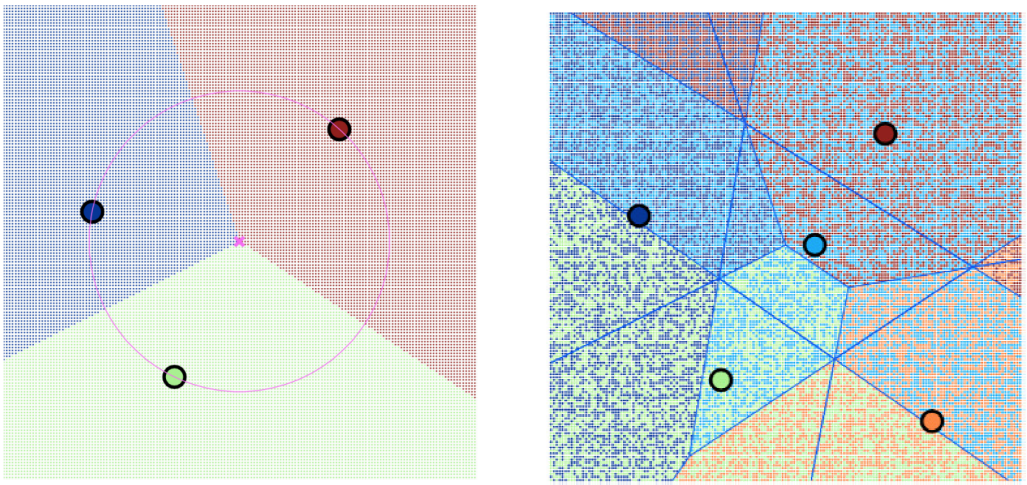
1 Nearest Neighbor Classification

In classification, it is reasonable to conjecture that data points that are sufficiently close to one another should be of the same class. For example, in fruit classification, perturbing a few pixels in an image of a banana should still result in something that looks like a banana. The **k-nearest-neighbors (k-NN)** classifier is based on this observation. Assuming that there is no preprocessing of the training data, the training time for k-NN is effectively $O(1)$. To train this classifier, we simply store our training data for future reference.[1] For this reason, k-NN is sometimes referred to as "lazy learning." The major work of k-NNs in done at testing time: to predict on a test data point $\mathbf{z}$, we compute the k closest training data points to $\mathbf{z}$, where "closeness" can be quantified in some distance function such as Euclidean distance — these are the k *nearest neighbors* to $\mathbf{z}$. We then find the most common class y among these k neighbors and classify $\mathbf{z}$ as y (that is, we perform a majority vote). For binary classification, k is usually chosen to be odd so we can break ties cleanly. Note that k-NN can also be applied to regression tasks — in that case k-NN would return the average label of the k nearest points.



(a) 1-NN on 3 classes

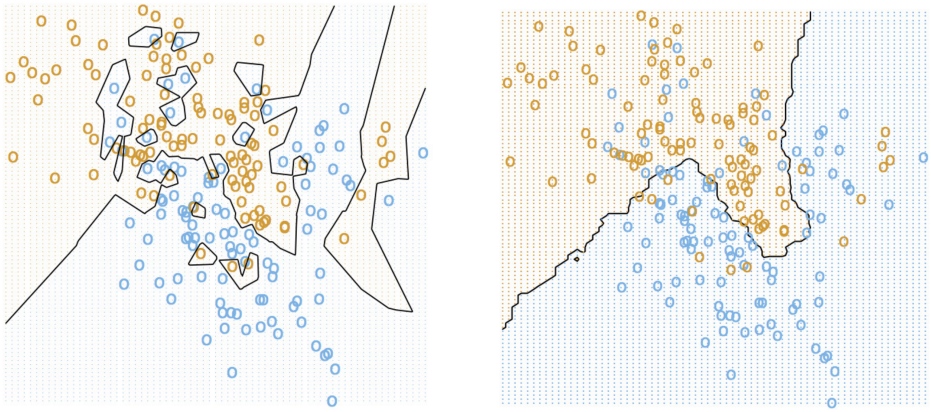
(b) 2-NN on 5 classes

Figure 1: **Voronoi diagram** for k-NN. Points in a region shaded a certain color will be classified as that color. Test points in a region shaded with a combination of 2 colors have those colors as their 2 nearest neighbors.

[1] Sometimes we store the data in a specialized structure called a *k-d tree*. This data structure is out of scope for this course, but it usually allows for faster (average-case $O(\log n)$) nearest neighbors queries.

1.1 Choosing k

Nearest neighbors can produce very complex decision functions, and its behavior is highly dependent on the choice of k.



(a) $k = 1$ (b) $k = 15$

Figure 2: **Voronoi diagram** for $k = 1$ vs. $k = 15$. Figure from Introduction to Statistical Learning.

Choosing $k = 1$, we achieve an optimal training error of 0 because each training point will classify as itself, thus achieving 100% accuracy on itself. However, $k = 1$ overfits to the training data, and is a terrible choice in the context of the bias-variance tradeoff. Increasing k leads to an increase in training error, but a decrease in testing error and achieves better generalization. At one point, if k becomes too large, the algorithm will underfit the training data, and suffer from huge bias. In general, in order to select k we use cross-validation.

Figure 3: Training and Testing error as a function of k. Figure from Introduction to Statistical Learning.

1.2 Bias-Variance Analysis

Let's justify this reasoning formally for k-NN applied to regression tasks. Suppose we are given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where the labels y_i are real valued scalars. We model our hypothesis $h(\mathbf{z})$ as

$$h(\mathbf{z}) = \frac{1}{k}\sum_{i=1}^{n} N(\mathbf{x}_i, \mathbf{z}, k)$$

where the function N is defined as

$$N(\mathbf{x}_i, \mathbf{z}, k) = \begin{cases} y_i & \text{if } \mathbf{x}_i \text{ is one of the } k \text{ closest points to } \mathbf{z} \\ 0 & o.w. \end{cases}$$

Suppose also we assume our labels $y_i = f(\mathbf{x}_i) + \epsilon$, where ϵ is the noise that comes from $\mathcal{N}(0, \sigma^2)$ and f is the true function. Without loss of generality, let $\mathbf{x}_1 \dots \mathbf{x}_k$ be the k closest points. Let's first derive the bias^2 of our model for the given dataset $\mathcal{D}$.

$$\begin{aligned}
\big(\mathbb{E}[h(\mathbf{z})] - f(\mathbf{z})\big)^2 &= \left(\mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{n} N(\mathbf{x}_i, \mathbf{z}, k)\right] - f(\mathbf{z})\right)^2 = \left(\mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} y_i\right] - f(\mathbf{z})\right)^2 \\
&= \left(\frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[y_i] - f(\mathbf{z})\right)^2 = \left(\frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[f(\mathbf{x}_i) + \epsilon] - f(\mathbf{z})\right)^2 \\
&= \left(\frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}_i) - f(\mathbf{z})\right)^2
\end{aligned}$$

When $k \longrightarrow \infty$, then $\frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}_i)$ goes to the average label for $\mathbf{x}$. When $k = 1$, then the bias is simply $f(\mathbf{x}_1) - f(\mathbf{z})$. Assuming $\mathbf{x}_1$ is close enough to $f(\mathbf{z})$, the bias would likely be small when $k = 1$ since it's likely to share a similar label. Meanwhile, when $k \longrightarrow \infty$, the bias doesn't depend on the training points at all which like will restrict it to be higher.

Now, let's derive the variance of our model.

$$\begin{aligned}
\text{Var}[h(\mathbf{z})] &= \text{Var}\left[\frac{1}{k}\sum_{i=1}^{k} y_i\right] = \frac{1}{k^2}\sum_{i=1}^{k} \text{Var}[f(\mathbf{x}_i) + \epsilon] \\
&= \frac{1}{k^2}\sum_{i=1}^{k} \text{Var}[\epsilon] \\
&= \frac{1}{k^2}\sum_{i=1}^{k} \sigma^2 = \frac{1}{k^2} k\sigma^2 = \frac{\sigma^2}{k}
\end{aligned}$$

The variance goes to 0 when $k \longrightarrow \infty$, and is maximized at $k = 1$.

1.3 Properties

Computational complexity: We require $O(n)$ space to store a training set of size n. There is no runtime cost during training if we do not use specialized data structures to store the data. However, predictions take $O(n)$ time, which is costly. There has been research into approximate nearest neighbors (ANN) procedures that quickly find an approximation for the nearest neighbor - some common ANN methods are *Locality-Sensitive Hashing* and algorithms that perform dimensionality reduction via randomized (Johnson-Lindenstrauss) distance-preserving projections.[2]

Flexibility: When $k > 1$, k-NN can be modified to output predicted probabilities $P(Y|X)$ by defining $P(Y|X)$ as the proportion of nearest neighbors to X in the training set that have class Y. k-NN can also be adapted for regression — instead of taking the majority vote, take the average of the y values for the nearest neighbors. k-NN can learn very complicated, non-linear decision boundaries.

Non-parametric: k-NN is a **non-parametric method**, which means that the number of parameters in the model grows with n, the number of training points. This is as opposed to parametric methods, for which the number of parameters is independent of n. Some examples of parametric models include linear regression, LDA, and neural networks.

Behavior in high dimensions: k-NN does not behave well in high dimensions. As the dimension increases, data points drift farther apart, so even the nearest neighbor to a point will tend to be very far away.

Theoretical properties: 1-NN has impressive theoretical guarantees for such a simple method. Cover and Hart, 1967 prove that as the number of training samples n approaches infinity, the expected prediction error for 1-NN is upper bounded by $2\epsilon^*$, where ϵ^* is the Bayes (optimal) error. Fix and Hodges, 1951 prove that as n and k approach infinity and if $\frac{k}{n} \to 0$, then the k nearest neighbor error approaches the Bayes error.

1.4 Curse of Dimensionality

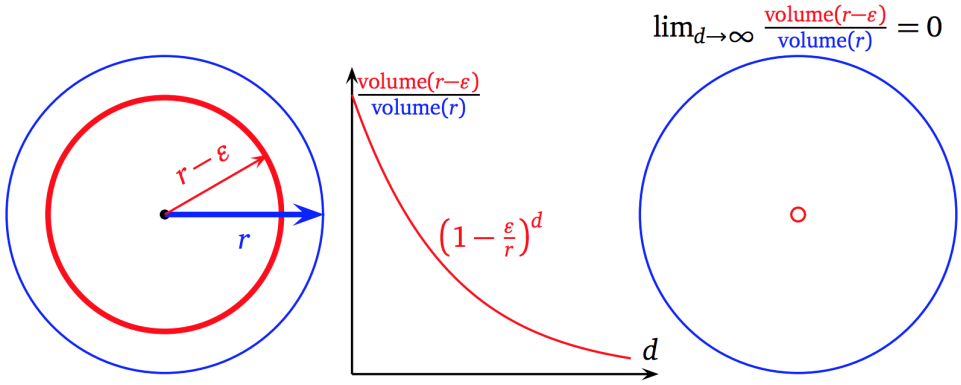
To understand why k-NN does not perform well in high-dimensional space, we first need to understand the properties of metric spaces. In high-dimensional spaces, much of our low-dimensional intuition breaks down. Here is one classical example. Consider a ball in $\mathbb{R}^d$ centered at the origin with radius r, and suppose we have another ball of radius $r - \epsilon$ centered at the origin. In low dimensions, we can visually see that much of the volume of the outer ball is also in the inner ball.

In general, the volume of the outer ball is proportional to r^d, while the volume of the inner ball is proportional to $(r - \epsilon)^d$. Thus the ratio of the volume of the inner ball to that of the outer ball is

$$\frac{(r-\epsilon)^d}{r^d} = \left(1 - \frac{\epsilon}{r}\right)^d \approx e^{-\epsilon d/r} \underset{d\to\infty}{\longrightarrow} 0$$

Hence as d gets large, most of the volume of the outer ball is concentrated in the annular region $\{x : r - \epsilon < x < r\}$ instead of the inner ball.

[2] ANN methods are beyond the scope of this course, but are useful in real applications.



High dimensions also make Gaussian distributions behave counter-intuitively. Suppose $X \sim \mathcal{N}(0, \sigma^2 I)$. If X_i are the components of X and R is the distance from X to the origin, then $R^2 = \sum_{i=1}^d X_i^2$. We have $E(R^2) = d\sigma^2$, so in expectation a random Gaussian will actually be reasonably far from the origin. If $\sigma = 1$, then R^2 is distributed chi-squared with d degrees of freedom. One can show that in high dimensions, with high probability $1 - O(e^{-d^\epsilon})$, this multivariate Gaussian will lie within the annular region $\{X : |R^2 - E(R^2)| \leq d^{1/2+\epsilon}\}$ where $E(R^2) = d\sigma^2$ (one possible approach is to note that as $d \to \infty$, the chi-squared approaches a Gaussian by the CLT, and use a Chernoff bound to show exponential decay). This phenomenon is known as *concentration of measure*. Without resorting to more complicated inequalities, we can show a simple, weaker result:

Theorem: If $X_i \sim \mathcal{N}(0, \sigma^2)$, $i = 1, ..., d$ are independent and $R^2 = \sum_{i=1}^d X_i^2$, then for every $\epsilon > 0$, the following holds:

$$\lim_{d\to\infty} P(|R^2 - E(R^2)| \geq d^{\frac{1}{2}+\epsilon}) = 0$$

Thus in the limit, the squared radius is concentrated about its mean.

Proof. From the formula for the variance of a chi-squared distribution, we see that $\text{Var}(R^2) = 2d\sigma^4$. Applying a Chebyshev bound yields

$$P(|R^2 - E(R^2)| \geq d^{\frac{1}{2}+\epsilon}) \leq \frac{2d\sigma^4}{d^{1+2\epsilon}} \underset{d\to\infty}{\longrightarrow} 0$$

□

Thus a random Gaussian will lie within a thin annular region away from the origin in high dimensions with high probability, even though the mode of the Gaussian bell curve is at the origin. This illustrates the phenomenon in high dimensions where random data is spread very far apart. The k-NN classifier was conceived on the principle that nearby points should be of the same class - however, in high dimensions, even the nearest neighbors that we have to a random test point will tend to be far away, so this principle is no longer useful.

1.5 Improving k-NN

There are two main ways to improve k-NN and overcome the shortcomings we have discussed.

1. Obtain more training data.
2. Reduce the dimensionality of the features and/or pick better features. Consider other choices of distance function.

One example of reducing the dimensionality in image space is to lower the resolution of the image — while this is throwing some of the original pixel features away, we may still be able to get the same or better performance with a nearest neighbors method.

We can also modify the distance function. For example, we have a whole family of **Minkowski distances** that are induced by the L^p norms:

$$D_p(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^{d} |x_i - z_i|^p \right)^{\frac{1}{p}}$$

Without preprocessing the data, 1-NN with the L^3 distance outperforms 1-NN with L^2 on MNIST.

We can also use kernels to compute distances in a different feature space. For example, if k is a kernel with associated feature map Φ and we want to compute the Euclidean distance from $\Phi(x)$ to $\Phi(z)$, then we have

$$\begin{aligned} \|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|_2^2 &= \Phi(\mathbf{x})^\top \Phi(\mathbf{x}) - 2\Phi(\mathbf{x})^\top \Phi(\mathbf{z}) + \Phi(\mathbf{z})^\top \Phi(\mathbf{z}) \\ &= k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{z}) + k(\mathbf{z}, \mathbf{z}) \end{aligned}$$

Thus if we define $D(\mathbf{x}, \mathbf{z}) = \sqrt{k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{z}) + k(\mathbf{z}, \mathbf{z})}$, then we can perform Euclidean nearest neighbors in Φ-space without explicitly representing Φ by using the kernelized distance function D.