1 Notation | Administrative

2 KNN

# CSC 311: Introduction to Machine Learning

Lecture 1 - Introduction & Nearest Neighbors

Rahul G. Krishnan & Amanjit Singh Kainth

University of Toronto, Fall 2024

1 Formalize 2 Notation 3 Ideas $\rightarrow$ Math

4 Objective functions 5 Solve

## Outline

## Introductions

## This course

- Methematically oriented introduction to machine learning
  - Part I (until midterm): Algorithms and principles for supervised learning
  - nearest neighbors, decision trees, ensembles, linear regression, logistic regression, neural nets
  - Part II (after midterm): Algorithms and principles for unsupervised learning
  - PCA, K-means, mixture models
- Coursework is aimed at *advanced* undergrads.
- This is a third year class. You *will* be expected to teach yourself any gaps in pre-requisite material.

## Expectations on pre-requisites

You are expected to be comfortable in knowing, using and combining ideas from the following mathematical concepts.

- Linear algebra: vector operations, matrix multiplications/determinants/traces/eigenvalues)
- Calculus: partial derivatives/gradient.
- Probability: common distributions (Gaussian, Exponential, Bernoulli, Multivariate Normal); Bayes Rule.
- Statistics: expectation, variance, covariance, median; maximum likelihood.
- Numerical Optimization: maximizing functions, minimizing functions, maxima, minima; maximum likelihood.

*This year:* We will be using YouTube videos created by Lisa Zhang to help you prepare for the mathematical rigor and notation required to keep up. Please watch these videos *before* class. They are found on the course website.

## Instructor

Rahul G. Krishnan

- Undergrad in ECE at UofT
- MS in CS at NYU
- PhD in EECS at MIT
- Formerly: Senior Researcher at Microsoft Research
- Research in Deep Learning, Causal Inference, Machine Learning for Healthcare

## Instructor

Amanjit Singh Kainth

- Undergrad in Mathematics at UofT
- MS in CS at UofT
- Currently: PhD in CS at UofT
- Formerly: Applied Scientist at Amazon; …started with Speech Recognition …now at Optimization, Geometry …
- Research in Machine Learning and NLP

## Admin Details

## Marking Scheme

| **Component** | **% Final Grade** |
|---|---|
| 3 Assignments | 30%, 10% each |
| Embedded Ethics assignments | 5% |
| Project | 10% |
| Midterm Exam | 20% |
| Final Exam | 35% (40% auto-fail threshold) |

5% embedded ethics assignments:

| **Assignment** | **% Final Grade** | **Marking** |
|---|---|---|
| Pre-module survey | 1% | Full credit for submitting. |
| Class participation | 0.5% | Full credit for 90% attendance. |
| Written Reflection | 2% | Full credit for a good-faith effort. |
| Post-module survey | 1.5% | Full credit for submitting. |

## Piazza and bonus grades

- Arts and Science only gives us a restricted number of TA hours to dedicate to answer questions on Piazza. Can be many unanswered questions during peak (hw or exams) times.
- *Use the search functionality on Piazza* - Very often, the answer to your question can be found this way.
- Pilot: encourage you to help answer *conceptual* questions that your peers have (particularly around midterm/assignments).
- **Do not give answers away. Hints on where to look in course content are OK**
- Top 5 students who assist on Piazza (ranking based on metrics collected by Piazza and manual review of responses by instructors) answering questions about the course material will *each* be awarded extra grades.
- **Bonus Grades:** 2% for the top two and 1% for the next three up to a cap of 100. Will be at instructors' discretion based on quality and number of student created and endorsed responses.

## Recommended Textbooks

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

Here are some recommended textbooks.

- Deisenroth, Faisal, and Ong: Math for ML. (Useful for brushing up on pre-requisites)
- Bishop: Pattern Recognition and Machine Learning.
- Hastie, Tibshirani, and Friedman: The Elements of Statistical Learning.
- MacKay: Information Theory, Inference, and Learning Algorithms.
- Barber: Bayesian Reasoning and Machine Learning.
- Sutton and Barto: Reinforcement Learning: An Introduction.
- Shalev-Shwartz and Ben-David: Understanding Machine Learning: From Theory to Algorithms.
- Kevin Murphy: Machine Learning: a Probabilistic Perspective.

## Course Components

- Course schedule up on website (subject to some minor changes).
- Assignment pdfs will be posted on the website (and announced on quercus) tentative dates.
- Assignment and project office hours will be updated.

## Assignments

- Theoretical and programming questions in python.
- Due on MarkUs before 6pm on due date.
- Late Policy: Deadlines are firm. Each person gets three grace days they can use throughout the semester. No credit after deadline+grace days.
- Collaboration Policy: You should absolutely help each other learn the course material. Discussing strategies to solve the homework is OK but the work you submit must be your own work. Do not give out your written material or use someone else's.

## Project

- Groups of 2-3.
- 4 weeks.
- Implement and evaluate several algorithms from the course.
- Propose and evaluate an extension of one algorithm or choose your own adventure (proposal and report)!
- Will post instructions and starter code sometime around mid-semester break.

## Exams

- Conceptual questions.
- Midterm
  - Held during the week of October 16. Keep an eye out on the course website.
  - Can bring one reference sheet (double-sided). Handwritten or printed.
- Final Exam
  - 3-hour exam.
  - Date/time/location will be released around November.
  - Can bring one reference sheet (double-sided). Handwritten or printed.
  - **Do not book travel plans** until your final exam schedule is released.

## Academic Integrity

- Cheating only cheats yourself!
    - Consult U of T Code of Behaviour on Academic Matters
- What you should do for assignments:
    - Ask questions during office hours.
    - Discuss ideas and code examples with others.
    - Write code on your own.
    - Say no to sending code to others.
- What you should do for tests and exams:
    - Create practice questions.
    - Test yourself/each other under time pressure.

## Generative AI and Learning TLDR

See full policy on course website.

- You do not need to use any such tools to succeed in the class.
- Your knowledge of the concepts will be tested on the final exam and midterm exam (55% of your grade) where you will *not* have access to generative AI tools.
- Use generative AI models for learning. But, any use of GPT4/ChatGPT must be documented.
- Use GPT4/ChatGPT to help understand and personalize concepts taught in homework and lectures!
- Models can hallucinate and can fail in unpredictable ways.
- Using GPT4/ChatGPT's output directly in any material handed in for homework constitutes an academic violation. You are expected to write own homework assignments even if such tools were used as aids to learn concepts.

## Strategies for Success

- **Time Management**
    1. The best time to do something was yesterday, the next best time is today, don't wait till tomorrow.
    2. Hard skill to master but will serve you well throughout your career.
    3. Keep reviewing the Math for ML textbook. Brushing up on your pre-requisites is key to success.
- **Study groups**: Virtual or in-person, they're a great way to keep yourself and your peers accountable. Teaching your peers is a good way to make sure you understand the foundational concepts.
- **Leverage resources**: Go to TA/instructor office hours *regularly* and not just before the tests/midterms/finals. Piazza is a good place to ask questions about course material!

## Special Considerations Policy

- Missing an assessment due to extraordinary circumstances? Submit form and supporting documentation.
- Acceptable reasons:
    - Late course enrollment
    - Medical conditions: physical/mental health, hospitalizations, injury, accidents
    - Non-medical conditions (i.e., family/personal emergency)
- Unacceptable reasons: heavy course loads, multiple assignments/tests during the same period, time management issues
- Accessibility students: Accommodations are listed in Accessibility documentation

## Remark Requests

- A marking error on assignment/test.
- Submit within two weeks after marks are released.
- For assignment, submit on MarkUs.
  For midterm, fill out a form and send it to course email.

## Course Information

Course Website & Quercus: Almost Everything.
https://www.cs.toronto.edu/~rahulgk/courses/csc311_f24/index.html

Piazza: Discussions.
https://piazza.com/utoronto.ca/fall2024/csc311

MarkUs: Assignments and Project.
https://markus.teach.cs.toronto.edu/markus/courses/4

CrowdMark: Midterm results

## Getting in Touch

Piazza

- Course related and no sensitive info → public post
- Course related and sensitive info → private post
- Fixed number of TAs per week.

Course email: `ticket-csc311-2024-09@teach.cs.toronto.edu`

- Special considerations requests.
- Remark requests for midterm.
- Any other matter.
- Course-related questions will get a response through the course email and not through emailing instructors individually.

## Office Hours

Instructors' Office Hours

- Will be posted on course website
- Instructor OH will prioritize conceptual questions about the course material.

TAs will hold office hours to help with assignments and the project, as well as preparing for the midterm and final exams.

## What is Machine Learning?

## What is Machine Learning?

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech
- Machine learning approach: Uses linear algebra, probability theory, numerical optimization and statistics to create programs that automatically learn from data, or from experience
- Why might you want to use a learning algorithm?
  - hard to code up a solution by hand (e.g. vision, speech)
  - system needs to adapt to a changing environment (e.g. spam detection)
  - want the system to perform *better* than the human programmers
  - privacy/fairness (e.g. ranking search results)

## Stats vs ML

- 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 good conclusions; ML is more concerned with building autonomous agents
  - 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.

## Types of Machine Learning

- **Supervised learning:** have labeled examples of the correct behavior
- **Unsupervised learning:** no labeled examples – instead, looking for “interesting” patterns in the data
- **Reinforcement learning:** (not covered) learning system (agent) interacts with the world and learns to maximize a scalar reward signal

## Computer Vision

Object detection, semantic segmentation, pose estimation, and almost every other task is done with ML.

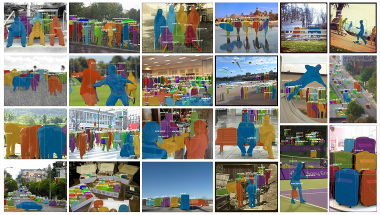
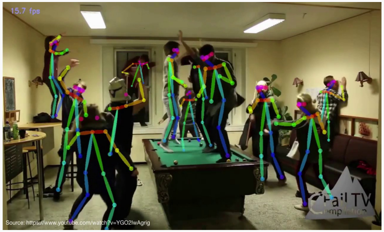
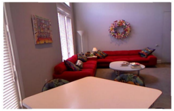


Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).



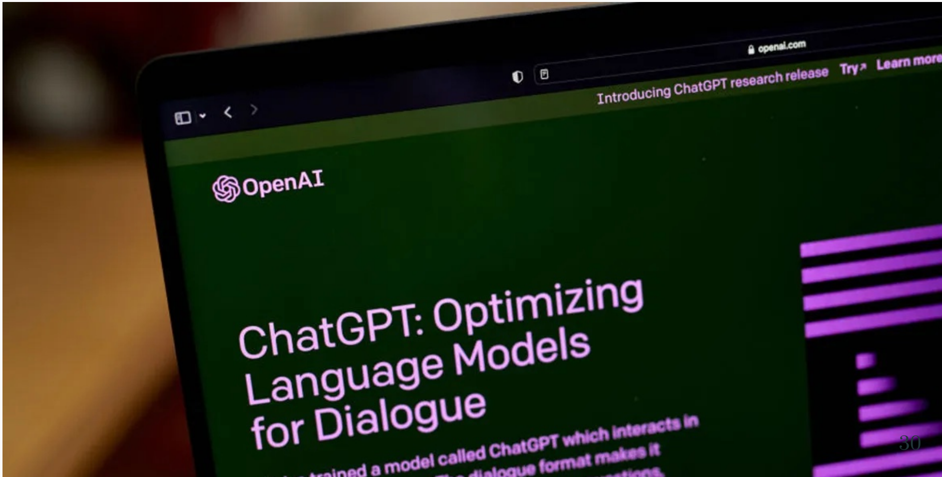


| DAQUAR 1553 | COCOQA 5078 |
|---|---|
| **What is there in front of the sofa?** | **How many leftover donuts is the red bicycle holding?** |
| Ground truth: table | Ground truth: three |
| IMG+BOW: table (0.74) | IMG+BOW: two (0.51) |
| 2-VIS+BLSTM: table (0.88) | 2-VIS+BLSTM: three (0.27) |
| LSTM: chair (0.47) | BOW: one (0.29) |

Instance segmentation - ▸ Link

## Natural Language Processing

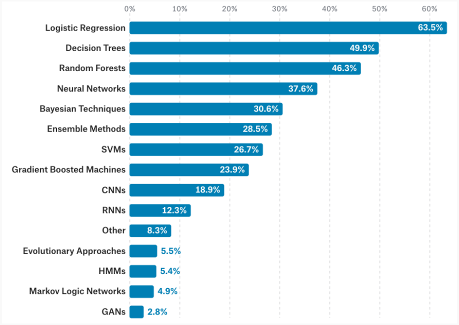
Machine translation, sentiment analysis, topic modeling, spam filtering, general purpose chatbots (ChatGPT/GPT4).


openai.com
Introducing ChatGPT research release Try↗ Learn more
OpenAI
ChatGPT: Optimizing Language Models for Dialogue
trained a model called ChatGPT which interacts in
dialogue format makes it

## Why This Class?

Why not jump straight to CSC 412/413, and learn Neural Nets first? Or use GPT4o/Claude for applying AI models directly?

- Foundations will be essential to understand and apply and build advanced tools to better understand mistakes LLMs make.
- Often the techniques in this course are the first things to try for a new ML problem.



## Programming Machine Learning Systems

- Neural net frameworks: PyTorch, TensorFlow, JAX, etc.
  - automatic differentiation
  - compiling computation graphs
  - libraries of algorithms and network primitives
  - support for graphics processing units (GPUs)
- 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!

## Implementing Machine Learning Models and Systems

Below is a categorization of ML problems that you will see time, and time-again throughout this semester.

- Step 1: Understand the problem (is it prediction, learning a good representation).
- Step 2: Formulate the problem mathematically (create notation for your inputs and outcomes and model).
- Step 3: Formulate an objective function that represents success for your model.
- Step 4: Find a strategy to solve the optimization problem on pencil and paper.
- Step 5: Translate the algorithm into code.
- Step 6: Analyze, iterate, improve design choices in your model and algorithm

We will see this taxonomy in action when we develop our very first learning algorithm.

## Nearest Neighbor Methods

## Supervised Learning

- Today (and for the first half of this course) we focus on **supervised learning**.
- This means we are given a **training set** consisting of **inputs** and corresponding **labels**, e.g.

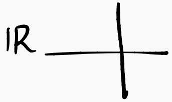
| Task | Inputs | Labels |
|---|---|---|
| • object recognition | image | object category |
| image captioning | image | caption |
| • document classification | text | document category |
| speech-to-text | audio waveform | text |
| ⋮ | ⋮ | ⋮ |

**Step 1: Our task is to build a system that can use the provided training set in order to make predictions on new data.**

## Inputs: Representing data as vectors

$x$ : Notation for input $d=2$

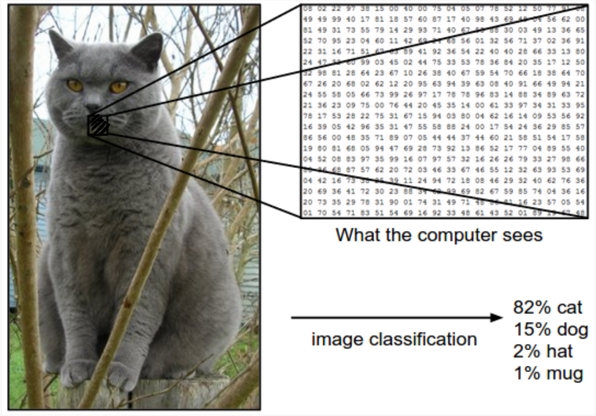
$y / t$ : Output.



- **Step 2: Create mathematical notation for the problem -** Understand how to represent data.
- Machine learning algorithms need to handle lots of types of data: images, text, audio waveforms, credit card transactions, etc.
- Represent inputs as **input vectors** in $\mathbb{R}^d$ $x \in \mathbb{R}^d$
- Vectors are a great tool since we can then use linear algebra to create transformations of inputs into desired outputs!
- **Representation** = mapping to another space that's easy to manipulate

## Example: Vector representation of an image

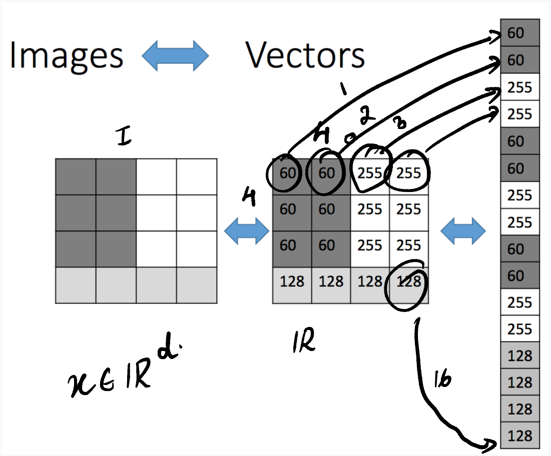
What an image looks like to the computer:



[Image credit: Andrej Karpathy]

## Example (cont'd): Vector representation of an image

Can use raw pixels:



|  |  |  |  |
|---|---|---|---|
| 60 | 60 | 255 | 255 |
| 60 | 60 | 255 | 255 |
| 60 | 60 | 255 | 255 |
| 128 | 128 | 128 | 128 |

|  |
|---|
| 60 |
| 60 |
| 255 |
| 255 |
| 60 |
| 60 |
| 255 |
| 255 |
| 60 |
| 60 |
| 255 |
| 255 |
| 128 |
| 128 |
| 128 |
| 128 |

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

## Notation: Training set

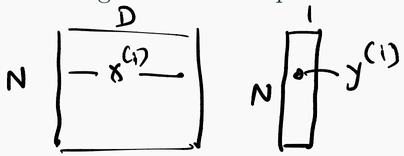
$$D = \{ (x^{(1)}, y^{(1)}) \ \ldots \ (x^{(N)}, y^{(N)}) \}$$

N elements

$(x^{(1)}, y^{(1)})$

- A training set consists of a collection of pairs of an input vector $\mathbf{x} \in \mathbb{R}^d$ *and* its corresponding **target**, or **label**, $y$
  - **Regression**: $y$ is a real number (e.g. stock price) $y \in \mathbb{R}$
  - **Classification**: $y$ is an element of a discrete set $\{1, \ldots, C\}$ $y \in \mathbb{R}^C$
  - $y$ can also be an image or graph
- Denote the training set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$
  - Superscript are used to indicate the index of the tuple in the training set. They have nothing to do with exponentiation!

$x^{(i)} \in \mathbb{R}^d \ \forall i$



## Recall: Understand problem (Step 1) and formulate notation (Step 2)

Recall our four step formulation for a machine learning problem? We just completed Step 1 and Step 2.

- Denote the training set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$.
- $\mathbf{x} \in \mathbb{R}^d$ is the $d$ dimensional vector of input data.
- $y \in \mathbb{R}^K$ ($K = 1$ if $y$ is a real number, $K = |C|$ if $y$ indicates which class $c \in C$ the label of the datapoint belongs to.

## Step 3: Crafting an optimization problem

- Suppose we're given a novel input vector $\mathbf{x}$ we'd like to classify.
- **Step 3: Crafting an optimization problem that represents our goal: supervised learning!**
- We're kicking things off with the first ML algorithm: Nearest Neighbor.

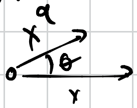
## Neighbors in vector spaces

- For each input $\mathbf{x} \in \mathbb{R}^d$, we have a corresponding label $y$.
- Our goal: For a new query input $\mathbf{x}$, we need to assign the point to a label.
- *Idea:* Set the label of the query input to be the label of the *closest* point (neighbor) in the training set.
- **Question 1**: How do we define a neighbor? — Euclidean distance.
- **Question 2**: How can we craft an optimization problem to find the nearest neighbor?

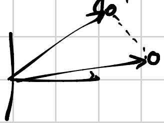
neighbour b/w $x^q$ & $x$

Dot product

$(x^q)^T x \sim \theta$



Hamming / Euclidean distance.



## Formulating an objective function to find the *Nearest Neighbors*

- **Answer 1**: Since we're working with inputs $\mathbf{x} \in \mathbb{R}^d$ we can use tools from linear algebra to formalize "neighbor" using the Euclidean distance (other choices possible).

  Euclidean dist.

$$\underline{\text{distance}(\mathbf{x}^{(a)}, \mathbf{x}^{(b)})} = \underline{||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2} = \underbrace{\sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}}$$

- **Answer 2**: Find the nearest input vector to $\mathbf{x}$ in the training set $\mathcal{D}_{tr} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$ and copy its label.

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

$$x^a \; \begin{array}{cc} j=1 & j=2 \\ \hline \boxed{5} & \boxed{-1} \end{array} \qquad x^b \; \begin{array}{cc} j=1 & j=2 \\ \hline \boxed{7} & \boxed{3} \end{array}$$

$x^a \in \mathbb{R}^2$ $\qquad$ $x^b \in \mathbb{R}^2$

$$d(x^a, x^b) = \sqrt{\underbrace{(5-7)^2}_{j=1} + \underbrace{(-1-3)^2}_{j=2}}$$

$$= \sqrt{4 + 16} = \sqrt{20}$$

## Algorithm for solving the *Nearest Neighbors* problem

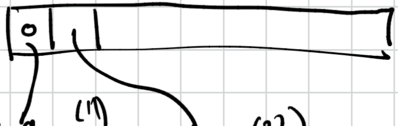
- **Algorithm**:
  1. Find example $(\mathbf{x}^n, y^n)$ (from $\mathcal{D}_{tr}$) closest to $\mathbf{x}$. That is:
  $$\cancel{\mathbf{x}}^n = \underset{i \in \{1,\ldots,n\}}{\operatorname{argmin}} \ \text{distance}(\mathbf{x}^{(i)}, \mathbf{x})$$
  2. Output $y = y^n$ .

- Note: we don't need to compute the square root. Why?
- This completes **Step 3:** Formulating an objective function that denotes success.

$\underset{x}{\text{argmin}}.\ f(x)$ $\quad$ $d(x^q, x)$

$N$ datapoints.

$dv =$



$\in \mathbb{R}^N$

$d(x^q, x^{(1)})$ $\quad$ $d(x^q, x^{(2)})$ $= \text{np.argmin}(dv)$

idx = argmin dist_vec.

return y[idx]

## Pseudocode to solve nearest neighbor

- Given query $\mathbf{x}^q \in \mathbb{R}^d$ and training dataset $|\mathcal{D}| = N$
- dv = [], midx = 0, mindist=$\infty$
- for $i = 0 \ldots, N-1$ (we'll assume zero-indexing here)
    - Calculate distance $d^i = d(x^q, x^i)$ and append to dv.
- for $i = 0 \ldots, N-1$
    - Update midx to i if dv[i]¡mindist
- return midx, dv[midx]

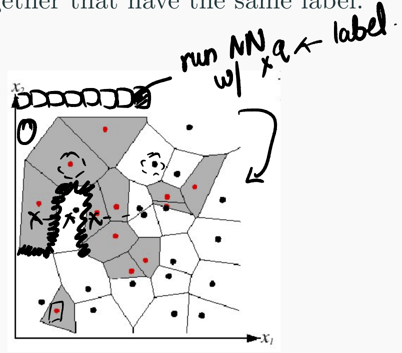
## Writing and running nearest neighbors in code

Lets walk through the algorithm from start to finish. `https://colab.research.google.com/drive/1i8nN5Syr4D7y-eDV3WXOcshWF6TQpb2n?usp=sharing` This completes **Step 5**: Translate the algorithm into code.

## Nearest Neighbors: Decision Boundaries

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

- In 2D, discretize space into tiny boxes and for each box, run the KNN algorithm to classify it.
- Merge boxes close together that have the same label.



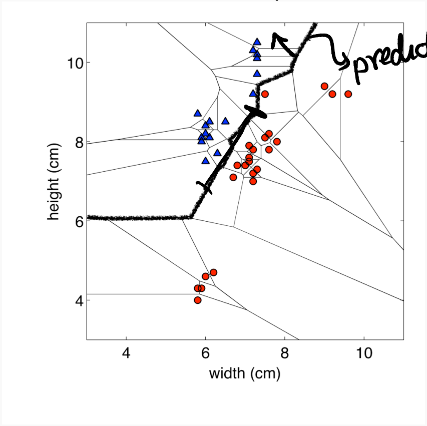
## Nearest Neighbors: Decision Boundaries

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

predict blue

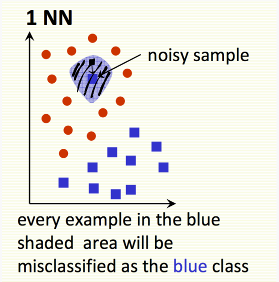
predict red.

$x^{(i)} \in \mathbb{R}^2$



## Nearest Neighbors

- **Sensitive to noise or mis-labeled data** (“class noise”). Solution?



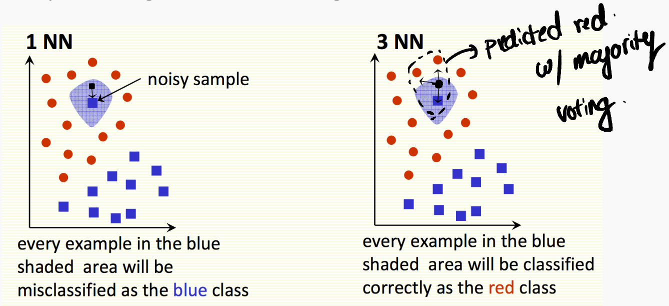
every example in the blue shaded area will be misclassified as the blue class

[Pic by Olga Veksler]

Algorithm is sensitive to label noise.

## Nearest Neighbors

- **Sensitive to noise or mis-labeled data** ("class noise"). Solution?
- Smooth by having $k$ nearest neighbors vote


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
predicted red w/ majority voting.

[Pic by Olga Veksler]

## $k$-Nearest Neighbors

- Nearest neighbors **sensitive to noise or mis-labeled data** ("class noise"). Solution?
- Smooth by having $k$ nearest neighbors vote
- Let $\mathcal{L}$ be the set of all class labels.

**Algorithm ($k$NN):**

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

$$y^* = \underbrace{\max_{t^{(z)} \in \mathcal{L}}}_{\text{Go through the class labels}} \underbrace{\sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})}_{\text{Count how often a label appears in the k-nearest neighbor set}}$$

$\mathbb{I}\{\text{statement}\}$ is the identity function and is equal to one whenever the statement is true. We could also write this as $\delta(t^{(z)}, t^{(i)})$, with $\delta(a, b) = 1$ if $a = b$, 0 otherwise.

np.argsort

$$v = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ \hline 4 & 5 & 7 & 0 & 1 \end{array}$$

np.argsort(v)

$$\text{midx} = \underbrace{\boxed{3\ 4\ 0}}_{K.}\boxed{1\ 2}$$

classification

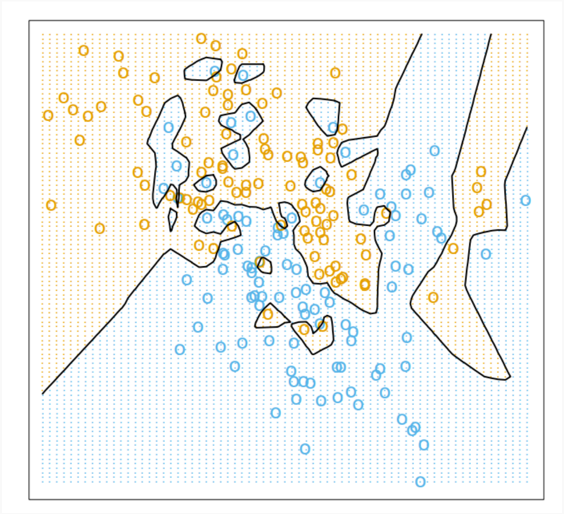
$$y^q = \text{majority}(y^3\ y^4\ y^0)$$

regression.

$$y^q = \text{avg}(y^3\ y^4\ y^0)$$

## *k*-Nearest Neighbors

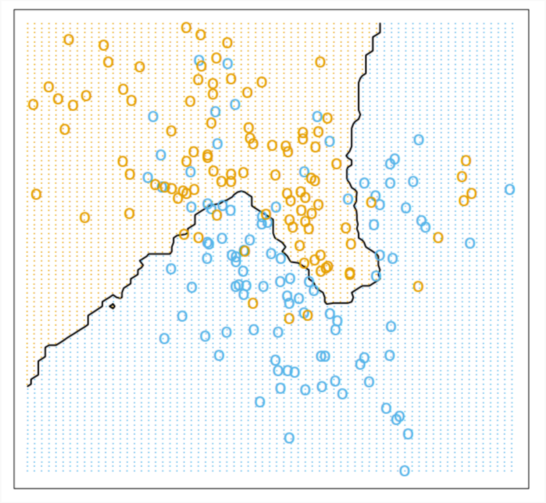
k=1



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

## $k$-Nearest Neighbors

k=15



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

## $k$-Nearest Neighbors

Tradeoffs in choosing $k$?

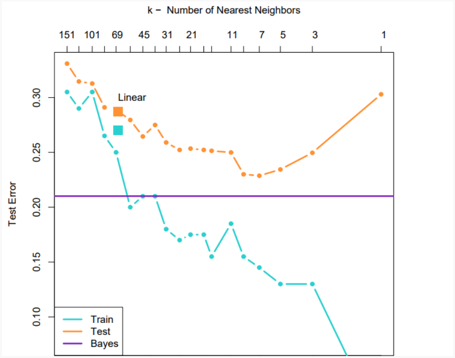
- Small $k$
  - Good at capturing fine-grained patterns
  - May **overfit**, i.e. be sensitive to random idiosyncrasies in the training data
- Large $k$
  - Makes stable predictions by averaging over lots of examples
  - May **underfit**, i.e. fail to capture important regularities
- Balancing $k$
  - Optimal choice of $k$ depends on number of data points $n$.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$.
  - Rule of thumb: choose $k < \sqrt{n}$.
  - We can choose $k$ using validation set (next slides).

## Quantifying success in $k$-Nearest Neighbors

- Previously, we used our entire dataset $\mathcal{D}$ to build a nearest neighbor algorithm.
- In reality, we want algorithms to **generalize** to new data – how do we encourage this behavior?
- *Idea:* Measure the **generalization error** (error rate on new examples) using a **test set** (data the model/algorithm has not seen before).
- Split the original data once into a **training** and **test** dataset. After finalizing the algorithm (on the training set), evaluate it on the test set.
- Use the test set performance as a proxy for how well the model will do in the future.

## Selecting hyperparameters in $k$-Nearest Neighbors

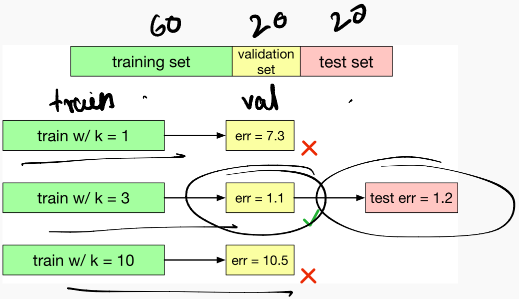
- Then how should we guide the design of hyperparameters?
- $k$ is an example of a **hyperparameter**, something we can't fit as part of the learning algorithm itself
- Selecting $k$ based on training error is a bad idea. (Why?)
- Purple line is Bayes Error (a lower-bound on error and the optimal error a model can have; more on this next week).



$$\begin{tikzcd}
 & D \arrow[dl] \arrow[d] \arrow[dr] & & \\
D_{tr} \arrow[d, bend right] & D_{val} \arrow[dr, bend left] & D_{test} \arrow[r] & \text{don't look at this.} \\
\text{use training.} & & \text{use for hyperparameters.} &
\end{tikzcd}$$

## Training, Validation and Test Sets

- We can tune hyperparameters using a **validation set**.
- Use train set to build model/run algorithm, use validation set to select hyperparameters, use test set (at the end) to quantify model generalization.
- Split the original dataset $\mathcal{D}$ into three non-overlapping sets (usually in a 60/30/10) ratio for train/valid/test.

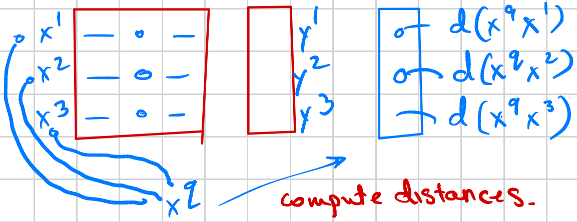


- The test set is used only at the very end, to measure the generalization performance of the final configuration.

## Recap

A HW1 out tonight. Due in 2 weeks. Start early.

B KNN - non parametric learning algorithm.

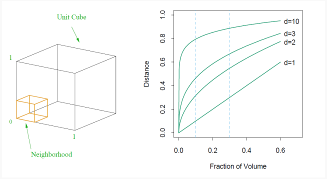


→ find indices of KNN
$idx^1 \dots idx^k$

→ return aggregation of labels
$\{y^{idx^1} \dots y^{idx^k}\}$

## Pitfalls: The Curse of Dimensionality

- Low-dimensional visualizations are misleading! In high dimensions, "most" points are far apart.
- Excercise: We want the nearest neighbor of *any* query $x$ to be closer than $\epsilon$. How many points do we in our space to guarantee it?
- Volume of a single ball of radius $\epsilon$ around a point in $\mathbb{R}^d$ is $\underline{\mathcal{O}(\epsilon^d)}$[1]
- The total volume of $[0,1]^d$ is 1. Therefore $\underline{\mathcal{O}\left(\left(\frac{1}{\epsilon}\right)^d\right)}$ points are needed to cover the volume.
- If $\epsilon = 0.1$, each increase of dimension means we need to have 10x more balls to cover the volume.

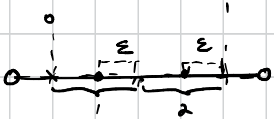


[Image credit: "The Elements of Statistical Learning"]

[1]Recall: A circle in $\mathbb{R}^2$ has volume $\pi\epsilon^2$ and a sphere has volume $\frac{4}{3}\pi\epsilon^3$

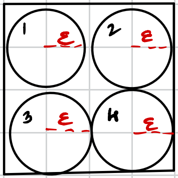
Thinking about space & packing d-dimensional spheres into d-dimensional cubes

1-dim.



Volume of 1-dim "sphere" $= O(\varepsilon)$ #spheres $= 1/\varepsilon$

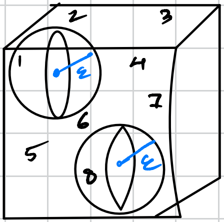
2-dim



Volume of 2-dim sphere $= 4\pi\varepsilon^2 = O(\varepsilon^2)$

#spheres $= O\left(1/\varepsilon^2\right)$

3-dim



Volume of 3-dim sphere $= 4/3\,\pi\varepsilon^3 = O(\varepsilon^3)$

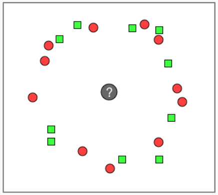
#spheres $= O\left(1/\varepsilon^3\right)$

As $d \to \infty$, I can fit exponentially more spheres $O\left(1/\varepsilon^d\right)$ in a d-dim hypercube.

- For K-NN it means I have exponentially more neighbours to choose from $\rightarrow$ less meaningful predictions

## 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. (Will show this in a homework question...)
- Picture to keep in mind:



## Pitfalls: The Curse of Dimensionality

- Nearest neighbors says that if two points are close in input space, their outputs must be close (the same).

(As $d$ increases, many more ways for $x_1$ & $x_2$ to be "similar".

- As dimension increases, all points appear equidistance – so how do we select the label for a test point?
- As dimension increases, so too do the number of irrelevant dimensions that nearest neighbor will compute distances based on.

## 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.

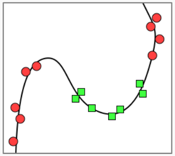
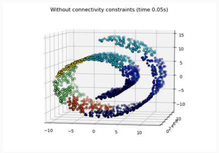
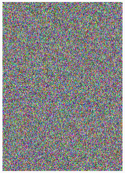



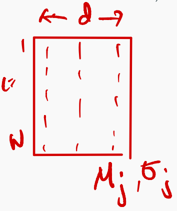
Image credit:
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html

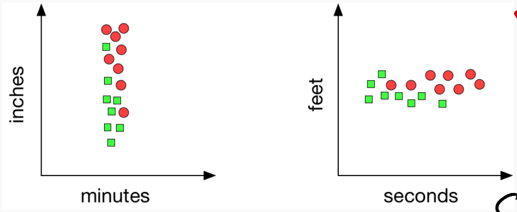
- The neighborhood structure (and hence the Curse of Dimensionality) depends on the intrinsic dimension.
- The space of megapixel images is 3 million-dimensional. The true number of degrees of freedom is much smaller.

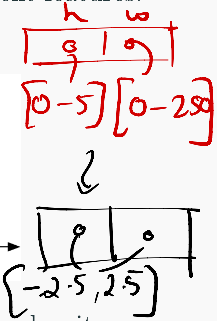


# Pitfalls: Normalization

- 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 $\mu_j$ 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!

## Pitfalls: 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 neighbors with high dimensions and/or large datasets.
- KNN is a **non-parametric** classifier (because it makes use of the data-points in order to do classification). Many of the upcoming models will be **parametric** classifiers.

## Summary - KNN

- 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.