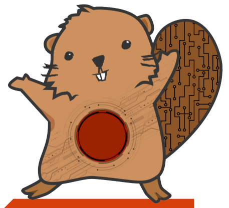
Machine Learning and Data Mining

Lecture 3.1: Perceptron, Evaluating Classifiers, Naïve Bayes



CS 434

RECAP

From Last Lecture

Beyond Lines but Still Linear?

Idea: Solve a linear regression problem in a feature space that is non-linear in the original input! For example, could add a x^2 term.

X		**Y**
	x	Y
1	-0.25	-4.47
1	0.9	11.08
1	0.46	-10.44
1	0.2	7.19
1	-0.69	7.08
1	-0.69	5.38
1	-0.88	11.55
1	0.73	10.85
1	0.2	7.85
1	0.42	-8.31

X'			**Y**
	x	x^2	Y
1	-0.25	0.06	-4.47
1	0.9	0.81	11.08
1	0.46	0.21	-10.44
1	0.2	0.04	7.19
1	-0.69	0.48	7.08
1	-0.69	0.48	5.38
1	-0.88	0.77	11.55
1	0.73	0.53	10.85
1	0.2	0.04	7.85
1	0.42	0.18	-8.31

Solve for w such that:

$$y_i = w_0 1 + w_1 x_i + w_2 x_i^2$$

Linear function of non-linear transformations of x

$$y_i = [1, x_i, x_i^2]\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Linear Regression over Basis Functions

Let's generalize this further.

Each data point has d features. Let $\mathbf{\Phi}\colon\ \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ be our basis function. Can make any arbitrary choice we want here based on how the data looks.

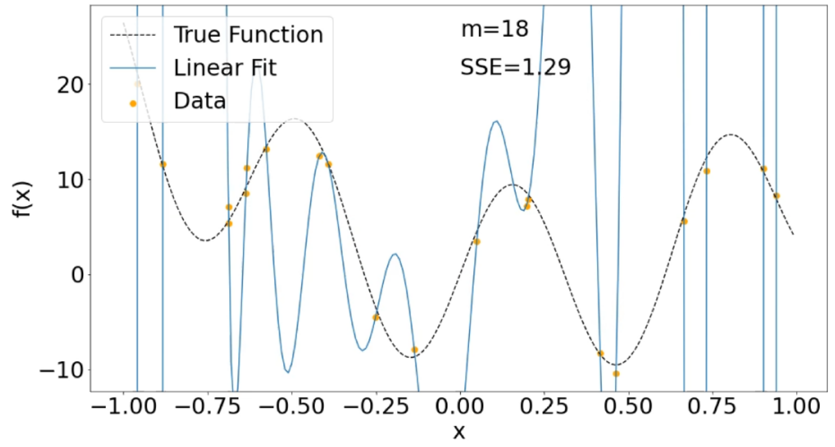
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^{\boldsymbol{T}}_{1\times d} \qquad \mathbf{\Phi}_{\mathrm{A}}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ \sin(x_1) \\ \cos(x_2) \end{bmatrix}^{\boldsymbol{T}} \qquad \mathbf{\Phi}_{\mathrm{B}}(x) = \begin{bmatrix} 1 \\ x_1 x_2 \\ x_1 x_3 \\ x_1 x_4 \\ \vdots \\ x_n x_n \end{bmatrix}^{\boldsymbol{T}} \qquad \mathbf{\Phi}_{\mathrm{C}}(x) = \begin{bmatrix} 1 \\ \prod x_i \\ \sqrt{x_1} \end{bmatrix}^{\boldsymbol{T}}$$

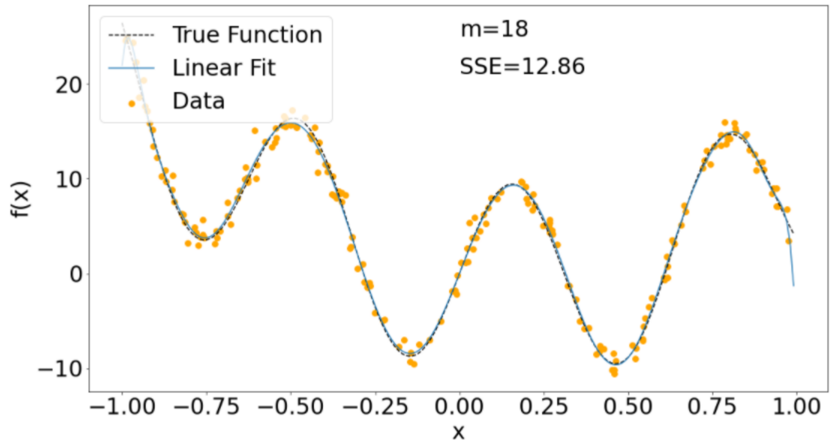
No matter what we choose, the procedure is the same. Replace each $\boldsymbol{x}$ (row) in our data matrix with $\mathbf{\Phi}(\mathbf{x})$, then solve the linear regression problem.

Techniques for Avoiding Overfitting

What can we do to reduce overfitting?

- View it as an estimation error
  - *Solution:* Get more data
- Do model selection to find a simpler model
  - Less complicated model, less data needed.
- **Regularization**
  - Add a *prior* belief that the model should be simple!
  - Okay. But how to encode this?





L2 Regularization

Regularizer View

Add a regularization penalty
$\lambda \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{w}$ to the SSE

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{w}$$

Bayesian View

Assume a Gaussian prior
$w \sim \mathcal{N}(\vec{0}, \beta I)$

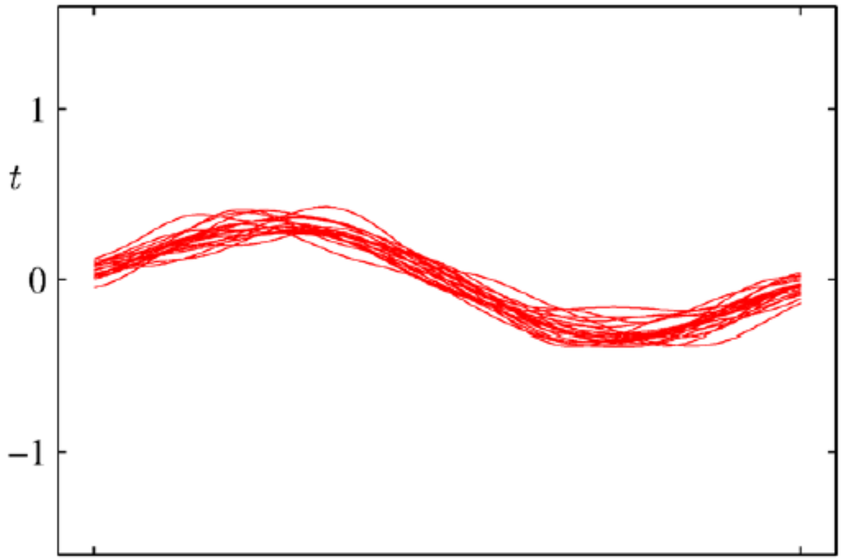
$$w^* = \underset{w}{argmax} \quad logP(D|w) + logP(w)$$

Arrive at the same solution for L2 regularized least squares.

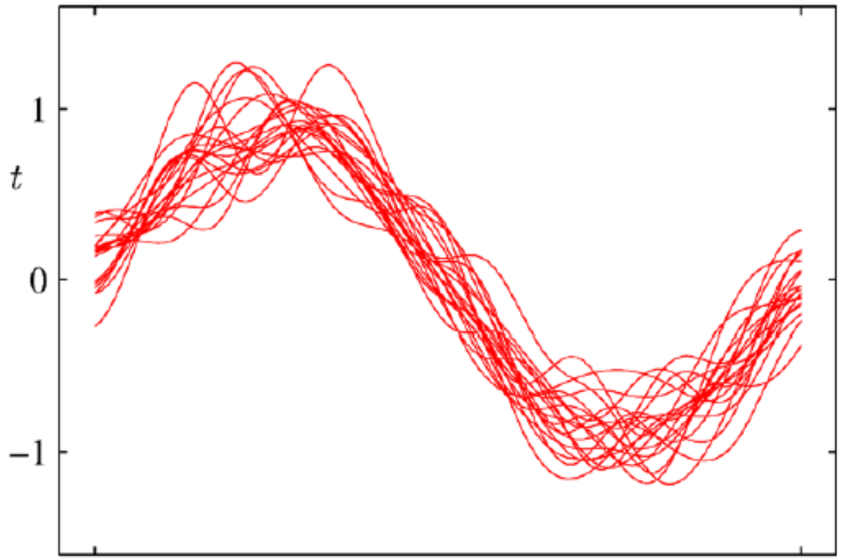
$$\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$

Effect of Regularization – Example 2

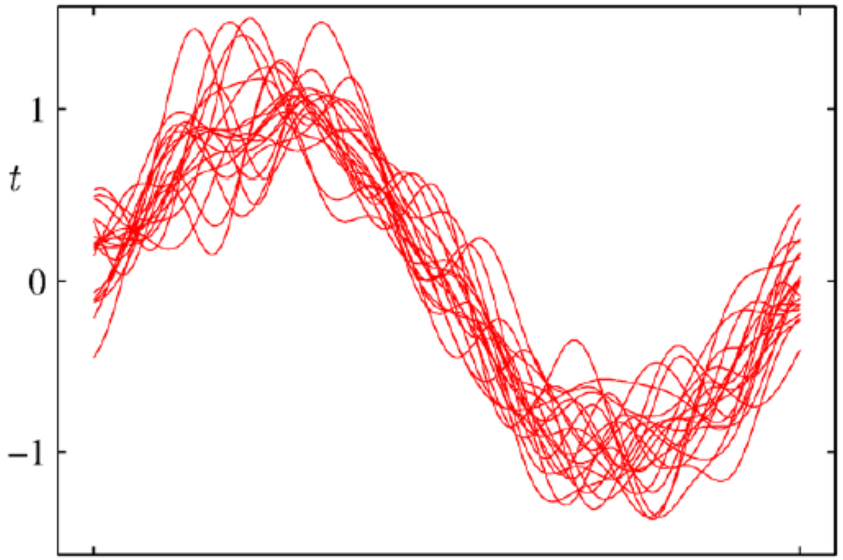
Each red line is the result of a 9th degree polynomial fit on 10 random points noisily sampled from the true curve.



$\lambda \approx 13.46$



$\lambda \approx 0.73$



$\lambda \approx 0.09$

Smaller λ resulted in more complex curves that better fit the data but has a high variance.

Summary of Regularization

Regularization is a much more general concept in machine learning. Consider any learning task to minimize a loss $\mathcal{L}(w)$ on the training data – could add a regularizer term.

$$w^* = \underset{w}{argmin}\, \mathcal{L}(w) + \lambda * regularizer(w)$$

Generally speaking, larger $\lambda's$ lead to simpler models and reduce overfitting, smaller $\lambda's$ lead to more complex models but improve fit on training data.

Most commonly used regularizers are based on the norm of the weight vector.

Why not regress probabilities directly?

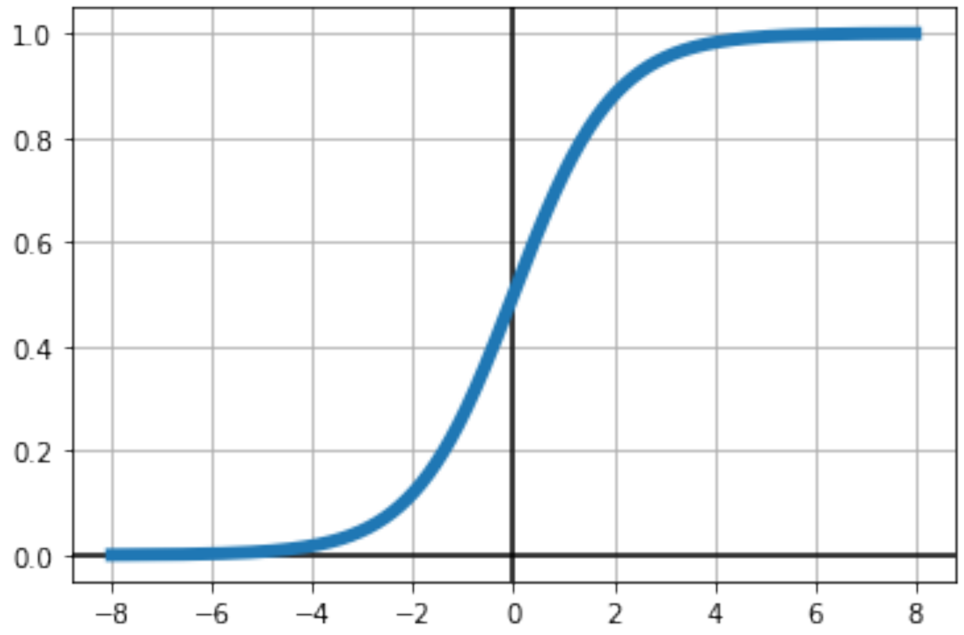
Introducing the logistic function:

- May also see it referred to as a sigmoid function or "logit" function

Logistic Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Maps $(-\infty, \infty)$ to $(0,1)$



Logistic Regression Model Assumption:

$$P(y_i = 1|\boldsymbol{x_i}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

$$P(y_i = 0|\boldsymbol{x_i}; \boldsymbol{w}) = 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

Logistic Regression Model's Decision Boundary:

Predict 1 if

$$P(y_i = 1|\boldsymbol{x}_i; \boldsymbol{w}) > P(y_i = 0|\boldsymbol{x}_i; \boldsymbol{w})$$

Divide both sides by $P(y_i = 0|\boldsymbol{x}_i; \boldsymbol{w})$

$$\Rightarrow \quad \frac{P(y_i = 1|\boldsymbol{x}_i; \boldsymbol{w})}{P(y_i = 0|\boldsymbol{x}_i; \boldsymbol{w})} > 1$$

Just some algebra

$$\Rightarrow \quad e^{\boldsymbol{w}^T \boldsymbol{x}} > 1$$

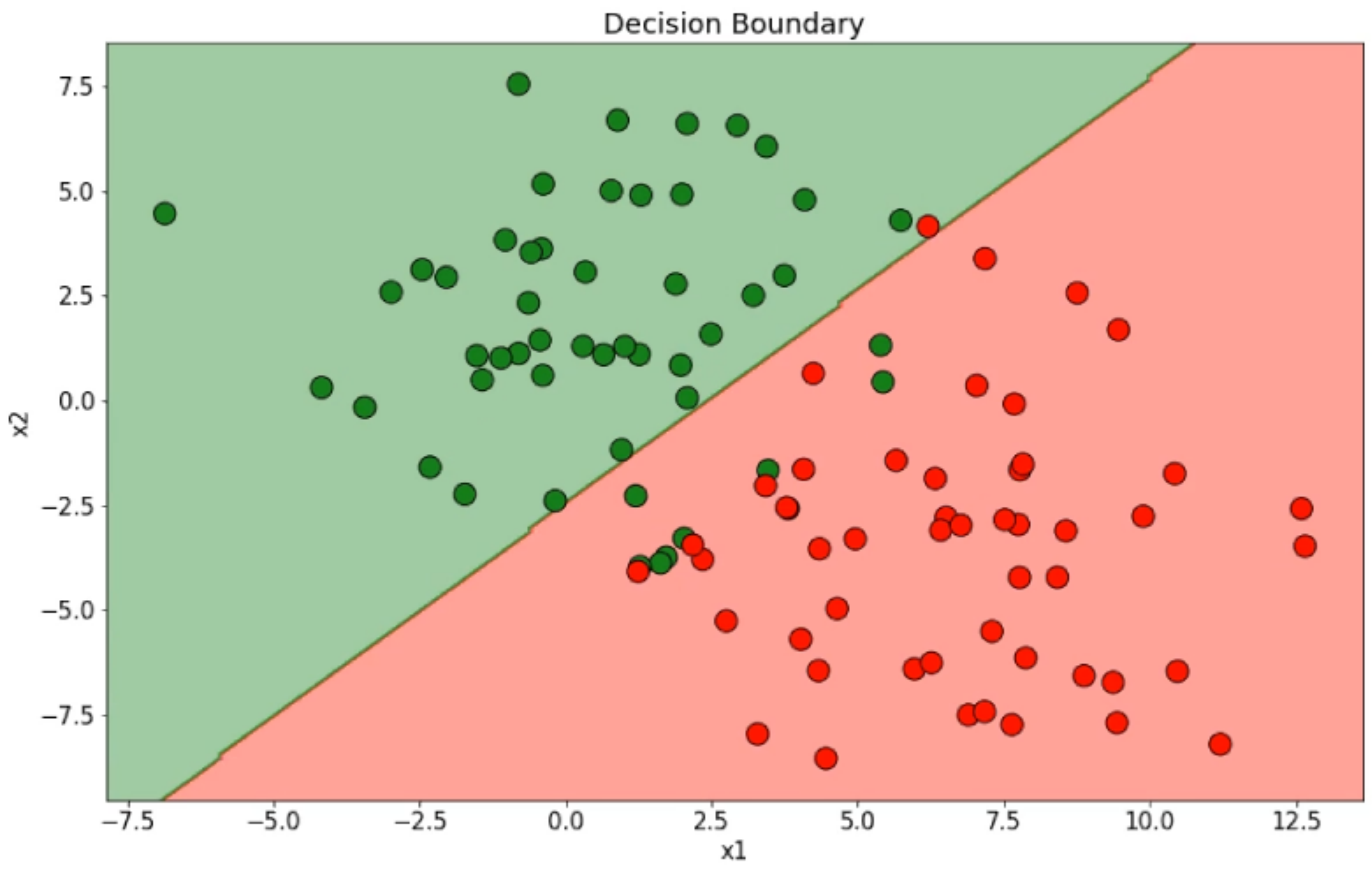
Log of both sides

$$\Rightarrow \quad \boldsymbol{w}^T \boldsymbol{x} > 0$$

This tells us the decision boundary is the line $w^T x = 0$

Logistic Regression

Example logistic regression decision boundary – linear in the features.



Today's Learning Objectives

Be able to answer:

- What is the perceptron linear classifier?
  - How is it updated?
- How to evaluate classification models?
  - Accuracy
  - Recall and Precision

Perceptron vs Logistic Regression -- History

1830s Logistic function invented

1943 Logistic Regression invented but in biology and largely ignored

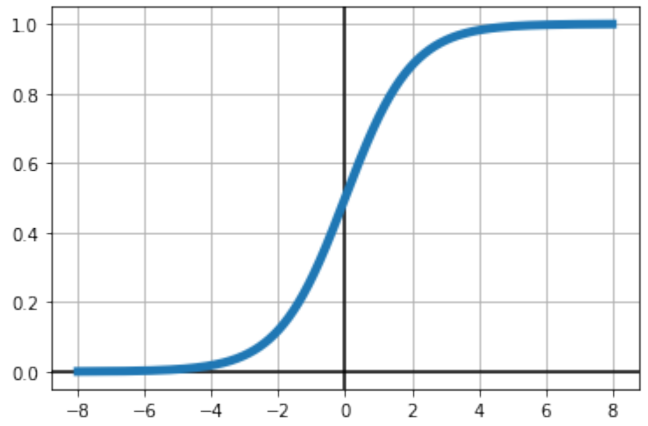
1958 Perceptron invented and popularized in AI/ML communities

1970s Logistic Regression reached widespread recognition

Perceptron vs Logistic Regression

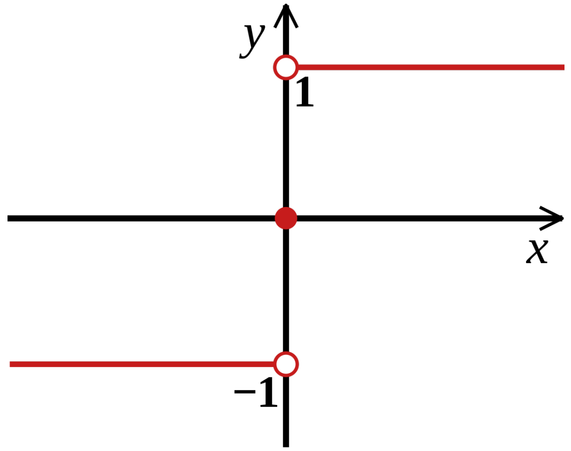
Logistic Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Sign Function

$$sign(z) = \begin{cases} 1 & z \geq 0 \\ -1 & else \end{cases}$$



Perceptron vs Logistic Regression

Logistic Regression

$$P(y_i = 1 | \boldsymbol{x}_i; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$$

$$y_i = argmax_y \, P(y | \boldsymbol{x}_i; \boldsymbol{w})$$

Linear Classifier

Probabilistic Interpretation

Trained with gradient descent or other optimization methods

Perceptron

$$y_i = sign(\boldsymbol{w}^T \boldsymbol{x}_i)$$

Linear Classifier

No useful probabilistic interpretation (mostly just geometry)

Trained with the "perceptron" algorithm

The Perceptron: Setup

Consider a binary prediction problem where we are given a dataset of $\boldsymbol{x_i}, y_i$ pairs where $y_i \in \{-1,1\}$. We want to find optimal linear weights $\boldsymbol{w}$ such that:

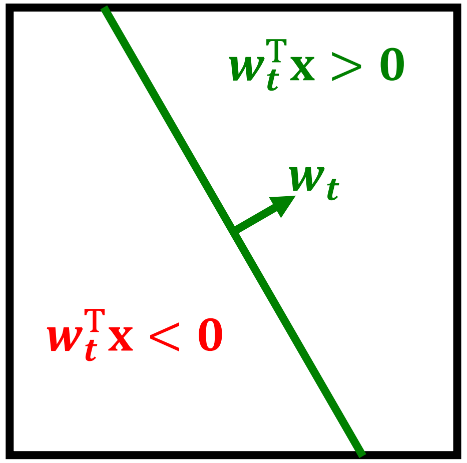
$$y_i = sign(\boldsymbol{w}^T \boldsymbol{x_i}) \quad \forall i$$

This is equivalent to saying we want:

$$y_i \boldsymbol{w}^T \boldsymbol{x_i} > 0 \quad \forall i$$

If y_i and $w^T x_i$ have same sign, then sign($w^T x_i$) will equal y_i

The Perceptron: Setup

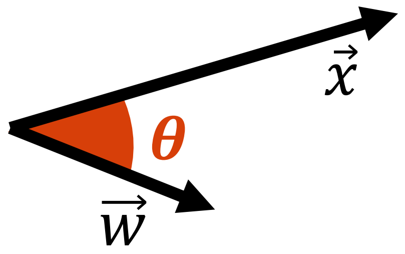


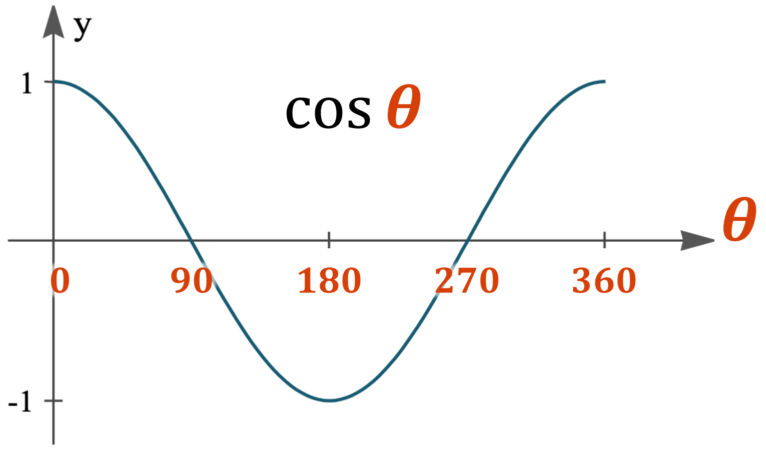
Note that the weight vector **w** is the *normal vector* of the decision boundary and **w** points to the side where $\boldsymbol{w}^T\boldsymbol{x} > \mathbf{0}$

Every point where $w^T x = 0$ lies along the line perpendicular to $\boldsymbol{w}$. **Why? Linear algebra.**

Linear Algebra Fact: Dot product of two vectors is proportional to the cosine of the angle between them.

$$\vec{v} \cdot \vec{x} = |\vec{x}||\vec{w}| \cos \boldsymbol{\theta}$$





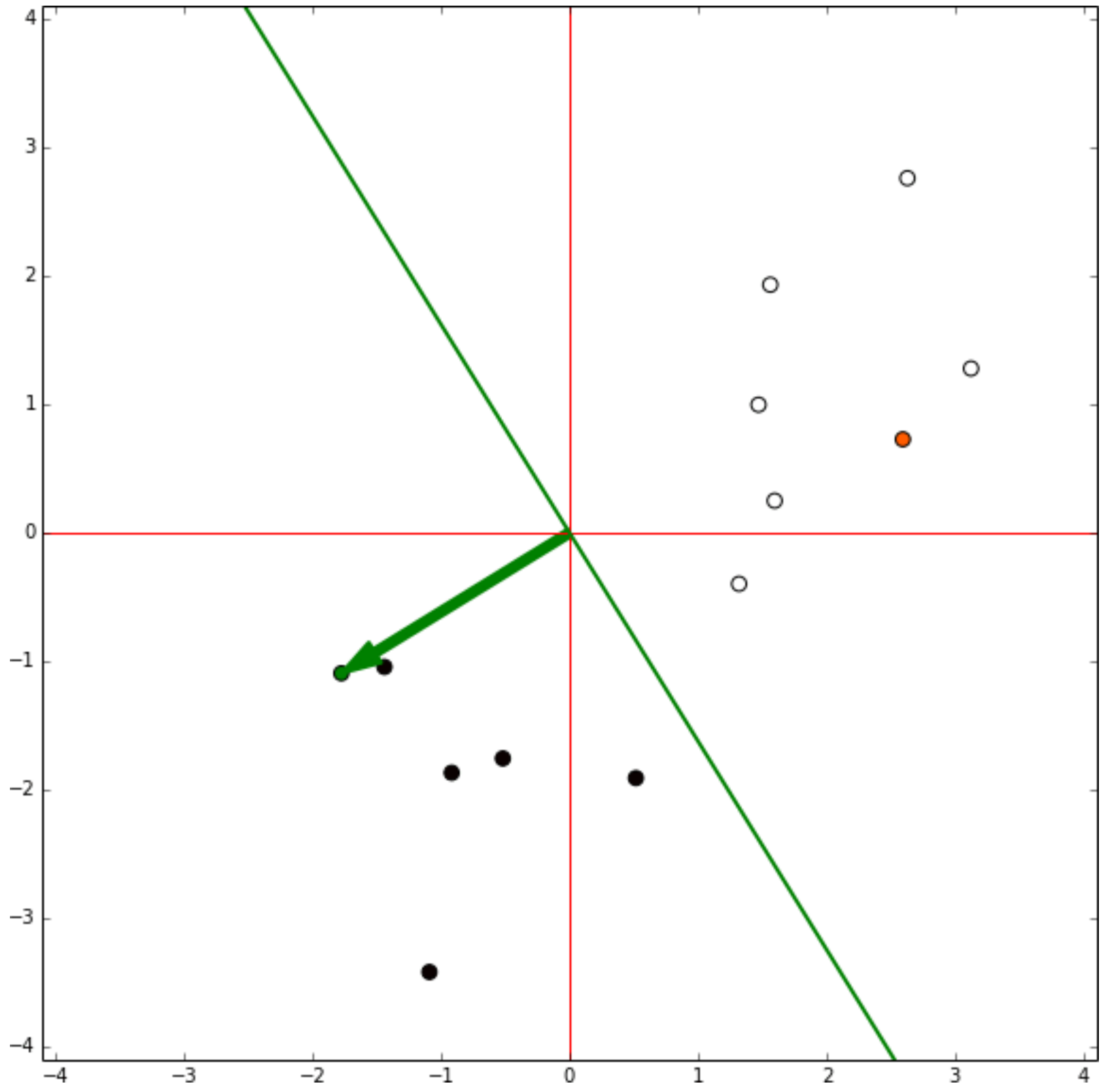
© Stefan Lee

The Perceptron: Learning Algorithm

Trained with a simple iterative algorithm

Perceptron Learning Algorithm

```
w = random(d)
Repeat:
    for each example $x_i, y_i \in D$:
        if $y_i w^T x_i < 0$: // if misclassification
            $w = w + \alpha y_i x_i$
Until no errors or max iterations
```



The Perceptron: Effect of Update Rule

Consider how adding $\alpha y_i \boldsymbol{x_i}$ to the weights changes the "error" of a point. Let w_t be the weights before the update and w_{t+1} be after the update -- $\boldsymbol{w_{t+1}} = \boldsymbol{w_t} + \alpha y_i \boldsymbol{x_i}$.

$$y_i \boldsymbol{w_{t+1}^T x_i} = y_i(\boldsymbol{w_t} + \alpha y_i \boldsymbol{x}_i)^T \boldsymbol{x_i}$$

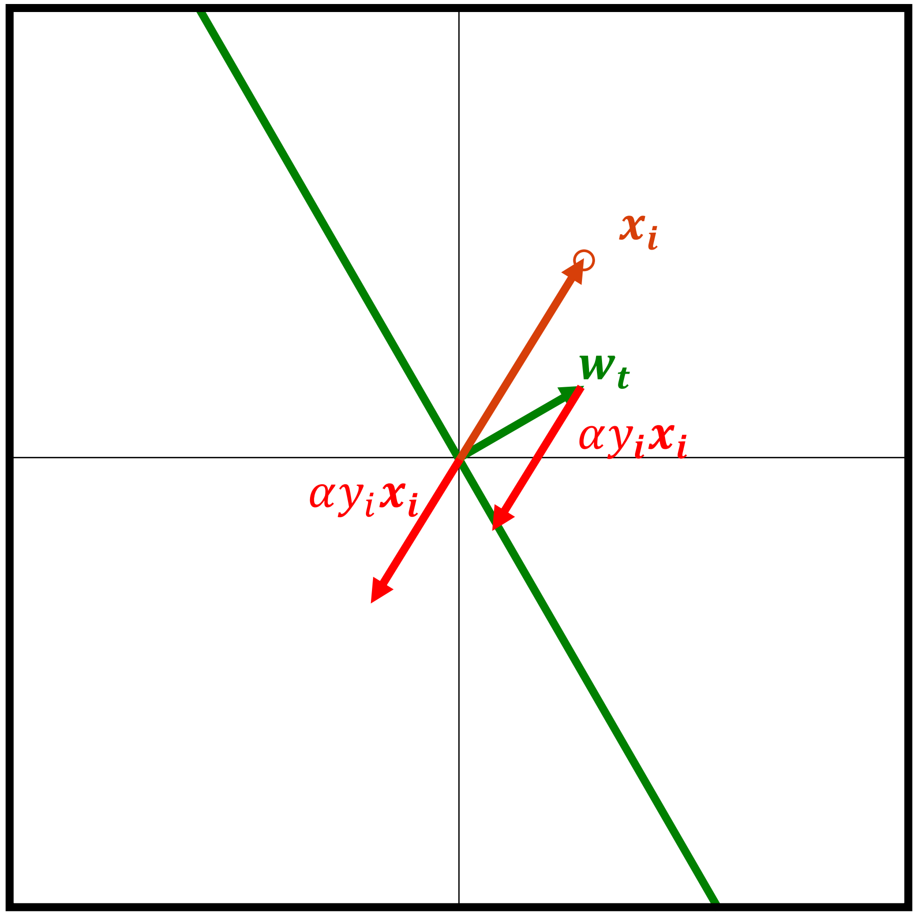
$$= y \boldsymbol{w_t^T x_i} + \alpha y_i^2 \boldsymbol{x_i^T x_i}$$

$$= y \boldsymbol{w_t^T x_i} + \alpha y^2 \|\boldsymbol{x_i}\|_2^2$$

This term is a non-negative scalar

$$\Rightarrow \quad y_i \boldsymbol{w_{t+1}^T x_i} \geq y_i \boldsymbol{w_t^T x_i}$$

We improve the score!



The Perceptron: Effect of Update Rule

Consider how adding $\alpha y_i \boldsymbol{x_i}$ to the weights changes the "error" of a point. Let w_t be the weights before the update and w_{t+1} be after the update -- $\boldsymbol{w_{t+1}} = \boldsymbol{w_t} + \alpha y_i \boldsymbol{x_i}$.

$$y_i \boldsymbol{w_{t+1}^T x_i} = y_i(\boldsymbol{w_t} + \alpha y_i \boldsymbol{x}_i)^T \boldsymbol{x_i}$$

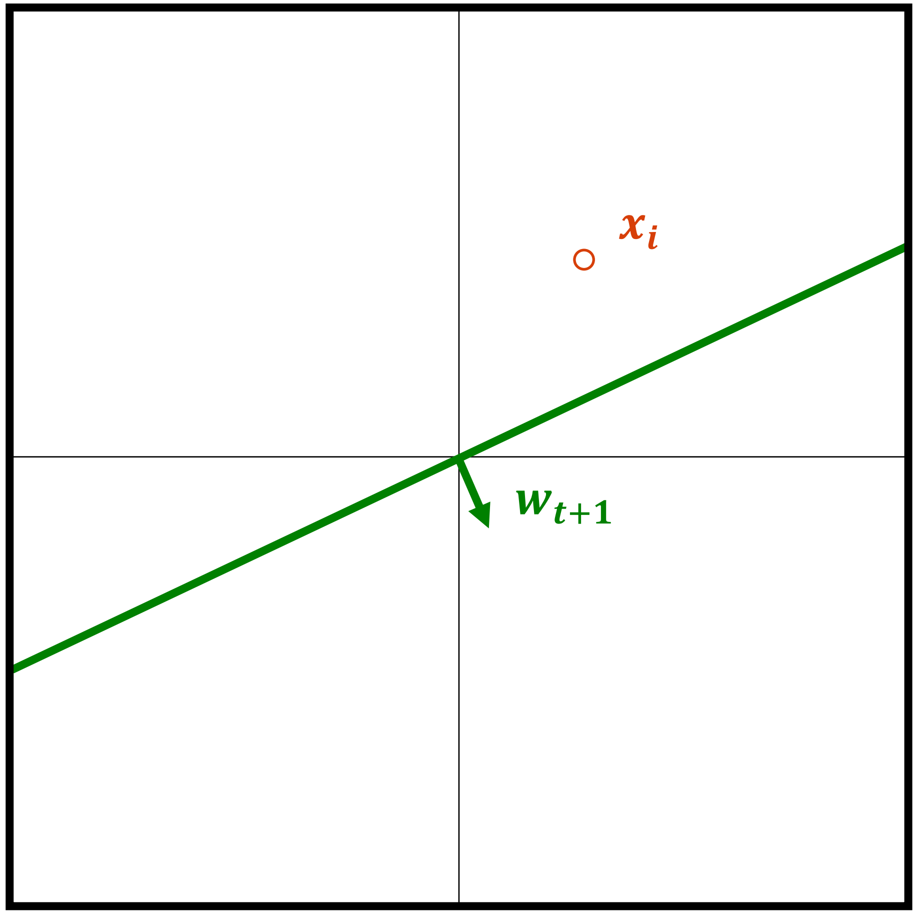
$$= y \boldsymbol{w_t^T x_i} + \alpha y_i^2 \boldsymbol{x_i^T x_i}$$

$$= y \boldsymbol{w_t^T x_i} + \alpha y^2 \|\boldsymbol{x_i}\|_2^2$$

This term is a non-negative scalar

$$\Rightarrow \quad y_i \boldsymbol{w_{t+1}^T x_i} \geq y_i \boldsymbol{w_t^T x_i}$$

We improve the score!

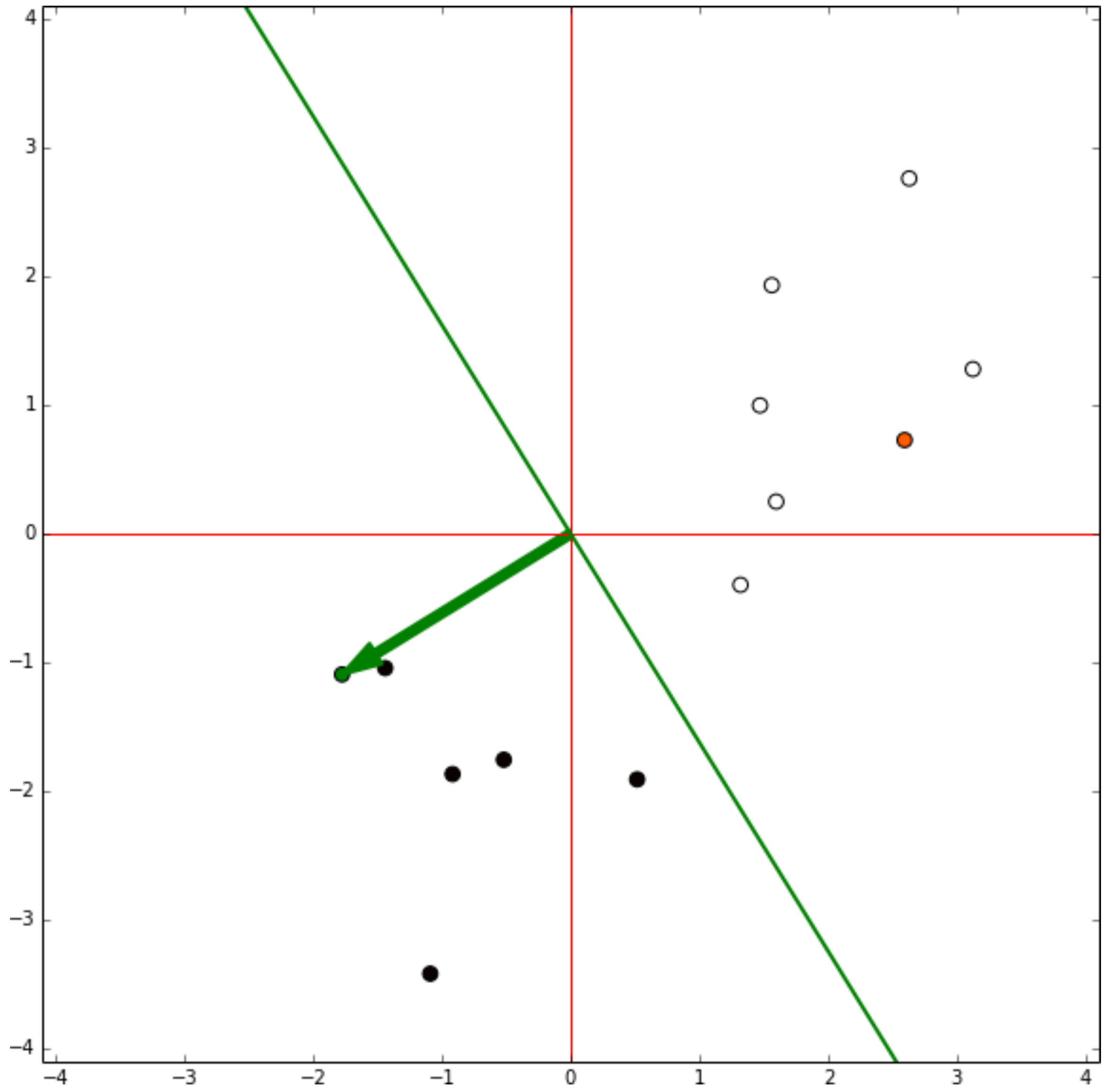


The Perceptron: Learning Algorithm Revisited

Trained with a simple iterative algorithm

Perceptron Learning Algorithm

```
w = random(d)
Repeat:
    for each example $x_i, y_i \in D$:
        if $y_i w^T x_i < 0$: // if misclassification
            $w = w + \alpha y_i x_i$
Until no errors or max iterations
```

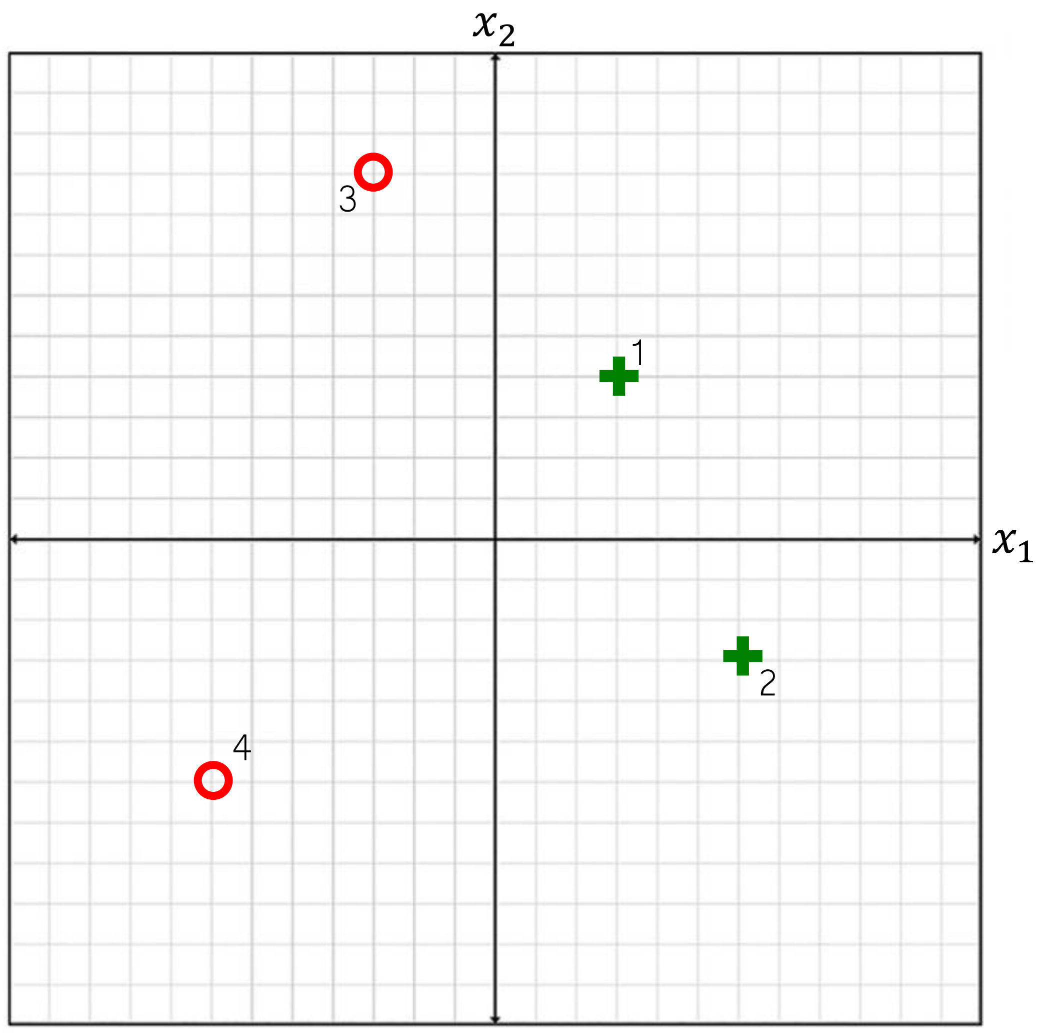


© Stefan Lee

The Perceptron: Learning Algorithm By Hand

	X1	X2	Y
1	3	4	1
2	6	-3	1
3	-3	9	-1
4	-7	6	-1

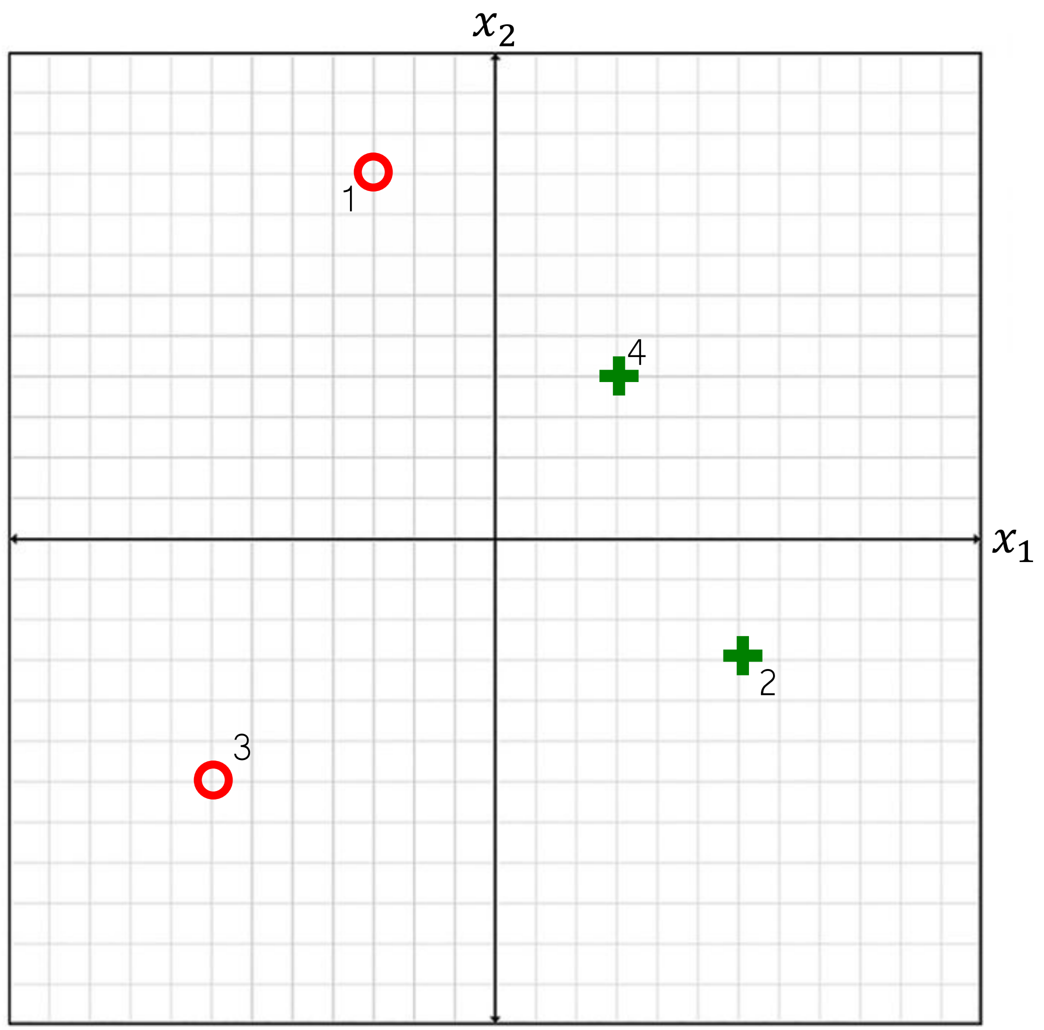
```
w = [0,3]
Repeat until no errors:
    for each example $\boldsymbol{x_i}, y_i \in D$:
        if $y_i \boldsymbol{w}^T \boldsymbol{x_i} < 0$: $\boldsymbol{w} = \boldsymbol{w} + 1 y_i \boldsymbol{x_i}$
```



The Perceptron: Learning Algorithm By Hand

	X1	X2	Y
4	3	4	1
2	6	-3	1
1	-3	9	-1
3	-7	6	-1

```
w = [-3,3]
Repeat until no errors:
    for each example $x_i, y_i \in D$:
        if $y_i w^T x_i < 0$: $w = w + 1 y_i x_i$
```



Online vs. Batch Learning

We refer to the perceptron algorithm as an **online learning** algorithm because it updates parameters (aka learns) each time it receives a training example.

In contrast, **batch learning** algorithms collect a set of training examples and learn from them all at once. For example, the gradient descent algorithm we described for logistic regression is a batch algorithm.

(Note that gradient descent also has multiple online versions)

Online vs. Batch Learning

Could modify the perceptron learning algorithm to be a batch learning algorithm by accumulating updates over the whole dataset before changing the weights.

Batched Perceptron Learning Algorithm

```
$\boldsymbol{w}$ = random(d)
Repeat:
    $\boldsymbol{u}$ = zero(d)
    for each example $\boldsymbol{x}_i, y_i \in D$:
        if $y_i \boldsymbol{w}^T \boldsymbol{x}_i < 0$: // if misclassification
            $\boldsymbol{u} \mathrel{+}= -y_i \boldsymbol{x}_i$
    $\boldsymbol{w} = \boldsymbol{w} - \alpha \boldsymbol{u}$
Until $|u| \leq \epsilon$
```

The result looks a lot like gradient descent.

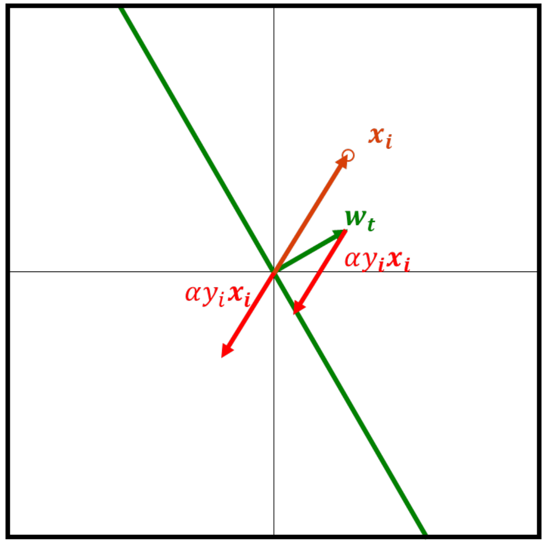
Note for the curious: Perceptron algorithm can be shown to be doing gradient descent to minimize the loss function

$$E(w) = \frac{1}{N}\sum_{i=1}^{N} max(-y_i w^T x_i, 0)$$

The Perceptron: Limitations

Perceptron Learning Algorithm

```
w = random(d)
While still errors or max iterations:
    for each example $x_i, y_i \in D$:
        if $y w^T x_i < 0$:    $w = w + \alpha y_i x_i$
```

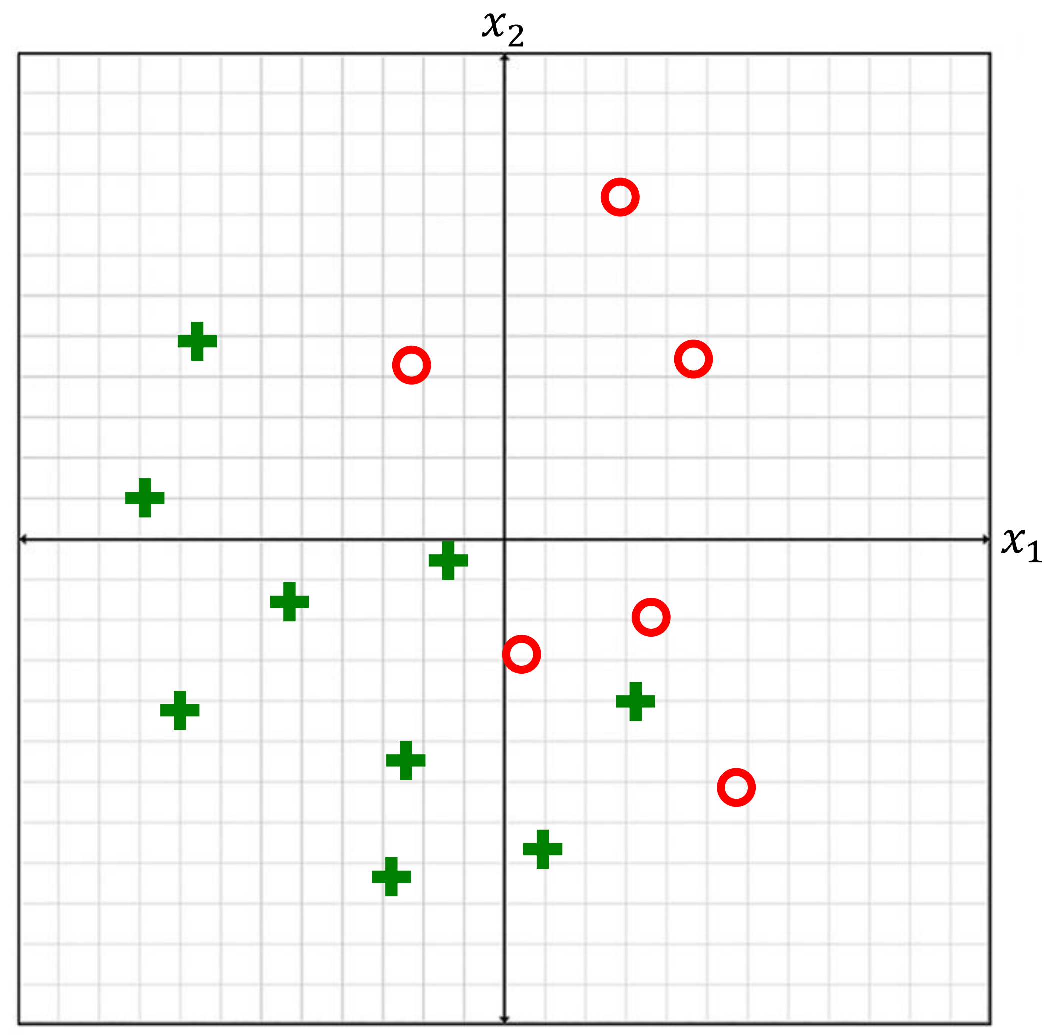


- Different solutions depending on initialization and order of visiting examples.
- Correcting for a misclassification could move the decision boundary so much that previously correct examples are now misclassified.
  - As such it must go over the training examples multiple times. Each time it goes through the whole training set, it is called an epoch.
- It will terminate if no update is made to ***w*** during one epoch

The Perceptron: Limitations

- This algorithm is guaranteed to converge if data is linearly separable – i.e., it reaches a solution in finitely many steps.
  - Why and how many steps? (proof)

- For non-linearly separable cases (like on right), algorithm fails to converge and may be arbitrarily bad if terminated at some maximum number of updates.



The Perceptron: The Voted-Perceptron Algorithm

One Idea: keep around intermediate hypothesis and have them “vote” -- i.e., a weighted average. [Freund and Schapire, 1998]

```
w = random(d)
n=0
While still errors or max iterations:
    for each example $\boldsymbol{x}_i, y_i \in D$:
        if $y\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}} < 0$:
            $\boldsymbol{w}_{\boldsymbol{n+1}} = \boldsymbol{w}_n + \alpha y_i \boldsymbol{x}_{\boldsymbol{i}}$
            $c_{n+1} = 0$
            $n += 1$
        else:
            $c_n += 1$
```

Store the weight vector after each error update along with the count of examples that weight was successful before making the error. Weights contribute relative to this value.

$$y_i = sign\left(\sum_{n=1}^{N} c_n sign(w_n^T x_i)\right)$$

Perceptron Summary

- Perceptron incrementally learns a **linear decision** boundary to separate positive from negative (two classes so binary classification).
- It begins with a random or zero weight vector, and incrementally updates the weight vector whenever it makes a mistake.
- For online perceptron, different orderings of the training examples can lead to different outputs. Algorithm will fail to converge if not linearly separable.
- Voted perceptron can handle non-linearly separable data, and is more robust to noise/outlier.

Today's Learning Objectives

Be able to answer:

- ~~What is the perceptron linear classifier?~~
  - ~~How is it updated?~~

- How to evaluate classification models?
  - Accuracy
  - Recall and Precision

Where are we?

We've covered a few predictive models so far in class:

Predictive Models:

<u>Classifiers (given x, produce discrete y)</u>

- k-Nearest Neighbors Classifier
- Logistic Regression (binary classification only)
- Perceptron (binary classification only)

<u>Regressors (given x, produce continuous y)</u>

- Linear Regression
- k-Nearest Neighbors Regressor

Evaluating Classifiers

Accuracy: # correct / total

If I tell you, I have a ML product that can tell whether someone has a rare disease or not with 99.99% accuracy – are you impressed?

This is a **binary classification** problem – 0 = no disease, 1 = disease

What if the actual percentage of people with the disease is only 0.0001%?

My algorithm is worse than a model that said no-one was a sick which would achieve 99.9999% accuracy.

Evaluating Classifiers

Let's consider all possible scenarios for a single prediction:

$Let\ y\ be\ the\ true\ label\ and\ \hat{y}\ be\ our\ prediction.$

- **False Positive** -- not sick but my model said they have the disease
  $y = 0, \qquad \hat{y} = 1$
- **False Negative** -- sick but my model said they don't have the disease
  $y = 1, \qquad \hat{y} = 0$
- **True Positive** – sick and my model said they have the disease
  $y = 1, \qquad \hat{y} = 1$
- **True Negative** – not sick and model said they don't have the disease
  $y = 0, \qquad \hat{y} = 0$

Evaluating Classifiers

These are often consolidated into what is called a **confusion matrix**.

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**)
	Positive **+**	False Negatives (F**N**)	True Positives (T**P**)

Evaluating Classifiers

These are often consolidated into what is called a **confusion matrix**.

y	$\hat{y}$
1	1
0	0
0	1
1	0
1	1
1	1
1	0
1	1

		Predicted	
		Negative (**N**) **-**	Positive (**P**) **+**
Actual	Negative **-**	True Negatives (T**N**)	False Positives (F**P**)
	Positive **+**	False Negatives (F**N**)	True Positives (T**P**)

Evaluating Classifiers

Two useful composite measures:

Recall: What fraction of positive does your model predict as positives:

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

Precision: Of things your model predicts as positives, what fraction are correct?

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

Question Break!

If I have a model with low recall but high precision, I should trust it __________________________________?

[✓ **A**] more when it predicts something as positive than if it predicts it as negative

B the same when it predicts something as positive or negative

C less when it predicts something as positive than if it predicts it as negative

D

Evaluating Classifiers

Let's go a step further and think about classifiers that output a score:

Let y be the true label and $\hat{y}$ be our prediction. Further let s be the score of our model.

We can write most of our models as doing the following:

$$\hat{y} = \begin{cases} 1 & if\ s \geq t \\ 0 & else \end{cases}$$

For example, linear classifiers like logistic regression or perceptron can be:

$$\hat{y} = \begin{cases} 1 & if\ w^T x \geq 0 \\ 0 & else \end{cases}$$

Evaluating Classifiers

Different settings of the threshold give different outputs.

$$\hat{y} = \begin{cases} 1 & if\ s \geq t \\ 0 & else \end{cases}$$

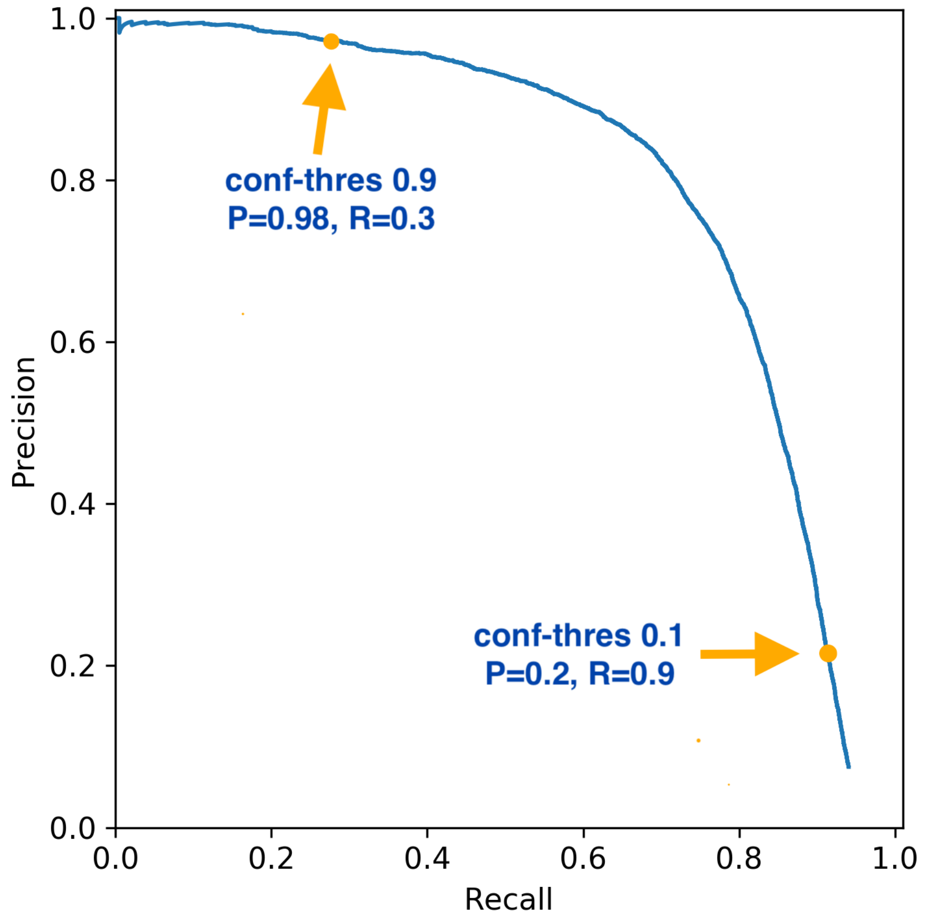
y	s
1	0.8
0	0.2
0	0.4
1	0.3
1	0.7
1	0.9
1	0.3
1	0.5

$\hat{y}$ t=0	$\hat{y}$ t=0.25	$\hat{y}$ t=0.5	$\hat{y}$ t=0.75	$\hat{y}$ t=1
1	1	1	1	0
1	0	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
1	1	0	0	0
1	1	1	0	0

And thus different recall/precision values – seems like something we can tune.

Evaluating Classifiers

Idea: Vary the threshold from its minimum to maximum value and plot the resulting sequence of recall and precision values on a curve.



Call this a precision-vs-recall plot.

Tells us the full range of precision and recall values our classifier can take.

Can be used to compare classifiers or to help us pick optimal settings for a classifier based on our needs.

Question Break!

If a disease can be easily treated early but has devastating effects if given time to develop, we might prefer a classifier with high _______?

A Accuracy

B Precision

C Recall

D

Questions Break!

Suppose you have an earthquake warning system with very high recall but low precision. How might people react to this system over time?

Today's Learning Objectives

Be able to answer:

- ~~What is the perceptron linear classifier?~~
  - ~~How is it updated?~~
- ~~How to evaluate classification models?~~
  - ~~Accuracy~~
  - ~~Recall and Precision~~

Next time

Next Time: We'll talk about Naïve Bayes and classification problems with more than two classes!