CSC 411: Introduction to Machine Learning

Lecture 2: Nearest Neighbours

Mengye Ren and Matthew MacKay

University of Toronto

Introduction

- 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.

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

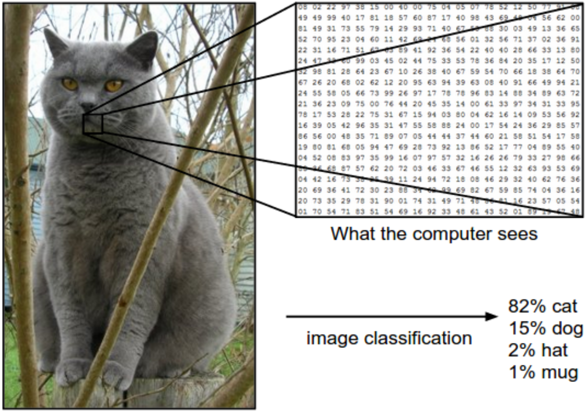
- Goal: correctly predict labels for data not in the training set ("in the wild") i.e. our ML algorithm must **generalize**

Input Vectors

- 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 $\mathbb{R}^d$
  - **Representation** = mapping to another space that's easy to manipulate
  - Vectors are a great representation since we can do linear algebra!

Input Vectors

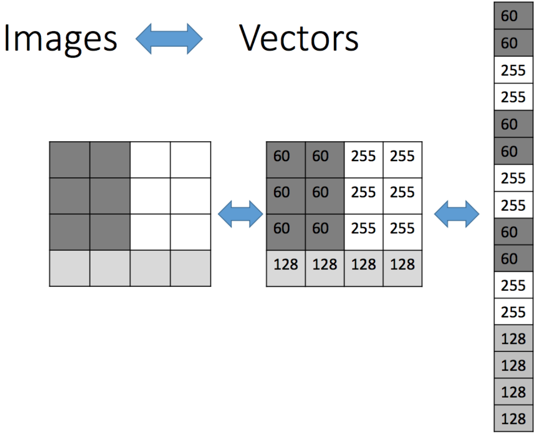
What an image looks like to the computer:



[Image credit: Andrej Karpathy]

Input Vectors

Can use raw pixels:



Can do much better if you compute a vector of meaningful features.

Input Vectors

- 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

$$||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2}$$

Algorithm:

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

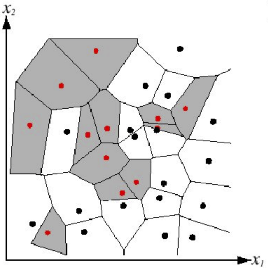
$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\operatorname{argmin}} \ \mathrm{dist}(\mathbf{x}^{(i)}, \mathbf{x})$$

2. Output $y = t^*$

- Note: we don't need to compute the square root. Why?

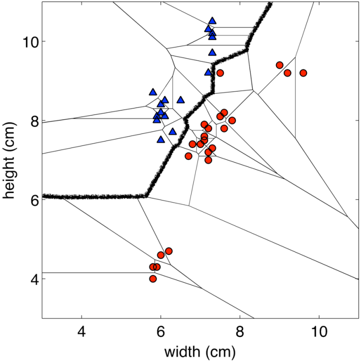
Nearest Neighbours: Decision Boundaries

We can visualize the behavior in the classification setting using a **Voronoi diagram**.

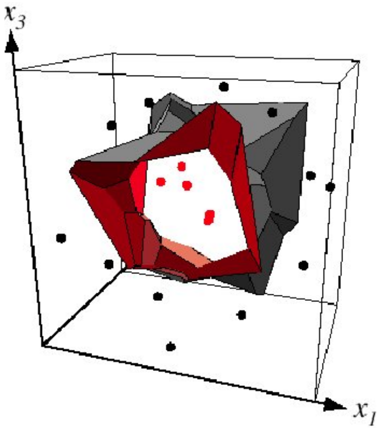


Nearest Neighbours: Decision Boundaries

Decision boundary: the boundary between regions of input space assigned to different categories.



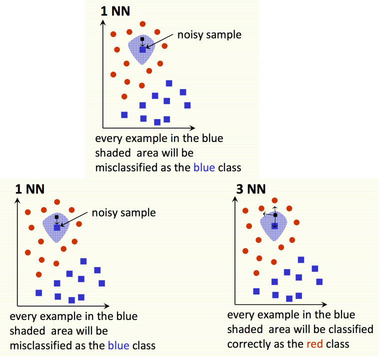
Nearest Neighbours: Decision Boundaries



Example: 3D decision boundary

k-Nearest Neighbours

[Pic by Olga Veksler]



- Nearest Neighbours **sensitive to noise or mis-labeled data** ("class noise"). Solution?
- Smooth by having k nearest Neighbours vote

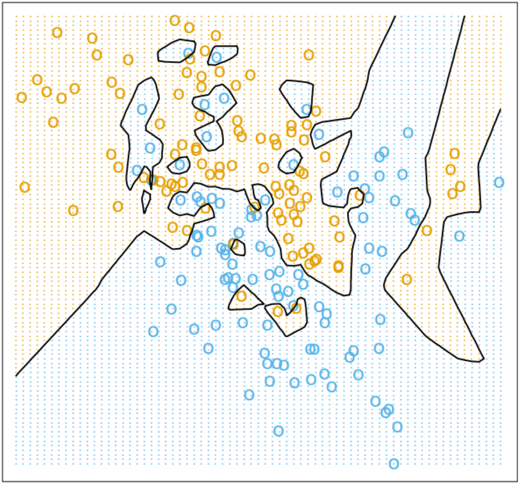
Algorithm (kNN):

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

$$y = arg\max_t \sum_{r=1}^{k} \mathbb{I}[t = t^{(r)}]$$

K-Nearest Neighbours

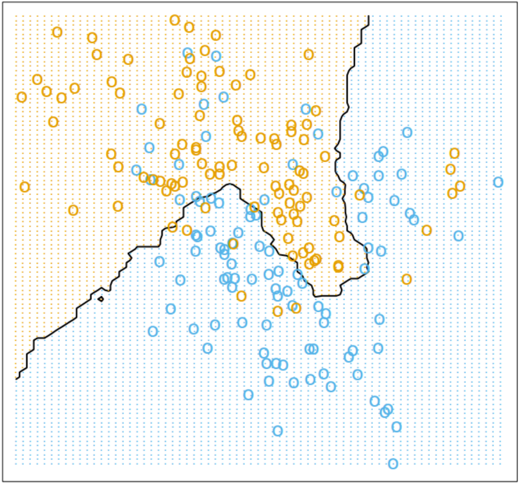
k=1



[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? Remember: goal is to correctly classify unseen examples

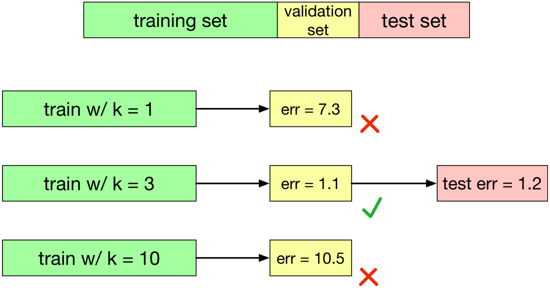
- Small k
  - Good at capturing fine-grained patterns
  - May **overfit**, i.e. be sensitive to random idiosyncrasies in the training data
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May **underfit**, i.e. fail to capture important regularities
- Rule of thumb: $k < sqrt(n)$, where n is the number of training examples

Choosing Hyperparameters using a Validation Set

- k is an example of a **hyperparameter**, something we can't fit as part of the learning algorithm itself, but which controls the behavior of the algorithm
- We want to choose hyperparameters based on how well the algorithm generalizes
- Thus, we separate some of our available data into a **validation set**, distinct from the training set
- Model's performance on the validation set indicates how well it generalizes
  - choose hyperparameters which leads to best performance (lowest error) on validation set
  - Note: error here means number of incorrectly classified examples

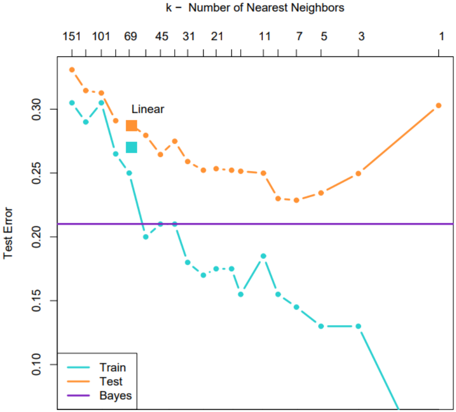
Test Set

- Now hyperparameters might have overfit to the validation set! Validation performance not good assessment of generalization of final algorithm
- Solution: separate an additional test set from the available data and evaluate on it once hyperparameters are chosen
  - Available data partitioned into 3 sets: training, validation, and test



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

K-Nearest Neighbours



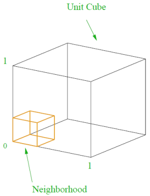
[Image credit: "The Elements of Statistical Learning"]

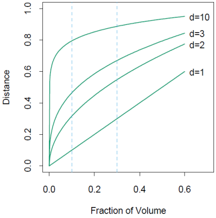
The Curse of Dimensionality

- Low-dimensional visualizations are misleading!
  - Given a new point, we want to classify it based on a point only a small distance away
  - But in high dimensions, "most" points are far apart.
- At least how many points are needed to guarantee the nearest neighbor is closer than ϵ?
  - The volume of a single ball of radius ϵ is $\mathcal{O}(\epsilon^d)$
  - The total volume of $[0,1]^d$ is 1.
  - Therefore $\mathcal{O}\left((\frac{1}{\epsilon})^d\right)$ balls are needed to cover the volume.
- Assuming data follows uniform distribution, training set size must grow exponentially with the number of dimensions for points to be close by!

The Curse of Dimensionality

- Edge length of hypercube required to occupy given fraction r of volume of unit hypercube $[0,1]^d$ is $r^{1/d}$
  - If $d = 10$ and $r = 0.1$, the edge length required is $0.1^{1/10} \approx 0.8$
  - To use 10% of the data to make our decision, must cover 80% of the range of each dimension!

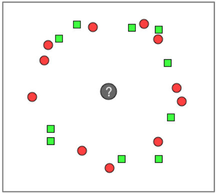




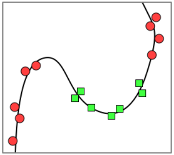
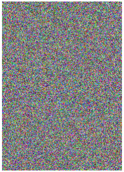
[Image credit: "The Elements of Statistical Learning"]

The Curse of Dimensionality

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

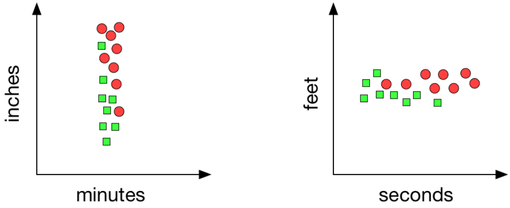


- Saving grace: some datasets (e.g. images) may have low **intrinsic dimension**, i.e. lie on or near a low-dimensional manifold. So nearest Neighbours sometimes still works in high dimensions.


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 μ_j and standard deviation σ_j, and take

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

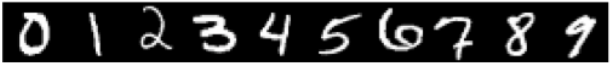
- Caution: depending on the problem, the scale might be important!

Computational Cost

- Number of computations at **training time**: 0
- Number of computations at **test time**, per query (naïve algorithm)
  - Calculuate D-dimensional Euclidean distances with N data points: $\mathcal{O}(ND)$
  - Sort the distances: $\mathcal{O}(N \log N)$
- 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

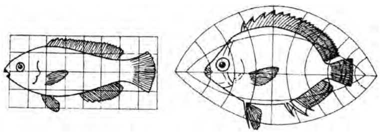


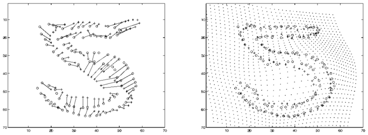
- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: d = 784
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

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

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.

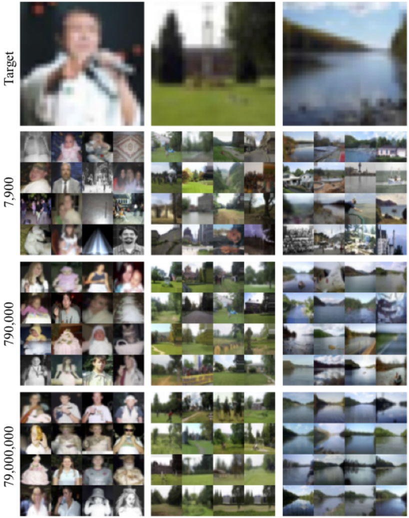




[Belongie, Malik, and Puzicha, 2002. Shape matching and object recognition using shape contexts.]

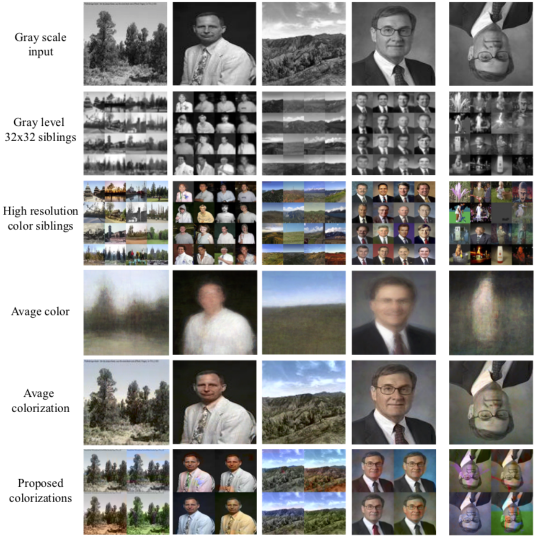
Example: 80 Million Tiny Images

- 80 Million Tiny Images was the first extremely large image dataset. It consisted of color images scaled down to 32×32.
- 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