Today (and for the next 5 weeks) we’re focused on **supervised learning**.

This means we’re given a **training set** consisting of **inputs** and corresponding **labels**.

Machine learning - learning a program. Labels are the expected output of the correct program when given the inputs.

<table>
<thead>
<tr>
<th>Task</th>
<th>Inputs</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>object recognition</td>
<td>image</td>
<td>object category</td>
</tr>
<tr>
<td>image captioning</td>
<td>image</td>
<td>caption</td>
</tr>
<tr>
<td>document classification</td>
<td>text</td>
<td>document category</td>
</tr>
<tr>
<td>speech-to-text</td>
<td>audio waveform</td>
<td>text</td>
</tr>
</tbody>
</table>

...
Input Vectors

What an image looks like to the computer:

[Image credit: Andrej Karpathy]
Machine learning algorithms need to handle lots of types of data: images, text, audio waveforms, credit card transactions, etc.

Common strategy: represent the input as an input vector in \( \mathbb{R}^d \)

- \textbf{Representation} = mapping to another space that’s easy to manipulate
- Vectors are a great representation since we can do linear algebra!
Input Vectors

Can use raw pixels:

Images ↔ Vectors

Can do much better if you compute a vector of meaningful features.
Mathematically, our training set consists of a collection of pairs of an input vector $\mathbf{x} \in \mathbb{R}^d$ and its corresponding target, or label, $t$

- **Regression**: $t$ is a real number (e.g. stock price)
- **Classification**: $t$ is an element of a discrete set $\{1, \ldots, C\}$
- These days, $t$ is often a highly structured object (e.g. image)

Denote the training set $\{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$

- Note: these superscripts have nothing to do with exponentiation!
Nearest Neighbours

- Suppose we’re given a novel input vector \( \mathbf{x} \) we’d like to classify.
- The idea: find the nearest input vector to \( \mathbf{x} \) in the training set and copy its label.
- Can formalize “nearest” in terms of Euclidean distance

\[
||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}
\]

**Algorithm:**

1. Find example \((\mathbf{x}^*, t^*)\) (from the stored training set) closest to \( \mathbf{x} \). That is:

\[
\mathbf{x}^* = \arg\min_{\mathbf{x}^{(i)} \in \text{train. set}} \text{dist}(\mathbf{x}^{(i)}, \mathbf{x})
\]

2. Output \( y = t^* \)

- Note: we don’t need to compute the square root. Why?
We can visualize the behavior in the classification setting using a **Voronoi diagram**.
**Decision boundary**: the boundary between regions of input space assigned to different categories.
Example: 3D decision boundary
k-Nearest Neighbours

Nearest Neighbours **sensitive to noise or mis-labeled data** ("class noise"). Solution?

- Smooth by having k nearest Neighbours vote

**Algorithm (kNN):**

1. Find k examples \( \{x^{(i)}, t^{(i)}\} \) closest to the test instance \( x \)
2. Classification output is majority class

\[
y = \arg\max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})
\]
K-Nearest Neighbours

$k=1$

[Image credit: "The Elements of Statistical Learning"]
K-Nearest Neighbours

k=15

[Image credit: "The Elements of Statistical Learning"]
k-Nearest Neighbours

Tradeoffs in choosing $k$?

- **Small $k$**
  - Good at capturing fine-grained patterns
  - May **overfit**, i.e. be sensitive to random idiosyncrasies in the training data

- **Large $k$**
  - Makes stable predictions by averaging over lots of examples
  - May **underfit**, i.e. fail to capture important regularities

- **Rule of thumb**: $k < \sqrt{n}$, where $n$ is the number of training examples
K-Nearest Neighbours

- We would like our algorithm to **generalize** to data it hasn’t before.
- We can measure the **generalization error** (error rate on new examples) using a **test set**.

[Image credit: "The Elements of Statistical Learning"]
Validation and Test Sets

- $k$ is an example of a **hyperparameter**, something we can’t fit as part of the learning algorithm itself.

- We can tune hyperparameters using a **validation set**:

  - Train with $k = 1$: $\text{err} = 7.3$ (✗)
  - Train with $k = 3$: $\text{err} = 1.1$ → $\text{test err} = 1.2$ (✓)
  - Train with $k = 10$: $\text{err} = 10.5$ (✗)

- The test set is used only at the very end, to measure the generalization performance of the final configuration.
The Curse of Dimensionality

- Low-dimensional visualizations are misleading! In high dimensions, “most” points are far apart.
- If we want the nearest Neighbour to be closer than $\epsilon$, how many points do we need to guarantee it?
- The volume of a single ball of radius $\epsilon$ is $O(\epsilon^d)$
- The total volume of $[0, 1]^d$ is 1.
- Therefore $O\left((\frac{1}{\epsilon})^d\right)$ balls are needed to cover the volume.

[Image credit: "The Elements of Statistical Learning"]
The Curse of Dimensionality

- In high dimensions, “most” points are approximately the same distance.

- Saving grace: some datasets (e.g. images) may have low intrinsic dimension, i.e. lie on or near a low-dimensional manifold. So nearest Neighbours sometimes still works in high dimensions.
Normalization

- Nearest Neighbours can be sensitive to the ranges of different features.
- Often, the units are arbitrary:

Simple fix: **normalize** each dimension to be zero mean and unit variance. I.e., compute the mean $\mu_j$ and standard deviation $\sigma_j$, and take

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

- Caution: depending on the problem, the scale might be important!
Computational Cost

- Number of computations at **training time**: 0

- Number of computations at **test time**, per query (naïve algorithm)
  - Calculate \(D\)-dimensional Euclidean distances with \(N\) data points: \(O(ND)\)
  - Sort the distances: \(O(N \log N)\)

- This must be done for each query, which is very expensive by the standards of a learning algorithm!

- Need to store the entire dataset in memory!

- Tons of work has gone into algorithms and data structures for efficient nearest Neighbours with high dimensions and/or large datasets.
Example: Digit Classification

- Decent performance when lots of data

Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples

<table>
<thead>
<tr>
<th>Test Error Rate (%)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear classifier (1-layer NN)</td>
<td>12.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean</td>
<td>5.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean, deskewed</td>
<td>2.4</td>
</tr>
<tr>
<td>K-NN, Tangent Distance, 16x16</td>
<td>1.1</td>
</tr>
<tr>
<td>K-NN, shape context matching</td>
<td>0.67</td>
</tr>
<tr>
<td>1000 RBF + linear classifier</td>
<td>3.6</td>
</tr>
<tr>
<td>SVM deg 4 polynomial</td>
<td>1.1</td>
</tr>
<tr>
<td>2-layer NN, 300 hidden units</td>
<td>4.7</td>
</tr>
<tr>
<td>2-layer NN, 300 HU, [deskewing]</td>
<td>1.6</td>
</tr>
<tr>
<td>LeNet-5, [distortions]</td>
<td>0.8</td>
</tr>
<tr>
<td>Boosted LeNet-4, [distortions]</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Example: Digit Classification

- KNN can perform a lot better with a good similarity measure.
- Example: shape contexts for object recognition. In order to achieve invariance to image transformations, they tried to warp one image to match the other image.
  - Distance measure: average distance between corresponding points on warped images
- Achieved 0.63% error on MNIST, compared with 3% for Euclidean KNN.
- Competitive with conv nets at the time, but required careful engineering.

[Belongie, Malik, and Puzicha, 2002. Shape matching and object recognition using shape contexts.]
80 Million Tiny Images was the first extremely large image dataset. It consisted of color images scaled down to $32 \times 32$.

With a large dataset, you can find much better semantic matches, and KNN can do some surprising things.

Note: this required a carefully chosen similarity metric.

[Torralba, Fergus, and Freeman, 2007. 80 Million Tiny Images.]
Example: 80 Million Tiny Images

Torralba, Fergus, and Freeman, 2007. 80 Million Tiny Images.
Conclusions

- Simple algorithm that does all its work at test time — in a sense, no learning!
- Can control the complexity by varying $k$
- Suffers from the Curse of Dimensionality
- Next time: decision trees, another approach to regression and classification