Data and Features, Supervised Learning by Computing Distances

CS771: Introduction to Machine Learning

Piyush Rai

Announcements

- Python + NumPy refresher: Aug 5 (Saturday), 6pm, RM-101
  - Conducted by CSE PhD students Avideep and Soumya
- Also plan to have a maths refresher next week (an extra session)
- Add-drop requests: Will be cleared by noon tomorrow (Aug 4)

Data and Features

Features represent semantics of the inputs. Being able to extract good features is key to the success of ML algos

- ML algos require a numeric feature representation of the inputs

- Features can be obtained using one of the two approaches
  - Approach 1: Extracting/constructing features manually from raw inputs
  - Approach 2: Learning the features from raw inputs

- Approach 1 is what we will focus on primarily for now

- Approach 2 is what is followed in Deep Learning algorithms (will see later)

- Approach 1 is not as powerful as Approach 2 but still used widely

Example: Feature Extraction for Text Data

Other similar approaches such as TF-IDF (term frequency, inverse document frequency) are also widely used

- Consider some text data consisting of the following sentences:
  - John likes to watch movies
  - Mary likes movies too
  - John also likes football

BoW is just one of the many ways of doing feature extraction for text data. Not the most optimal one, and has various flaws (can you think of some?), but often works reasonably well

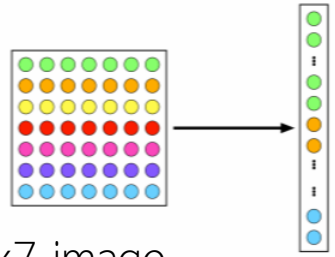
- Want to construct a feature representation for these sentences
- Here is a "bag-of-words" (BoW) feature representation of these sentences

	John	likes	to	watch	movies	Mary	too	also	football
Sentence 1	1	1	1	1	1	0	0	0	0
Sentence 2	0	1	0	0	1	1	1	0	0
Sentence 3	1	1	0	0	0	0	0	1	1

- Each sentence is now represented as a binary vector (each feature is a binary value, denoting presence or absence of a word). BoW is also called "unigram" rep.

Example: Feature Extraction for Image Data

- A very simple feature extraction approach for image data is flattening

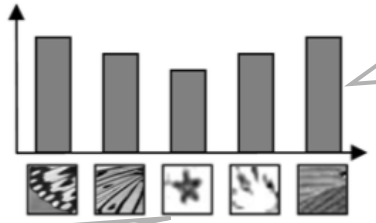


7x7 image (49 pixels)

Vector of pixel intensities

Flattening and histogram based methods destroy the spatial information in the image but often still work reasonably well

- Histogram of visual patterns is another popular feature extr. method for images



Bar heights in the histogram denote how the frequency of occurrence of each of the patterns in the given image (this vector of frequencies can be used as the extracted feature vector for this image)

Suppose these are typical patterns in the images in the dataset

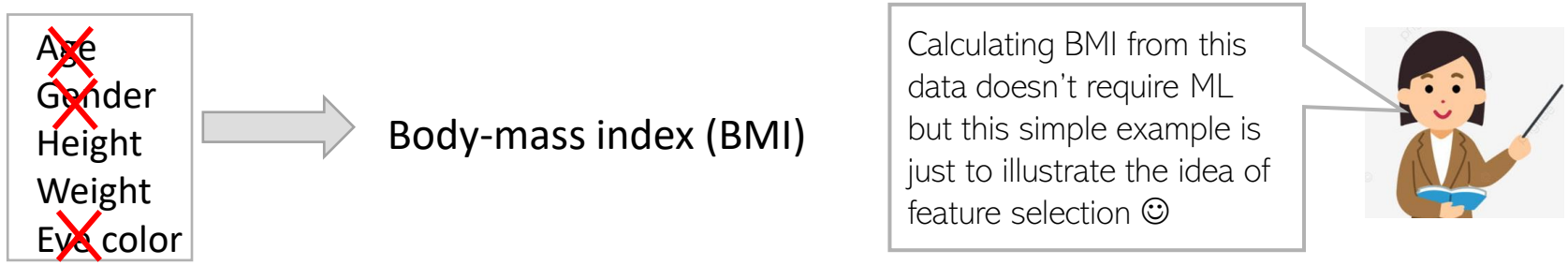
- Many other manual feature extraction techniques developed in computer vision and image processing communities (SIFT, HoG, and others)

Pic credit: cat.uab.cat/Research/object-recognition

Feature Selection

- Not all the extracted features may be relevant for learning the model (some may even confuse the learner)

- Feature selection (a step after feature extraction) can be used to identify the features that matter, and discard the others, for more effective learning

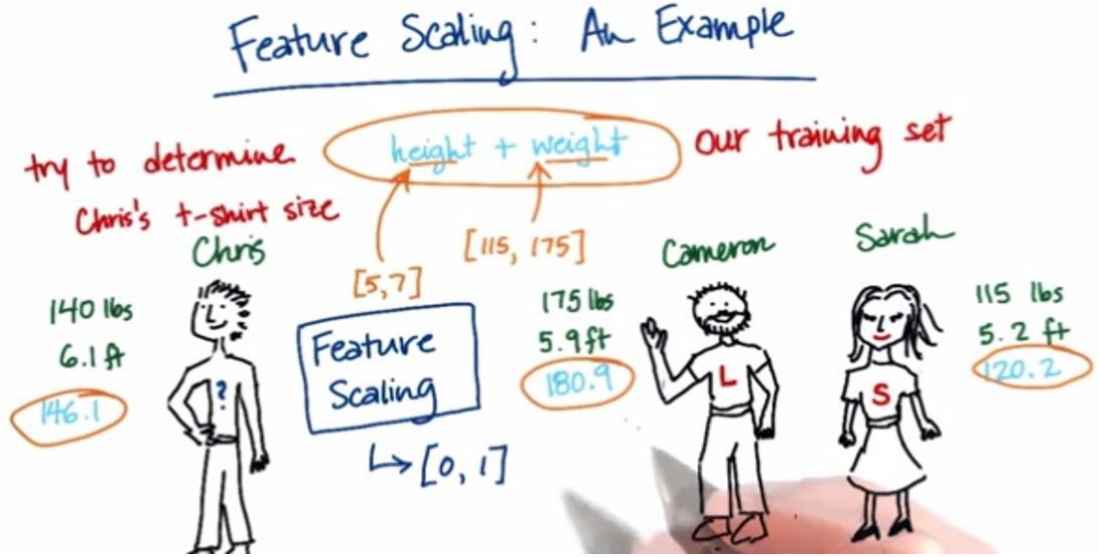


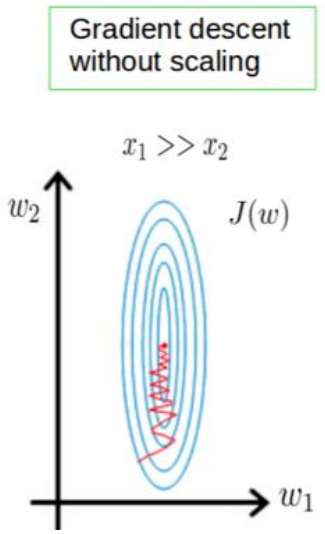
- Many techniques exist – some based on intuition, some based on algorithmic principles (will visit feature selection later)

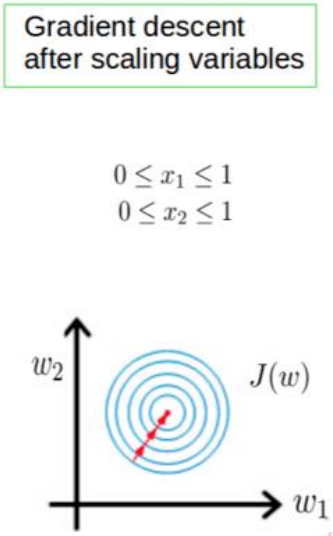
- More common in supervised learning but can also be done for unsup. learning

Some More Postprocessing: Feature Scaling

- Even after feature selection, the features may not be on the same scale
- This can be problematic when comparing two inputs – features that have larger scales may dominate the result of such comparisons
- Therefore helpful to standardize the features (e.g., by bringing all of them on the same scale such as between 0 to 1)







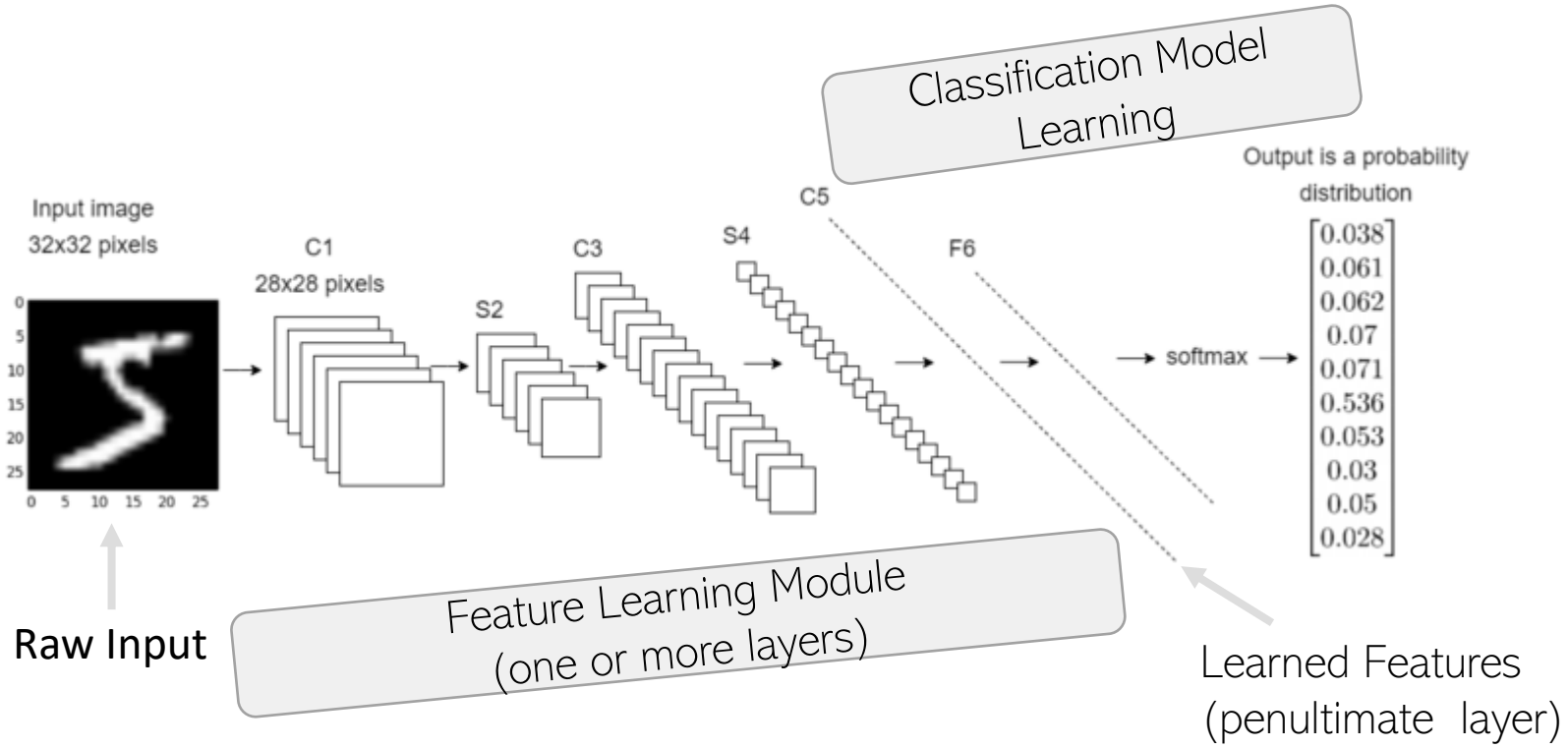
- Also helpful for stabilizing the optimization techniques used in ML algos

Pic credit: https://becominghuman.ai/demystifying-feature-scaling-baff53e9b3fd, https://stackoverflow.com/

Deep Learning: An End-to-End Approach to ML

Deep Learning = ML with automated feature learning from the raw inputs

Feature extraction part is automated via the feature learning module



Pic an adaptation of the original from: https://deepai.org/

Some Notation/Nomenclature/Convention

- Sup. learning requires training data as N input-output pairs $\{(\mathbf{x_n}, y_n)\}_{n=1}^{N}$
- Unsupervised learning requires training data as N inputs $\{\mathbf{x_n}\}_{n=1}^{N}$

RL and other flavors of ML problems also use similar notation

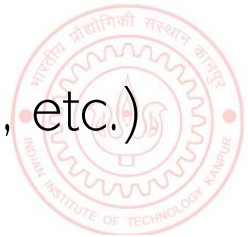
- Each input $\mathbf{x_n}$ is (usually) a vector containing the values of the features or attributes or covariates that encode properties of the it represents, e.g.,
  - For a 7 × 7 image: $\mathbf{x_n}$ can be a 49 × 1 vector of pixel intensities

Size or length of the input $\mathbf{x_n}$ is commonly known as data/input dimensionality or feature dimensionality

- (In sup. learning) Each y_n is the output or response or label associated with input $\mathbf{x_n}$ (and its value is known for the training inputs)
  - Output can be a scalar, a vector of numbers, or even an structured object (more on this later)

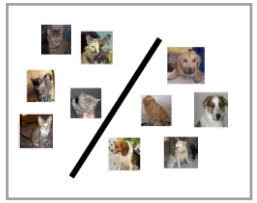
Types of Features and Types of Outputs

- Features as well as outputs can be real-valued, binary, categorical, ordinal, etc.
- Real-valued: Pixel intensity, house area, house price, rainfall amount, temperature, etc
- Binary: Male/female, adult/non-adult, or any yes/no or present/absent type value
- Categorical/Discrete: Zipcode, blood-group, or any "one from a finite many choices" value
- Ordinal: Grade (A/B/C etc.) in a course, or any other type where relative values matter
- Often, the features can be of mixed types (some real, some categorical, some ordinal, etc.)



Supervised Learning

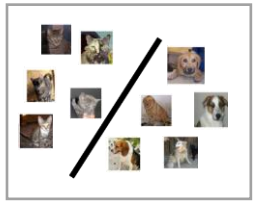
Supervised Learning Algorithm



Cat vs Dog
Prediction model

Important: In ML (not just sup. learning but also unsup. and RL), training and test datasets should be "similar" (we don't like "out-of-syllabus" questions in exams ☺)

A test image



Cat vs Dog
Prediction model

Predicted Label
(cat/dog)

In the above example, it means that we can't have test data with BnW images or sketches of cats and dogs

More formally, the train and test data distributions should be the same

Of course not. ☺ Many ML techniques exist to handle such situations (a bit advanced but will touch upon those later)

Will give you just the names for now – domain adaptation, covariate shift, transfer learning, etc

Does it mean ML is useless if this assumption is violated?

Some Types of Supervised Learning Problems

- Consider building an ML module for an e-mail client
- Some tasks that we may want this module to perform
  - Predicting whether an email of spam or normal: Binary Classification
  - Predicting which of the many folders the email should be sent to: Multi-class Classification
  - Predicting all the relevant tags for an email: Tagging or Multi-label Classification
  - Predicting what's the spam-score of an email: Regression
  - Predicting which email(s) should be shown at the top: Ranking
  - Predicting which emails are work/study-related emails: One-class Classification
- These predictive modeling tasks can be formulated as supervised learning problems
- Today: A very simple supervised learning model for binary/multi-class classification
  - This model doesn't require any fancy maths – just computing means and distances

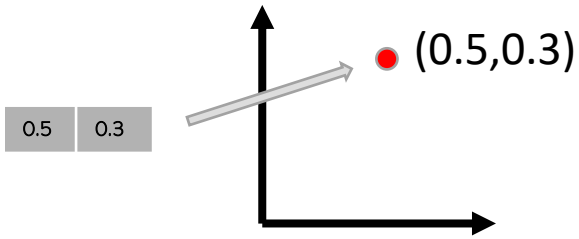
Some Notation and Conventions

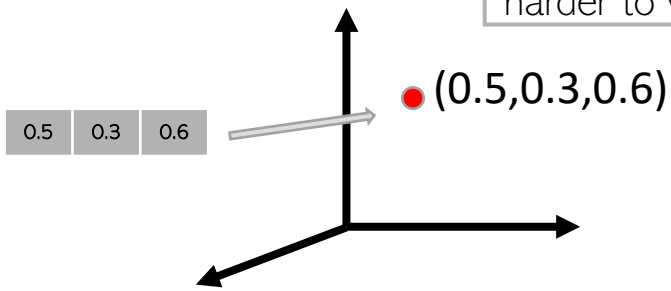
- In ML, inputs are usually represented by vectors
- A vector consists of an array of scalar values

0.5	0.3	0.6	0.1	0.2	0.5	0.9	0.2	0.1	0.5

- Geometrically, a vector is just a point in a vector space, e.g.,
  - A length 2 vector is a point in 2-dim vector space
  - A length 3 vector is a point in 3-dim vector space

Likewise for higher dimensions, even though harder to visualize





- Unless specified otherwise
  - Small letters in bold font will denote vectors, e.g., $\mathbf{x}, \mathbf{a}, \mathbf{b}$ etc.
  - Small letters in normal font to denote scalars, e.g. x, a, b, etc
  - Capital letters in bold font will denote matrices (2-dim arrays), e.g., $\mathbf{X}, \mathbf{A}, \mathbf{B}$, etc

Some Notation and Conventions

- A single vector will be assumed to be of the form $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
- Unless specified otherwise, vectors will be assumed to be column vectors
  - So we will assume $\mathbf{x} = [x_1, x_2, \ldots, x_D]$ to be a column vector of size $D \times 1$
  - Assuming each element to be real-valued scalar, $\mathbf{x} \in \mathbb{R}^{D \times 1}$ or $\mathbf{x} \in \mathbb{R}^{D}$ ($\mathbb{R}$: space of reals)
- If $\mathbf{x} = [x_1, x_2, \ldots, x_D]$ is a feature vector representing, say an image, then
  - D denotes the dimensionality of this feature vector (number of features)
  - x_i (a scalar) denotes the value of i^{th} feature in the image
- For denoting multiple vectors, we will use a subscript with each vector, e.g.,
  - N images denoted by N feature vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, or compactly as $\{\mathbf{x}_n\}_{n=1}^{N}$
  - The vector $\mathbf{x}_n$ denotes the n^{th} image
  - x_{ni} (a scalar) denotes the i^{th} feature ($i = 1,2, \ldots, D$) of the n^{th} image

Some Basic Operations on Vectors

- Addition/subtraction of two vectors gives another vector of the same size

- The mean $\boldsymbol{\mu}$ (average or centroid) of $\boldsymbol{N}$ vectors $\{\mathbf{x}_n\}_{n=1}^N$

$$\mu = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$$ (of the same size as each $\mathbf{x}_n$)

- The inner/dot product of two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top \boldsymbol{b} = \sum_{i=1}^{D} a_i b_i$$ (a real-valued number denoting how "similar" $\boldsymbol{a}$ and $\boldsymbol{b}$ are)

Assuming both $\boldsymbol{a}$ and $\boldsymbol{b}$ have unit Euclidean norm

- For a vector $\boldsymbol{a} \in \mathbb{R}^D$, its Euclidean norm is defined via its inner product with itself

$$\|\boldsymbol{a}\|_2 = \sqrt{\boldsymbol{a}^\top \boldsymbol{a}} = \sqrt{\sum_{i=1}^{d} a_i^2}$$

- Also the Euclidean distance of $\boldsymbol{a}$ from origin
- Note: Euclidean norm is also called L2 norm

Computing Distances

- Euclidean (L2 norm) distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$d_2(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_2 = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top (\boldsymbol{a} - \boldsymbol{b})} = \sqrt{\boldsymbol{a}^\top \boldsymbol{a} + \boldsymbol{b}^\top \boldsymbol{b} - 2\boldsymbol{a}^\top \boldsymbol{b}}$$

Sqrt of Inner product of the difference vector!

Another expression in terms of inner products of individual vectors

- Weighted Euclidean distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

Useful tip: Can achieve the effect of feature scaling by using weighted Euclidean distances!

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

$\mathbf{W}$ is a DxD diagonal matrix with weights w_i on its diagonals. Weights may be known or even learned from data (in ML problems)

Note: If $\mathbf{W}$ is a DxD symmetric matrix then it is called the Mahalanobis distance (more on this later)

- Absolute (L1 norm) distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

L1 norm distance is also known as the Manhattan distance or Taxicab norm (it's a very natural notion of distance between two points in some vector space)

Yes. Another, although less commonly used, distance is the L-infinity distance (equals to max of abs-value of element-wise difference between two vectors

$$d_1(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_1 = \sum_{i=1}^{D} |a_i - b_i|$$

Apart from L2 and L1. there other ways of defining distances?

Our First Supervised Learner

Prelude: A Very Primitive Classifier

The idea also applies to multi-class classification: Use one image per class, and predict label based on the distances of the test image from all such images

- Consider a binary classification problem – cat vs dog
- Assume training data with just 2 images – one  and one 
- Given a new test image (cat/dog), how do we predict its label?
- A simple idea: Predict using its distance from each of the 2 training images

$d($ Test image $,$  $) < d($ Test image $,$  $)$? Predict cat else dog

Wait. Is it ML? Seems to be like just a simple "rule". Where is the "learning" part in this?

Some possibilities: Use a feature learning/selection algorithm to extract features, and use a Mahalanobis distance where you learn the W matrix (instead of using a predefined W), using "distance metric learning" techniques

Excellent question! Glad you asked! Even this simple model can be learned. For example, for the feature extraction/selection part and/or for the distance computation part

Improving Our Primitive Classifier

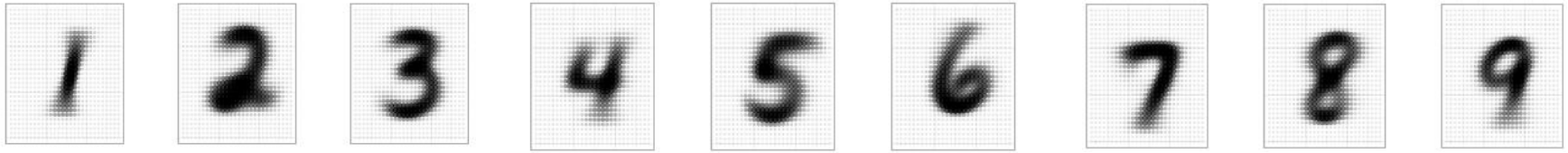
- Just one input per class may not sufficiently capture variations in a class
- A natural improvement can be by using more inputs per class

- We will consider two approaches to do this
  - Learning with Prototypes (LwP), also called "Nearest Class Mean" (NCM)
  - Nearest Neighbors (NN – not "neural networks", at least not for now ☺)
- Both LwP and NN will use multiple inputs per class but in different ways

Learning with Prototypes (LwP)

- Basic idea: Represent each class by a "prototype" vector

- Class Prototype: The "mean" or "average" of inputs from that class



Averages (prototypes) of each of the handwritten digits 1-9

- Predict label of each test input based on its distances from the class prototypes
  - Predicted label will be the class that is the closest to the test input

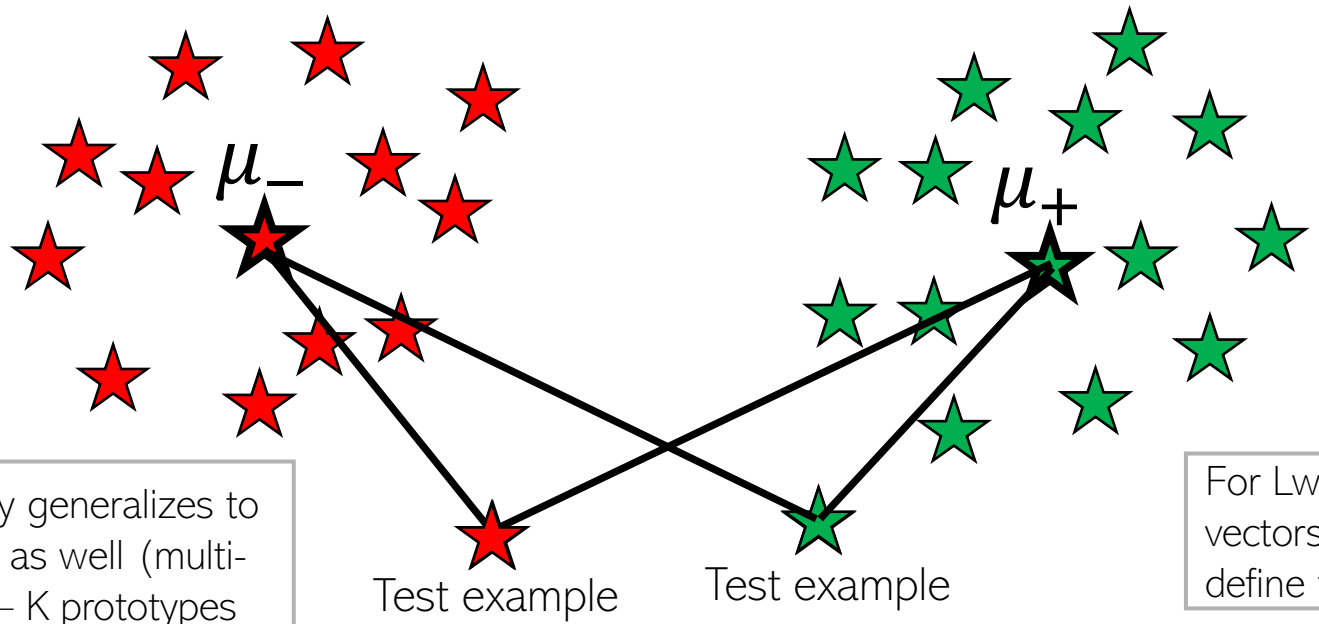
- How we compute distances can have an effect on the accuracy of this model (may need to try Euclidean, weight Euclidean, Mahalanobis, or something else)

Pic from: https://www.reddit.com/r/dataisbeautiful/comments/3wgbv9/average_handwritten_digit_oc/

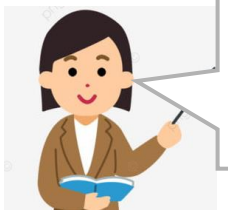
Learning with Prototypes (LwP): An Illustration

- Suppose the task is binary classification (two classes assumed pos and neg)
- Training data: N labelled examples $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, $\mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \{-1, +1\}$
  - Assume N_+ example from positive class, N_- examples from negative class
  - Assume green is positive and red is negative

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n$$



$$\mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$



LwP straightforwardly generalizes to more than 2 classes as well (multi-class classification) – K prototypes for K classes

For LwP, the prototype vectors (μ_+ and μ_- here) define the "model"

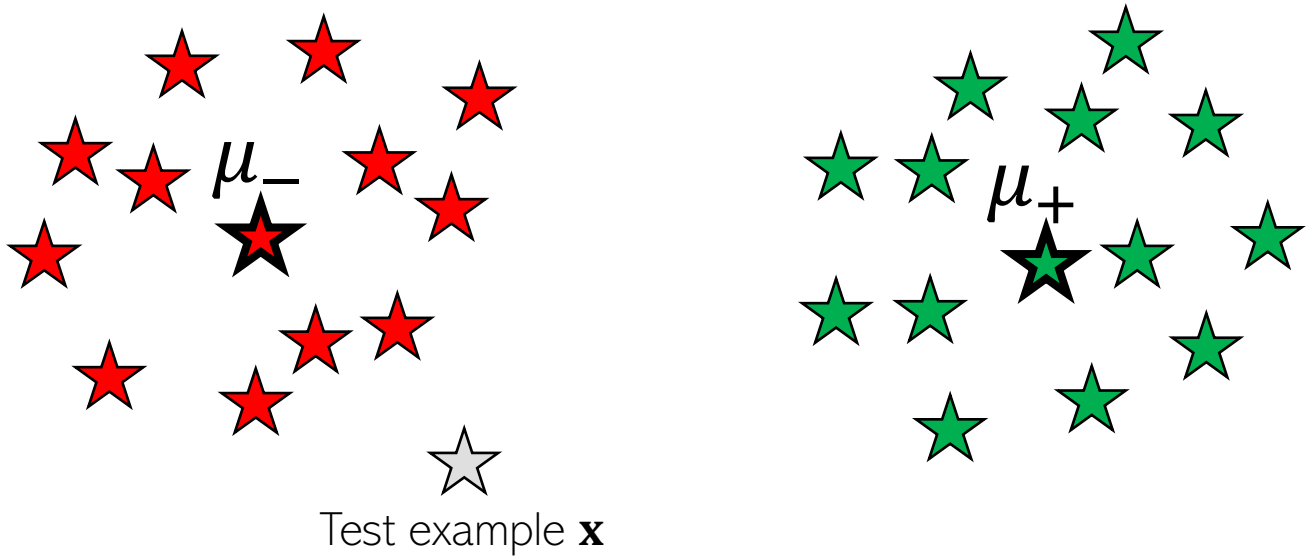
LwP: The Prediction Rule, Mathematically

- What does the prediction rule for LwP look like mathematically?

- Assume we are using Euclidean distances here

$$||\boldsymbol{\mu}_- - \mathbf{x}||^2 = ||\boldsymbol{\mu}_-||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_-, \mathbf{x} \rangle$$

$$||\boldsymbol{\mu}_+ - \mathbf{x}||^2 = ||\boldsymbol{\mu}_+||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_+, \mathbf{x} \rangle$$



Prediction Rule: Predict label as +1 if $f(\mathbf{x}) = ||\boldsymbol{\mu}_- - \mathbf{x}||^2 - ||\boldsymbol{\mu}_+ - \mathbf{x}||^2 > 0$ otherwise -1

LwP: The Prediction Rule, Mathematically

- Let's expand the prediction rule expression a bit more

$$
\begin{aligned}
f(\mathbf{x}) &= \left|\left|\boldsymbol{\mu}_- - \mathbf{x}\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+ - \mathbf{x}\right|\right|^2 \\
&= \left|\left|\boldsymbol{\mu}_-\right|\right|^2 + \left|\left|\mathbf{x}\right|\right|^2 - 2\langle\boldsymbol{\mu}_-, \mathbf{x}\rangle - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 - \left|\left|\mathbf{x}\right|\right|^2 + 2\langle\boldsymbol{\mu}_+, \mathbf{x}\rangle \\
&= 2\langle\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x}\rangle + \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 \\
&= \langle\mathbf{w}, \mathbf{x}\rangle + b
\end{aligned}
$$

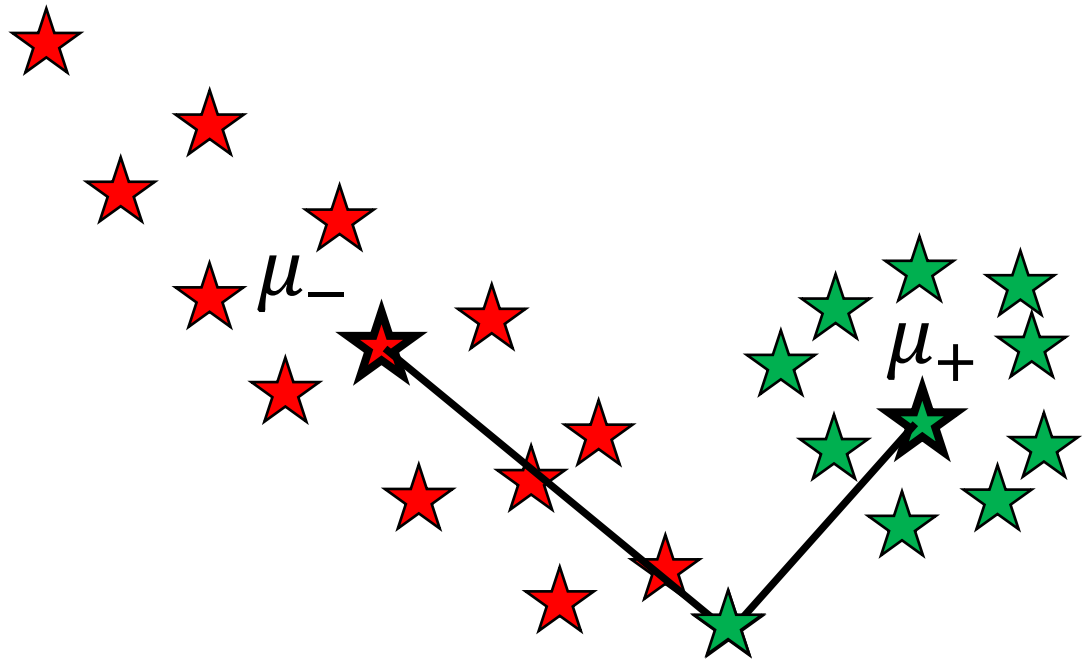
- Thus LwP with Euclidean distance is equivalent to a linear model with
  - Weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
  - Bias term $b = \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2$

Will look at linear models more formally and in more detail later

- Prediction rule therefore is: Predict +1 if $\langle\mathbf{w}, \mathbf{x}\rangle + b > 0$, else predict -1

LwP: Some Failure Cases

- Here is a case where LwP with Euclidean distance may not work well



Can use feature scaling or use Mahalanobis distance to handle such cases (will discuss this in the next lecture)

- In general, if classes are not equisized and spherical, LwP with Euclidean distance will usually not work well (but improvements possible; will discuss later)

LwP: Some Key Aspects

- Very simple, interpretable, and lightweight model
  - Just requires computing and storing the class prototype vectors
- Works with any number of classes (thus for multi-class classification as well)
- Can be generalized in various ways to improve it further, e.g.,
  - Modeling each class by a probability distribution rather than just a prototype vector
  - Using distances other than the standard Euclidean distance (e.g., Mahalanobis)
- With a learned distance function, can work very well even with very few examples from each class (used in some "few-shot learning" models nowadays – if interested, please refer to "Prototypical Networks for Few-shot Learning")

Next Class

- Nearest Neighbors
- Decision Trees and Forests/Ensembles