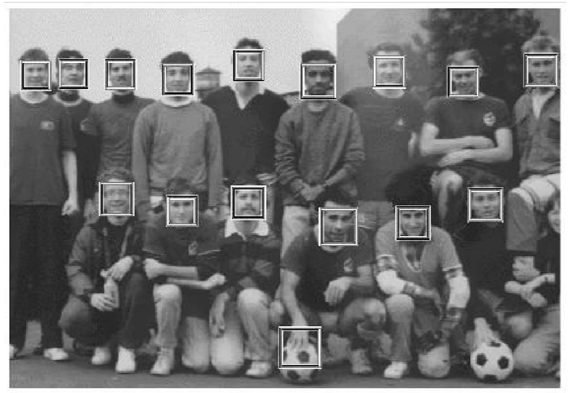
Intro to Machine Learning Algorithms

Outline

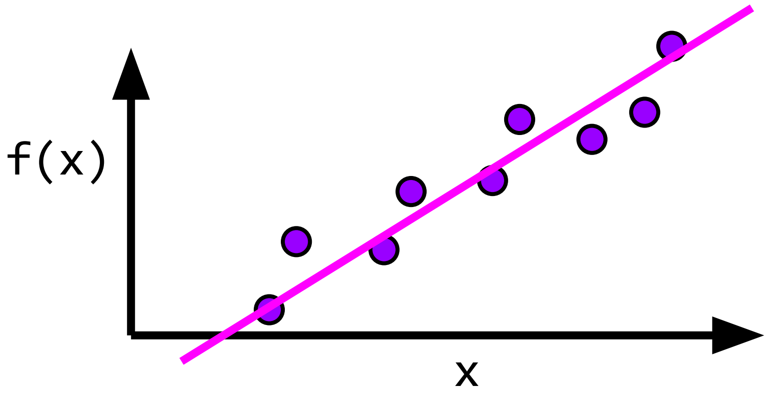
- Linear Regression
- K-nearest neighbour
- Decision Tree
- Logistic Regression



Acknowledgement : Most of slide credits go to CSE455, University of Washington and CS131, Stanford University

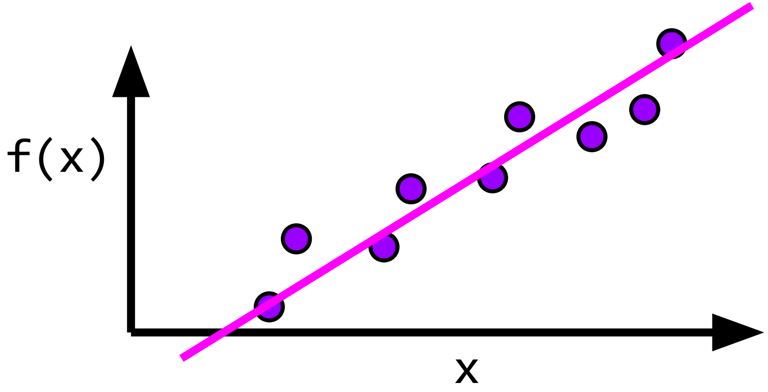
Linear regression

- $f^*(x) = ax + b$
- Learn a and b from data (how?)



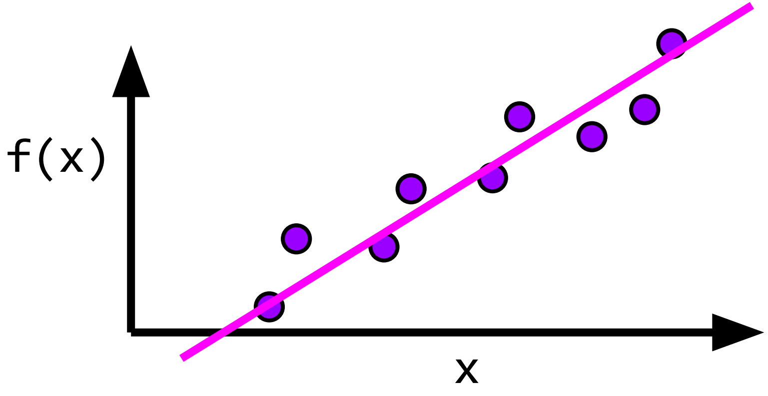
Linear regression

- $f^*(x) = ax + b$
- Learn a and b from data (how?)
  - Minimize squared error!
  - Loss function $L(f^*) = \Sigma_i \; ||f(x_i) - f^*(x_i)||^2$



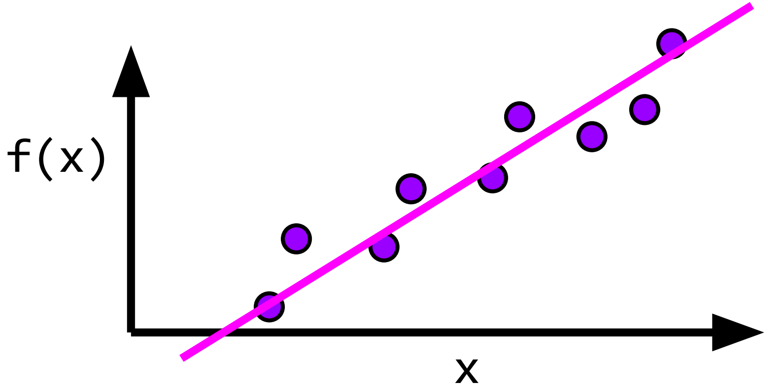
Linear regression

- $f^*(x) = ax + b$
- Learn a and b from data (how?)
  - Minimize squared error!
  - Loss function $L(f^*) = \Sigma_i \, ||f(x_i) - f^*(x_i)||^2$
  - Want $\text{argmin}_{a,b}[L(f^*)]$
  - Extrema when derivative = 0



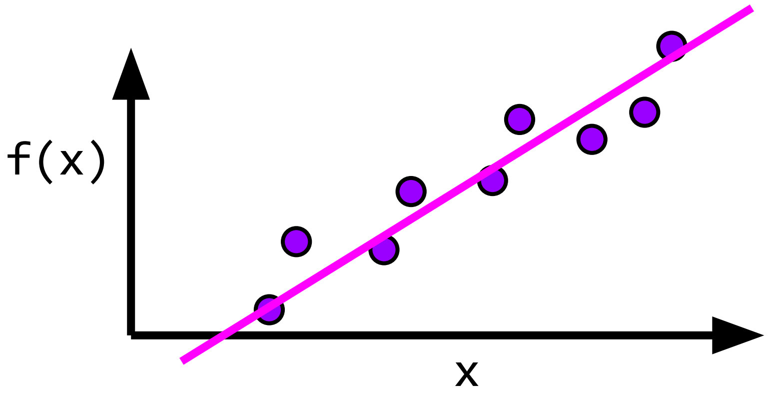
Linear regression

- $f^*(x) = ax + b*1$
- Learn a and b from data (how?)
  - Minimize squared error!
  - Loss function $L(f^*) = \Sigma_i \ ||f(x_i) - f^*(x_i)||^2$
  - Want $\text{argmin}_{a,b}[L(f^*)]$
  - Extrema when derivative = 0
  - Solve linear system of equations
    - **Ma** = **b**
  - Already did this!



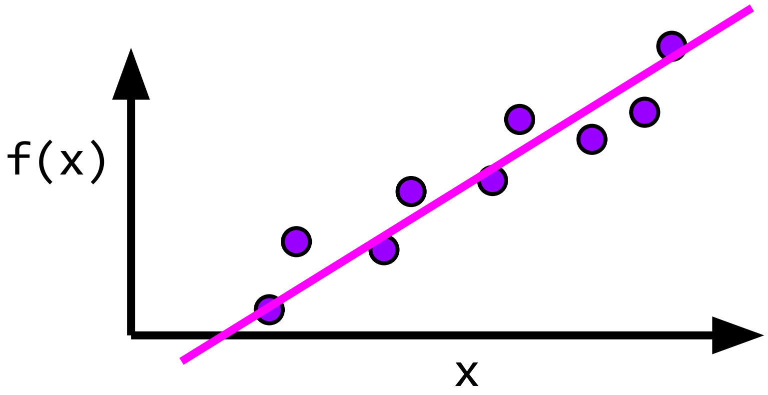
Linear regression

- f*(x) = ax + b
- Learn a and b from data (how?)
- High bias: linear assumption
- Low variance
- Benefits:
  - Closed form solution
  - Fast to compute for new data



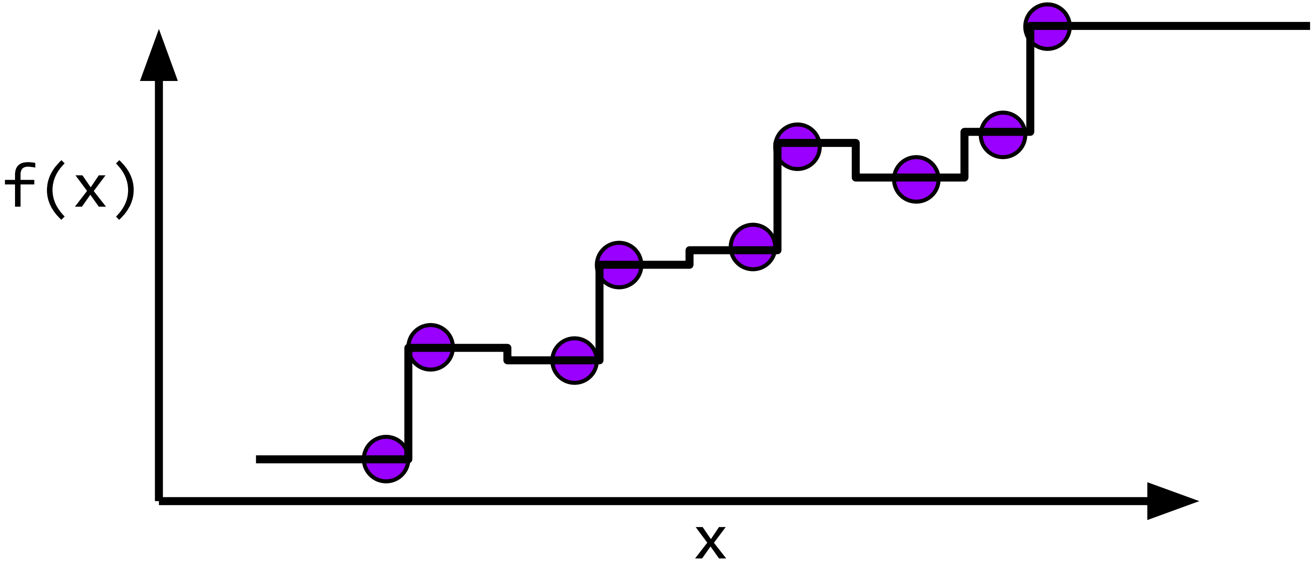
Linear regression

- $f^*(x) = ax + b$
- Learn a and b from data (how?)
- High bias: linear assumption
- Low variance
- Benefits:
  - Closed form solution
  - Fast to compute for new data
- Weaknesses:
  - Not very powerful, **assumes linear**
  - **Underfit** more interesting data



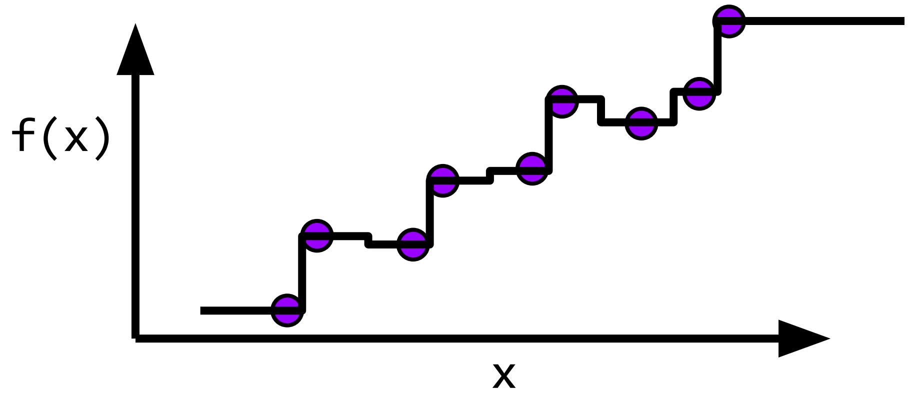
Nearest neighbor

- f*(x) = f(x’) for nearest x’ in training set



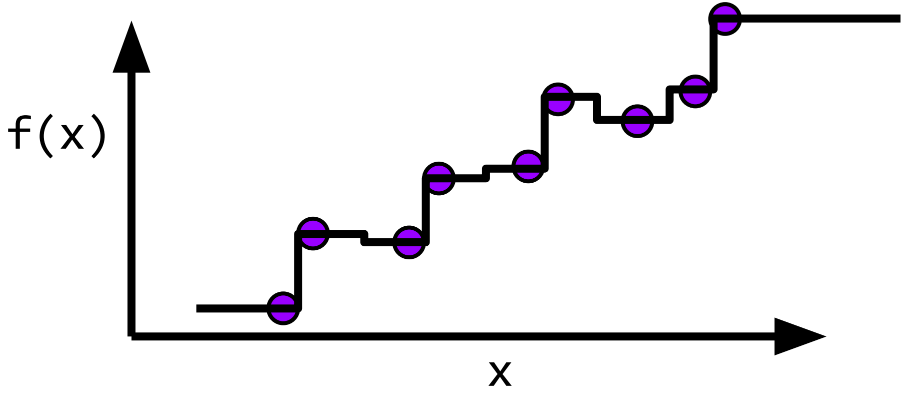
Nearest neighbor

- $f^*(x) = f(x')$ for nearest x’ in training set
- Low bias: no assumptions about data
- High variance: very sensitive to training set



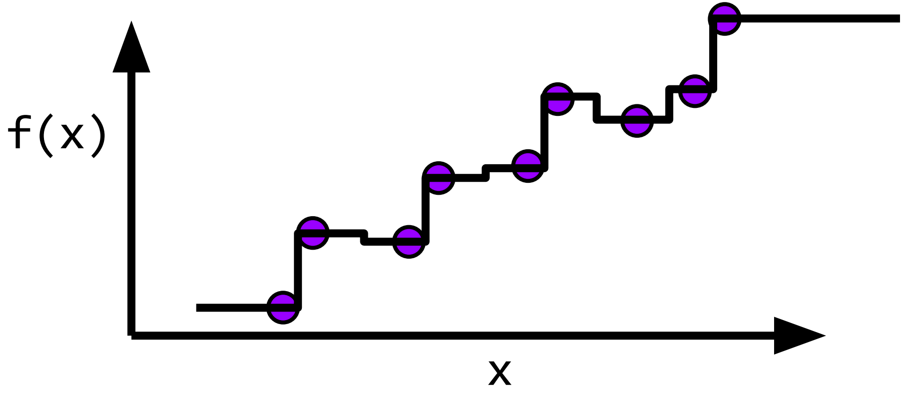
Nearest neighbor

- f*(x) = f(x’) for nearest x’ in training set
- Low bias: no assumptions about data
- High variance: very sensitive to training set
- Benefits:
    - Super easy to implement
    - Easy to understand
    - Arbitrarily powerful, esp with lots of data



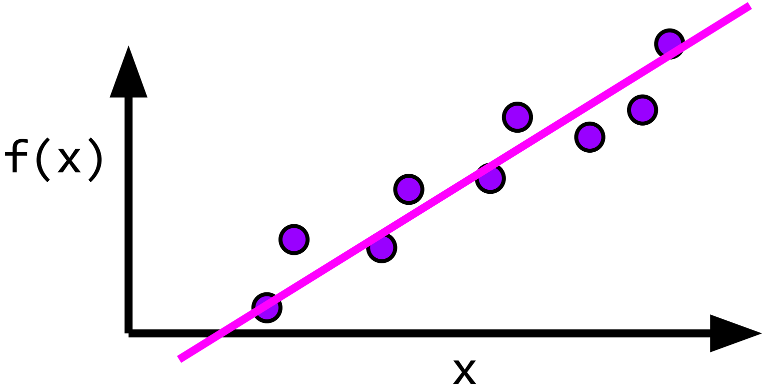
Nearest neighbor

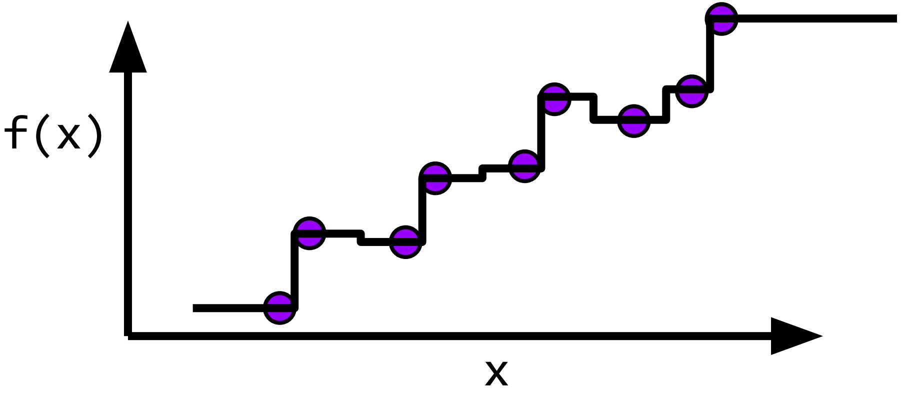
- $f^*(x) = f(x')$ for nearest x' in training set
- Low bias: no assumptions about data
- **High variance**: very sensitive to training set
- Benefits:
  - Super easy to implement
  - Easy to understand
  - Arbitrarily powerful, esp with lots of data
- Weaknesses:
  - Hard to scale
  - Prone to **overfitting to noise**



These are examples of *regression*

- Given training data
  - input variables **X**, output variables **Y**
- And new data point x’
- Predict corresponding output variables y’





A different task: *Classification*

- Training data: points associated with a class
  - Also other data about that point
- Example: Does patient have the flu?
  - Binary classification (yes or no)
  - Different types of variables (continuous, discrete)

sore throat	runny nose	nausea	temp	chills	pain	age	days	diagnosis
no	yes	yes	101.3	yes	7	15	5	flu
yes	yes	no	98.8	no	3	74	3	not flu
yes	yes	no	100.1	yes	4	46	4	flu
yes	yes	yes	99.8	yes	6	27	1	flu
yes	no	no	98.4	yes	5	35	2	not flu
yes	yes	yes	99.0	no	3	42	4	not flu

One approach: partitions

- Find best split or splits to data along one variable
- One possibility, Pr(flu) = (temp > 99.5)
  - Pretty accurate on our training data

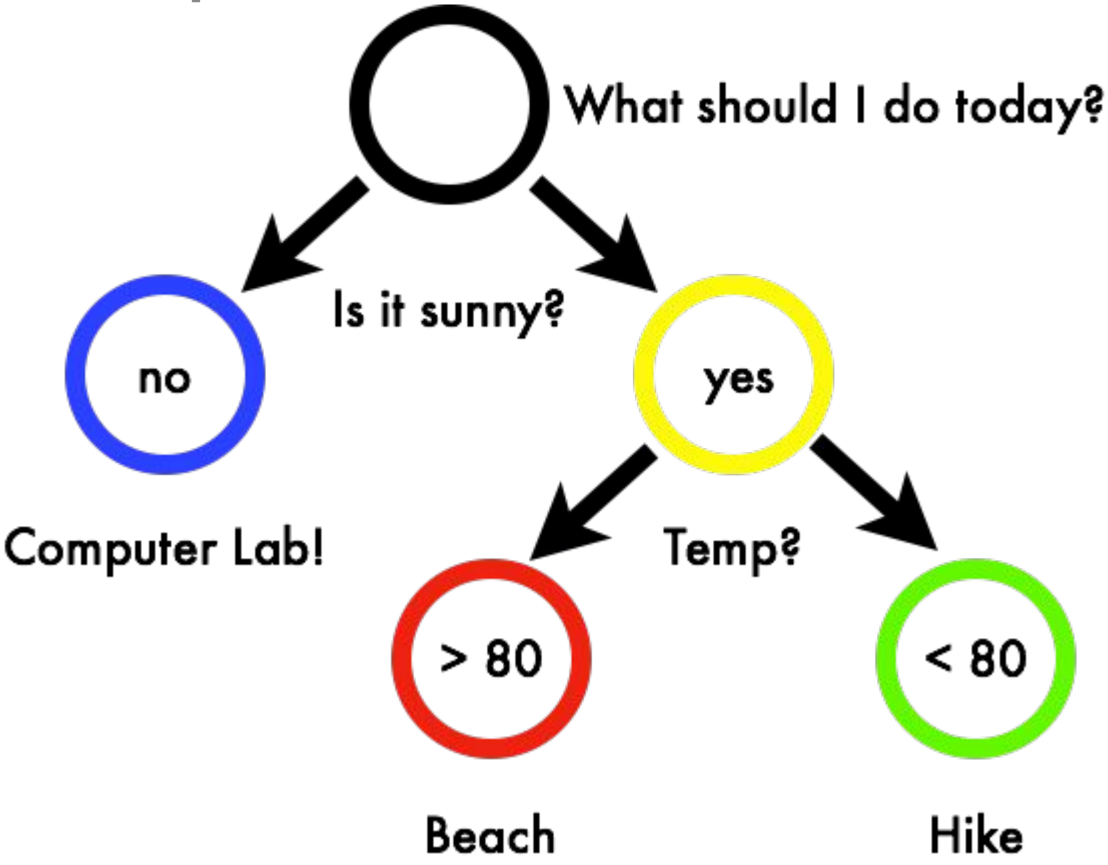
sore throat	runny nose	nausea	temp	chills	pain	age	days	diagnosis
no	yes	yes	101.3	yes	7	15	5	flu
yes	yes	no	98.8	no	3	74	3	not flu
yes	yes	no	100.1	yes	4	46	4	flu
yes	yes	yes	99.8	yes	6	27	1	flu
yes	no	no	98.4	yes	5	35	2	not flu
yes	yes	yes	99.9	no	3	42	4	not flu

Trees: layers of partitions

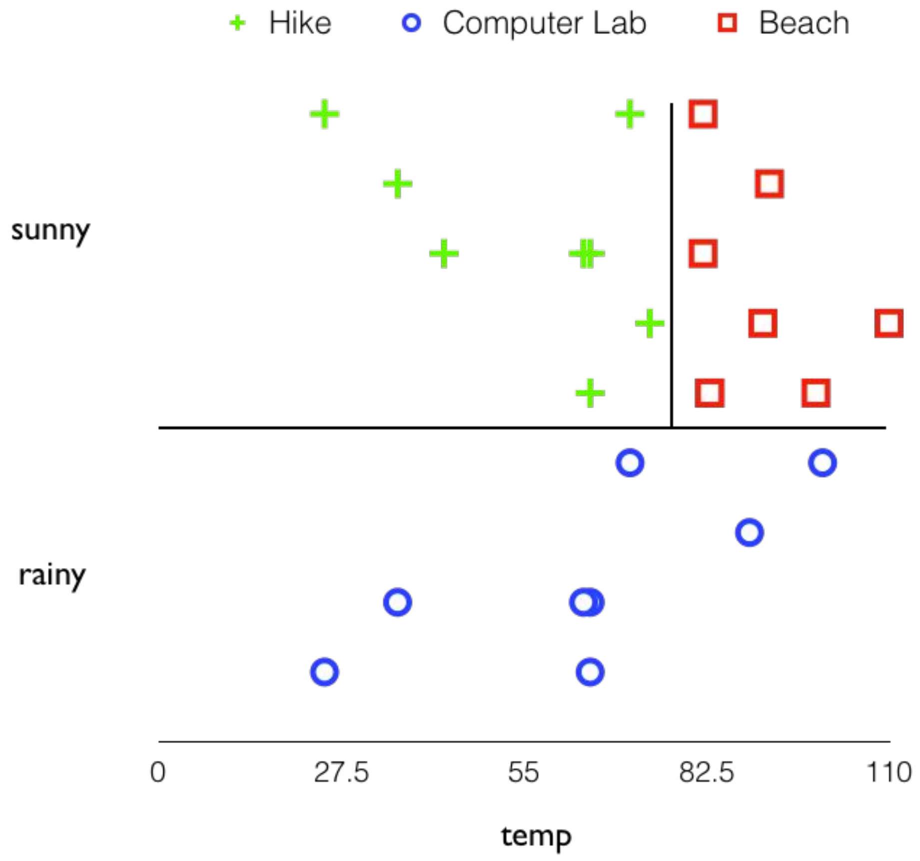
Very simple models

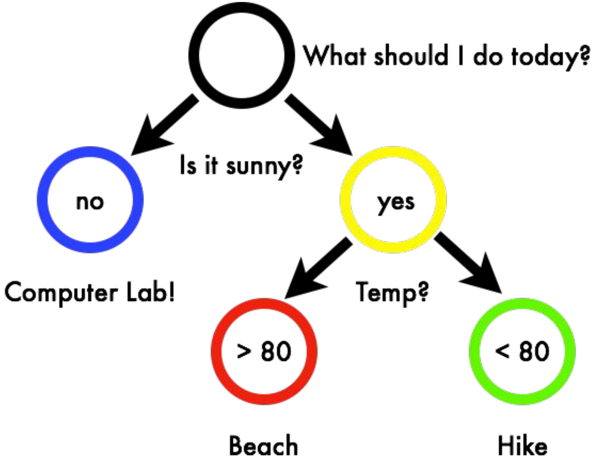
Benefits:

- Interpretable
- Easy to use
- Good for applications
  - E.g. **medicine**

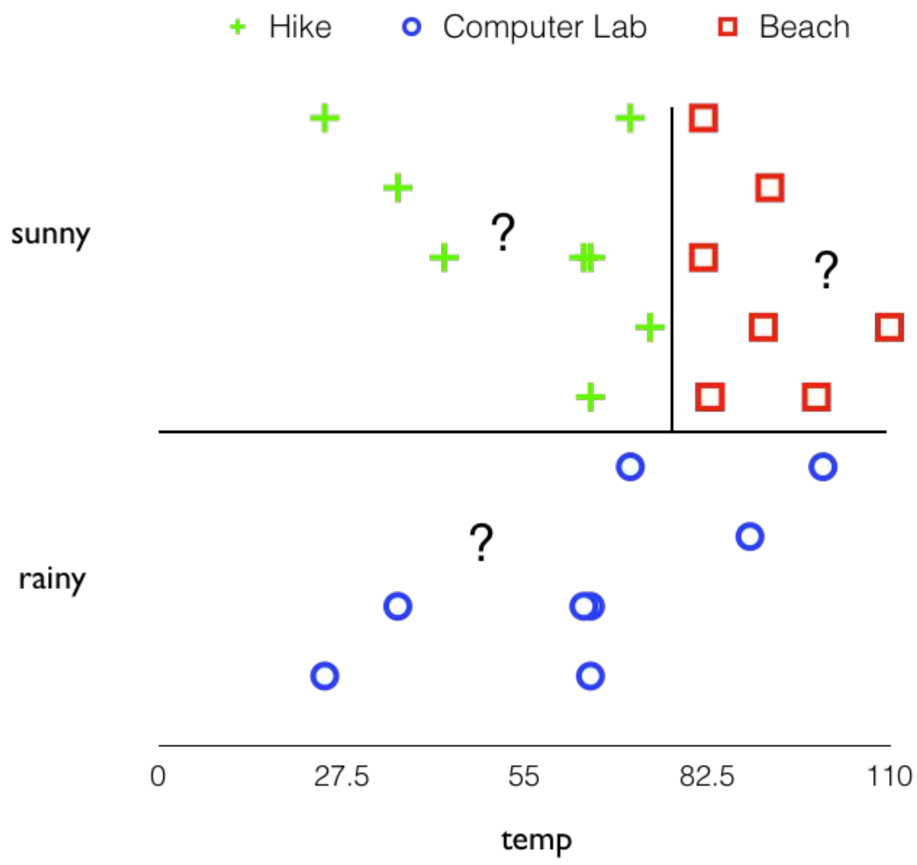


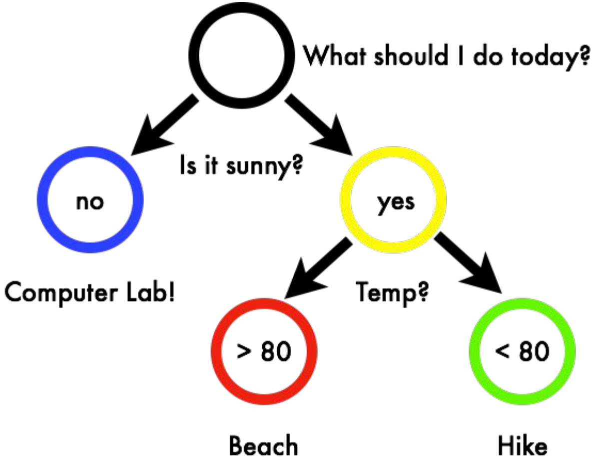
Trees are partitions of data



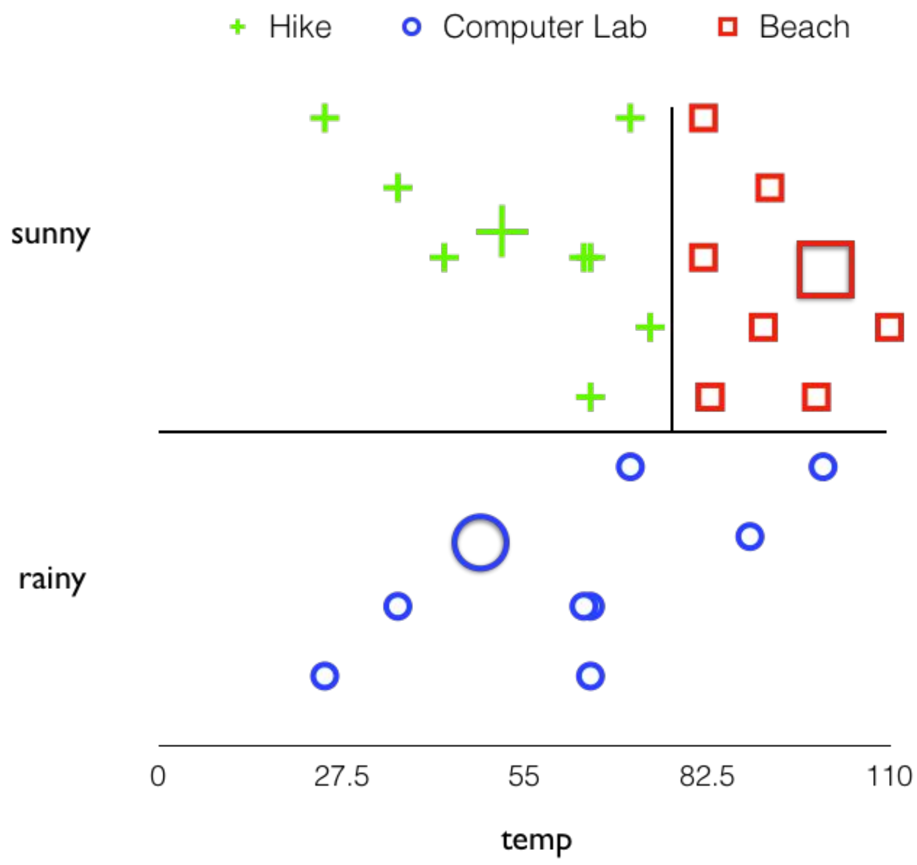


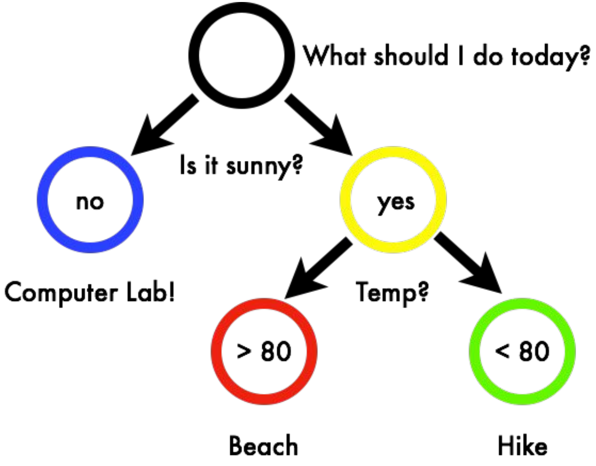
Predict new data based on what region it falls into



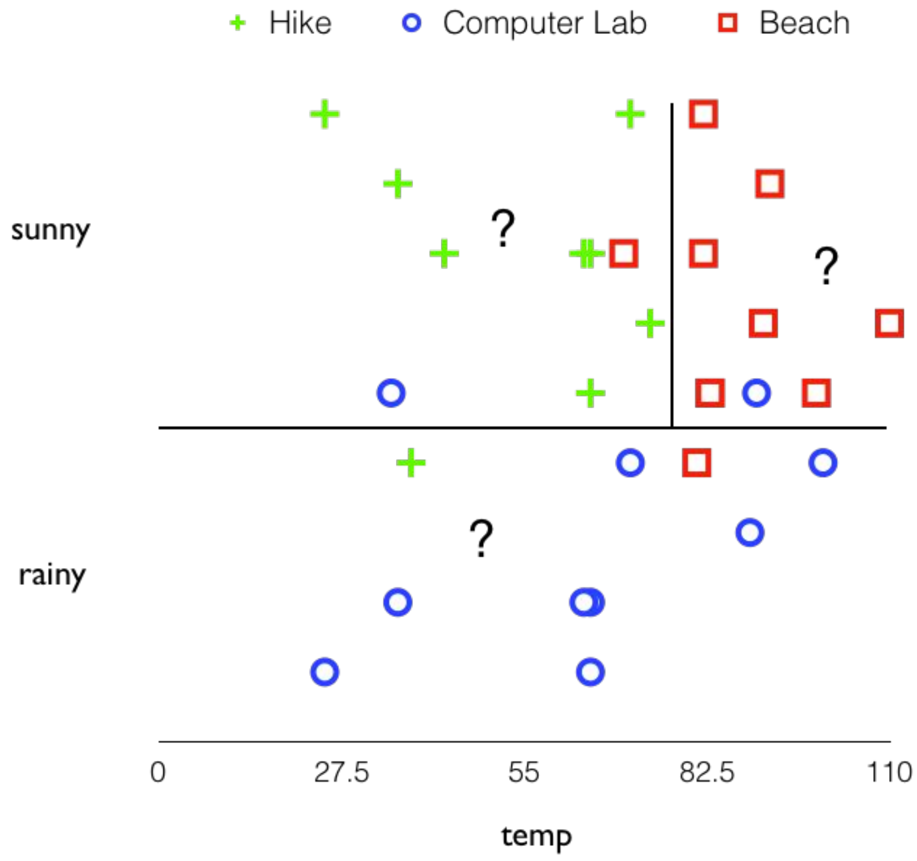


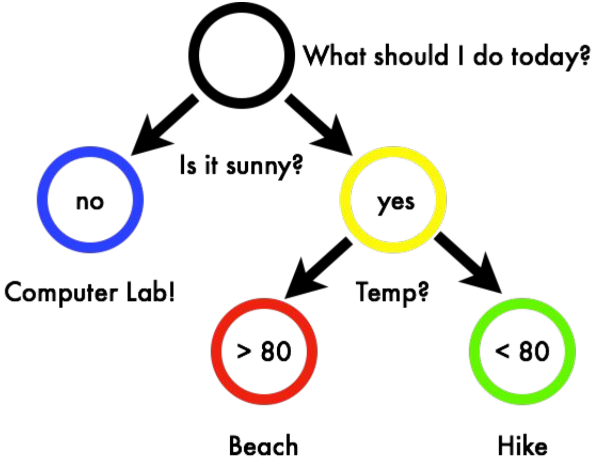
Predict new data based on what region it falls into



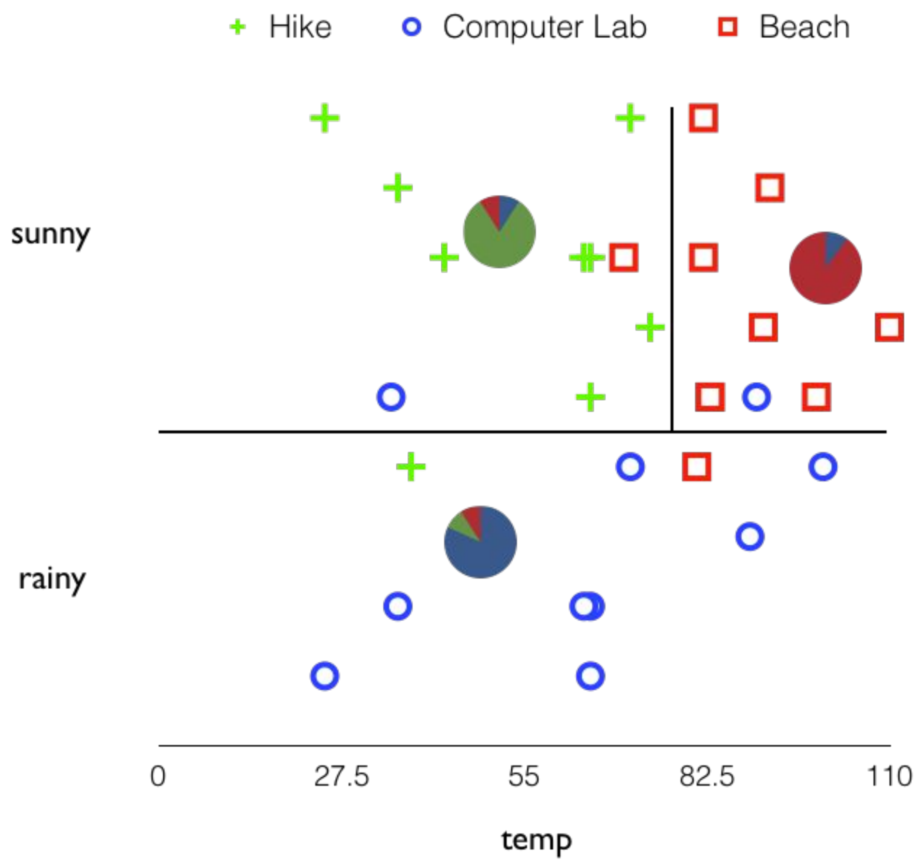


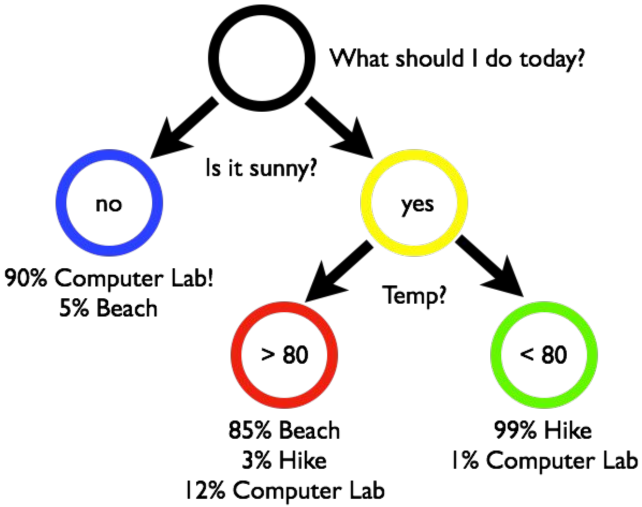
Data might be noisy, use soft assignments





Data might be noisy, use soft assignments





Case study: Viola-Jones Face detection

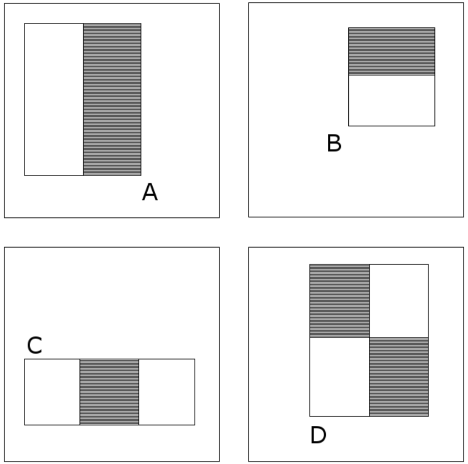
Want it to be very fast and accurate

Run on a camera or cell phone, low cost

Use simple features and simple classifiers

Haar features:

Response = Σ pix in black region - Σ pix in white region



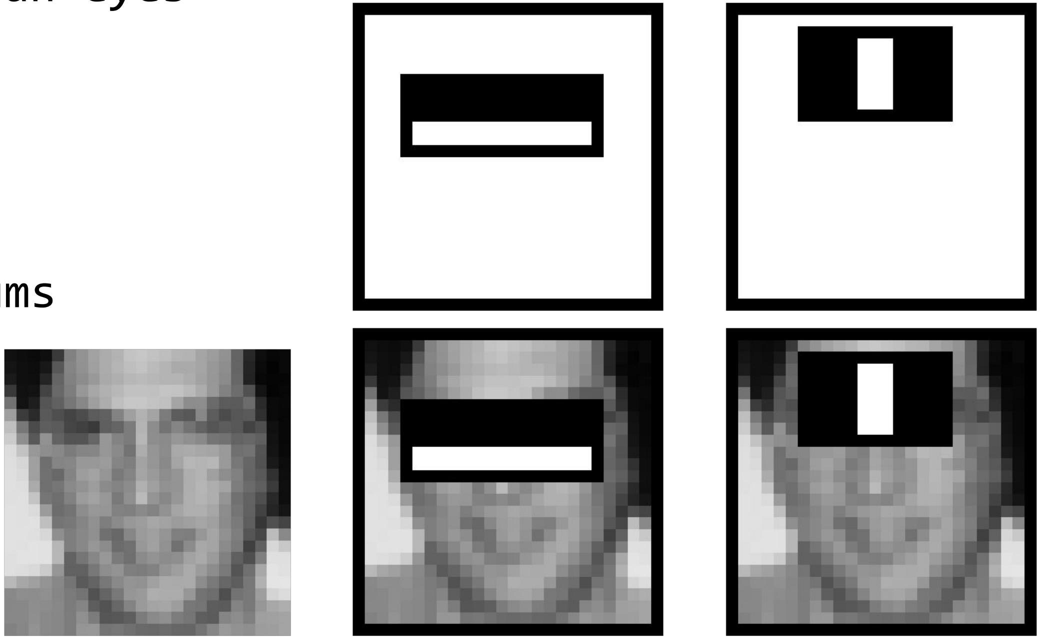
Case study: Viola-Jones Face detection

Why do Haar features work?

- Eyes are generally darker than cheeks
- Bridge of nose lighter than eyes
- Etc.

Also, fast to compute!

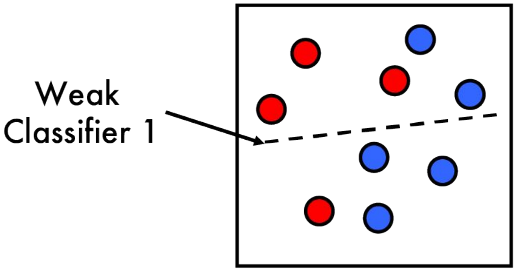
- *Integral images* - fast sums over regions.



https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

Case study: Viola-Jones Face detection

Classifier: *boosted* partitions



Boosting

Way to make weak classifiers better

Train a weak classifier

https://www.cs.ubc.ca/~lowe/425/slides/13-ViolaJones.pdf

Case study: Viola-Jones Face detection

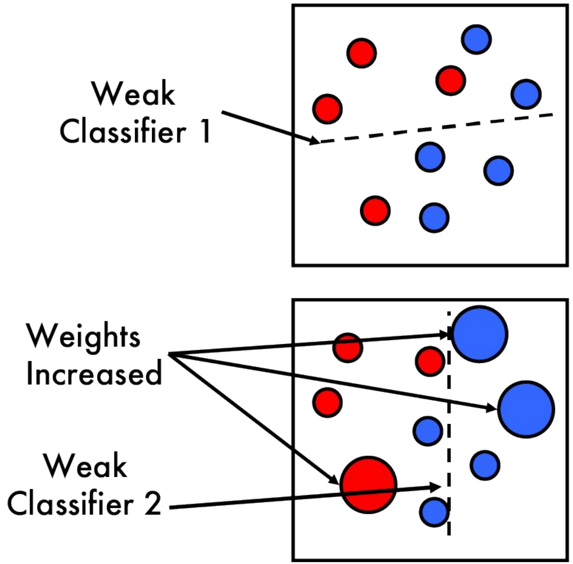
Classifier: *boosted* partitions

Boosting

Way to make weak classifiers better

Train a weak classifier

Reweight data we got wrong, train again



Classifier: *boosted* partitions

Boosting

Way to make weak classifiers better

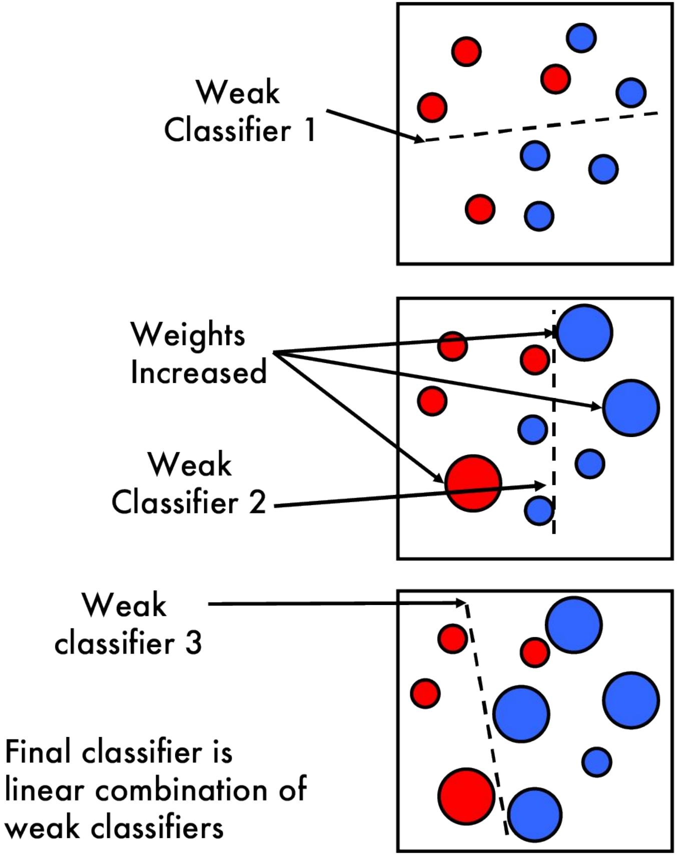
Train a weak classifier

Reweight data we got wrong, train again

...and again

Until you feel like stopping

Final classifier is combination of all



Case study: Viola-Jones Face detection

Finally, use a **cascade of classifiers**

1st classifier

Very fast, throws out easy negatives

2nd classifier

Fast, throws out harder negatives

3rd classifier

Slower, throws out hard negatives

Only run slow, good classifiers on hard examples

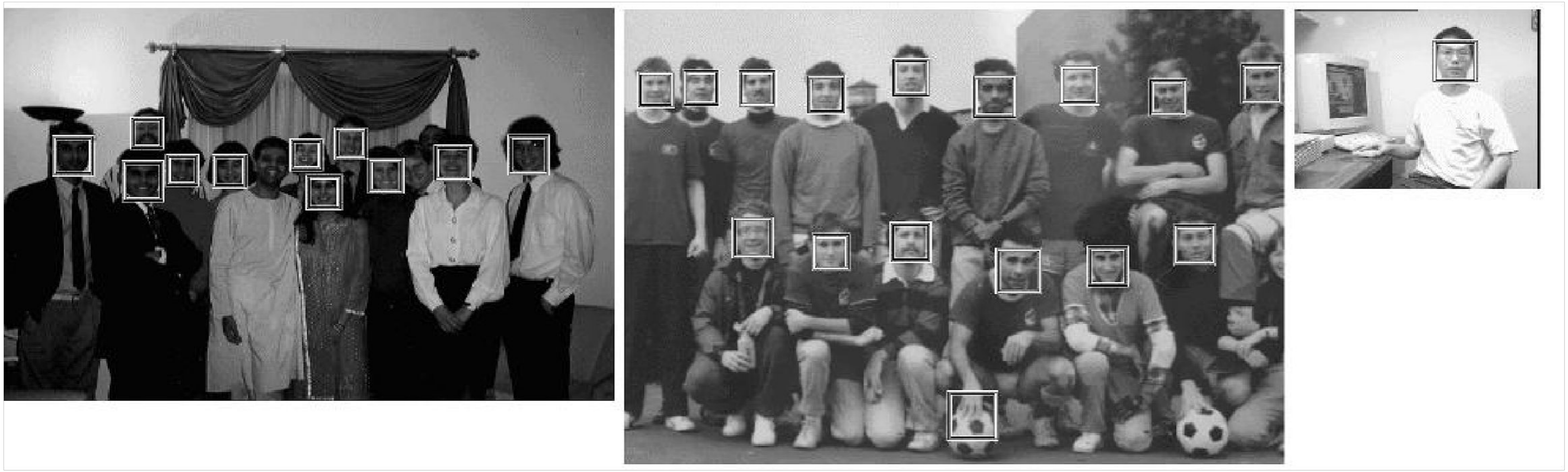
Fast classifier that is still very accurate

Case study: Viola-Jones Face detection

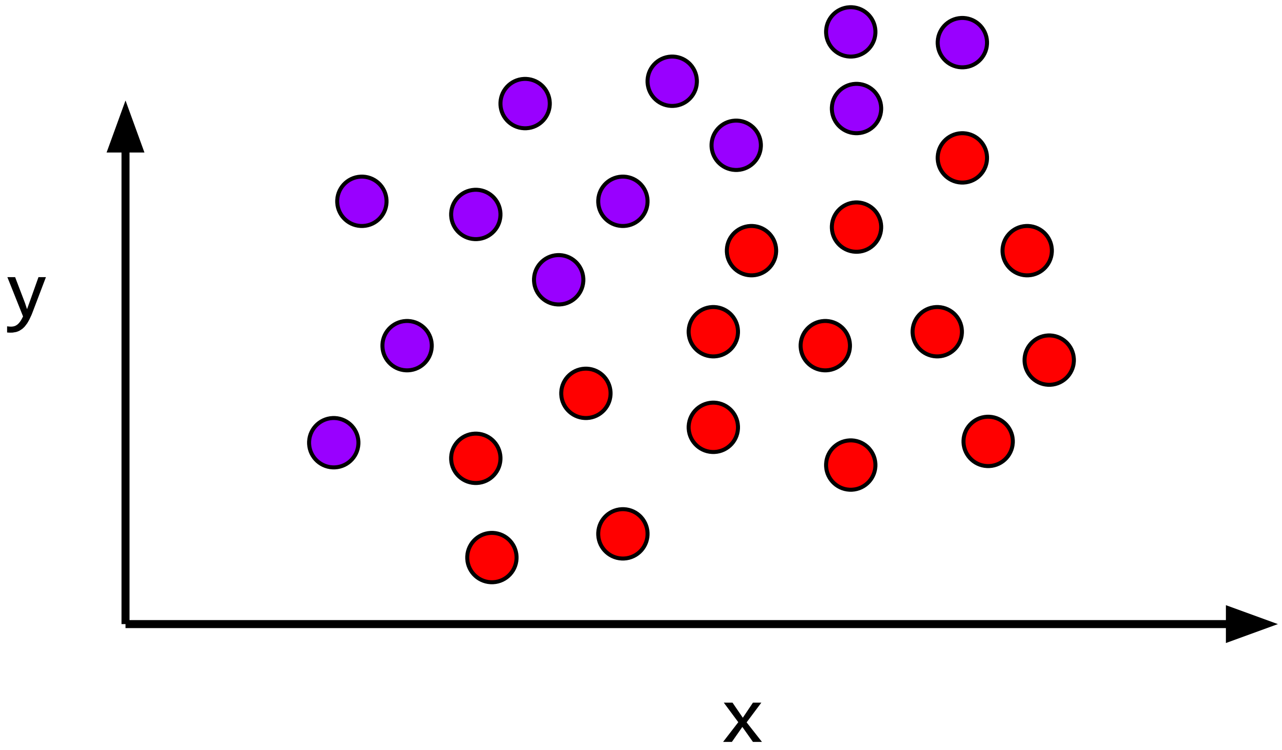
Haar features

Cascade

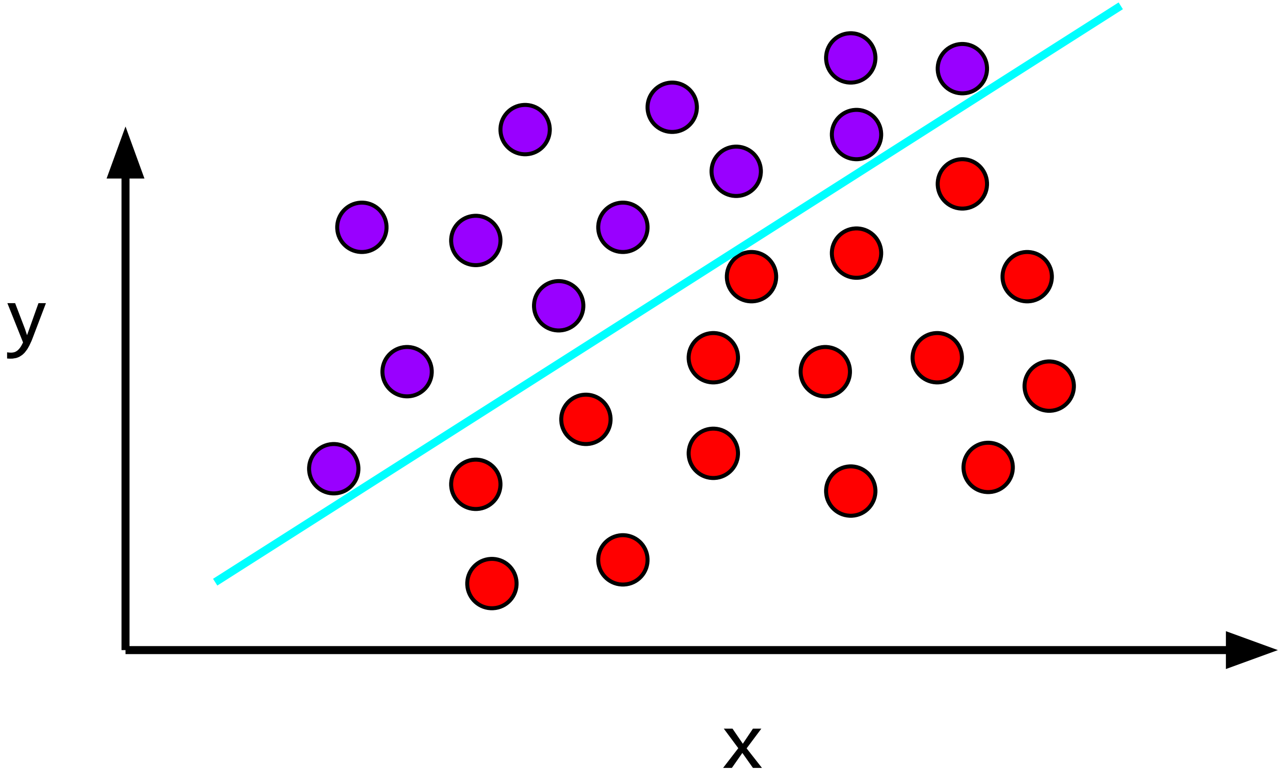
- Of boosted classifiers



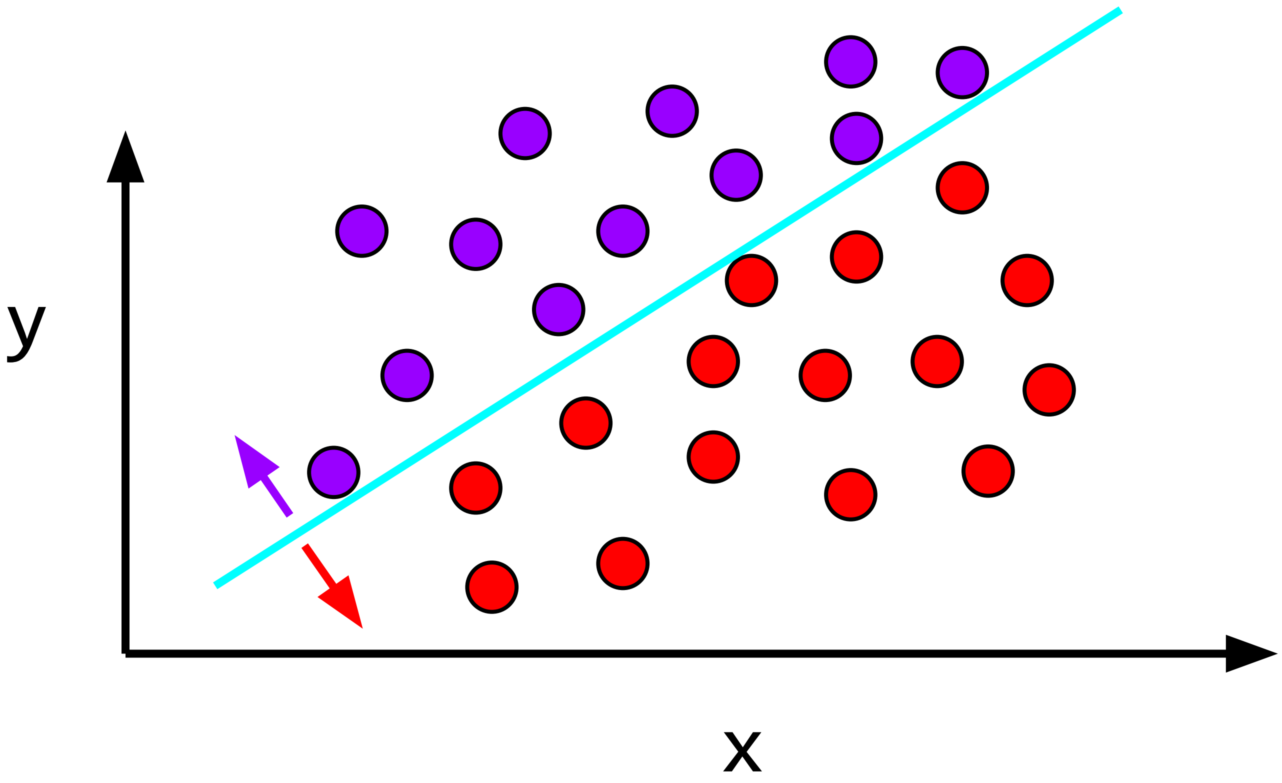
Classification in two dimensions



Classification in two dimensions

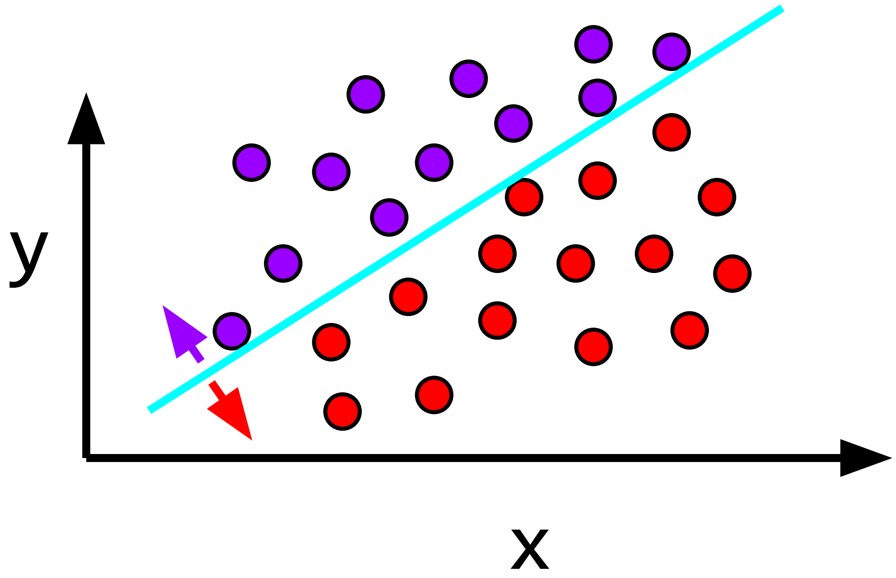


Classification in two dimensions



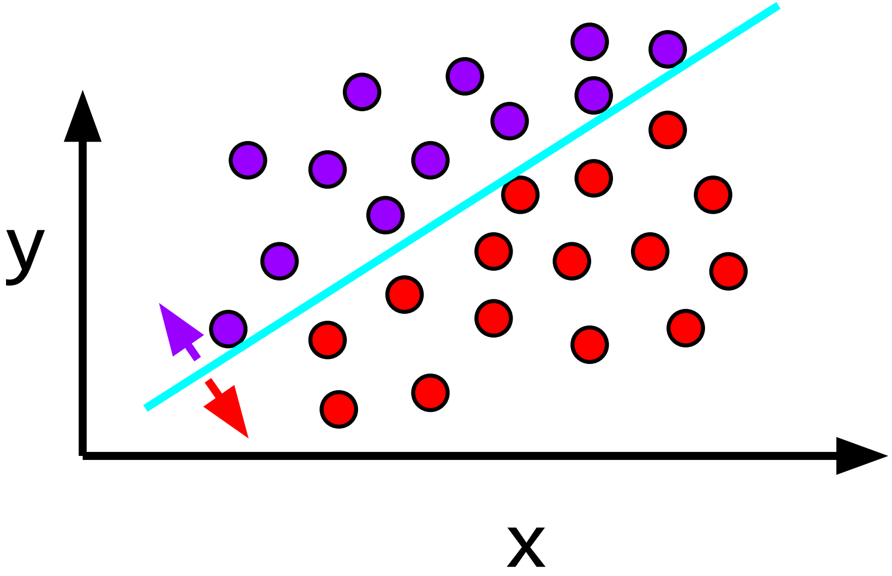
Classification in two dimensions

- Linear classifier
  - Given dataset, learn weights **w**
  - Output of model is weighted sum of inputs
  - $P(\text{purple} \mid \mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x}) = f(\Sigma_i (w_i x_i))$
  - Where f is some function (a few options)



Classification in two dimensions

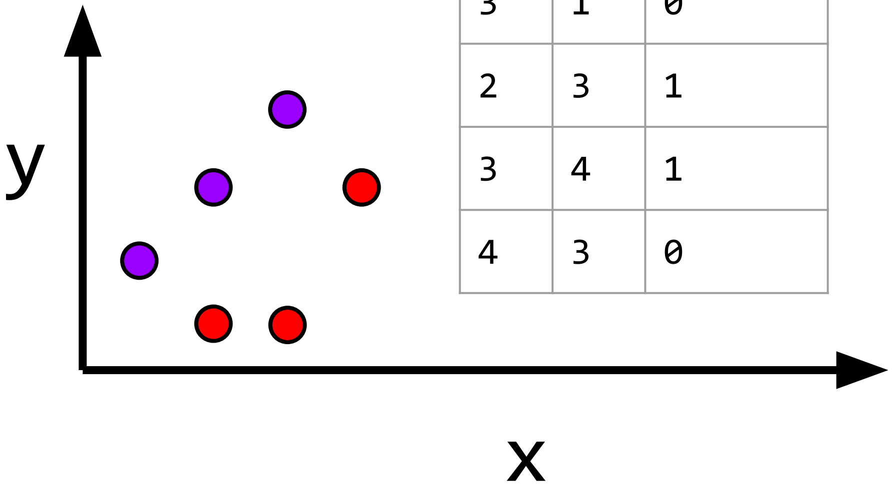
- Linear classifier
  - Given dataset, learn weights **w**
  - Output of model is weighted sum of inputs
  - P(purple | **x**) = f(**w**·**x**) = $f(\Sigma_i(w_i x_i))$
  - Where f is some function (a few options)
  - Typically a bias term:
    - $f(\Sigma_i(w_i x_i) + w_{bias})$



Classification in two dimensions

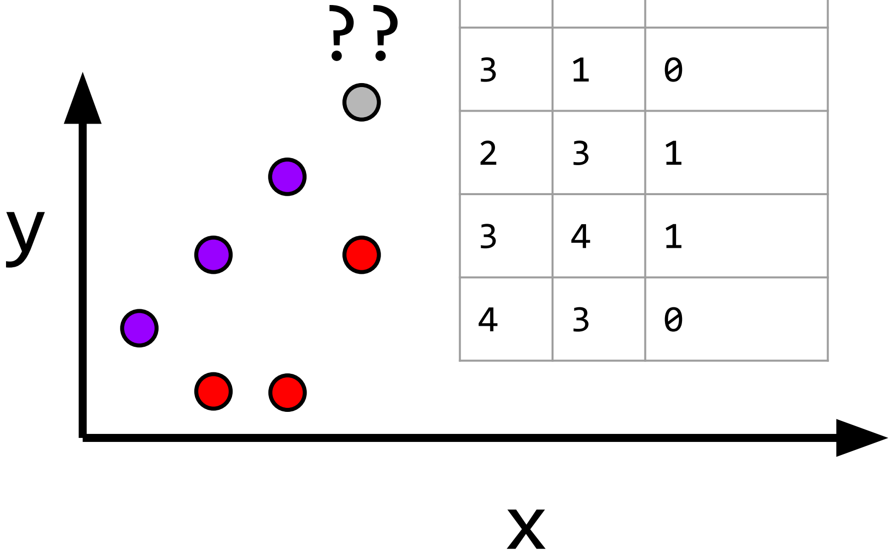
- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
  -

x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0



Classification in two dimensions

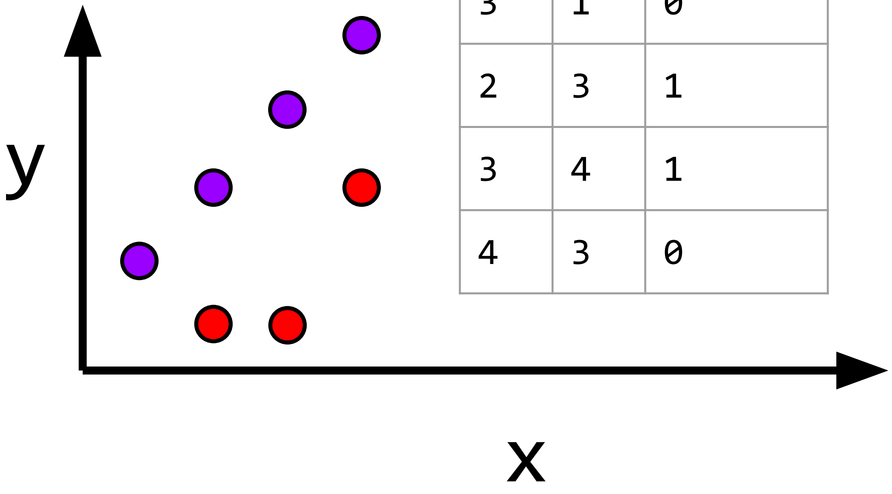
- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - (**w**·**x**) = (4,5)·(-1, 1)
    = 4*-1 + 5*1 = 1



x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0

Classification in two dimensions

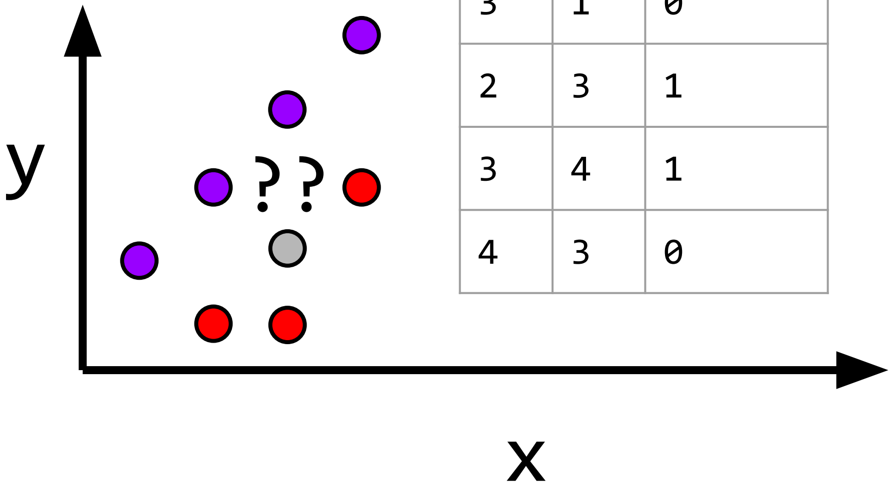
- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - (**w**·**x**) = (4,5)·(-1, 1)
    = 4*-1 + 5*1 = 1
  - f(**w**·**x**) = f(1) = 1



x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0

Classification in two dimensions

- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - $f(\mathbf{w}\cdot\mathbf{x}) = 1$
- New data point (3, 2)
  - $(\mathbf{w}\cdot\mathbf{x}) = (3,2)\cdot(-1, 1)$
    $= 3*-1 + 2*1 = -1$
  - $f(\mathbf{w}\cdot\mathbf{x}) = f(-1) = 0$
  -

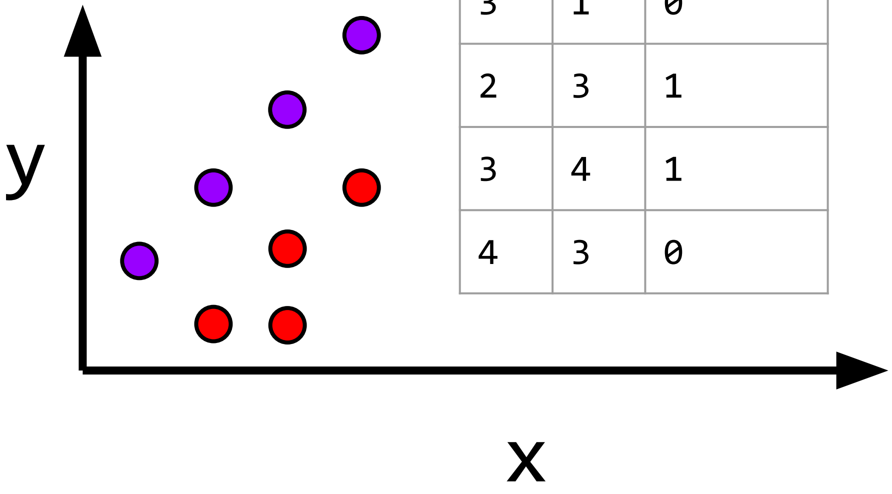


x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0

Classification in two dimensions

- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - f(**w**·**x**) = 1
- New data point (3, 2)
  - f(**w**·**x**) = 0

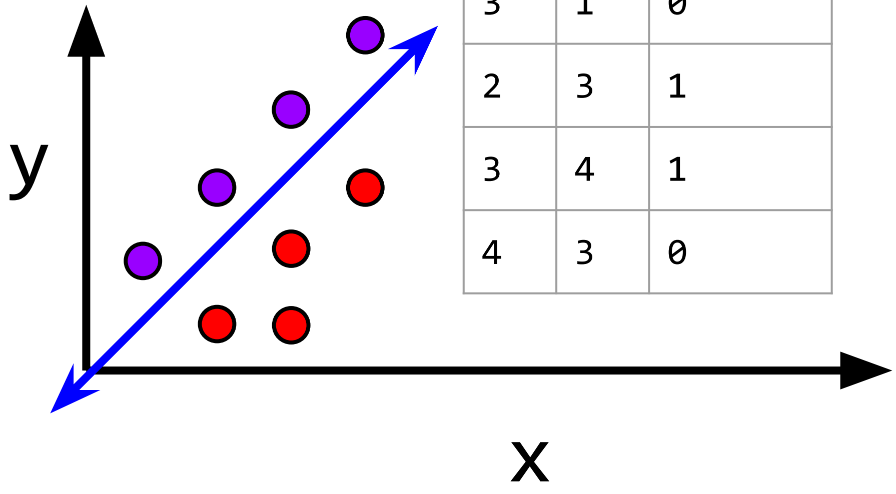
x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0



Classification in two dimensions

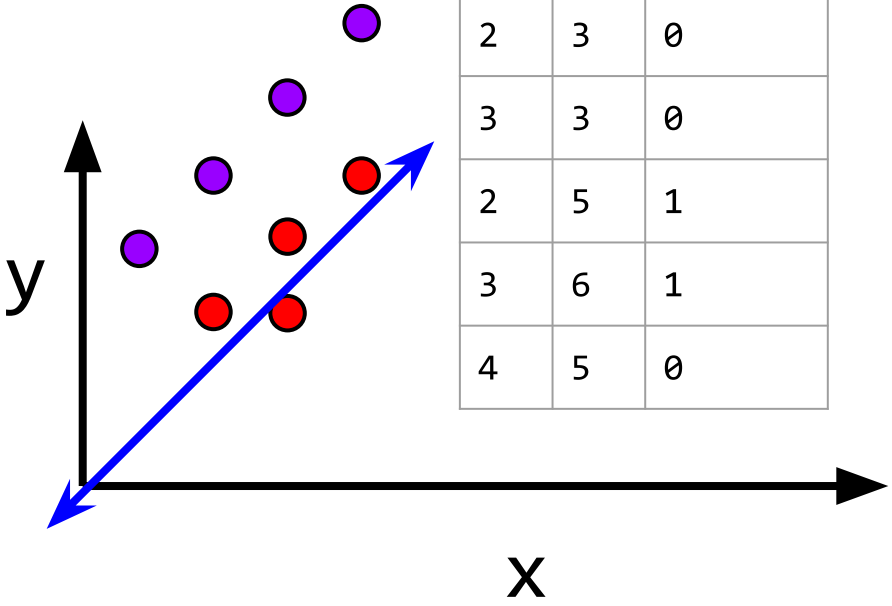
- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - $f(\mathbf{w} \cdot \mathbf{x}) = 1$
- New data point (3, 2)
  - $f(\mathbf{w} \cdot \mathbf{x}) = 0$
- Decision boundary: x=y

x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0



What if data is shifted up by two?

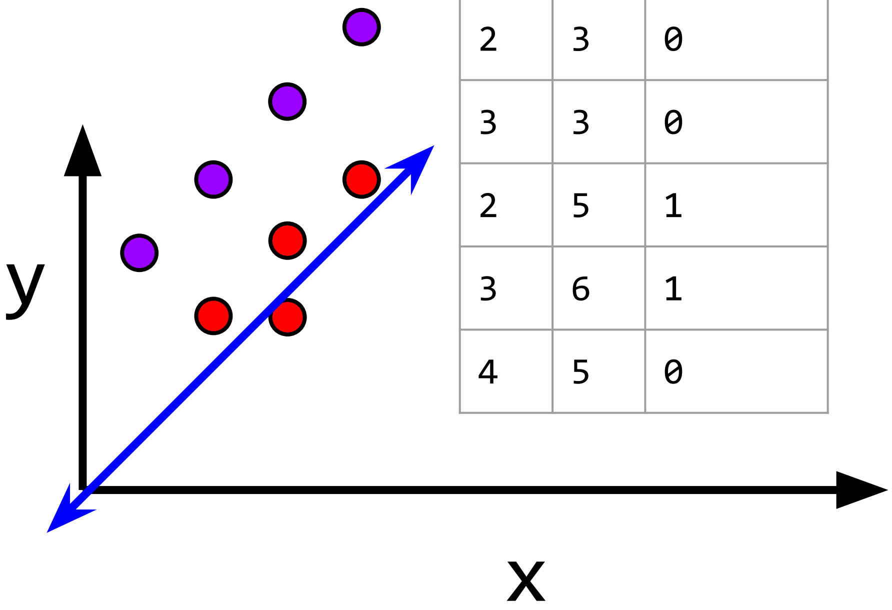
- Need a bias!:
  - Learned weights: [-1, 1]
  - f is threshold at 0



x	y	purple
1	4	1
2	3	0
3	3	0
2	5	1
3	6	1
4	5	0

What if data is shifted up by two?

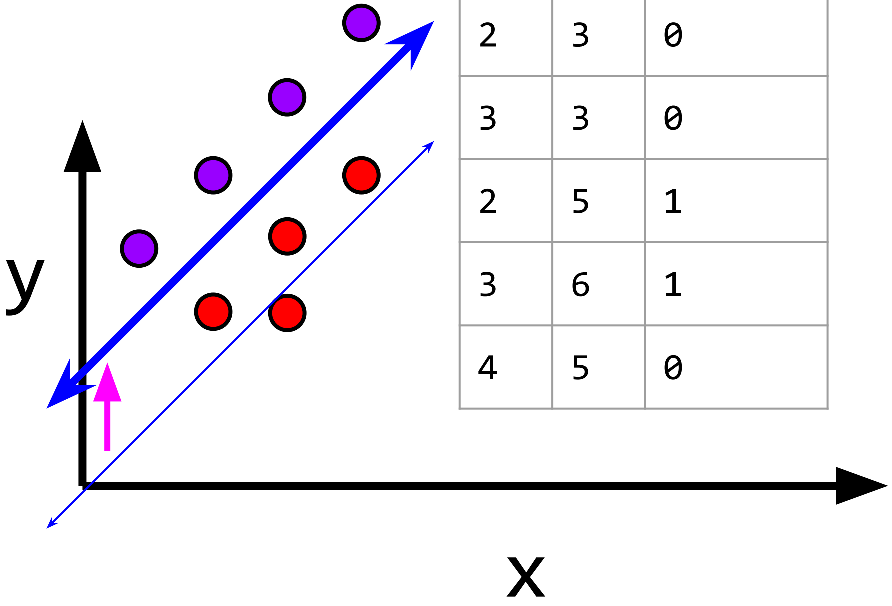
- Need a bias!:
  - Learned weights: [-1, 1, -2]
  - f is threshold at 0



x	y	purple
1	4	1
2	3	0
3	3	0
2	5	1
3	6	1
4	5	0

What if data is shifted up by two?

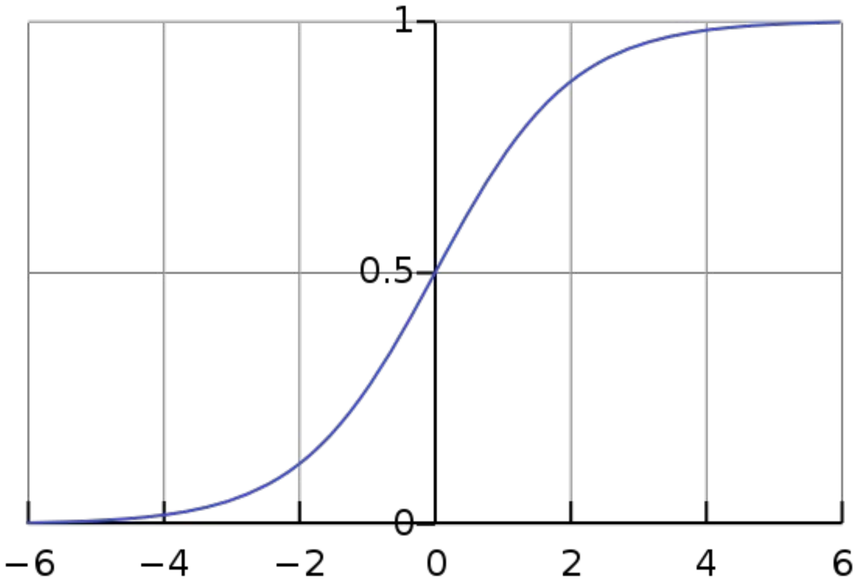
- Need a bias!:
  - Learned weights: [-1, 1, -2]
  - f is threshold at 0
- New data point (4, 7, 1)
  - (w·x) = (4,7,1)·(-1,1,-2)
    = 4*-1 + 7*1 + 1*-2 = 1
  - f(w·x) = f(1) = 1



x	y	purple
1	4	1
2	3	0
3	3	0
2	5	1
3	6	1
4	5	0

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^x/(1 + e^x)$
  - Maps all reals -> [0,1], **probabilities!**



Logistic regression

- Linear classifier, f is logistic function
    - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
    - Good choice: how well our model fits the data, likelihood

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - **X**: training input variables, **Y**: training dependant variables
  - Want to learn **w** that models training data well

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^x/(1 + e^x)$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - **X**: training input variables, **Y**: training dependant variables
  - Want to learn **w** that models training data well
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \Pr(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \prod_i \Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w})$

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^x/(1 + e^x)$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - **X**: training input variables, **Y**: training dependant variables
  - Want to learn **w** that models training data well
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = Pr(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \mathbf{\Pi}_i Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w})$
  - $Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w}) =$
    - If $Y_i = 1$, $\sigma(\mathbf{w}\cdot\mathbf{X}_i)$

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - **X**: training input variables, **Y**: training dependant variables
  - Want to learn **w** that models training data well
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \Pr(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \mathbf{\Pi}_i \Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w})$
  - $\Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w}) =$
    - If $\mathbf{Y}_i = 1$, $\sigma(\mathbf{w}\cdot\mathbf{X}_i)$
    - If $\mathbf{Y}_i = 0$, $1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i)$

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - **X**: training input variables, **Y**: training dependant variables
  - Want to learn **w** that models training data well
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \Pr(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \boldsymbol{\Pi}_i \Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w})$
  - $\Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w}) =$
    - If $\mathbf{Y}_i = 1$, $\sigma(\mathbf{w}\cdot\mathbf{X}_i)$
    - If $\mathbf{Y}_i = 0$, $1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i)$
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \boldsymbol{\Pi}_i[(\sigma(\mathbf{w}\cdot\mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i))^{(1-Yi)}]$

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - Good choice: how well our model fits the data, likelihood
  - $\mathbf{X}$: training input variables, $\mathbf{Y}$: training dependant variables
  - Want to learn $\mathbf{w}$ that models training data well
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = Pr(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \mathbf{\Pi}_i Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w})$
  - $Pr(\mathbf{Y}_i \mid \mathbf{X}_i,\mathbf{w}) =$
    - If $\mathbf{Y}_i = 1$, $\sigma(\mathbf{w}\cdot\mathbf{X}_i)$
    - If $\mathbf{Y}_i = 0$, $1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i)$
  - $L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \mathbf{\Pi}_i[(\sigma(\mathbf{w}\cdot\mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i))^{(1-Yi)}]$
  - In practice we use log likelihood, it’s simpler later!!

Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - $\log L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \log \mathbf{\Pi}_i[(\sigma(\mathbf{w}\cdot\mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i))^{(1-Yi)}]$
  - $= \Sigma_i \log[(\sigma(\mathbf{w}\cdot\mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i))^{(1-Yi)}]$
  - $= \Sigma_i [\mathbf{Y}_i\log(\sigma(\mathbf{w}\cdot\mathbf{X}_i)) + (1-\mathbf{Y}_i)\log(1 - \sigma(\mathbf{w}\cdot\mathbf{X}_i))]$

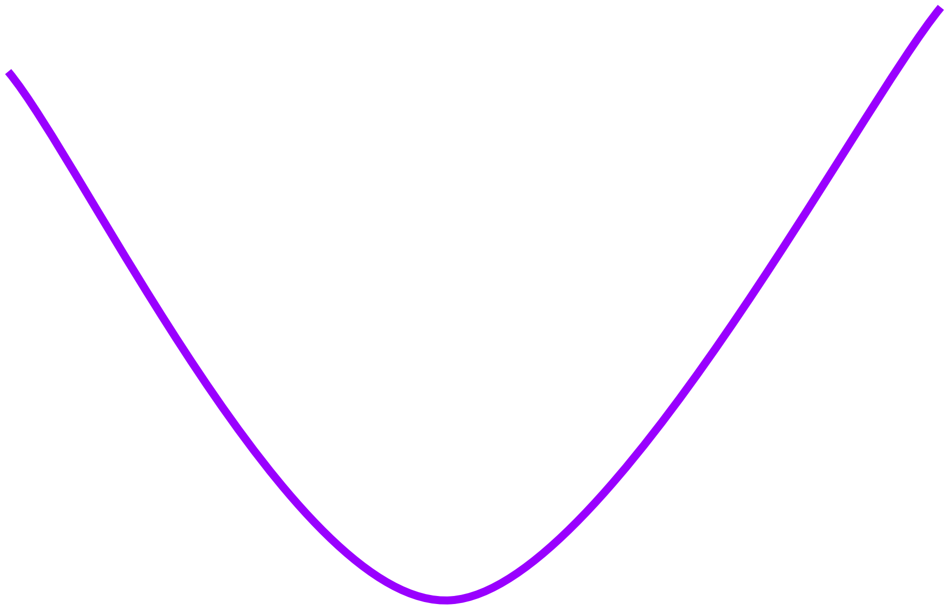
Logistic regression

- Linear classifier, f is logistic function
  - $\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$
- Want something to optimize!
  - $\log L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \log \prod_i [(\sigma(\mathbf{w} \cdot \mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w} \cdot \mathbf{X}_i))^{(1-Yi)}]$
  - $= \Sigma_i \log[(\sigma(\mathbf{w} \cdot \mathbf{X}_i))^{Yi} * (1 - \sigma(\mathbf{w} \cdot \mathbf{X}_i))^{(1-Yi)}]$
  - $= \Sigma_i [\mathbf{Y}_i \log(\sigma(\mathbf{w} \cdot \mathbf{X}_i)) + (1-\mathbf{Y}_i)\log(1 - \sigma(\mathbf{w} \cdot \mathbf{X}_i))]$
- Can we take derivative and set to 0?
  - No! :-(no closed form solution
  - BUT! We can still optimize

Gradient descent

For some loss function $L(\mathbf{w})$, gradient $\nabla L(\mathbf{w})$ points towards in direction of steepest ascent.

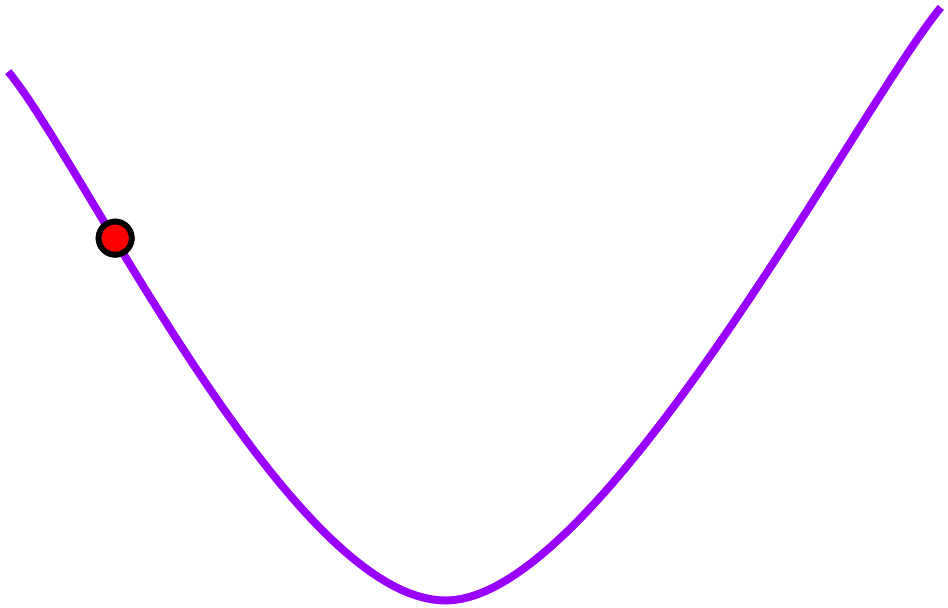
In 1d, either points left or right



Gradient descent

For some loss function $L(\mathbf{w})$, gradient $\nabla L(\mathbf{w})$ points towards in direction of steepest ascent.

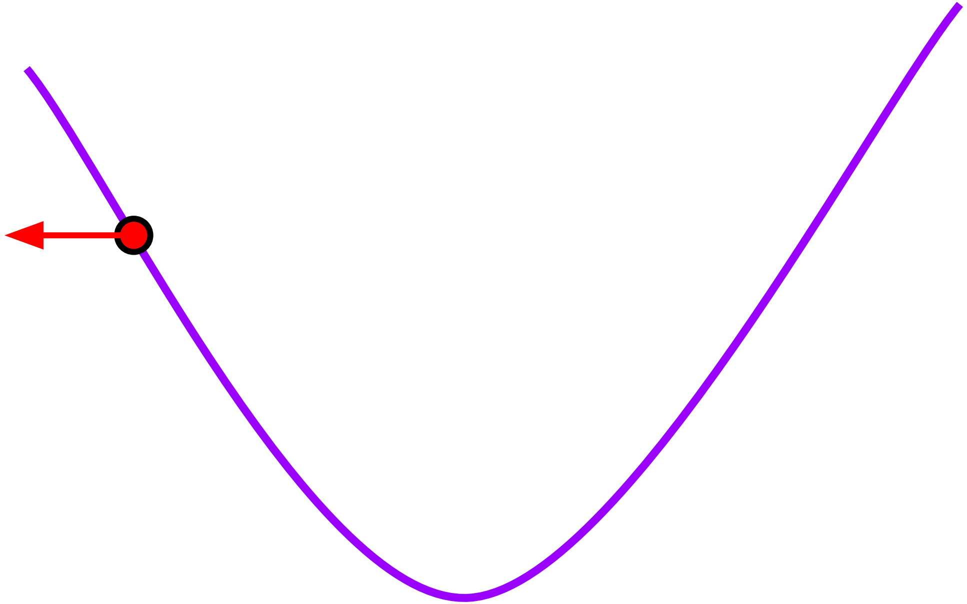
In 1d, either points left or right



Gradient descent

For some loss function $L(\mathbf{w})$, gradient $\nabla L(\mathbf{w})$ points towards in direction of steepest ascent.

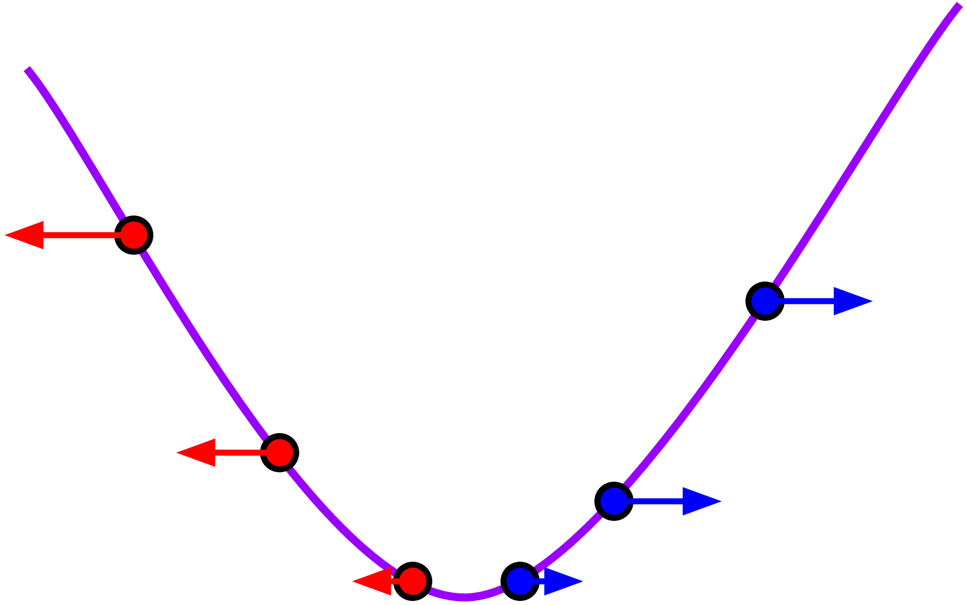
In 1d, either points left or right



Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

In 1d, either points left or right



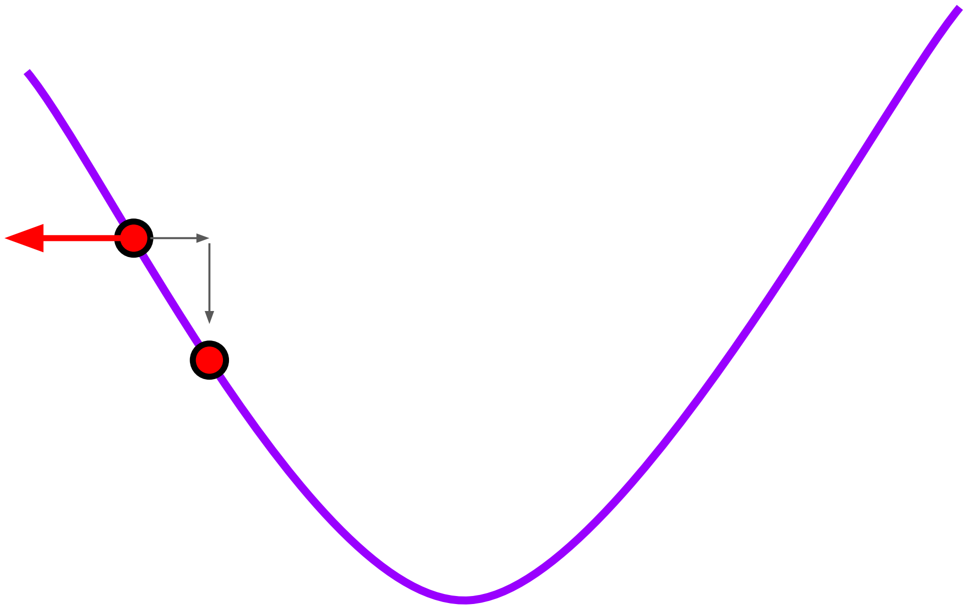
Gradient descent

For some loss function L(**w**), gradient ∇L(**w**) points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



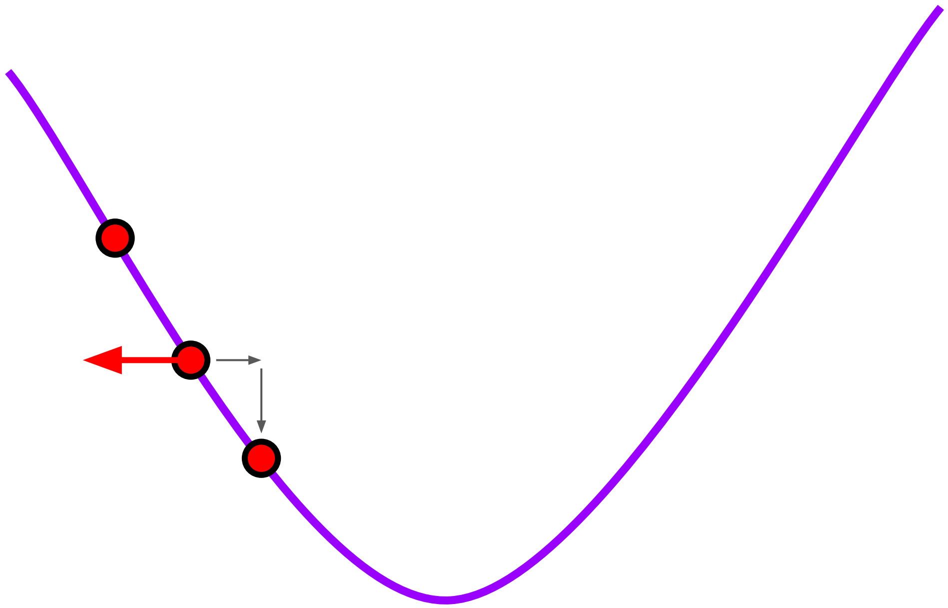
Gradient descent

For some loss function $L(\mathbf{w})$, gradient $\nabla L(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat

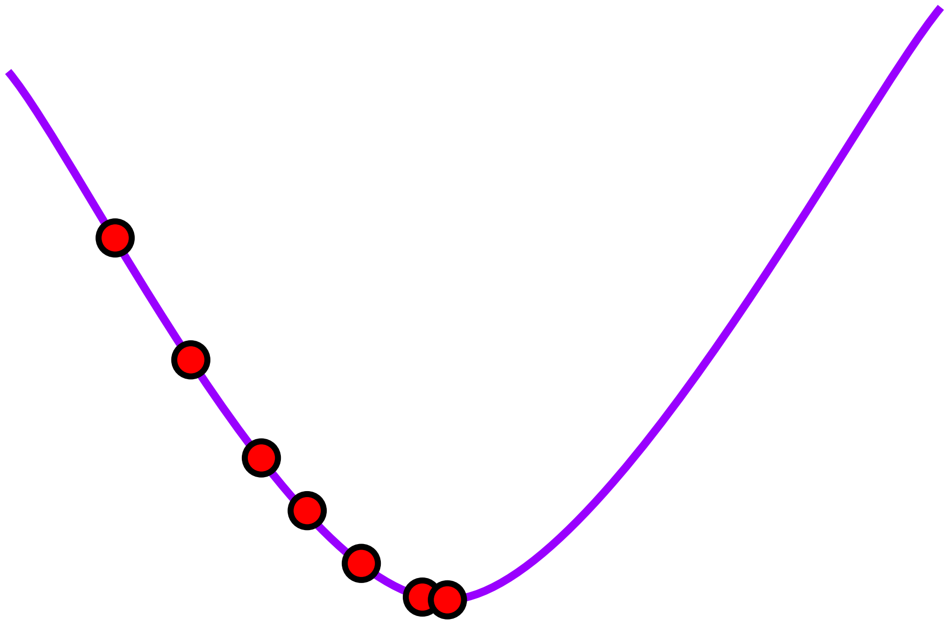


Gradient descent

Algorithm:

Take derivative
Move slightly in other direction
Repeat

End up at local optima

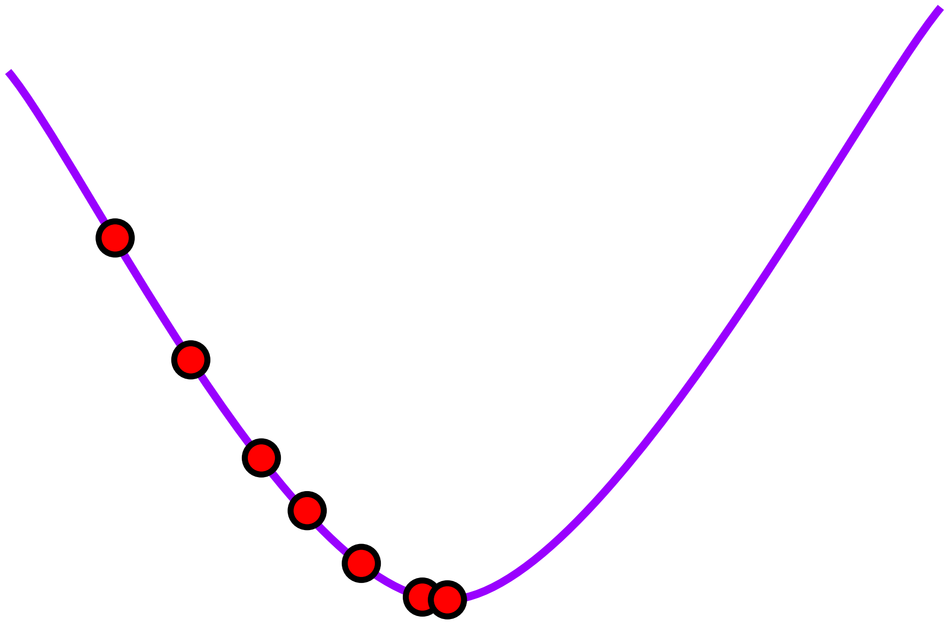


Gradient descent

Formally:

$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L(\mathbf{w})$

Where η is *step size*, how far to step relative to the gradient



Gradient descent

Calculating $\nabla L(\mathbf{w})$ can be hard, especially for big data, |data| very large.

What if we estimate it it instead?

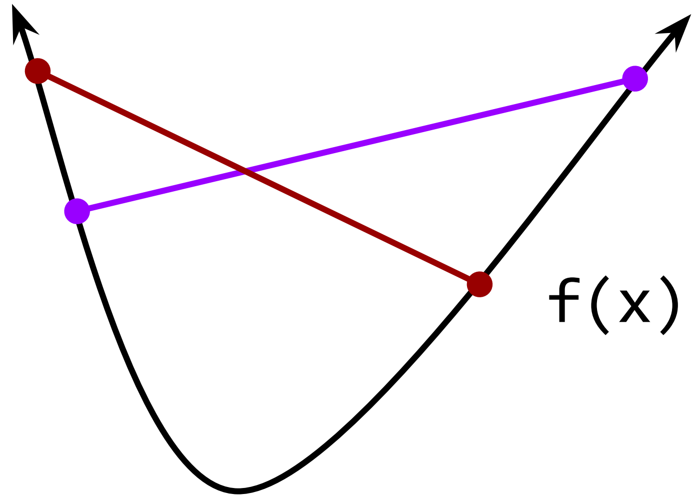
How do we estimate things?

Convex vs Non-convex

Convex function: connect any two points on graph with a line, that line lies above function everywhere

Why is it important? Any local extrema is global extrema!

If our loss function is convex, can set derivative = 0, solve for parameters (sometimes still no closed-form)

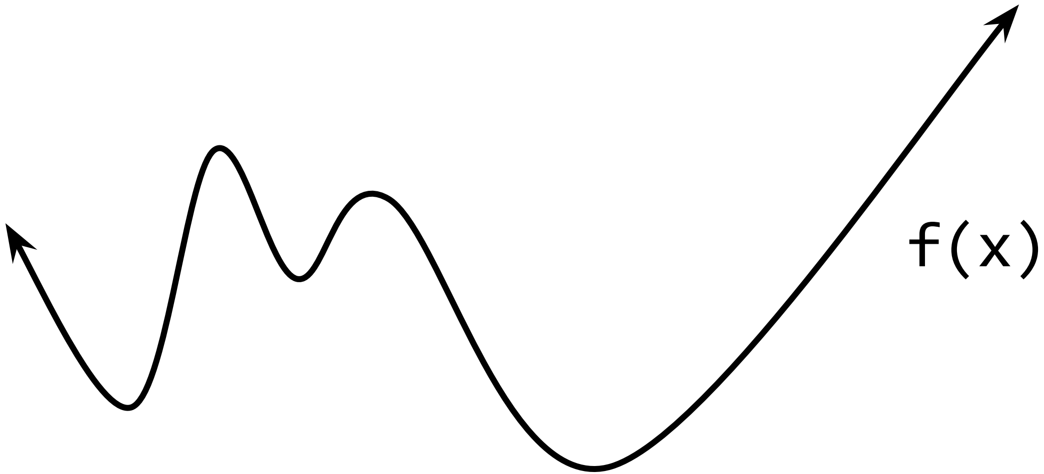


Convex vs Non-convex

Non-convex function: no rules!

Local optima are not global optima

Usually no easy way to find global or local optima, harder to optimize



Loss	Likelihood
Measure of how wrong our model is	How probable our model thinks our training data is
Want a smaller loss	Want high likelihood, model that explains the data well
Try to find local or global minima of our loss function	Find local or global maxima of likelihood function
Derivative = 0, gradient descent	Derivative = 0, gradient ascent

What if we have multiple classes?

Use an extension of logistic regression to multiple classes

For each class k we have weights $\mathbf{w}_k$

Want to predict probability distribution over classes, what’s wrong with:

$\Pr(Y_i=1) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, $\Pr(Y_i=2) = \sigma(\mathbf{w}_2 \cdot \mathbf{X}_i)$, … etc

What if we have multiple classes?

Use an extension of logistic regression to multiple classes

For each class k we have weights $\mathbf{w}_k$

Want to predict probability distribution over classes, what's wrong with:

$\Pr(Y_i=1) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, $\Pr(Y_i=2) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, … etc

No normalization! Might sum to <> 1.

What if we have multiple classes?

What if we normalized logistic regression across classes?

Softmax!

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

If we have 2 classes and we assume $z_0 = 1$, $z_1 = \mathbf{w} \cdot \mathbf{X}$ then this is normal logistic regression.

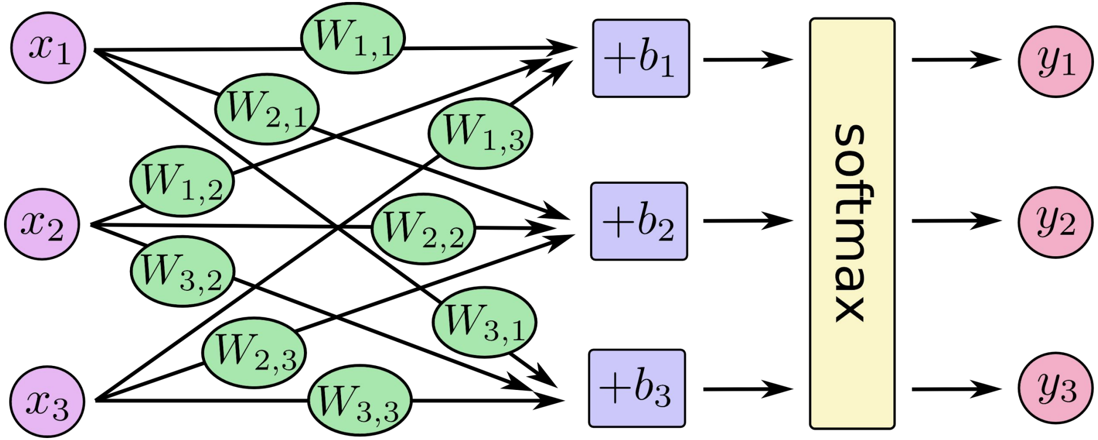
Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.



Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

MNIST: Handwriting recognition

50,000 images of handwriting

28 x 28 x 1 (grayscale)

Numbers 0-9

10 class softmax regression

Input is 784 pixel values

Train with SGD

> 95% accuracy

Q & A

External Readings :

- Viola Jones Face Detection with OpenCV

https://towardsdatascience.com/viola-jones-algorithm-and-haar-cascade-classifier-ee3bfb19f7d8

- Introduction to Linear Regression

https://thuraaung-1601.medium.com/introduction-to-linear-regression-with-normal-equation-98e6c1f839f8

- Mnist Logistic Regression

https://aigeekprogrammer.com/binary-classification-using-logistic-regression-and-keras/

- Scikit-learn Image Classification

https://youtu.be/bwZ3Qiuj3i8

- Deep Learning Introduction (Optional)

https://youtu.be/hl1XtO2jRWM

Thank You !