STA 314: Statistical Methods for Machine Learning I

Lecture 1 - Introduction and Nearest Neighbours

Chris J. Maddison

University of Toronto

About me

- Assistant professor in computer science and statistics.
- Study machine learning, with a particular interest in large models, drug discovery, and medicine.
- Did my Master's here with Geoffrey Hinton, my PhD at Oxford, and I've worked at DeepMind.
- I'm masking because I got very sick from COVID-19.
- Very excited to be here!

This course

- This course is a broad introduction to machine learning.
- We cover two major learning paradigms, supervised learning and unsupervised learning.
- We cover a variety of important methods that are used by learning algorithms.
- Even though we don't cover large language models, the ideas in this course are still relevant in understanding AI today.
- Coursework is aimed at advanced undergrads. We will use multivariate calculus, probability, and linear algebra.

Do I have the appropriate background?

- Linear algebra: vector/matrix manipulations, properties.
- Calculus: partial derivatives/gradient.
- Probability: common distributions; Bayes Rule.
- Statistics: expectation, variance, covariance, median; maximum likelihood.

Do I have the appropriate background?

- We are using the Python programming language in this course.
- How much do you need to know?
  - The emphasis will be on the use of the numerical computing package NumPy (tutorial 2) and on the implementation of the key subroutines of ML methods.
  - You will **not need to write an entire Python package**. We will provide you with very complete starter code.
  - You will **need to confidently modify / complete the body of a Python function** to make the code perform the algorithm. correctly.
- Why not R?
  - I don't know R.
  - The machine learning community mostly uses Python.
  - Follow-up courses like STA414 typically use Python.

Course Information

Most information: website is main source of information; check regularly!

`https://www.cs.toronto.edu/~cmaddis/courses/sta314_f25/`

Announcements, grades, & links: Quercus.

- Did you receive the announcement?

Discussions: Piazza.

- Sign up: `https://piazza.com/utoronto.ca/fall2025/sta314`
- Your grade does not depend on your participation on Piazza. It's a good way for asking questions, discussing with the course community. Only discuss course materials/assignments/exams, but **do not give homework hints**.

Course Information

Delivery instructions:

- All lectures and office hours are in-person.
  - **Check ACORN for lecture and office hour time and place information.**
  - Office hours are held during lecture hours.
- Tutorials are in-person.
  - **Check ACORN for tutorial time and place information.**
  - Some weeks will not have tutorials. The course schedule on the website has a preliminary schedule for tutorials, and we will make announcements if there are any changes.
- Office hours are **not mandatory** and you can attend any office hour, regardless of which section you're enrolled in.
- You must attend the lecture and tutorial sections that you're enrolled in.

Marking

- (30%) 4 assignments
  - Combination of pen & paper derivations and light-weight programming exercises.
  - Weighted equally.
  - Hand-in on MarkUs.
- (30%) 1-hour midterm held during normal class time.
  - See website for times and dates.
  - Taken in-person.
  - You must attend the midterm with your section.
- (40%) FAS-proctored final exam held during exam period.
  - Taken in-person.

Course Information

- Lectures will be recorded for asynchronous viewing by enrolled students. You should be able to find this through the OCCS Student App on Quercus.
- You may download recorded lectures for your own academic use, but you should not copy, share, or use them for any other purpose.
- In case of illness, you should fill out the absence declaration form on ACORN and notify the instructors to request special consideration.
- For accessibility services: If you require additional academic accommodations, please contact UofT Accessibility Services as soon as possible, `studentlife.utoronto.ca/as`.
- Let's review the syllabus: `https://www.cs.toronto.edu/~cmaddis/courses/sta314_f25/sta314_f25_syllabus.pdf`.

Course Information

Check Quercus and website regularly.

Suggested Readings

Suggested readings will be given for each lecture. These are **completely optional**, but useful. The following will be useful throughout the course:

- Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning*.
- Christopher Bishop. *Pattern Recognition and Machine Learning*.
- Kevin Murphy. *Machine Learning: a Probabilistic Perspective*.
- David MacKay. *Information Theory, Inference, and Learning Algorithms*.
- Shai Shalev-Shwartz & Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*.
- David Barber. *Bayesian Reasoning and Machine Learning*.

There are lots of freely available, high-quality ML resources.

What is the difference between this course and CSC311?

- STA314 and CSC311 content depends somewhat on the instructor.
- Here are major differences between CSC311 and how I teach STA314:
    - I am not planning to cover neural networks nor reinforcement learning.
    - I am planning to cover the probabilistic interpretation in a bit more detail.
    - The emphasis in the homeworks is more on proofs and less on coding.
- In other words, **STA314 takes a more statistical perspective than CSC311** while covering the same core of material.

Advanced Courses

This course will help prepare you for the following courses.

- **STA414** (Statistical Methods for Machine Learning II)
    - This course is the follow-up course, which delves deeper into the probabilistic interpretation of machine learning that we cover in the last few weeks.
- **CSC413** (Neural Networks and Deep Learning)
    - This course covers deep learning and automatic differentiation.
- **CSC412** (Probabilistic Learning and Reasoning)
    - The CSC analogue of STA414.

Questions?

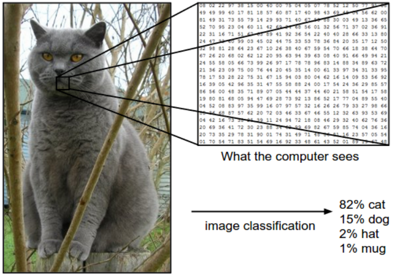
What is machine learning?

- For many problems, it's difficult to program the correct behavior by hand, e.g.,
  - recognizing people and objects,
  - understanding human speech.
- Machine learning approach: program an algorithm that itself automatically learns from data, or from experience.
- Our focus will be on the formalization of machine learning as the problem of learning good predictors and good generative models.
- Today, we will focus on predictors.

Data begins with real-world measurements

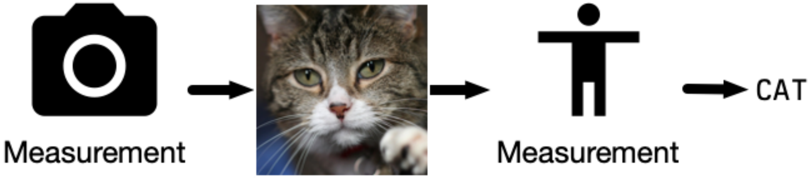
- A measurement is an action that determines a property of a system.
  - E.g., silver halide crystals in film reducing to metallic silver determine light intensity.
- Stored measurements are called data.
- In computers, we store data in a digital representation, i.e., a list of integers.

What an image looks like to the computer:



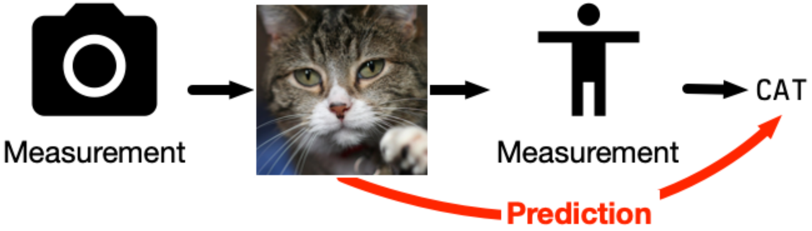
[Image credit: Andrej Karpathy]

Data relate to each other



- Data relate to each other because they store measurements from the same underlying world.
- Taking a picture is a measurement.
- Asking a human to classify an image is a measurement.

Predictions allow us to skip measurement



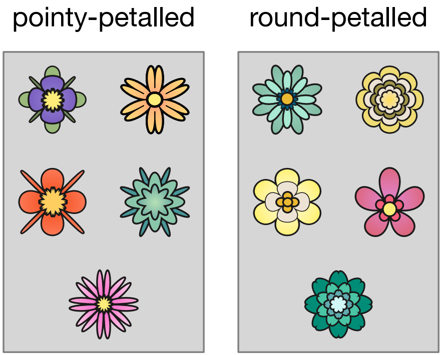
- If the measurement is cheap, there's no need for machine learning.
- If we can already make perfect predictions, as in many areas of physics, there's no need for machine learning.

Why use machine learning for prediction?

- For many important prediction problems, we don't know of a simple solution.
- Machine learning studies algorithms that construct predictors from examples.
- Let's do a very toy example to get a feeling for what this looks like.
  - This is an example of classification, which we'll learn more about.

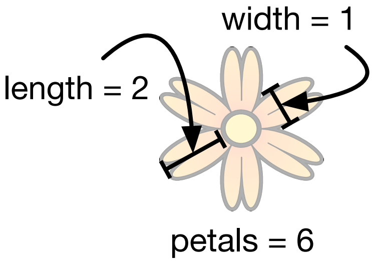
A toy classification example

- Let's say we want to classify flowers as pointy or round.
- We have a dataset of 10 flowers, each determined to be pointy or round by a human.
- To classify a new flower, we want an algorithm that doesn't rely on humans that nevertheless predicts the human determination.



A toy classification example

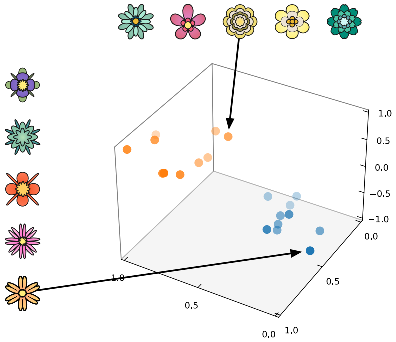
- First, we measure the representation of each flower.
- E.g., for flowers in our toy case:
  - the number of petals
  - the length of the longest petal
  - the width of the narrowest petal



representation = (6, 2, 1)

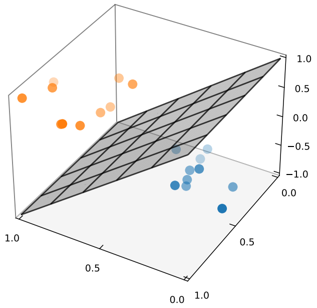
A toy classification example

- A flower has a coordinate now, which we some times call its representation.
- Each flower also has a label associated with it, which is either pointy or round.



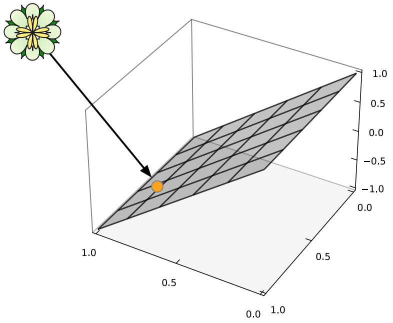
A toy classification example

- What label should we predict for a new, unseen flower?
- We can use mathematics to find a hyperplane that separates our current data.
    - The surface that separates the orange dots from the blue dots.



A toy classification example

- The hyperplane is our prediction algorithm! For a new flower:
  - measure its representation,
  - determine which side of the hyperplane its on,
  - and read out the label!
- That's machine learning in a nutshell!



Relations to statistics

- It's similar to statistics...
  - Both fields try to uncover patterns in data
  - Both fields draw heavily on calculus, probability, and linear algebra, and share many of the same core algorithms
- it's not *exactly* statistics...
  - Stats is more concerned with helping scientists and policymakers draw rigorous conclusions about data
  - ML is more concerned with making predictions that are very accurate
- and the communities are somewhat different...
  - Stats puts more emphasis on interpretability and mathematical rigor
  - ML puts more emphasis on predictive performance, scalability, and autonomy
- ...but machine learning and statistics rely on similar mathematics.

Relations to AI

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)
- Classical AI does not always imply a learning based system
  - Symbolic reasoning
  - Rule based system
  - Tree search
  - etc.
- Learning based system → learned based on the data → more flexibility, good at solving pattern recognition problems.
- Nowadays, with large language models like ChatGPT, AI almost always is synonymous with machine learning.

Relations to human learning

- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems, e.g., neural networks.
- It may perform better or worse than humans.

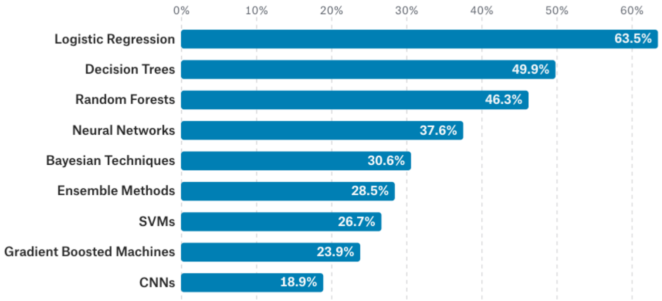
Why this class?

“I’ve heard that ChatGPT solves everything, can we just learn how to make ChatGPT?”

- The principles you learn in this course will be essential to understand and apply large language models.
- The techniques in this course are still the first things to try for a new ML problem.
  - E.g., try logistic regression before a neural network!
- One day, you may want to contribute at the forefront of AI research, and the basics you learn in this course will help.

Why this class?

2017 Kaggle survey of data science and ML practitioners: what data science methods do you use at work? We are covering 6/9 of the top methods.



Why this class?

- There are now very convenient and powerful frameworks: sklearn, PyTorch, TensorFlow, JAX, etc.
  - pre-build ML workflows
  - automatic differentiation
  - libraries of algorithms and network primitives
- Why take this class if these frameworks do so much for you?
  - So you know what to do if something goes wrong!
  - Debugging learning algorithms requires sophisticated detective work, which requires understanding what goes on beneath the hood.
  - That's why we derive things by hand in this class!

Preliminaries and Nearest Neighbor Methods

Introduction

- Today (and for much of this course) we focus on supervised learning, which is largely framed in the language of prediction.
- For many tasks of interest we are given some data and are interested in predicting something about that data.

Task	Input data	We wish to predict
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

- Supervised learning is applicable when we have many examples of good predictions, i.e., we can supervise the learner by telling it exactly what to predict.

Introduction

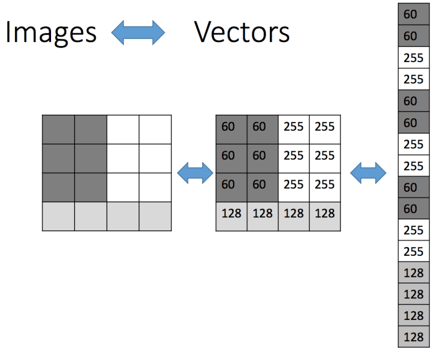
More precisely, in supervised learning, we are given

- training set consisting of
- inputs x and corresponding
- labels t.

Our goal is to predict the label or learn the mapping from $x \rightarrow t$. Let's unpack this carefully.

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!
- Can use raw pixels as representation, but sometimes you can compute more meaningful feature representations.



Labels

- You can think of labels as answers to questions about the data.
  - e.g., what is the ambient temperature in the picture?
  - e.g., what is the primary object in the image?
- We can use numbers to represent labels as well.
- Traditionally, we use different names depending on the type of label t.
  - Regression: $t \in \mathbb{R}$ is a real number, e.g., the ambient temperature
  - Classification: $t \in \{1, \ldots, C\}$ is an element of a discrete set, e.g., let 1 mean "dog", 2 mean "cat", etc.
  - Structured prediction: these days, t is often a highly structured object (e.g., image can also be a label)

Training sets

- To summarize, mathematically, our training set consists of a collection of pairs of an input $\mathbf{x} \in \mathbb{R}^d$ and its corresponding label t.
- Denote the training set $\{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$
    - Note: these superscripts have nothing to do with exponentiation!
- Our goal is to learn a mapping from $\mathbf{x}^{(i)} \to t^{(i)}$ that performs well (will be more precise about this later).

Side note: arg max and arg min

For a function f, we will often use the following notation:

$$\arg\max_{x \in C} f(x) \qquad \text{and} \qquad \arg\min_{x \in C} f(x)$$

argmax means "the arguments that maximize the function". More formally,

$$\arg\max_{x \in C} f(x) = \{x \in C : f(x) \geq f(y) \text{ for all } y \in C\}.$$

argmin is defined analogously.

Nearest Neighbors

- Suppose we're given a novel input vector $\mathbf{x}$ we'd like to classify.
- Key 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:

$$(\mathbf{x}^*, t^*) = \arg \min_{(\mathbf{x}^{(i)}, t^{(i)}) \in \text{train. set}} ||\mathbf{x}^{(i)} - \mathbf{x}||_2$$

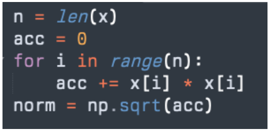
2. Output $y = t^*$

- Note: this algorithm wouldn't change if we used squared distances. Why?

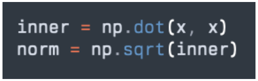
Side note: implementing machine learning

- A key part of learning machine learning involves modifying existing methods to make them fit your setting.
- This will often involve deriving an algorithm (with pencil and paper), and then translating the math into code.
- One of the key ideas in machine learning is array processing or vectorized computations, i.e., express the algorithm in terms of matrix/vector operations to exploit hardware efficiency (more in next week's tutorial on NumPy).

$$||x||$$



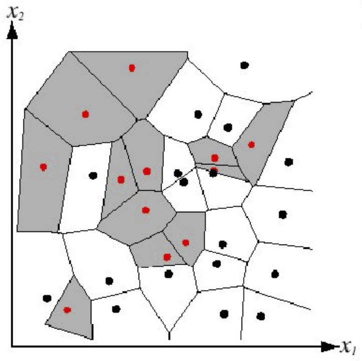
```
n = len(x)
acc = 0
for i in range(n):
    acc += x[i] * x[i]
norm = np.sqrt(acc)
```



```
inner = np.dot(x, x)
norm = np.sqrt(inner)
```

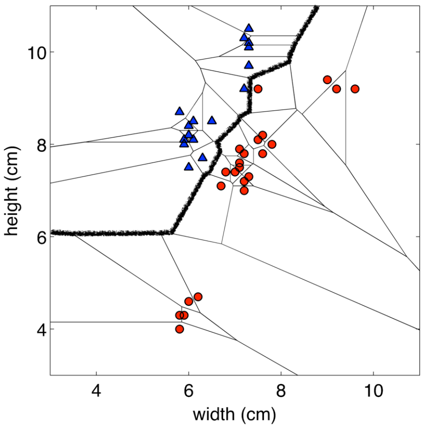
Nearest Neighbors: Decision Boundaries

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

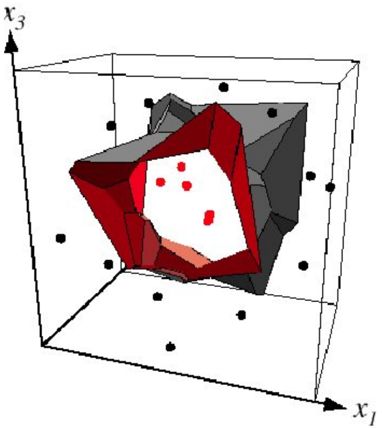


Nearest Neighbors: Decision Boundaries

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

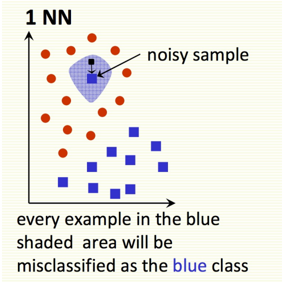


Nearest Neighbors: Decision Boundaries



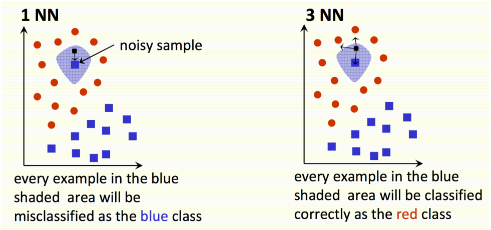
Example: 2D decision boundary

Nearest Neighbors



- Nearest neighbors sensitive to noise in the labels ("class noise").
- Solution? Smooth by having k nearest neighbors vote

k-Nearest Neighbors


1 NN
noisy sample
every example in the blue shaded area will be misclassified as the blue class
3 NN
every example in the blue shaded area will be classified correctly as the red class

Algorithm (kNN):

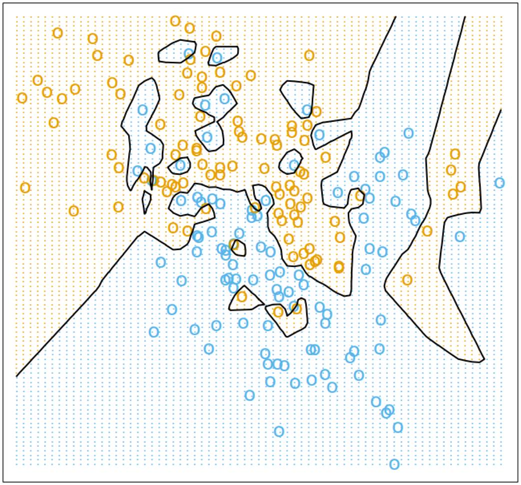
1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg \max_{t^{(z)}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

$\mathbb{I}\{\text{statement}\} = 1$ when the statement is true, and 0 otherwise.

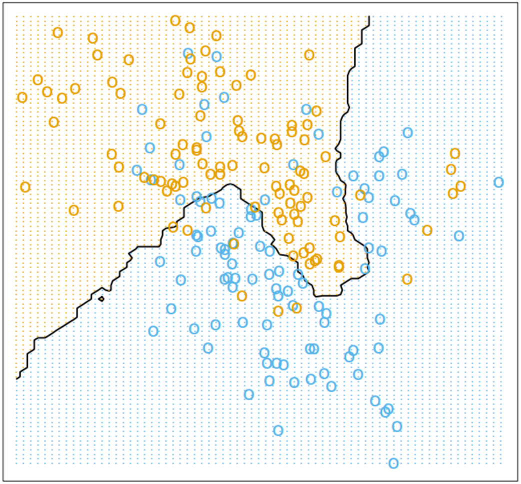
k-NN

k=1



k-NN

k=15



k-NN

Tradeoffs in choosing k?

- Small k
  - Good at capturing fine-grained patterns
  - May be sensitive to random idiosyncrasies in the training data, we call this overfitting.
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May fail to capture important regularities, we call this underfitting.
- Balancing k
  - Optimal choice of k depends on number of data points n.
  - k shouldn't increase much faster than n.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$ (ESL 2.4).
  - Rule of thumb: choose $k < \sqrt{n}$.

Side note: error rate

Let $\hat{y}_N^*(\mathbf{x})$ be the prediction of k-NN for input $\mathbf{x}$. The train error rate is

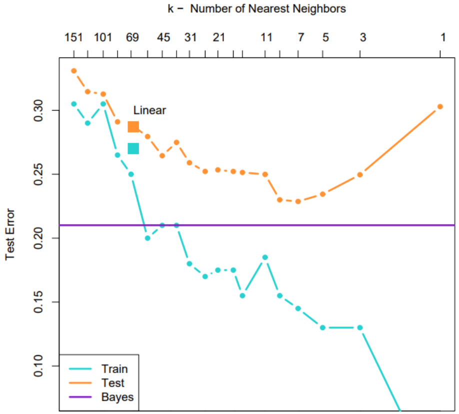
$$\frac{\sum_{(\mathbf{x}^{(i)},t^{(i)})\in \text{train. set}} \mathbb{I}(\hat{y}_N^*(\mathbf{x}^{(i)}) \neq t^{(i)})}{\sum_{(\mathbf{x}^{(i)},t^{(i)})\in \text{train. set}} 1}.$$

The test error rate is

$$\frac{\sum_{(\mathbf{x}^{(i)},t^{(i)})\in \text{test set}} \mathbb{I}(\hat{y}_N^*(\mathbf{x}^{(i)}) \neq t^{(i)})}{\sum_{(\mathbf{x}^{(i)},t^{(i)})\in \text{test set}} 1}.$$

Generalization error for k-NN

- We would like our algorithm to generalize to data it hasn't seen before.
- How can we measure the generalization error (error rate on new examples)?
- Use a new, unseen set of input-label pairs called a test set.

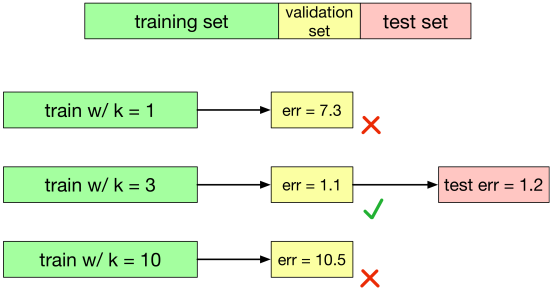


Hyperparameters

- How should we pick k?
- Maybe we can use the test set to pick k? However, once we use the test set of pick k, it is no longer an unbiased way of measuring of how well we will do on *unseen* data.
- k is an example of a hyperparameter, something we can't fit as part of the learning algorithm itself.

Validation sets

- We can tune hyperparameters using a validation set, which is a third, unseen set of input-label pairs.
- Key idea: pick the hyperparameters that perform best on the validation set. We call this model selection.



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

Pitfalls of k-NN: The Curse of Dimensionality

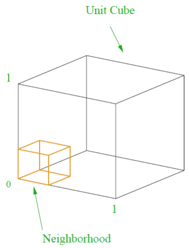
Low-dimensional visualizations are misleading!

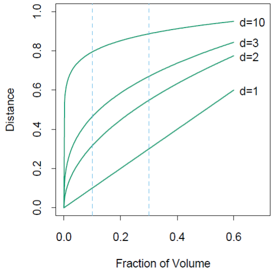
In high dimensions, “most” points are far apart.

Pitfalls of k-NN: The Curse of Dimensionality

Suppose we want we want the nearest neighbor of *every query* $x \in [0,1]^d$ to be closer than ϵ, how many points do we need in our training set to guarantee it?

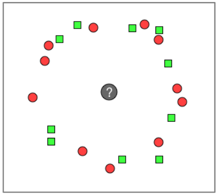
- The volume of a single ball of radius ϵ around each point is $\mathcal{O}(\epsilon^d)$
- The total volume of $[0,1]^d$ is 1.
- $\mathcal{O}\left(\left(\frac{1}{\epsilon}\right)^d\right)$ points are needed to cover the volume, i.e., increasing exponentially in d.





Pitfalls: The Curse of Dimensionality

- In high dimensions, “most” points are approximately the same distance.
- We can show this by applying the rules of expectation and covariance of random variables in surprising ways.
- Picture to keep in mind:



Pitfalls: The Curse of Dimensionality

- Saving grace: some datasets (e.g. images) may have low intrinsic dimension, i.e. lie on or near a low-dimensional manifold.

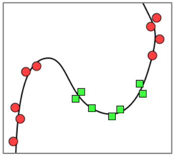
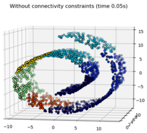
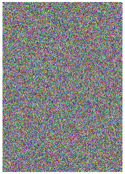



Image credit: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html`

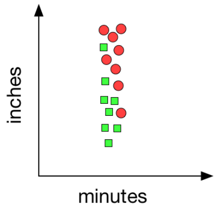
- The space of megapixel images is 3 million-dimensional. The true number of degrees of freedom is much smaller.

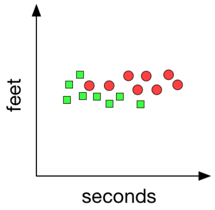


- It turns out that the performance of KNN depends on the intrinsic dimensionality, not the ambient one: `https://francisbach.com/quest-for-adaptivity/`

Pitfalls: Normalization

- Nearest neighbors 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

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

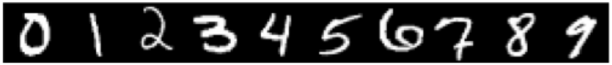
- Caution: depending on the problem, the scale might be important!

Pitfalls: 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: $\mathcal{O}(ND)$
  - 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 neighbors with high dimensions and/or large datasets.

Example: Digit Classification

- Decent performance when lots of data

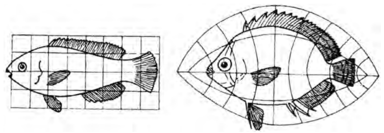


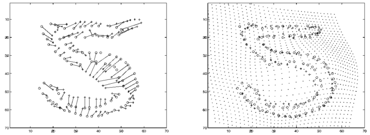
- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

	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

- Changing the similarity measure can really improve k-NN.
- 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 the state of the art at the time, but required careful engineering.





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

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: parametric models, which learn a compact summary of the data rather than referring back to it at test time.

Questions?