Lecture 13:
Intro to Machine Learning

Administrative

- My Office Hours today cancelled due to travel
- HW2 was due yesterday 2/19
- HW3 due date changed:
  - Old due date: Friday 2/28, 11:59pm
  - New due date: Wednesday 3/4, 11:59pm
  - You are <u>strongly encouraged</u> to finish the assignment before Spring Break
  - **GSIs and IAs will not be checking Piazza over Spring Break**

Last Time: Creating Panoramas

Creating Panoramas

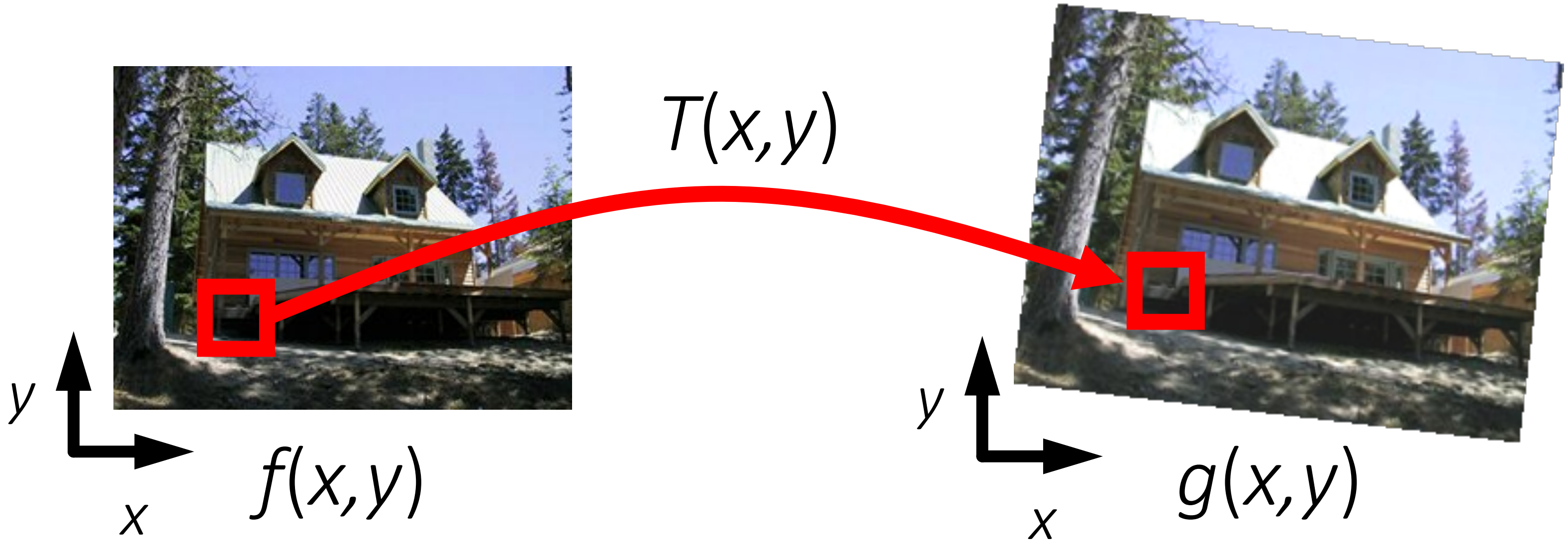
Categories of Transformations

Fitting Transformations

Applying Transformations

Blending Images

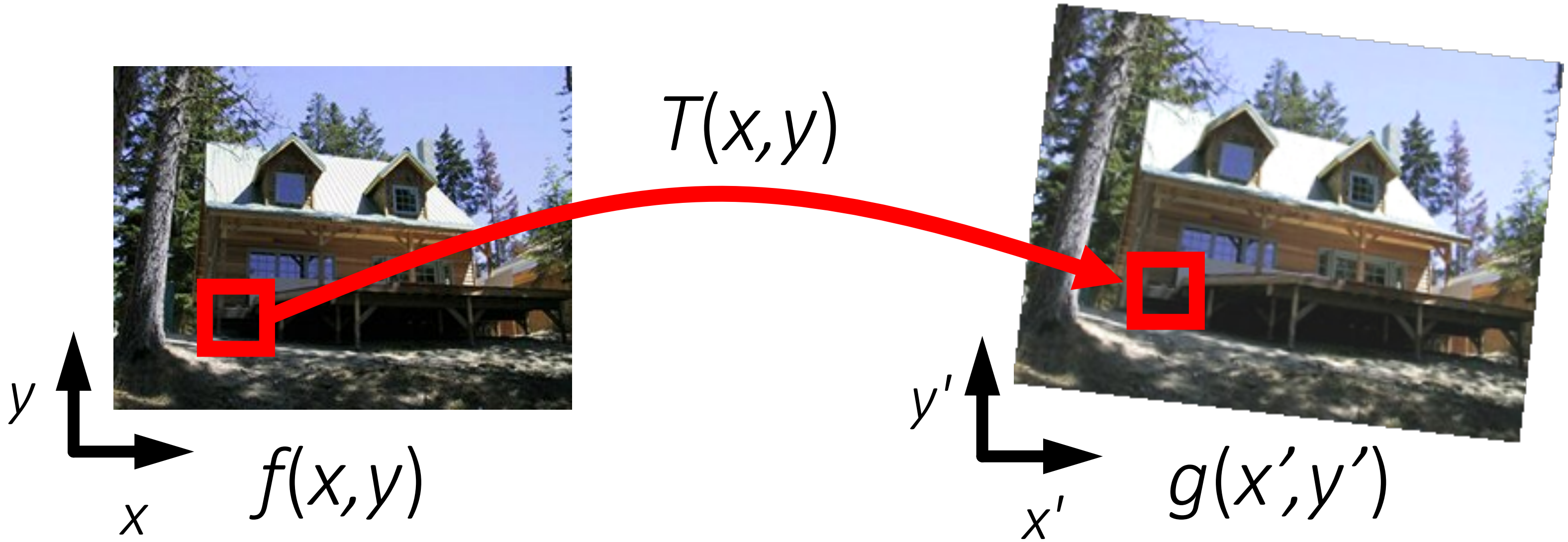
Image Warping



Given a coordinate transform $(x',y') = T(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(T(x,y))$?

Slide Credit: A. Efros

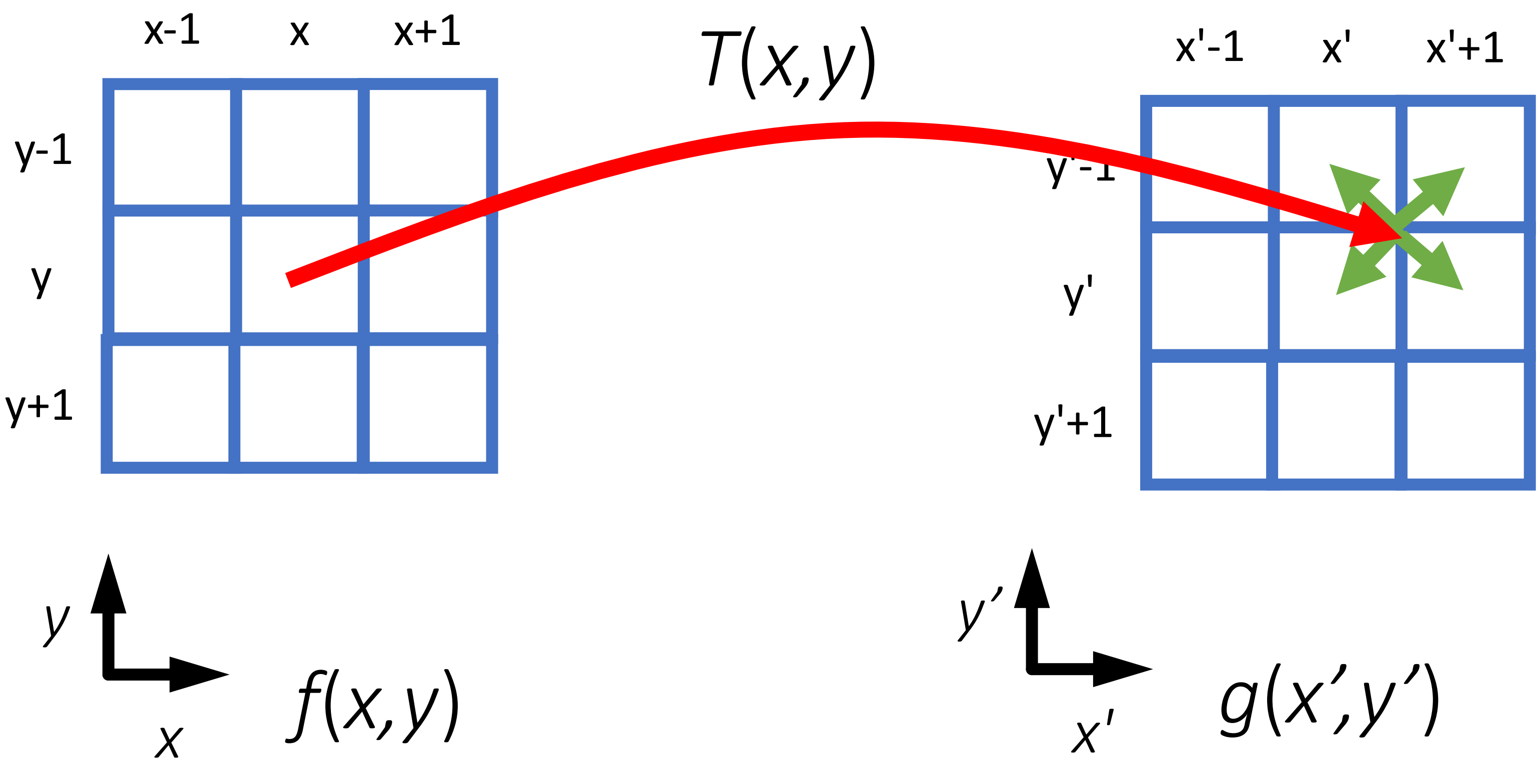
Forward Warping



Send the value at each pixel (x,y) to the new pixel (x',y') = T([x,y])

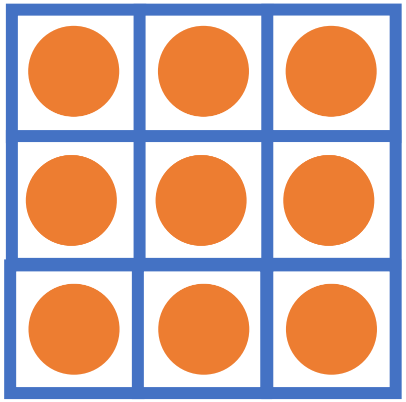
Slide Credit: A. Efros

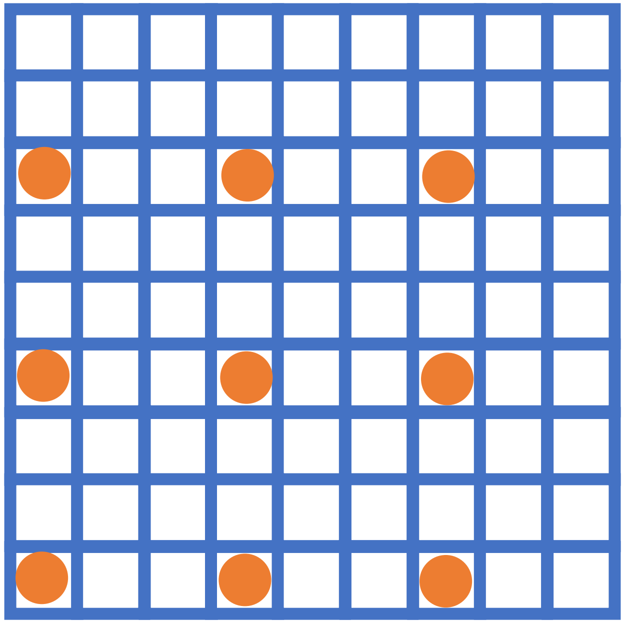
Forward Warping



If you don't hit an exact pixel, give the value to each of the neighboring pixels ("splatting").

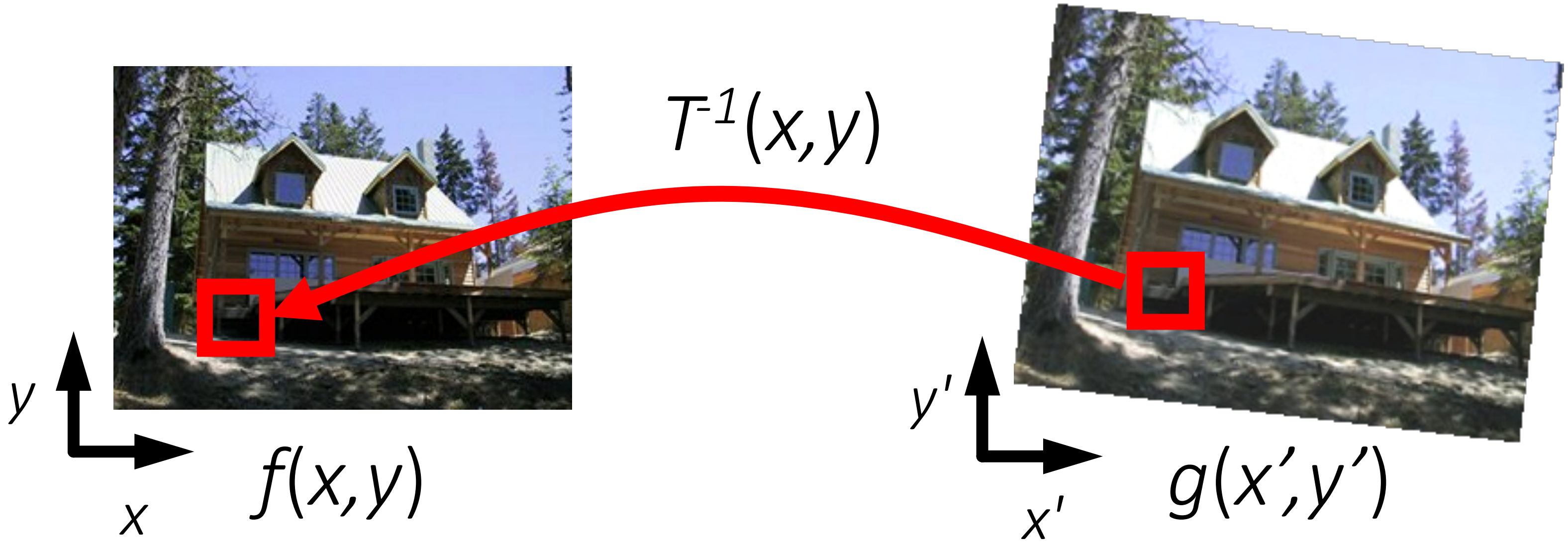
Forward Warping





Suppose T(x,y) scales by a factor of 3.
Hmmmm.

Backward Warping

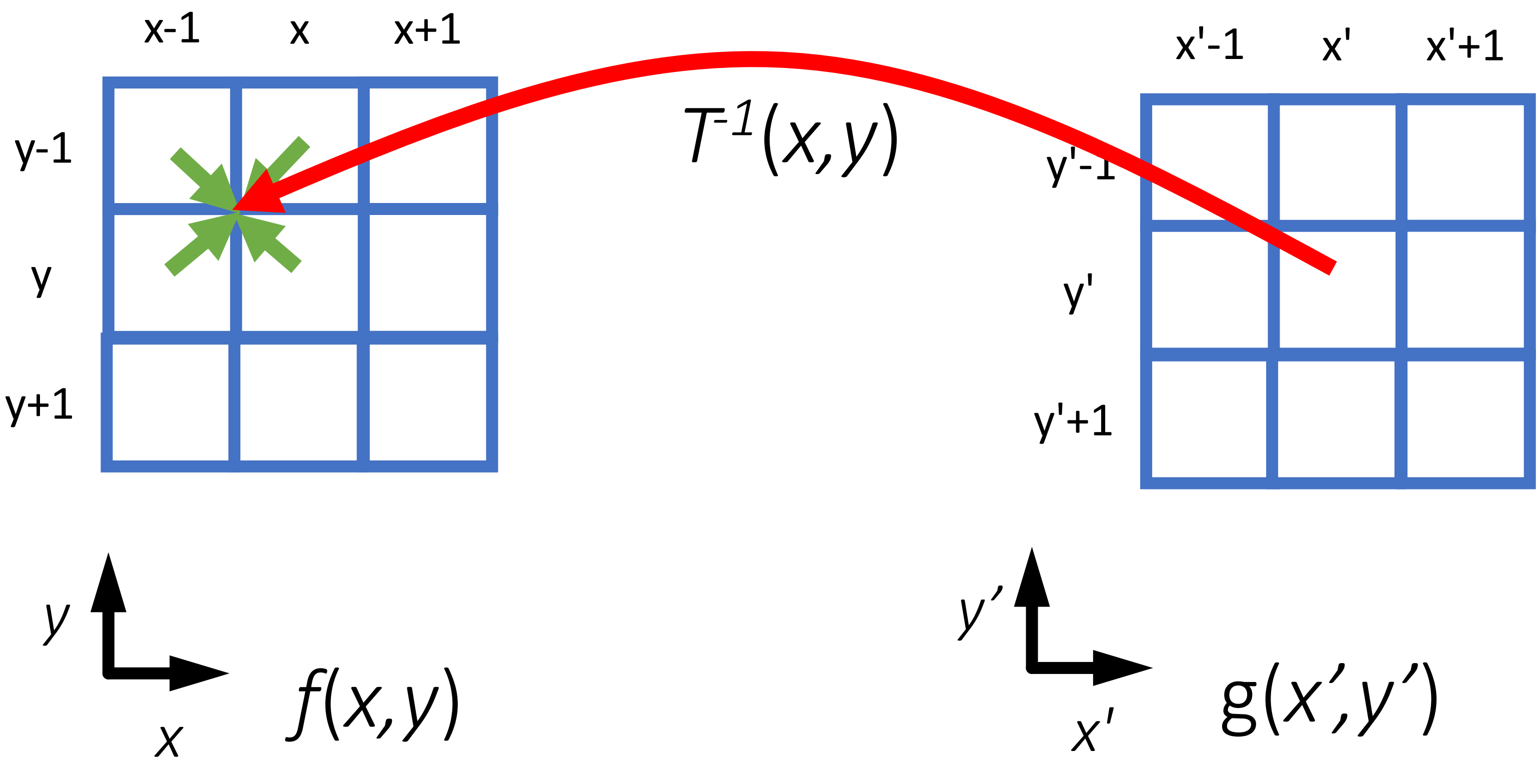


Find out where each pixel g(x',y') should get its value from, and steal it.

Note: requires ability to invert T

Slide Credit: A. Efros

Backward Warping



If you don't hit an exact pixel, figure out how to take it from the neighbors.

Creating Panoramas

Categories of Transformations

Fitting Transformations

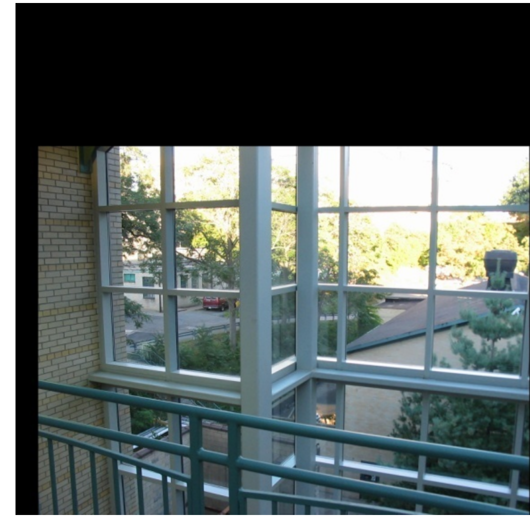
Applying Transformations

Blending Images

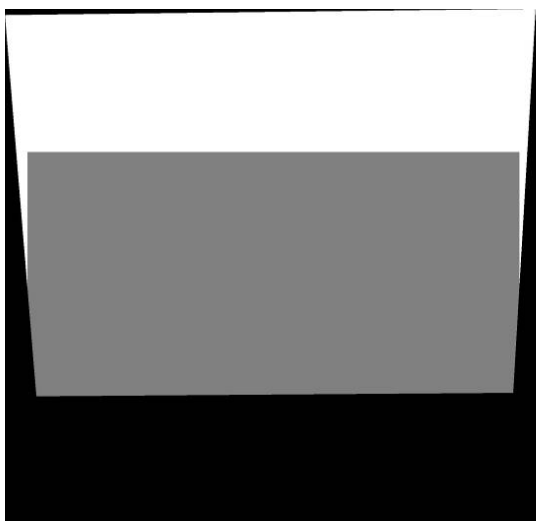
Blending Images

Warped Input 1 I_1

Warped Input 2 I_2



α



$\alpha I_1 + (1-\alpha) I_2$

Slide Credit: A. Efros

Simple Approach: Two-Band Blending

- Brown & Lowe, 2003
  - Break up each image into high frequency + low frequency
  - Linearly blend low-frequency information
  - No blending for high-frequency: at each pixel take from one image or the other

Figure Credit: Brown & Lowe

Simple Approach: Two-Band Blending

Low frequency ($\lambda > 2$ pixels)

High frequency ($\lambda < 2$ pixels)

Linear Blending

2-band Blending

Creating Panoramas

Categories of Transformations

Fitting Transformations

Applying Transformations

Blending Images

Putting It All Together

How do you make a panorama?

Step 1: Find “features” to match

Step 2: Describe Features

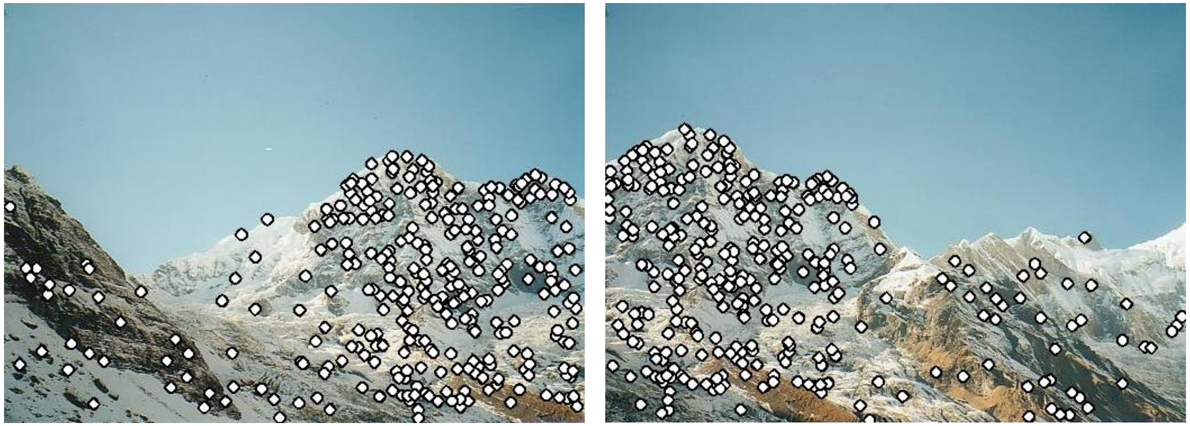
Step 3: Match by Nearest Neighbor

Step 4: Fit H via RANSAC

Step 5: Blend Images

Putting It All Together: Step 1

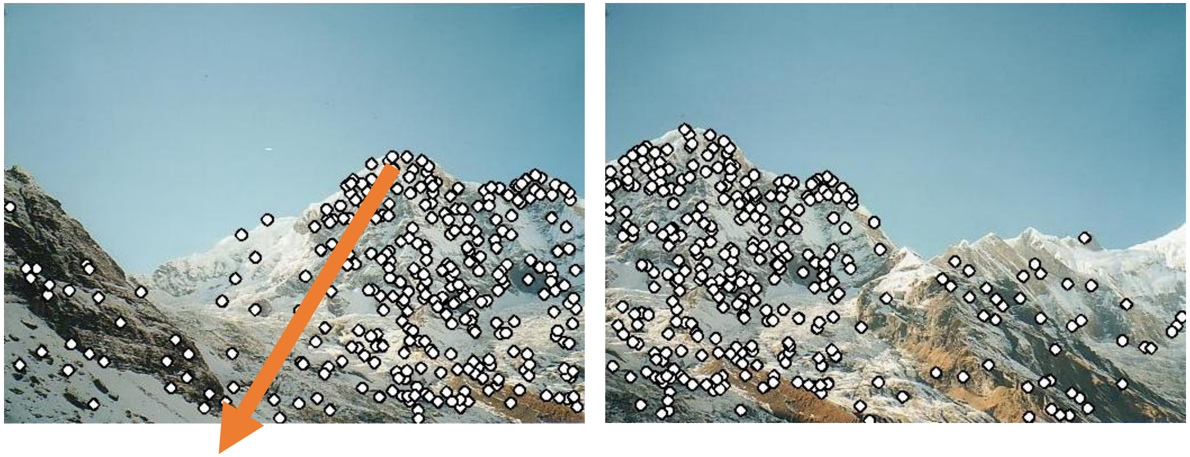
Find corners/blobs



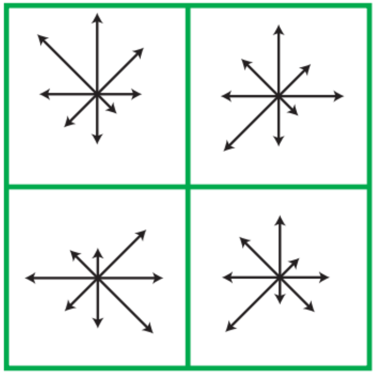
- (Multi-scale) Harris; or
- Laplacian of Gaussian

Putting It All Together: Step 2

Describe Regions Near Features



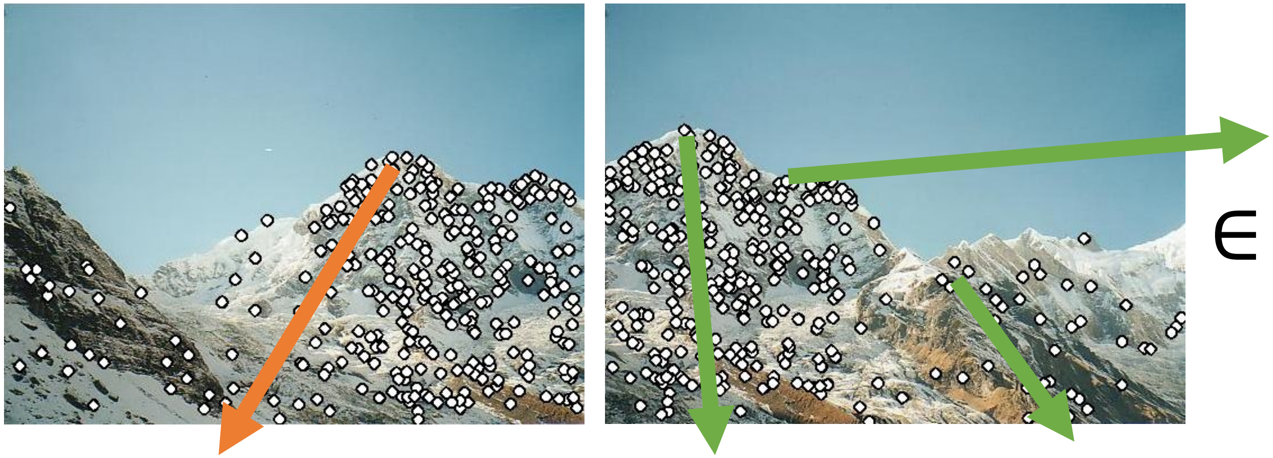
$x_q \in R^{128}$



Build histogram of gradient orientations (SIFT)

Putting It All Together: Step 3

Match Features Based On Region



$x_3 \in R^{128}$

$x_q \in R^{128}$ $\quad x_1 \in R^{128}$ $\quad x_2 \in R^{128}$

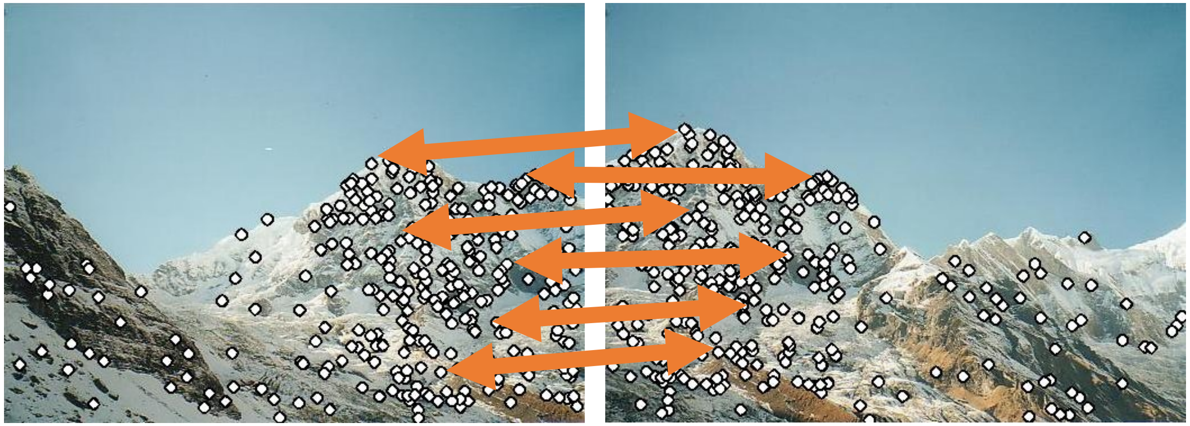
Sort by distance to: x_q $\quad \|x_q - x_1\| < \|x_q - x_2\| < \|x_q - x_3\|$

Accept match if: $\quad \|x_q - x_1\| / \|x_q - x_2\|$

Nearest neighbor is far closer than 2nd nearest neighbor

Putting It All Together: Step 4

Fit transformation H via RANSAC



for trial in range(Ntrials):
    Pick sample
    Fit model
    Check if more inliers
Re-fit model with most inliers

$$\arg\min_{\|\boldsymbol{h}\|=1} \|\boldsymbol{A}\boldsymbol{h}\|^2$$

Putting It All Together: Step 5

Warp images together

Resample images with inverse warping and blend

Course Roadmap

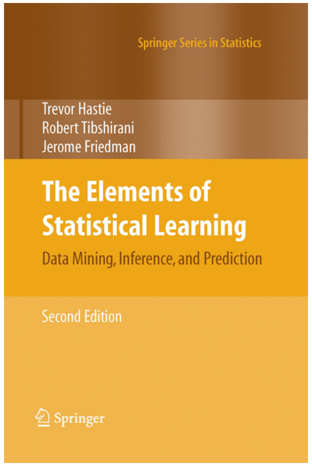
- Basics (Done)
- Image Stitching/Warping (Done)
- Learning-based Vision (Most of March)
- 3D Vision (April)

Machine Learning

Next Few Classes

- Machine Learning (ML) Crash Course
- I can’t cover everything
- ML really won’t solve all problems and is incredibly dangerous if misused
- But ML is a powerful tool and not going away

Pointers



Useful book (Free too!):
The Elements of Statistical Learning
Hastie, Tibshirani, Friedman
https://web.stanford.edu/~hastie/ElemStatLearn/

Useful set of data:
UCI ML Repository
https://archive.ics.uci.edu/ml/datasets.html

A lot of important and hard lessons summarized:
https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

Image Classification: Core Vision Task

Input: image

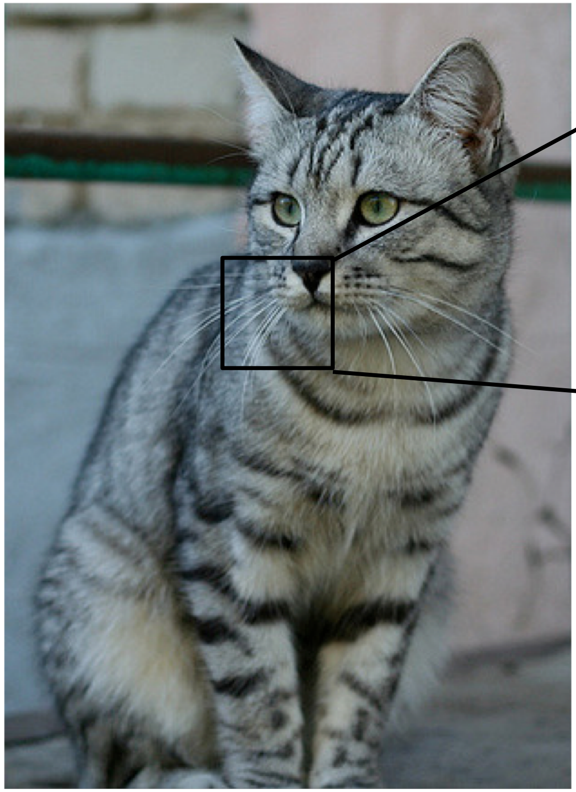
→

Output: Assign image to one of a fixed set of categories

cat
bird
deer
dog
truck

Problem: Semantic Gap

Input: image




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

An Image Classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

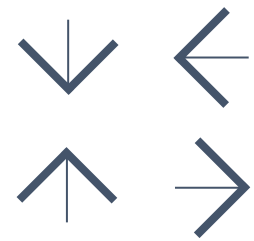
Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

An Image Classifier

Find edges

Find corners



?

Machine Learning: Data-Driven Approach

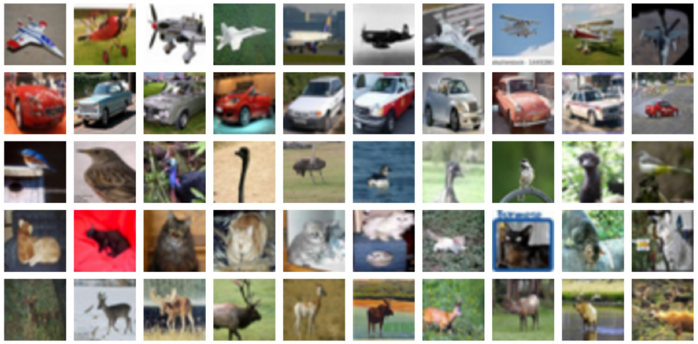
1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set

airplane
automobile
bird
cat
deer



Supervised Learning

Input: **x**

Feature vector/Data point: Vector representation of datapoint. Each dimension or "**feature**" represents some aspect of the data.

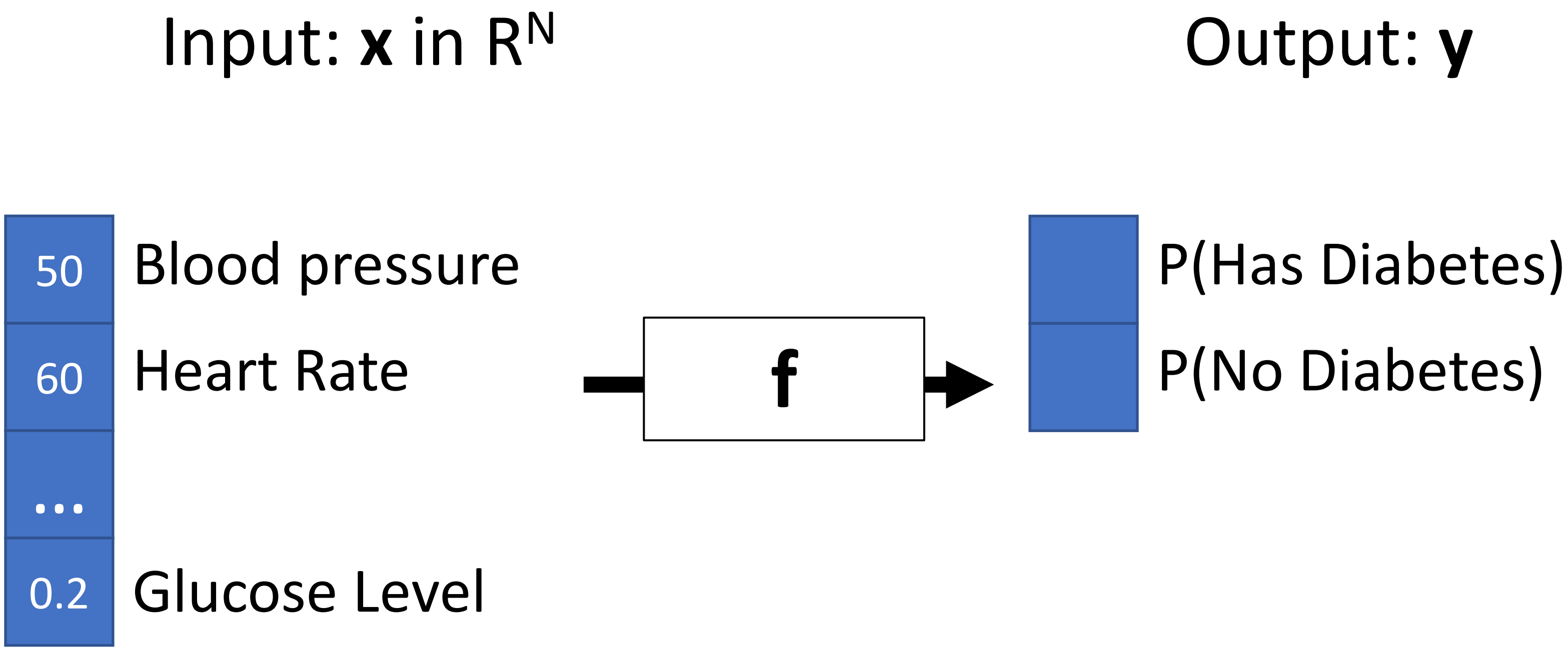
Output: **y**

Label / target: Fixed length vector of desired output. Each dimension represents some aspect of the output data

Goal: Given a dataset (x, y), learn a function f that maps from inputs to outputs

An **objective function** evaluates functions f

Example: Health

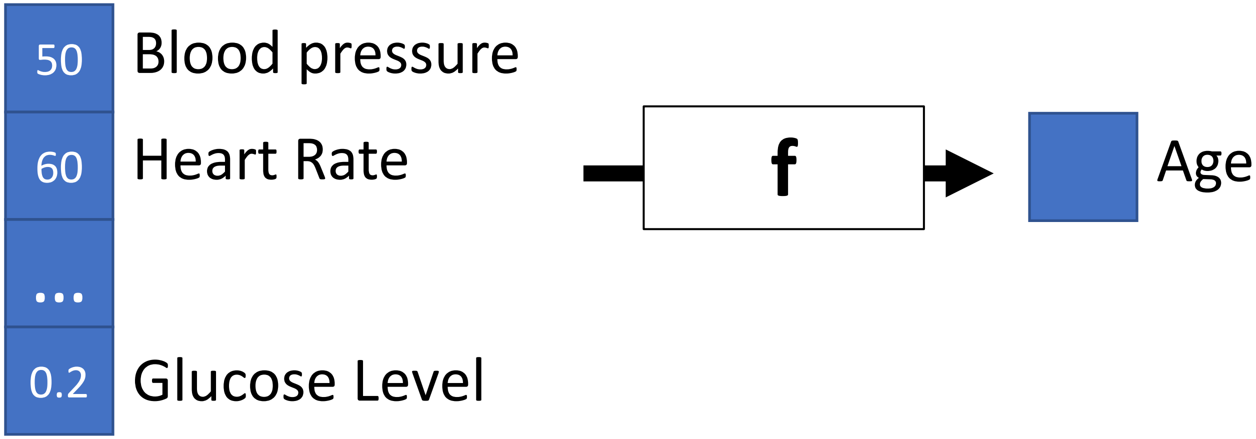


Intuitive objective function: Want correct category to be likely with our model.

Example: Health

Input: $\mathbf{x}$ in R^N

Output: $\mathbf{y}$

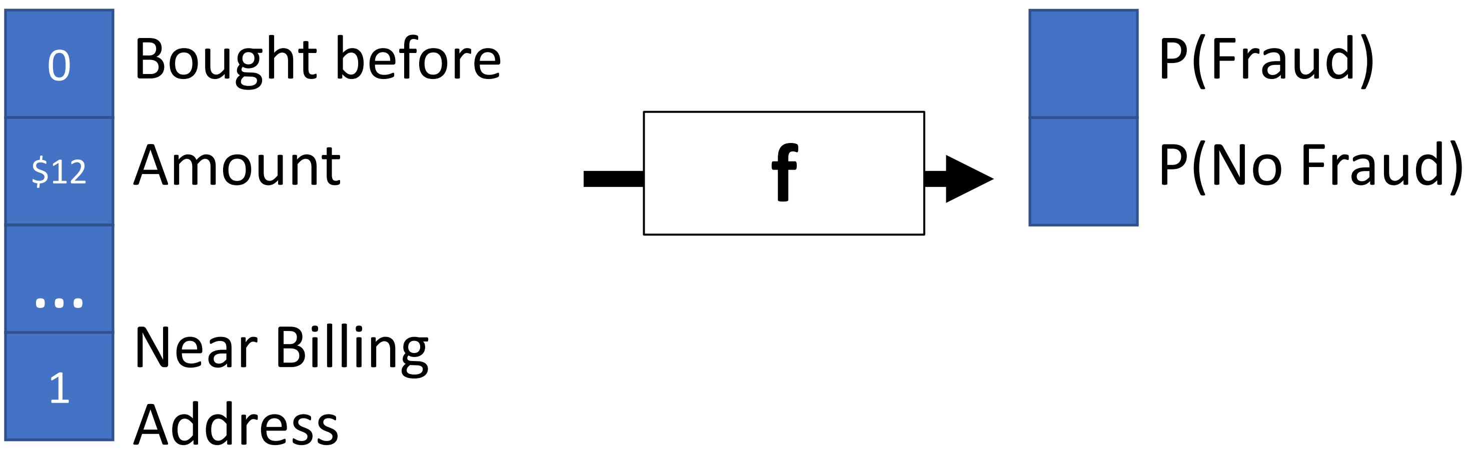


Intuitive objective function: Want our prediction of age to be "close" to true age.

Example: Credit Card Fraud

Input: **x** in R^N

Output: **y**



Intuitive objective function: Want correct category to be likely with our model.

Unsupervised Learning

Input: **x**

Feature vector/Data point: Vector representation of datapoint. Each dimension or "**feature**" represents some aspect of the data.

Output: **y**

Label / target: Fixed length vector of desired output. Each dimension represents some aspect of the output data

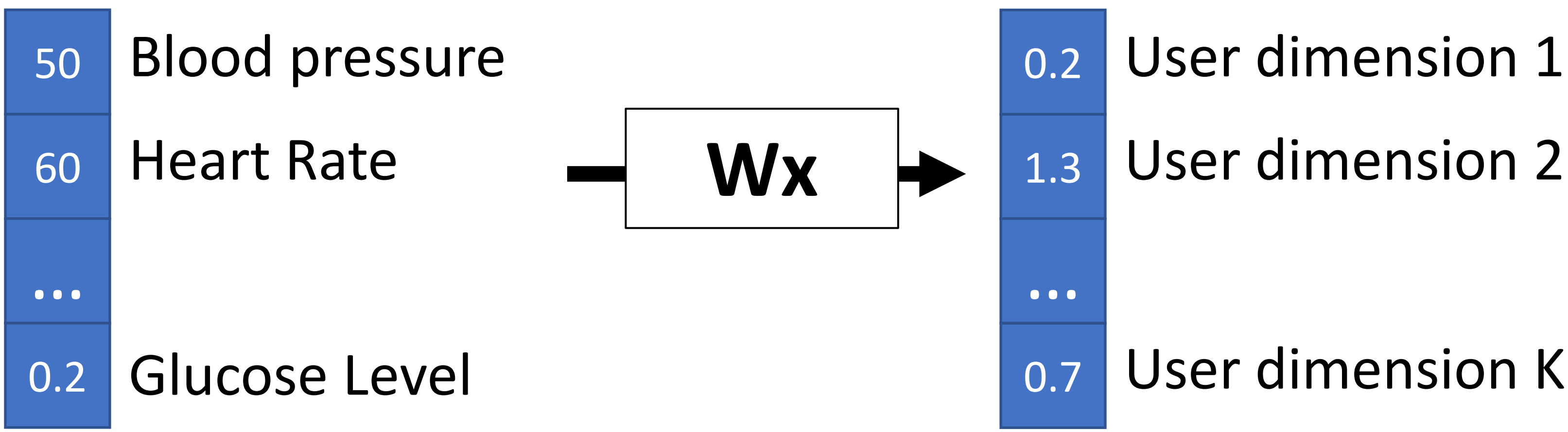
Goal: Given a dataset of only x, learn a function f that uncovers structure in the data

An **objective function** evaluates functions f

Example: Health

Input: **x** in R^N

Output: **continuous y (discovered)**

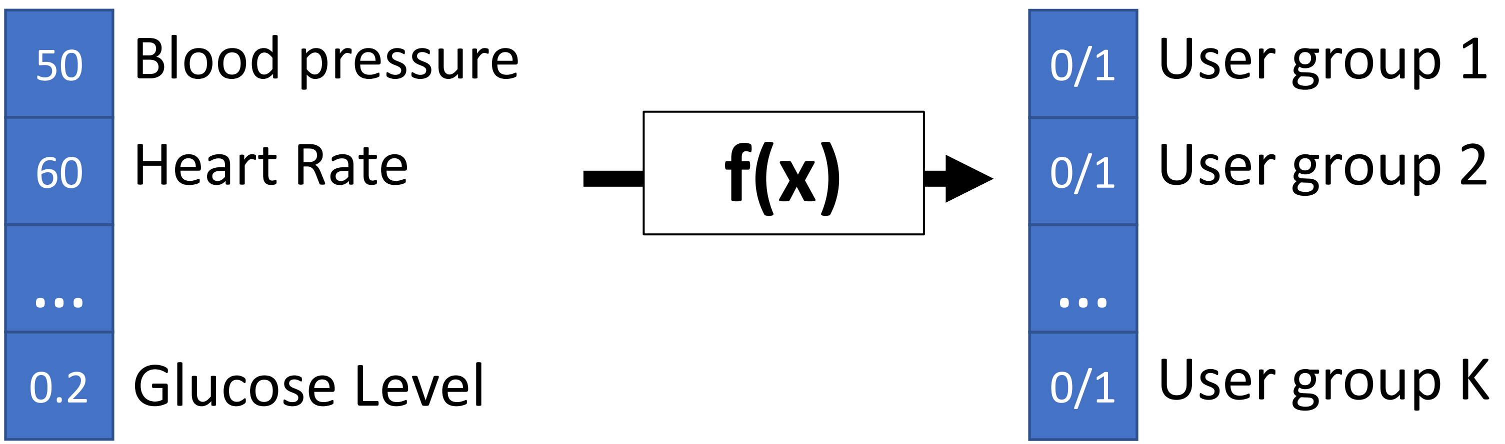


Intuitive objective function: Want to K dimensions (often two) that are easier to understand but capture the variance of the data.

Example: Health

Input: **x** in R^N

Output: **discrete y (unsupervised)**



Intuitive objective function: Want to find K groups that explain the data we see.

ML Problems in Vision

Image credit: Wikipedia

ML Problems in Vision

	Supervised (Data+Labels)	Unsupervised (Just Data)
Discrete Output	Classification/ Categorization	
Continuous Output		

Slide adapted from J. Hays

ML Problems in Vision

***Cat**egorization/Classification*

Binning into K mutually-exclusive categories

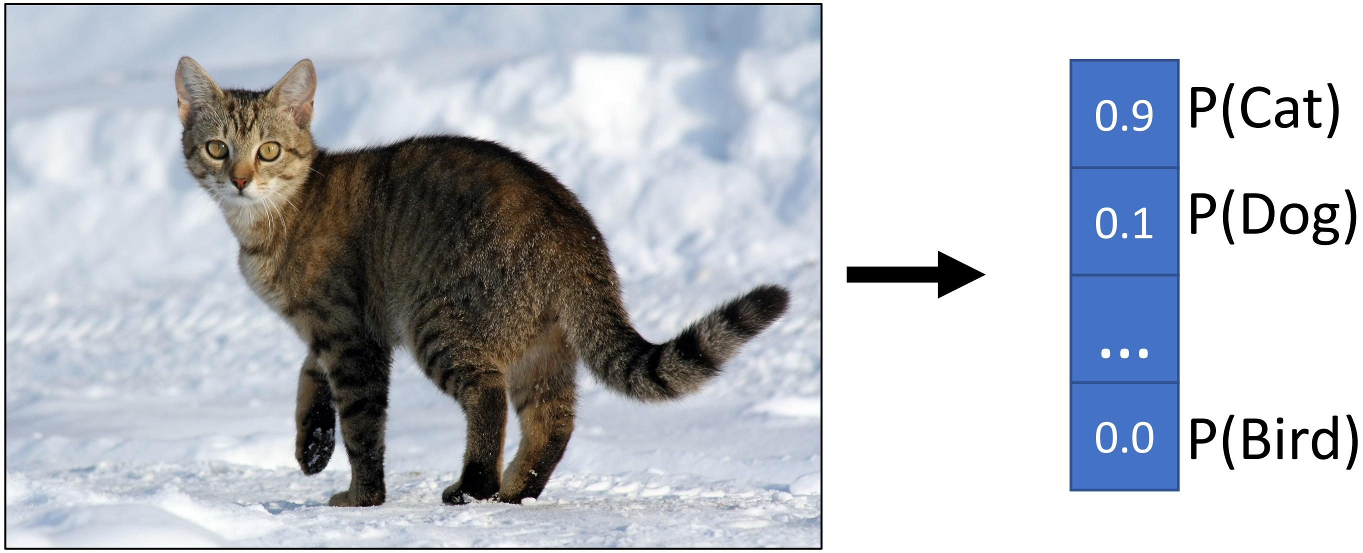


Image credit: Wikipedia

ML Problems in Vision

	Supervised (Data+Labels)	Unsupervised (Just Data)
Discrete Output	Classification/ Categorization	
Continuous Output	**Regression**	

Slide adapted from J. Hays

ML Problems in Vision

Regression

Estimating continuous variable(s)

→ 3.6 kg Cat weight

Image credit: Wikipedia

ML Problems in Vision

	Supervised (Data+Labels)	**Unsupervised (Just Data)**
Discrete Output	Classification/ Categorization	**Clustering**
Continuous Output	Regression	

Slide adapted from J. Hays

ML Problems in Vision

Clustering

Given a set of cats, automatically discover clusters or *cat*egories.

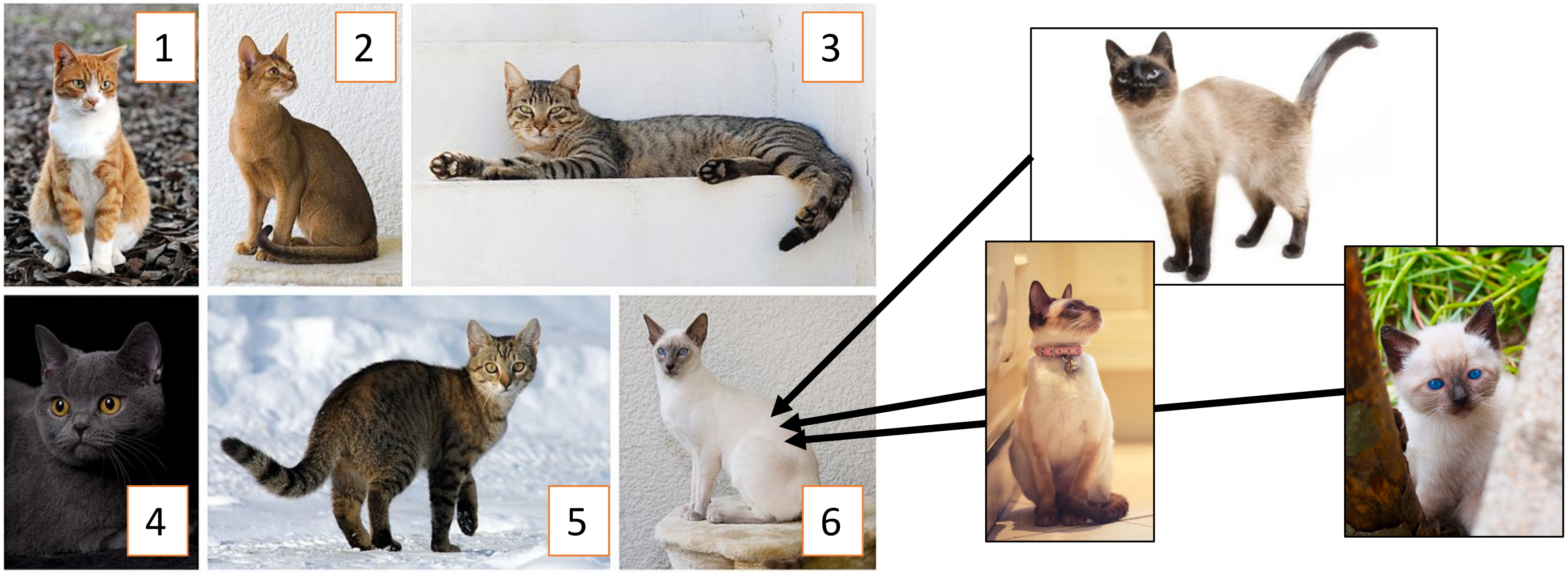


Image credit: Wikipedia, cattime.com

ML Problems in Vision

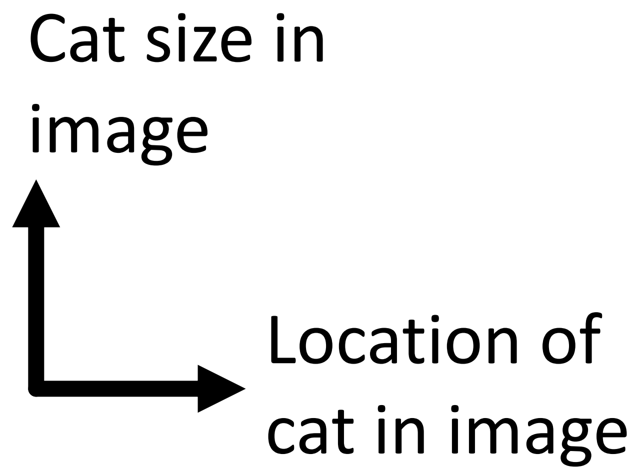
	Supervised (Data+Labels)	**Unsupervised (Just Data)**
Discrete Output	Classification/ Categorization	Clustering
Continuous Output	Regression	**Dimensionality Reduction**

Slide adapted from J. Hays

ML Problems in Vision

Dimensionality Reduction

Find dimensions that best explain the whole image/input



For ordinary images, this is currently a totally hopeless task. For certain images (e.g., faces, this works reasonably well)

Image credit: Wikipedia

Practical ML Example

- Let’s start with:
  - A model you learned in middle/high school (a line)
  - Least-squares
- One thing to remember:
  - N eqns, <N vars = overdetermined (will have errors)
  - N eqns, N vars = exact solution
  - N eqns, >N vars = underdetermined (infinite solns)

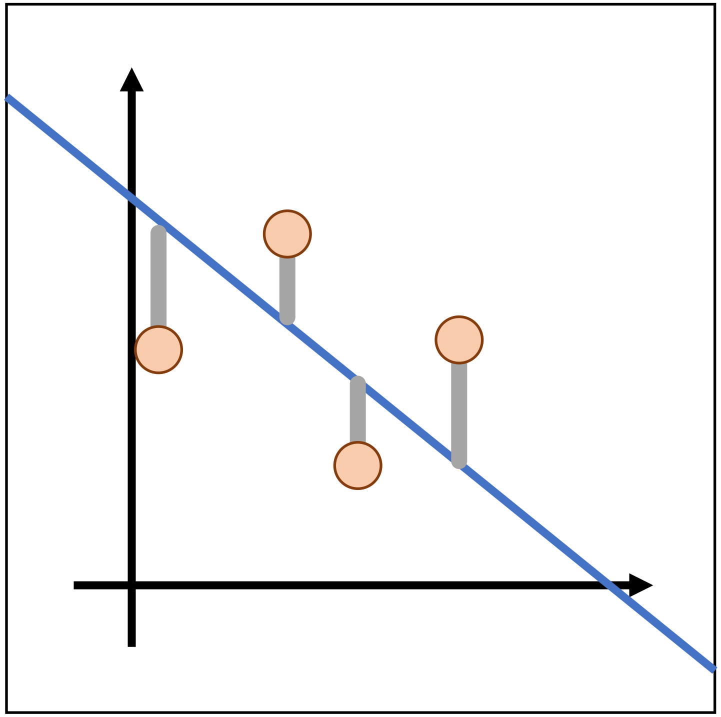
Example: Least Squares

Let's make the world's **worst** weather model

Data: (x_1,y_1), (x_2,y_2), …, (x_k,y_k)

Model: (m,b) $y_i=mx_i+b$
Or $(\mathbf{w})$ $y_i = \mathbf{w}^T\mathbf{x}_i$

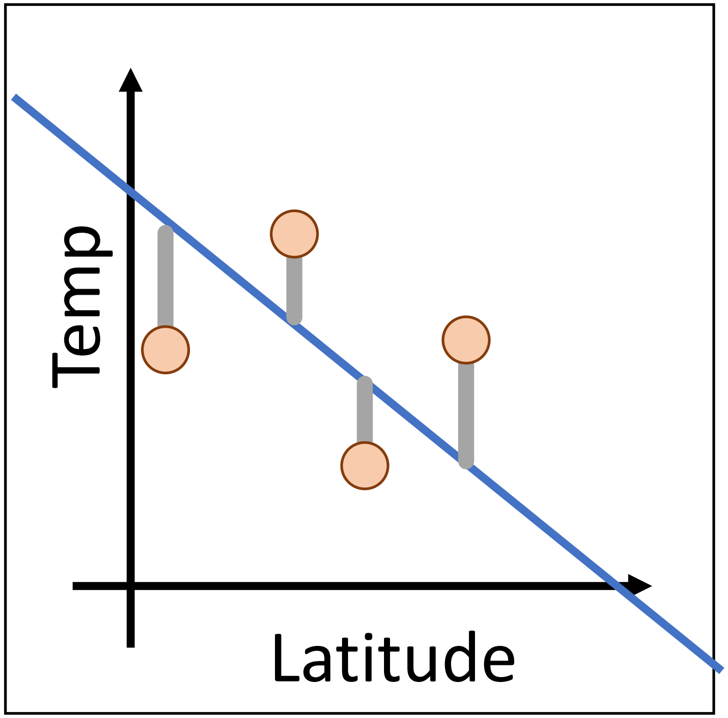
Objective function:
$(y_i - \mathbf{w}^T\mathbf{x}_i)^2$



World’s Worst Weather Model

Given latitude (distance above equator), predict temperature by fitting a line

City	*Latitude (°)*	*Temp (F)*
Ann Arbor	42	33
Washington, DC	39	38
Austin, TX	30	62
Mexico City	19	67
Panama City	9	83



Example: Least Squares

$$\sum_{i=1}^{k}(y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2 \quad \longrightarrow \quad \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$

Output: Temperature

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Inputs: Latitude, 1

$$\boldsymbol{X} = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_k & 1 \end{bmatrix}$$

Model/Weights: Latitude, “Bias”

$$\boldsymbol{w} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Example: Least Squares

Training ($\mathbf{x}_i$,y_i):

$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$ **or**

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \|\boldsymbol{w}^T \boldsymbol{x_i} - y_i\|^2$$

Loss function/objective: evaluates correctness.
Here: Squared L2 norm / Sum of Squared Errors

Training/Learning/Fitting: try to find model that *optimizes/minimizes* an objective / loss function

Optimal **w*** is $$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Example: Least Squares

Training ($\mathbf{x}_i$,y_i):

$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 \quad \textbf{or}$$

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \|\boldsymbol{w}^T \boldsymbol{x_i} - y_i\|^2$$

Inference (x):

$$\boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + \cdots + w_F x_F$$

Testing/Inference:
Given a new output,
what's the prediction?

Least Squares: Learning

Data

City	*Latitude*	*Temp*
Ann Arbor	42	33
Washington, DC	39	38
Austin, TX	30	62
Mexico City	19	67
Panama City	9	83

Model

Temp =
-1.47*Lat + 97

$$\boldsymbol{X}_{5x2} = \begin{bmatrix} 42 & 1 \\ 39 & 1 \\ 30 & 1 \\ 19 & 1 \\ 9 & 1 \end{bmatrix} \quad \boldsymbol{y}_{5x1} = \begin{bmatrix} 33 \\ 38 \\ 62 \\ 67 \\ 83 \end{bmatrix}$$

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$$w_{2x1} = \begin{bmatrix} -1.47 \\ 97 \end{bmatrix}$$

Least Squares: Prediction

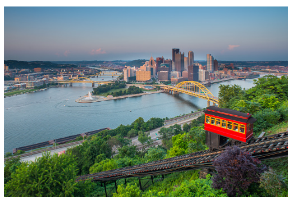
The EECS 442 **Weather Channel**

City	*Latitude*	*Temp*	*Temp*	*Error*
Ann Arbor	42	33	35.3	2.3
Washington, DC	39	38	39.7	1.7
Austin, TX	30	62	52.9	10.9
Mexico City	19	67	69.1	2.1
Panama City	9	83	83.8	0.8

Is this a good idea?

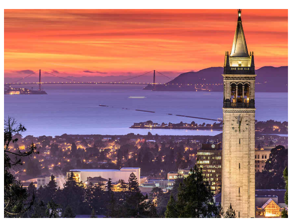
The EECS 442 **Weather Channel**

The Weather Channel



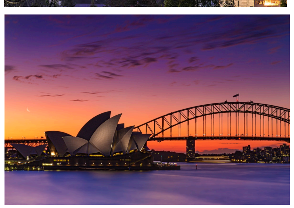
Pittsburgh:
Temp = -1.47*40 + 97 = 38

Actual Pittsburgh:
45



Berkeley:
Temp = -1.47*38 + 97 = 41

Actual Berkeley:
53



Sydney:
Temp = -1.47*-33 + 97 = **146**

Actual Sydney:
74

Won’t do so well in the Australian market…

What is going wrong?

Data

City	*Latitude*	*Temp*
Ann Arbor	42	33
Washington, DC	39	38

Model

Temp =
-1.66*Lat + 103

How well can we predict Ann Arbor and DC and why?

Testing the model

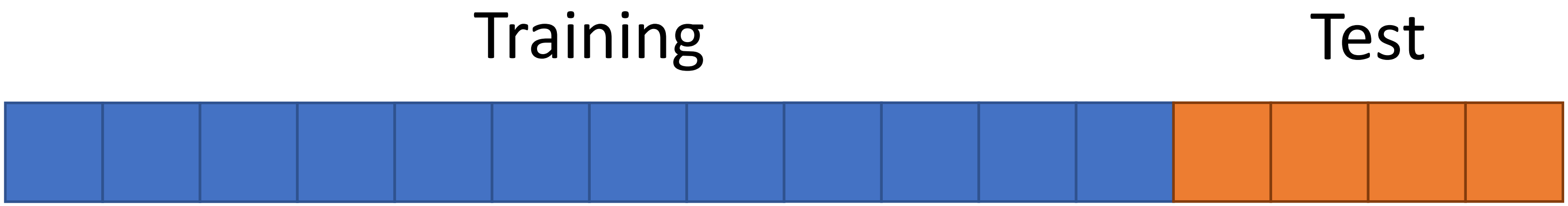
Overfitting: Model might be fit data too precisely
Remember: #datapoints = #params = perfect fit

Model may only work under some conditions
(e.g., trained on northern hemisphere).

Sydney:
Temp = -1.47*-33 + 97 = **146**

Training and Testing

Fit model parameters on **training** set;
evaluate on *entirely unseen* **test** set.



Nearly any model can predict data it's seen. If your model can't accurately interpret "unseen" data, it's probably useless. We have no clue whether it has just memorized.

Adding More Features

If one feature does ok, what about more features!?

City Name	*Latitude (deg)*	*Avg July High (F)*	*Avg Snowfall*	*Temp (F)*
Ann Arbor	42	83	58	33
Washington, DC	39	88	15	38
Austin, TX	30	95	0.6	62
Mexico City	19	74	0	67
Panama City	9	93	0	83

$\boldsymbol{X}_{5x4}$ 4 features + a feature of 1s for intercept/bias

$\boldsymbol{y}_{5x1}$

Adding More Features

All the math works out!

Data

$\boldsymbol{X}_{5x4} \quad \boldsymbol{y}_{5x1}$

$\boxed{\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$

$\Rightarrow$

Model

$\boldsymbol{w}_{4x1}$

New EECS 442 Weather Rule:

w_1*latitude + w_2*(avg July high) + w_3*(avg snowfall) + w_4*1

In general called linear regression

Adding More Features

If one feature does ok, what about **LOTS** of features!?

City Name	*Latitude (deg)*	*Avg July High (F)*	*Avg Snowfall*	*Day of Year*	*Elevation (ft)*	*% Letter M*	*Temp (F)*
Ann Arbor	42	83	58	45	840	100	33
Washington, DC	39	88	15	45	409	3	38
Austin, TX	30	95	0.6	45	489	2	62
Mexico City	19	74	0	45	7200	4	67
Panama City	9	93	0	45	7	1	83

$\boldsymbol{X}_{5x7}$

6 features + a feature of 1s for intercept/bias

$\boldsymbol{y}_{5x1}$

Adding More Features

Data

$$\boldsymbol{X}_{5x7} \quad \boldsymbol{y}_{5x1}$$

$\boxed{\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$

Model

$$\boldsymbol{w}_{7x1}$$

$$\boldsymbol{w}^* = \underbrace{\left(\boldsymbol{X}^T\boldsymbol{X}\right)}^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

X^TX is a 7x7 matrix but is **rank deficient** (rank 5) *and has no inverse. There are an infinite number of solutions.*

Have to express some preference for which of the infinite solutions we want.

Exercise for the mathematically-inclined folks: derive what the space of solutions looks like.

The Fix: Regularization

Add **regularization** to objective that prefers some solutions:

Before: $\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$ → Loss

After: $\arg\min_{\boldsymbol{w}} \underbrace{\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2}_{\text{Loss}} + \underbrace{\lambda}_{\text{Trade-off}} \underbrace{\|\boldsymbol{w}\|_2^2}_{\text{Regularization}}$

Want model "smaller": pay a penalty for **w** with big norm

Intuitive Objective: accurate model (low loss) but not too complex (low regularization). λ controls how much of each.

The Fix: Regularization

Objective: $\arg\min_{\boldsymbol{w}} \underbrace{\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2}_{\text{Loss}} + \underbrace{\lambda}_{\text{Trade-off}} \underbrace{\|\boldsymbol{w}\|_2^2}_{\text{Regularization}}$

Take $\frac{\partial}{\partial \boldsymbol{w}}$, set to **0**, solve

$$\boldsymbol{w}^* = \underbrace{\left(\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda I}\right)}_{}^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$X^TX+\lambda I$ is full-rank (and thus invertible) for $\lambda>0$

Called *lots of things:* regularized least-squares, Tikhonov regularization (after Andrey Tikhonov), ridge regression, Bayesian linear regression with a multivariate normal prior.

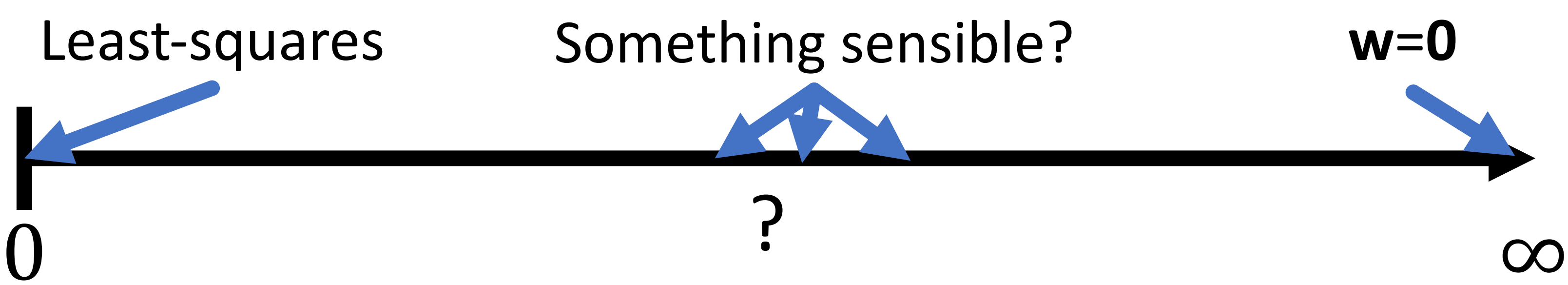
The Fix: Regularization

Objective: $\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2$

Loss Trade-off Regularization

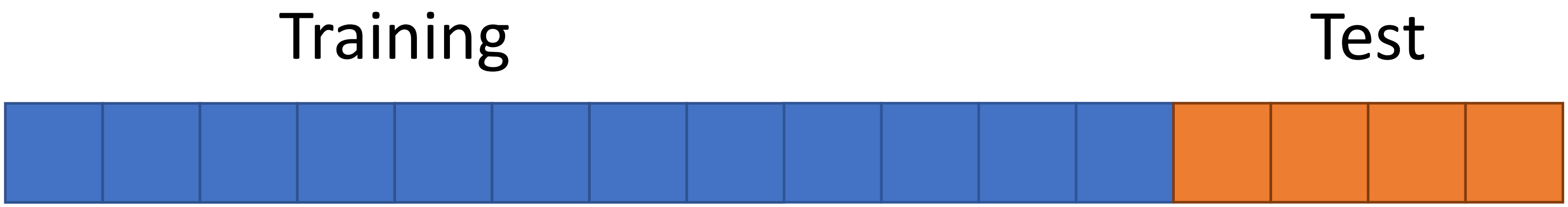
What happens (and why) if:

- $\lambda=0$
- $\lambda=\infty$



Training and Testing

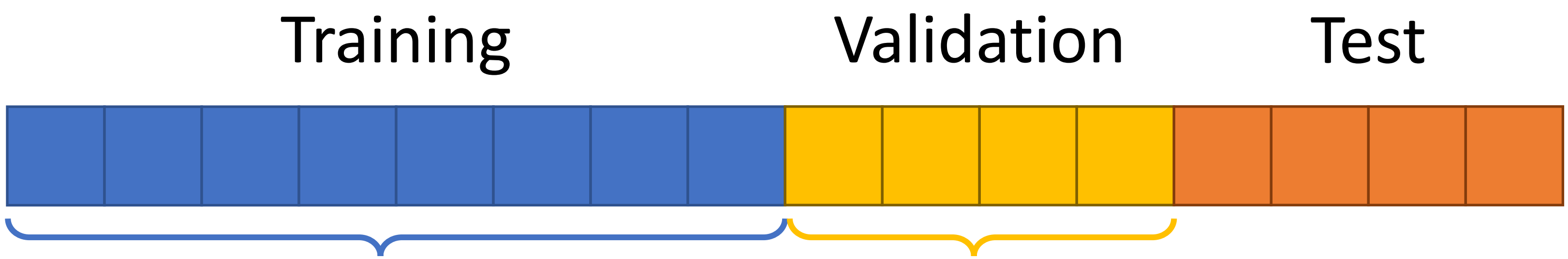
Fit model parameters on training set;
evaluate on *entirely unseen* test set.



How do we pick λ?
(hyperparameter)

Training and Testing

Fit model parameters on training set;
find *hyperparameters* by testing on validation set;
evaluate on *entirely unseen* test set.



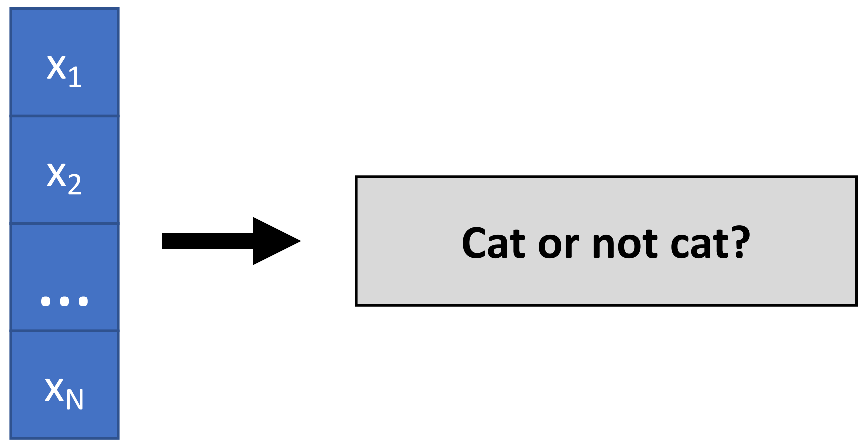
Use these data points to fit $\mathbf{w}^*=(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Evaluate on these points for different λ, pick the best

Image Classification

Start with simplest example: binary classification

Actually: a feature vector representing the image



Classification with Least Squares

Treat as regression: x_i is image feature; y_i is 1 if it's a cat, 0 if it's not a cat. Minimize least-squares loss.

Training ($\mathbf{x}_i$,y_i): $$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \|\boldsymbol{w}^T \boldsymbol{x}_{\boldsymbol{i}} - y_i\|^2$$

Inference (x): $$\boldsymbol{w}^T \boldsymbol{x} > t$$

Unprincipled in theory, but often effective in practice

The reverse (regression via discrete bins) is also common

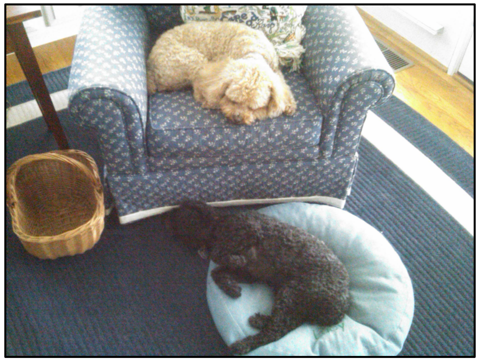
Rifkin, Yeo, Poggio. *Regularized Least Squares Classification* (http://cbcl.mit.edu/publications/ps/rlsc.pdf). 2003

Redmon, Divvala, Girshick, Farhadi. *You Only Look Once: Unified, Real-Time Object Detection.* CVPR 2016.

Classification via Memorization

Just **memorize** (as in a Python dictionary)
Consider cat/dog/hippo classification.

If this:
cat.



If this:
dog.

If this:
hippo.

Classification via Memorization

Where does this go wrong?

Rule: if this, then cat

Hmmm. Not quite the same.

Classification via Memorization

Known Images
Labels

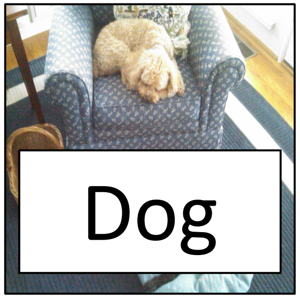
$\boldsymbol{x}_1$

$D(\boldsymbol{x}_1, \boldsymbol{x}_T)$

$\boldsymbol{x}_T$

Test
Image

...



$\boldsymbol{x}_N$

$D(\boldsymbol{x}_N, \boldsymbol{x}_T)$

(1) Compute distance between feature vectors
(2) find nearest
(3) use label.

Nearest Neighbor

"Algorithm"

Training ($\mathbf{x}_i$,y_i): Memorize training set

Inference (x):

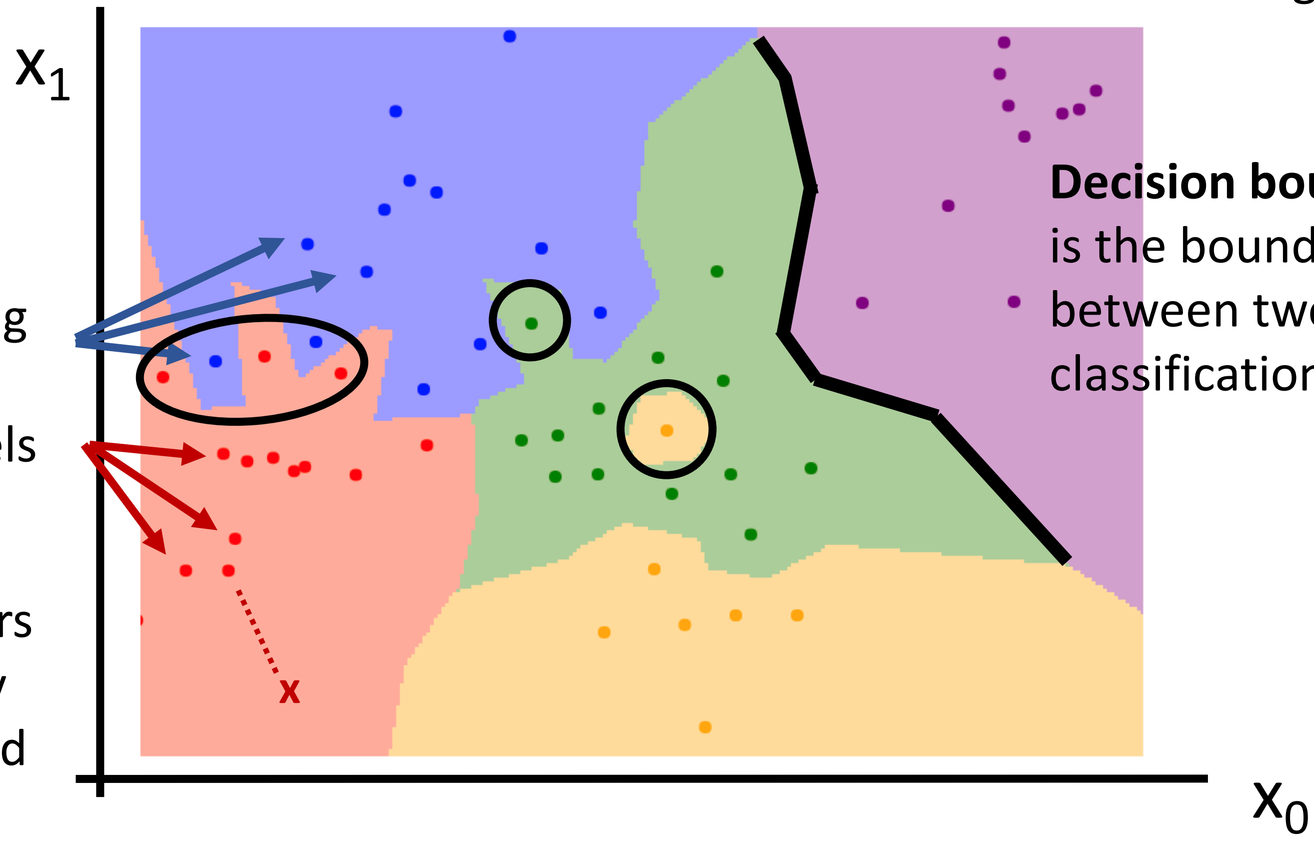
```
bestDist, prediction = Inf, None
for i in range(N):
    if dist(x_i,x) < bestDist:
        bestDist = dist(x_i,x)
        prediction = y_i
```

Nearest Neighbor

Nearest neighbors in two dimensions

Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!



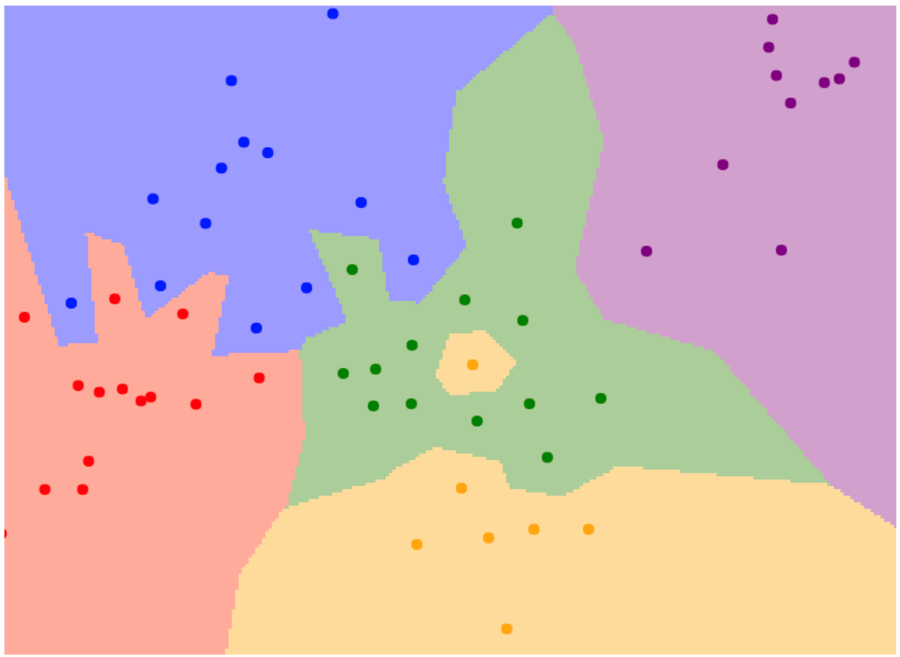
Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

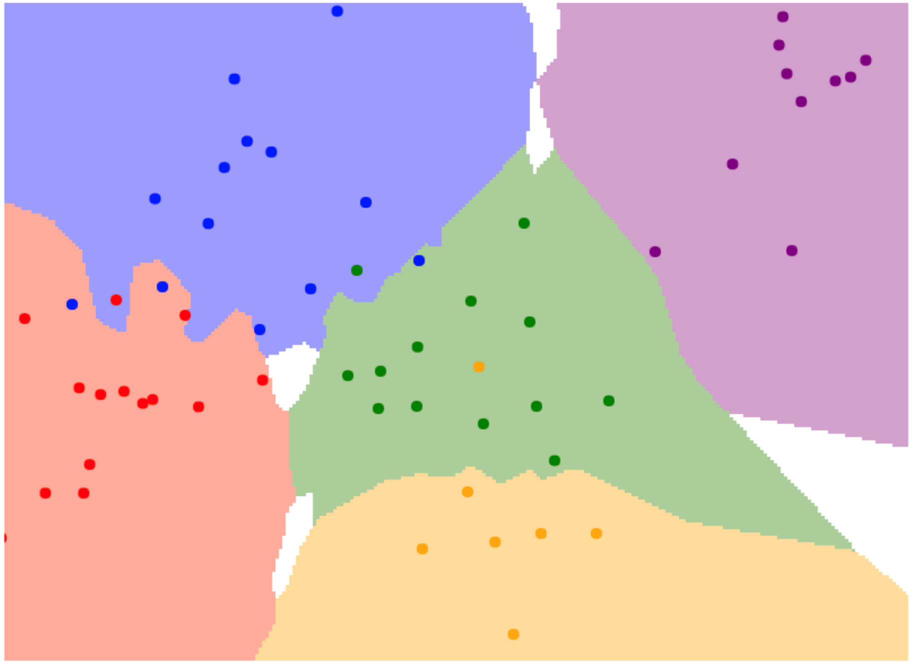
Decision boundary is the boundary between two classification regions

K-Nearest Neighbors

K = 1

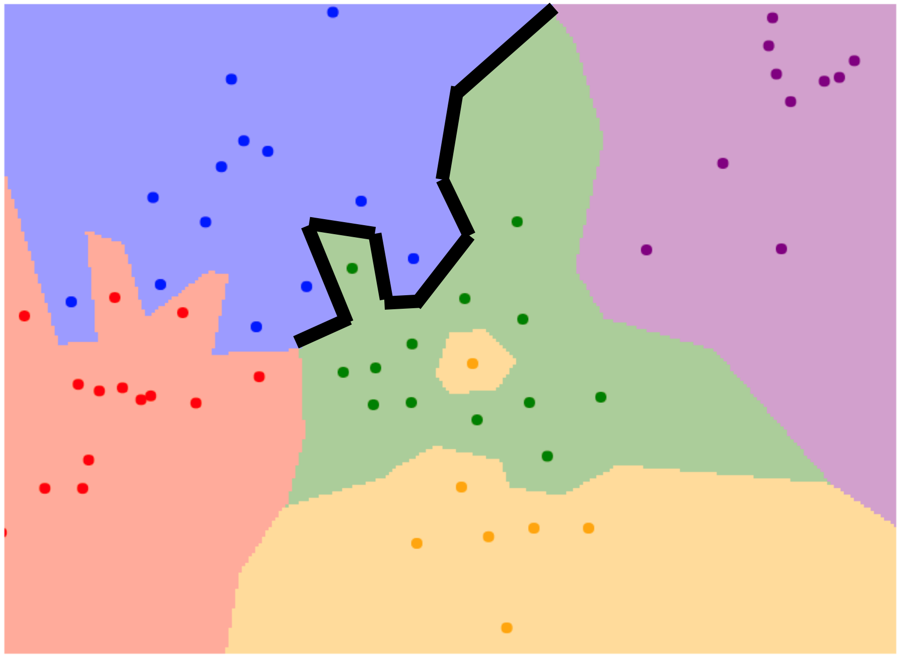


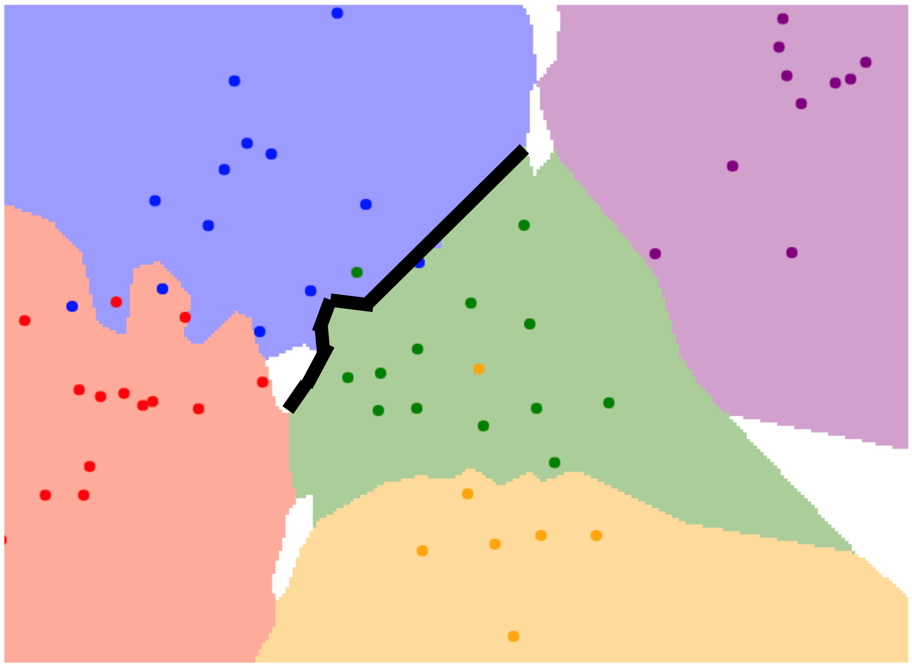
K = 3



Instead of copying label from nearest neighbor, take **majority vote** from K closest points

K-Nearest Neighbors

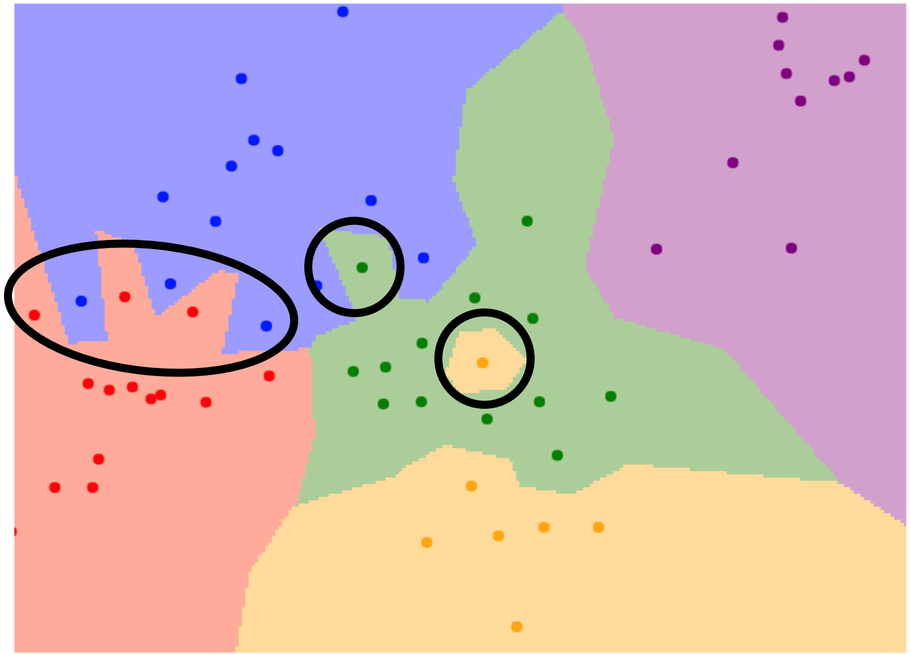




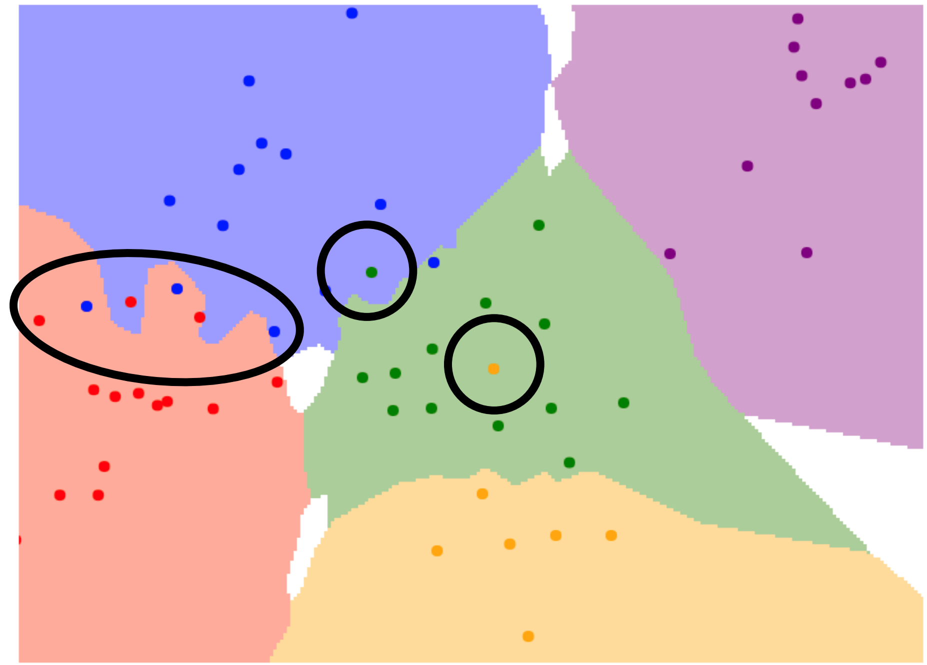
Using more neighbors helps smooth out rough decision boundaries

K-Nearest Neighbors

K = 1

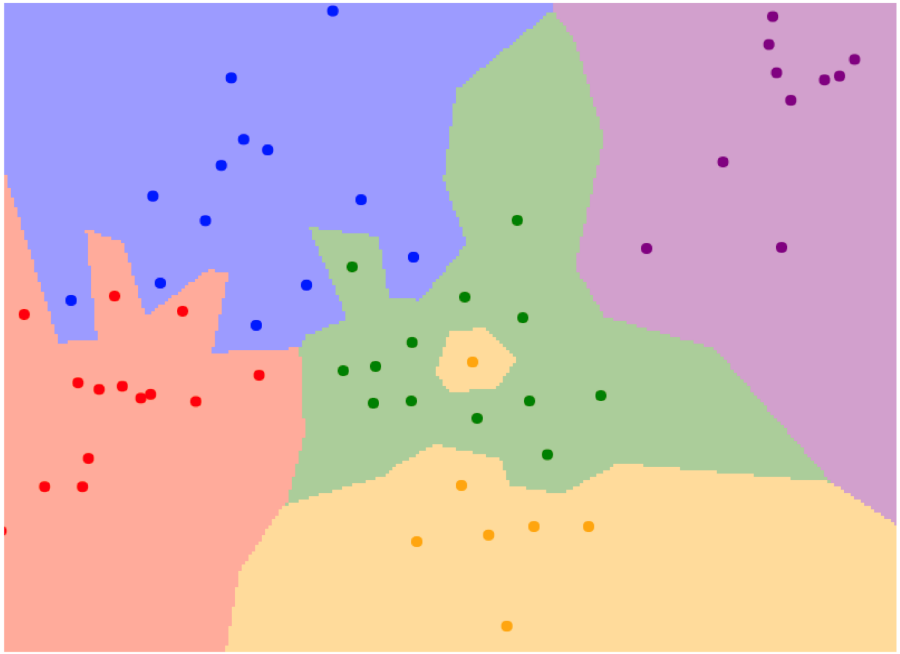


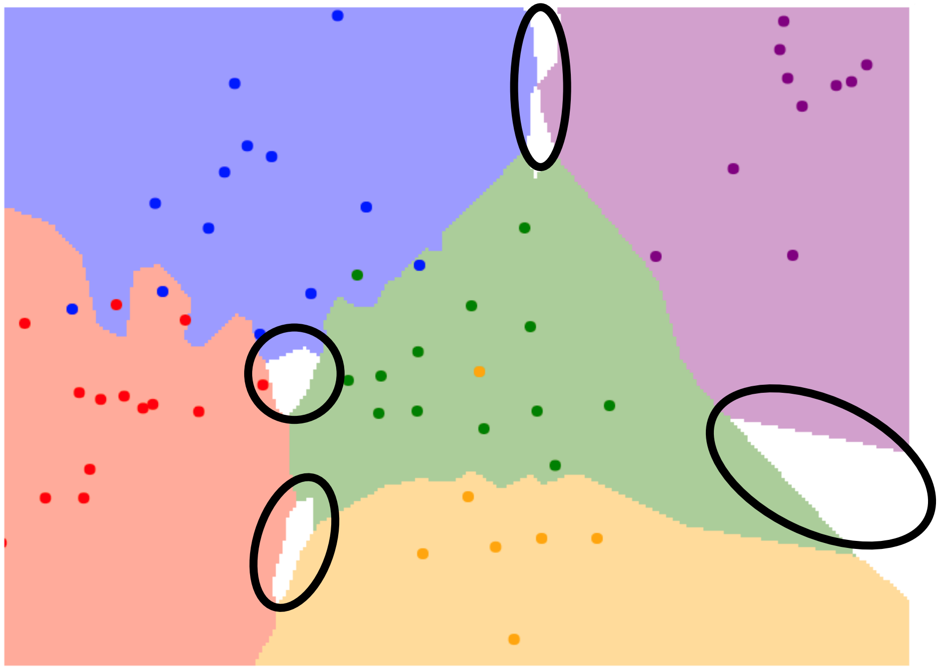
K = 3



Using more neighbors helps
reduce the effect of outliers

K-Nearest Neighbors



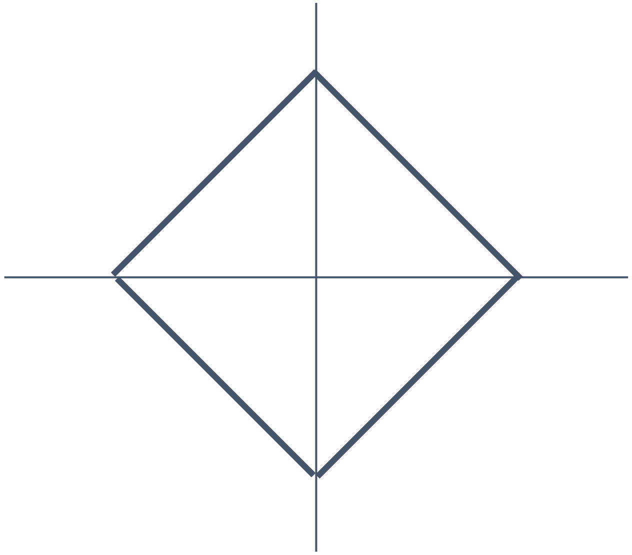


When K > 1 there can be ties!
Need to break them somehow

K-Nearest Neighbors: Distance Metric

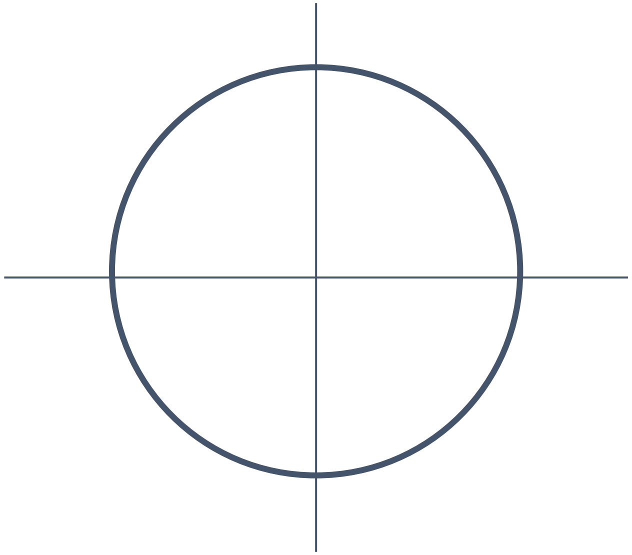
L1 (Manhattan) Distance

$$d(x, y) = \sum_i |x_i - y_i|$$



L2 (Euclidean) Distance

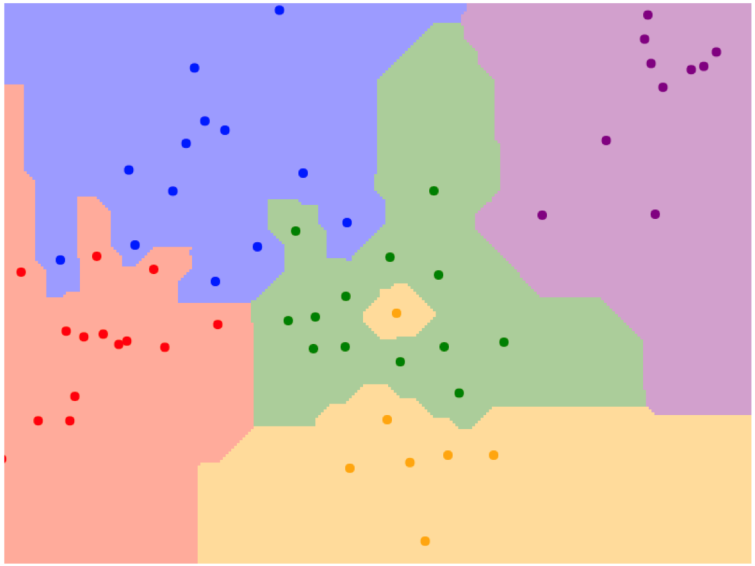
$$d(x, y) = \left(\sum_i (x_i - y_i)^2\right)^{1/2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) Distance

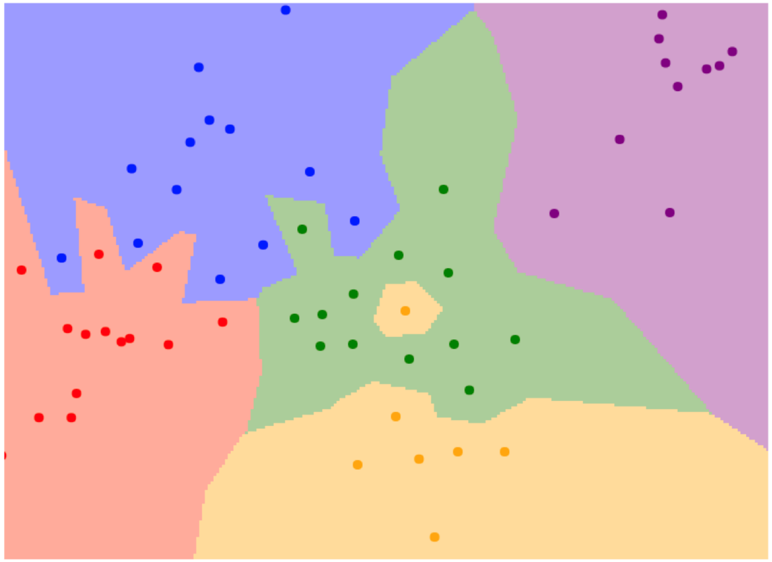
$$d(x, y) = \sum_i |x_i - y_i|$$



K = 1

L2 (Euclidean) Distance

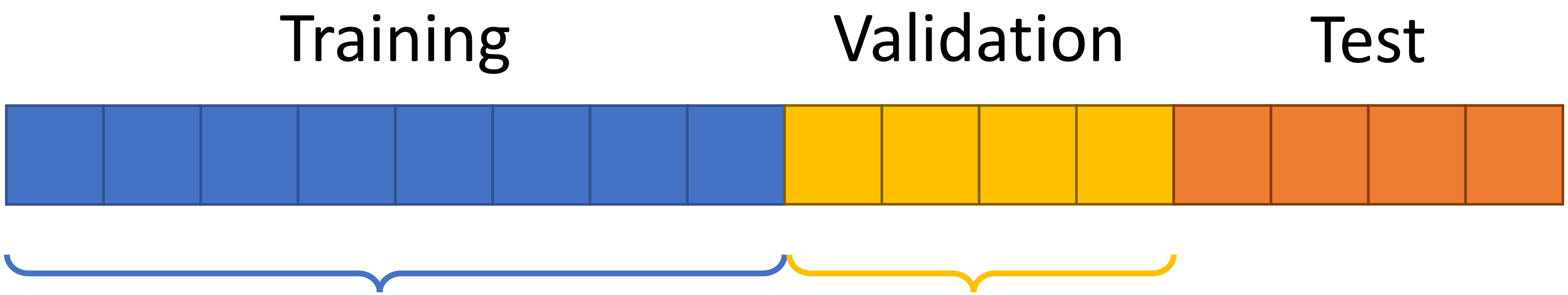
$$d(x, y) = \left(\sum_i (x_i - y_i)^2\right)^{1/2}$$



K = 1

K-Nearest Neighbors

What distance? What value for K?



Use these data points for lookup

Evaluate on these points for different k, distances

K-Nearest Neighbors

- No learning going on but usually effective
- Same algorithm for every task
- As number of datapoints $\rightarrow \infty$, error rate is guaranteed to be at most 2x worse than optimal you could do on data
- Training is fast, but inference is slow. Opposite of what we want!

Next Time:
Linear Classification