension, which is equivalent to feature scaling. In the case of the Euclidean distance, this would look as follows:

$$d_w(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} w_j \left(x_j^{[a]} - x_j^{[b]}\right)^2}, \tag{19}$$

where $w_j \in \mathbb{R}$.

To implement this efficiently in code, we can express the weighting as a transformation matrix, where the transformation matrix is a diagonal matrix consisting of the $m$ weight coefficient for the $m$ features:

$$\mathbf{W} \in \mathbb{R}^{m \times m} = \text{diag}(w_1, w_2, ..., w_m). \tag{20}$$

In particular, note that we can express the standard Euclidean distance as a dot product of the distance vector $x$:

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{(\mathbf{x}^{[a]} - \mathbf{x}^{[b]})^T (\mathbf{x}^{[a]} - \mathbf{x}^{[b]})}. \tag{21}$$

Then, the distance-weighted Euclidean distance can be expressed as a follows:

$$d_w(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{(\mathbf{x}^{[a]} - \mathbf{x}^{[b]})^T \mathbf{W} (\mathbf{x}^{[a]} - \mathbf{x}^{[b]})}. \tag{22}$$

## 2.9 Distance-weighted *k*NN

A variant of $k$NN is distance-weighted $k$NN. In "regular" $k$NN, the all $k$ neighbors participate similarly in the plurality voting or averaging. However, especially if the radius enclosing a

set of neighbors is large, we may want to give a stronger weights to neighbors that are "closer" to the query point. For instance, we can assign a weight $w$ to the neighbors in $k$NN classification,

$$h(\mathbf{x}^{[t]}) = \arg \max_{j \in \{1,\dots,p\}} \sum_{i=1}^{k} w^{[i]} \delta(j, f(\mathbf{x}^{[i]})). \tag{23}$$

Simiarly, we can define the following equation for $k$NN regression:

$$h(\mathbf{x}^{[t]}) = \frac{\sum_{i=1}^{k} w^{[i]} f(\mathbf{x}^{[i]})}{\sum_{i=1}^{k} w^{[i]}}. \tag{24}$$

As described by Tom Mitchell[14], a popular weighting scheme is using the inverse squared distance

$$w^{[i]} = \frac{1}{d(\mathbf{x}^{[i]}, \mathbf{x}^{[t]})^2}, \tag{25}$$

where $h(\mathbf{x}) = f(\mathbf{x})$ is used for an exact match. Other strategies include adding a small constant to the denominator for avoiding zero-division errors.

Also, using a weighting scheme as described above, we can also turn the $k$NN algorithm into a *global* method by considering all data points instead of $k$, as defined by Shepard[15].

## 2.10 Improving Predictive Performance

There are several different ways of how we can improve the predictive performance of the $k$NN algorithms:

- Choosing the value of $k$.
- Scaling of the feature axes.
- Choice of distance measure.
- Weighting of the distance measure.

However, other techniques such as "editing" (removing noisy data points) and so forth also affect the generalization performance (i.e., the predictive performance on unseen, that is, non-training data) of $k$NN.

Choosing between different settings of an algorithm is also known as "hyperparameter tuning" or "model selection," which is typically performed by cross-validation.

[14] T.M. Mitchell. "Machine Learning". In: McGraw-Hill International Editions. McGraw-Hill, 1997. Chap. 8. ISBN: 9780071154673. URL: https://books.google.com/books?id=EoYBngEACAAJ.

[15] Donald Shepard. "A two-dimensional interpolation function for irregularly-spaced data". In: *Proceedings of the 1968 23rd ACM national conference*. ACM. 1968, pp. 517–524.

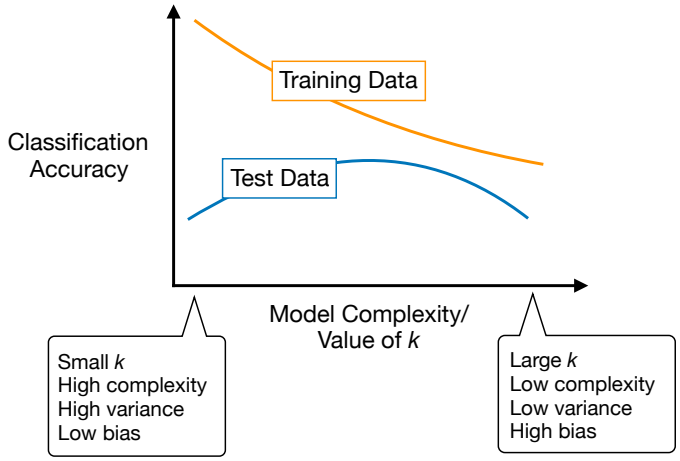


**Figure 10:** By changing the value of $k$, we affect the complexity of a $k$NN model. In practice, we try to find a good trade-off between high bias (the model is not complex enough to fit the data well) and high variance (the model fits the training data too closely). We will discuss overfitting and the bias-variance trade-off in more detail in future lectures.

In particular, model selection helps us with reducing the effect of the curse of dimensionality and overfitting to the training data, if done properly. We will discuss model selection and cross-validation later in this course.

In practice, values of $k$ between 3-15 seem reasonable choices. Also, if we are working on binary classification problems, it is a good idea to avoid even numbers for $k$ to prevent ties.

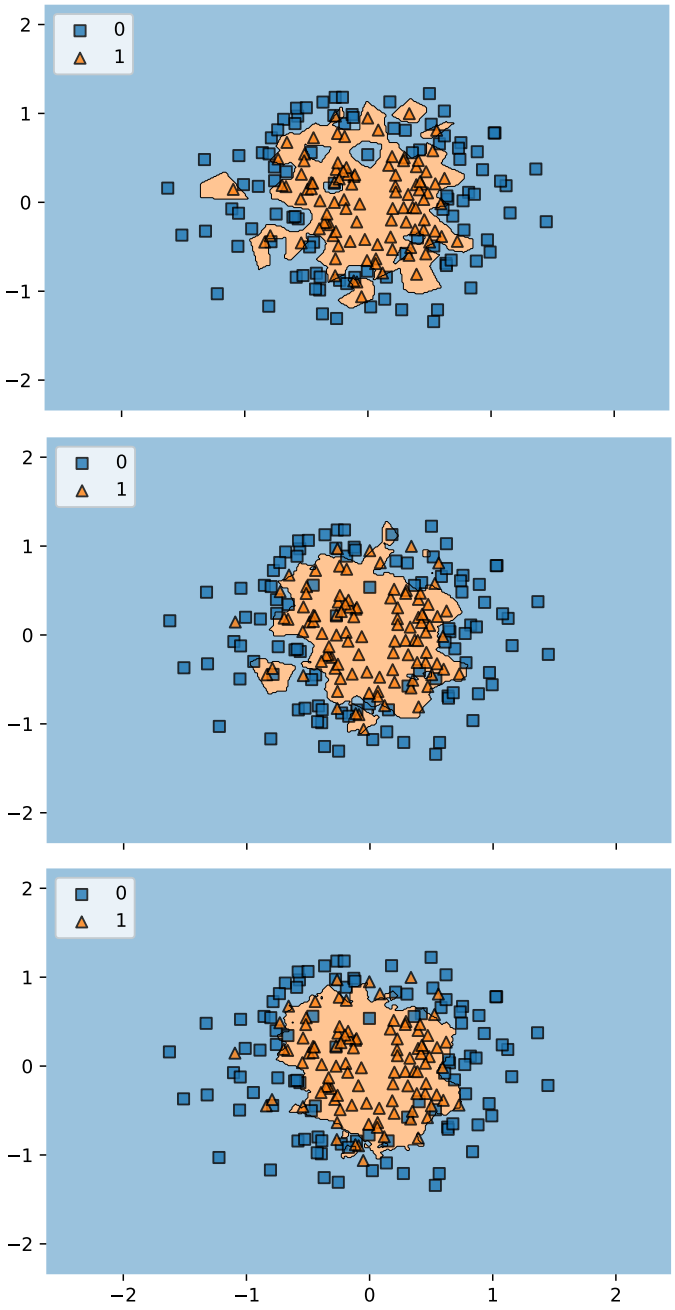


**Figure 11:** An illustration of the effect of changing the value of $k$ on a simple toy dataset.

## 2.11 Error Bounds

Cover and Hart have proved that the 1-NN algorithm is guaranteed to be no worse than twice the Bayes error[16]. The Bayes error is the minimum possible error that can be achieved; we will discuss the Bayes error in future lectures. We will not cover the proof of the error bounds in class, but interested students are encouraged to read more in Chapter 4 (Nonparametric

[16]Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

Techniques) of Duda, Hart, and Stork's *Pattern Classification* book[17].

## 2.12 *k*NN from a Bayesian Perspective

As a statistics student, you may be interested in looking at $k$NN from a Bayesian perspective. Duda, Hard, and Stork provide an excellent theoretical foundation together with helpful illustrations in their book "Pattern Classification[18]" (Chapter 4, Section 4.4) – this is out of the scope of this class but recommended if you are interested. If you are not familiar with Bayes theorem, yet (no worries if you are not, we will cover it later in this course), do not worry about it, and please feel free to skip this section.

In my attempt below, I try to describe the $k$NN concept from a Bayesian perspective using our familiar notation.

First, recall the Bayes theorem

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}. \tag{26}$$

In our context, that is

$$\text{Posterior Prob.} = \frac{\text{Likelihood} \times \text{Prior Prob.}}{\text{Marginal Prob.}} \tag{27}$$

or

$$P(h(\mathbf{x}) = i|\mathbf{x}) = \frac{p(\mathbf{x}|h(\mathbf{x}) = i) \times p(h(\mathbf{x}) = i)}{p(\mathbf{x})}. \tag{28}$$

Suppose we have a dataset $\mathcal{D}$ with $n$ data points and $t$ classes. $\mathcal{D}_i$ denotes the subset of data points with class label $h(\mathbf{x}) = i$, $i \in \{1, ..., t\}$.

Now, to classify a new data point $\mathbf{x}$, we draw a tight sphere around the $k$ nearest neighbors of $\mathbf{x}$. This sphere has the volume $V$.

Then

$$p\big(\mathbf{x}|h(\mathbf{x}) = i\big) = \frac{k_i}{|D_i| \times V} \tag{29}$$

is a density estimate of class $i$, where $k_i$ denote the $k$ neighbors with class label $i$. The marginal probability, $p(\mathbf{x})$ can then be computed as

$$p(\mathbf{x}) = \frac{k}{|\mathcal{D}| \times V}. \tag{30}$$

The class priors are then computed as

$$p(h(\mathbf{x}) = i) = \frac{|\mathcal{D}_\rangle|}{|\mathcal{D}|}. \tag{31}$$

Combining the equations, we get the posterior probability that the data point $\mathbf{x}$ belongs to class $i$:

$$P(h(\mathbf{x}) = i|\mathbf{x}) = \frac{p(\mathbf{x}|h(\mathbf{x}) = i) \times p(h(\mathbf{x}) = i)}{p(\mathbf{x})} = \frac{k_i}{k}. \tag{32}$$

Stable approximation under the assumption that

$$|\mathcal{D}|, k \to \infty \tag{33}$$

and

$$\frac{k}{|\mathcal{D}|} \to 0. \tag{34}$$

[17] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.
[18] Duda, Hart, and Stork, *Pattern classification*.

## 2.13 Advantages and Disadvantages of *k*NN

One of the most significant advantages of $k$NN is that it is relatively easy to implement and interpret. Also, with its approach to approximate complex global functions locally, it can be a powerful predictive model.

The downsides are that $k$NN is very sensitive to the curse of dimensionality and expensive to compute with a $O(n)$ prediction step – however, smart implementations and use of data structures such as KD-trees and Ball-trees can make $k$NN substantially more efficient.

Compared to other machine learning algorithms, the basic $k$NN algorithm has relatively few hyperparameters, namely $k$ and the distance metric; however, the choice of an appropriate distance metric is not always obvious. Additional hyperparameters are added if we consider rescaling the feature axes and weighting neighbors by their distance from the query point, for example.

## 2.14 Other Forms of Instance-based Learning

### 2.14.1 Locally Weighted Regression

Another popular example of instance-based learning is *locally weighted regression*. The idea behind locally weighted regression is straightforward. Similar to $k$NN, training examples neighboring a query point are selected. Then, a regression model is fit to that selected set of training examples to predict the value of a given data point. Essentially, we are approximating the target function locally, which is easier than learning a hypothesis that approximates the target function well on a global scale.

The regression model could have any form, really; however, linear regression models are popular in the context of locally weighted regression because it is simple, computationally efficient, and performs sufficiently well for approximating the local neighborhood of a given data point.

Interestingly, the concept behind locally weighted regression has been recently used to approximate decision functions locally to help interpret decisions made by "complex" models (random forests, multi-layer networks, etc.), for example, via the LIME technique (LIME is an acronym for Local Interpretable Model-Agnostic Explanations)[19].

### 2.14.2 Kernel Methods

The term kernel may be a bit confusing due to its varied use, but in the context of machine learning, kernel methods are usually associated with what is known as the "kernel trick." The probably most widely used kernel method is the support vector machine (its standard form can be interpreted as a special case using a linear kernel). We will discuss this in more detail later in this course. In a nutshell, kernel methods use a kernel (as mentioned earlier, a "similarity function") to measure the similarity or distance between pairs of data points, and the "trick" refers to an implicit transformation into a higher dimensional space where a linear classifier can separate the data points. However, while SVM can be considered an instance-based learner, it is not a "lazy" learner such as $k$NN.

[19] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

## 2.15 *k*NN in Python

Please see the complimentary Jupyter Notebook for code examples at

https://github.com/rasbt/stat479-machine-learning-fs18/blob/master/02_knn/02_knn_demo.ipynb

## 2.16 Resources

## 2.17 Assigned Reading

- Elements of Statistical Learning, Ch 02 sections 2.0-2.3
- Scikit-learn documentation http://scikit-learn.org/stable/modules/neighbors.html Section "1.6.4. Nearest Neighbor Algorithms," which has a nice discussion on the advantages and disadvantages of the different neighbor search approaches (brute force, KD Tree, and Ball Tree)

## 2.18 Further Reading

Listed below are optional reading materials for students interested in the theory of 1NN and *k*NN, including the Bayes error rate proofs of 1NN.


- Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21-27.
- Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons. Chapter 4