Lecture 4

CS436/536: Introduction to Machine Learning

Zhaohan Xi

Binghamton University

zxi1@binghamton.edu

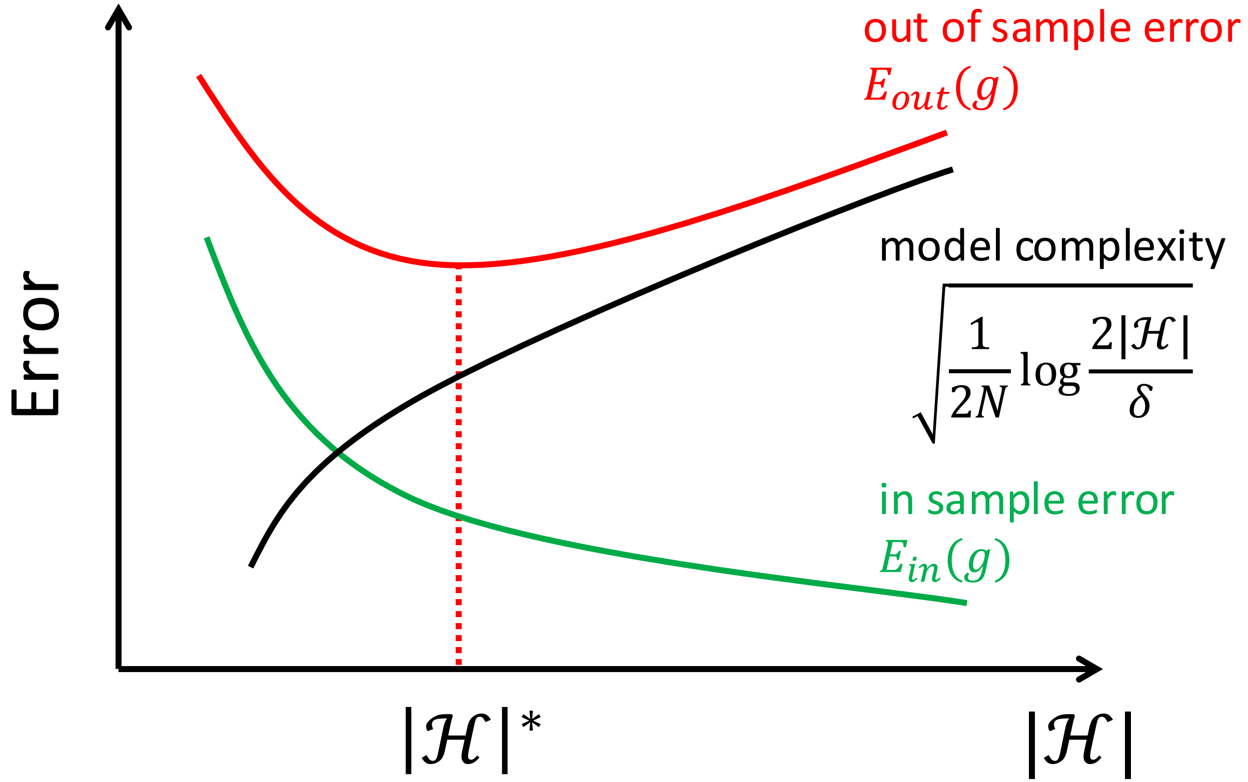
Two Step Approach to "Real" Learning ($E_{out} \approx 0$)

Step 1: Ensure $E_{out} \approx E_{in}$

Step 2: Make E_{in} small

$$E_{out}(g) \leq E_{in}(g) + \underbrace{\sqrt{\frac{1}{2N}\log\frac{2|\mathcal{H}|}{\delta}}}_{\text{generalization error bar}}$$

For Fixed N, δ:



The complexity of f

More complex target functions are harder to learn

- Simple f $\Rightarrow$ can use small $\mathcal{H}$ to get $E_{in}(g) \approx 0$ using smaller N
- Complex f $\Rightarrow$ need large $\mathcal{H}$ to get $E_{in}(g) \approx 0$ and need larger N

The Issue of Noise

- Measurement error: When $y_n \neq f(\boldsymbol{x}_n)$
- Non-deterministic target function: the target is a distribution $P(y|\boldsymbol{x})$

$$f(\boldsymbol{x}_n) \begin{array}{l} \nearrow +1 \\ \searrow -1 \end{array}$$

Data points $(\boldsymbol{x}, y)$ drawn from joint distribution $P(\boldsymbol{x}, y) = P(\boldsymbol{x})P(y|\boldsymbol{x})$

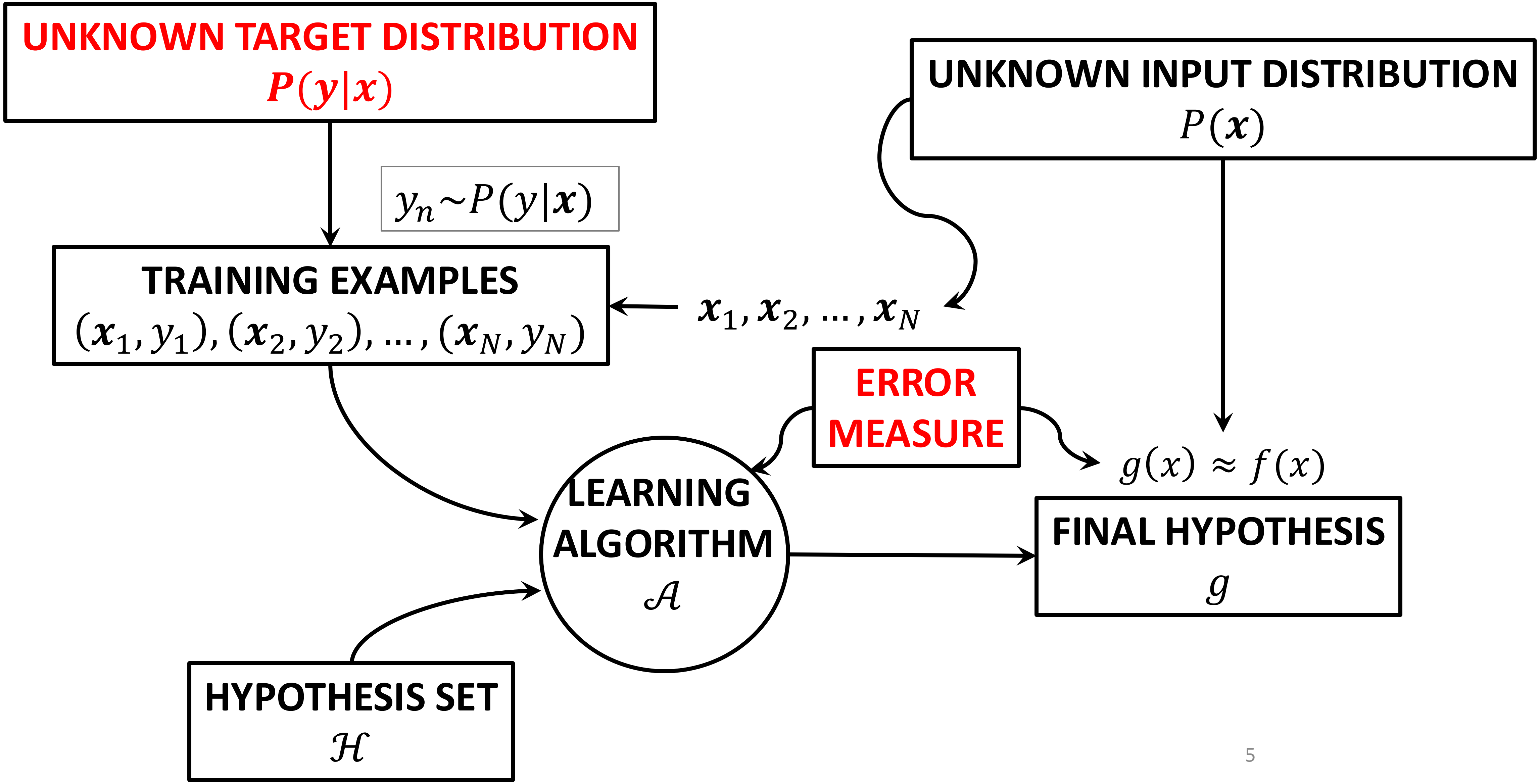
Our theory works for non-deterministic target functions!

- Applies to any particular random realization of the target function

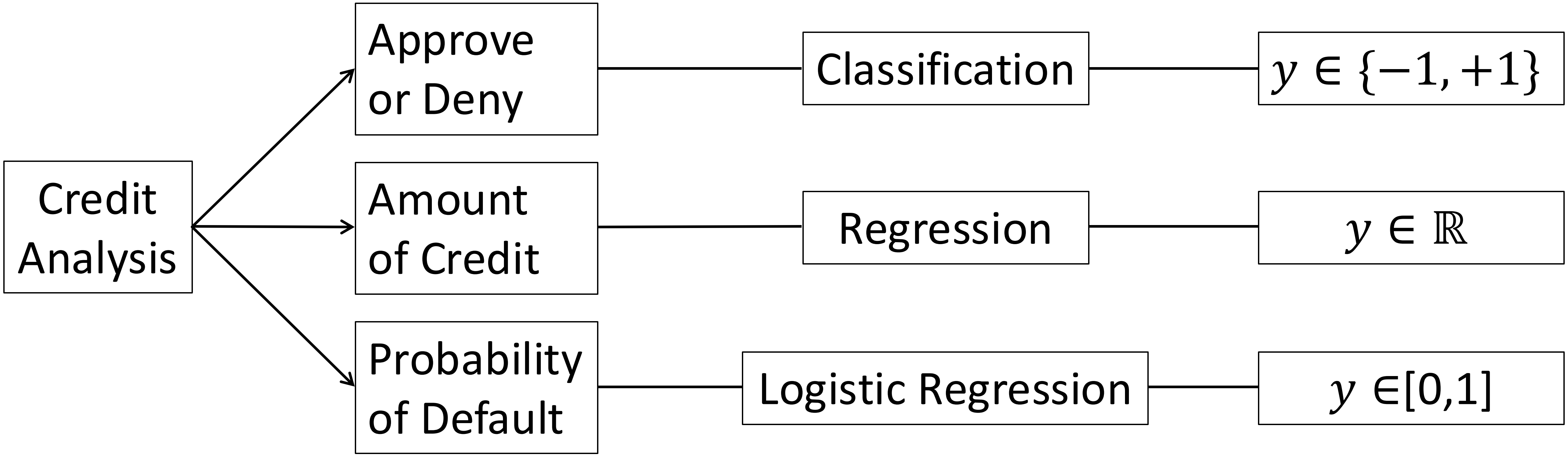
- Can learn the target $P(y|\boldsymbol{x})$

so long as data points are drawn from $P(\boldsymbol{x})$ i.i.d.

Learning Problem with Error Measure, Noisy Target



Linear Models for Three Learning Problems

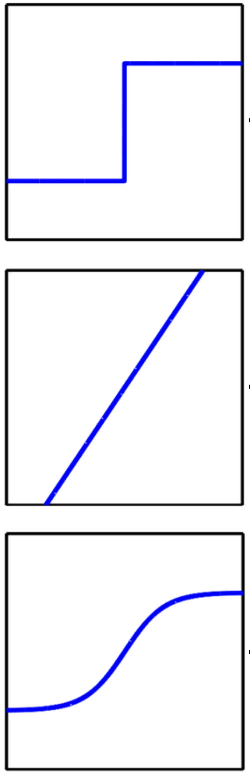


- Fundamental: Building block for more complex models
- First model to try!

Linear Models: The Linear Signal

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}, \qquad \text{where } \boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{x} \in 1 \times \mathbb{R}^d$$

Linear Models: The Linear Signal



$$\mathbf{w}^T\mathbf{x} \longrightarrow \begin{cases} sign(\mathbf{w}^T\mathbf{x}) \longrightarrow \{-1,+1\} \\ \mathbf{w}^T\mathbf{x} \longrightarrow \mathbb{R} \\ \theta(\mathbf{w}^T\mathbf{x}) \longrightarrow [0,1] \end{cases}$$

- $sign(\mathbf{w}^T\mathbf{x}) \longrightarrow \{-1,+1\}$ ← Classification
- $\mathbf{w}^T\mathbf{x} \longrightarrow \mathbb{R}$ ← Regression
- $\theta(\mathbf{w}^T\mathbf{x}) \longrightarrow [0,1]$ ← Logistic Regression

Linear Model for Classification

$$\mathcal{H} = \{h: h(x) = sign(w^T x)\}$$

Classification error on point $\boldsymbol{x}$: $e\big(h(\boldsymbol{x}), f(\boldsymbol{x})\big) = [\![h(\boldsymbol{x}) \neq f(\boldsymbol{x})]\!]$
Overall error is average value of point-wise error

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N} e(h(\boldsymbol{x}_n), f(\boldsymbol{x}_n))$$

$$E_{out}(h) = \mathbb{E}_{\boldsymbol{x}}[e(h(\boldsymbol{x}), f(\boldsymbol{x}))]$$

Ultimate Goal: Minimize $E_{out}(h)$
Typical Learning Algorithm Goal: Minimize $E_{in}(h)$

How to Learn a Final Hypothesis g from $\mathcal{H}$?

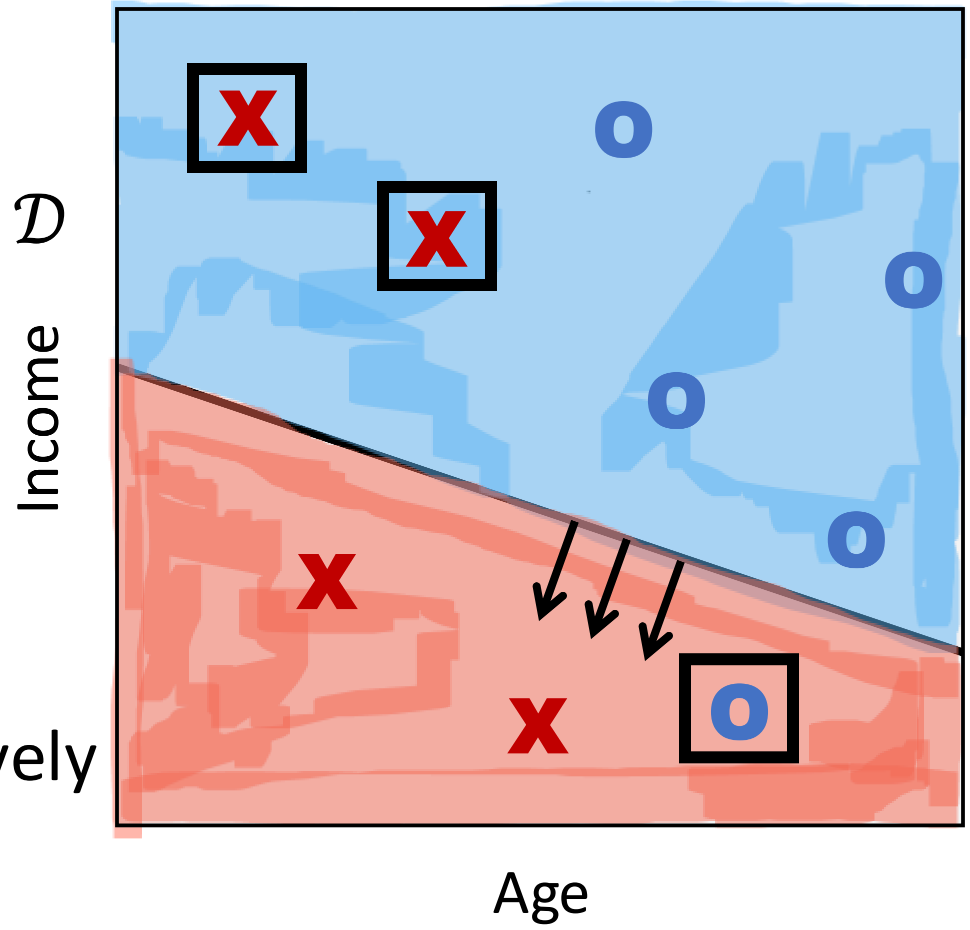
- Want: Select g from $\mathcal{H}$ so that $g \approx f$
- Certainly want $g \approx f$ on the dataset $\mathcal{D}$, i.e.,

$$g(\boldsymbol{x}_n) = y_n \text{ for each } (x_n, y_n) \text{ in } \mathcal{D}$$

- But $\mathcal{H}$ is uncountably infinite **(more on this later)**

How to find g in the infinite hypothesis set $\mathcal{H}$?

Start with *some* weights and improve it iteratively



The Perceptron Learning Algorithm (PLA)

A simple iterative algorithm

1. $\boldsymbol{w}(0) = \boldsymbol{0}$ Start at some weights
2. **for** iteration $t = 1, 2, 3, \dots$ **do**
3. the weight vector is $\boldsymbol{w}(t)$
4. **from** $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$ pick *any* misclassified example
5. let (x_*, y_*) be the misclassified example Observe a misclassification
$$sign(\boldsymbol{w}(t) \cdot \boldsymbol{x}_*) \neq y_*$$
6. update the weights
$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y_* \boldsymbol{x}_*$$
The update rule: Correct a misclassification
7. $t \leftarrow t + 1$

"incremental learning" one example at a time

The Perceptron Update Rule

- $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y_* \boldsymbol{x}_*$

Mistake when $y_* = +1$: Increase the score of $\boldsymbol{x}_*$

$$\boldsymbol{w}^T(t+1)\boldsymbol{x}_* = (\boldsymbol{w}^T(t) + \boldsymbol{x}_*^T)\boldsymbol{x}_* = \boldsymbol{w}^T(t)\boldsymbol{x}_* + \boldsymbol{x}_*^T\boldsymbol{x}_*$$
$$> \boldsymbol{w}^T(t)\boldsymbol{x}_*$$

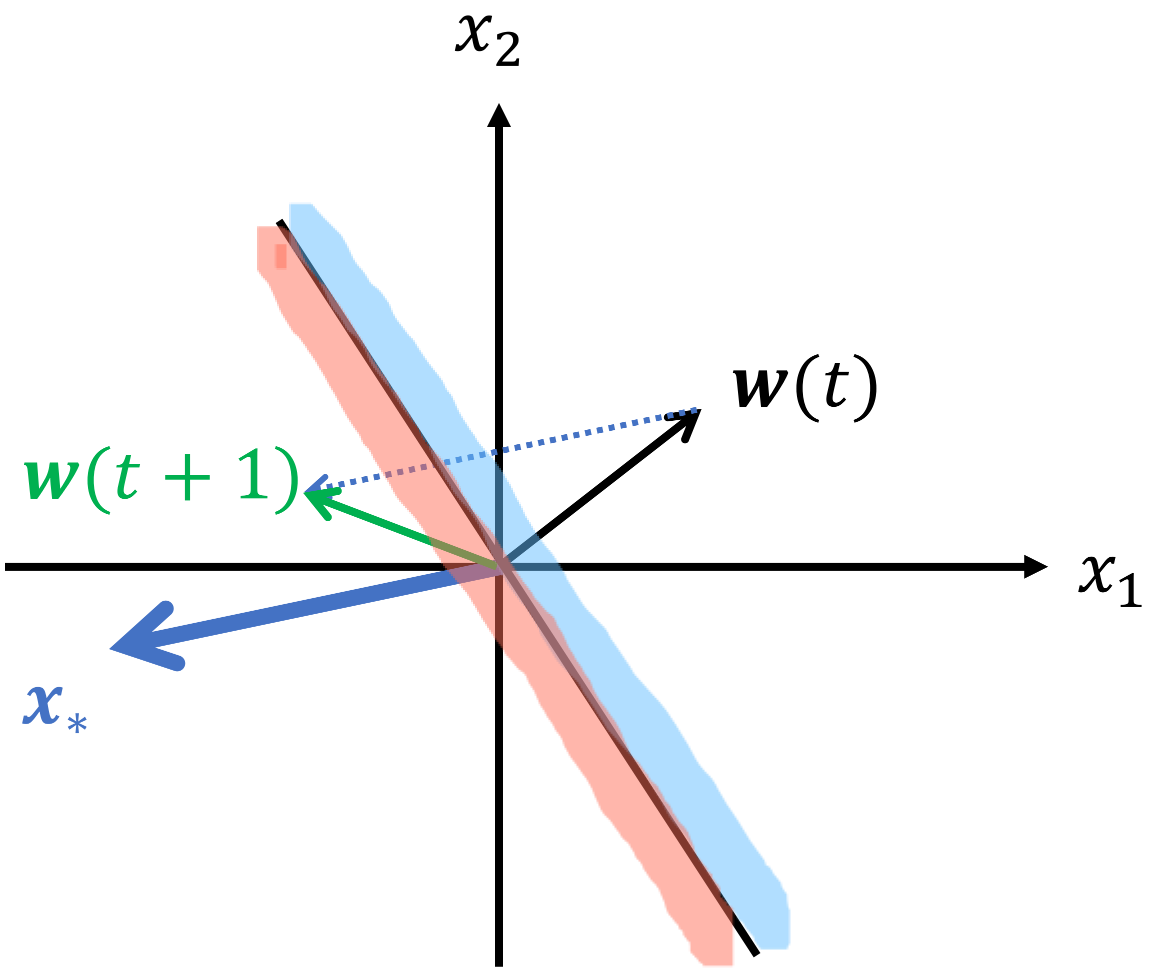
Mistake when $y_* = -1$: Decrease the score of $\boldsymbol{x}_*$

$$\boldsymbol{w}^T(t+1)\boldsymbol{x}_* = (\boldsymbol{w}^T(t) - \boldsymbol{x}_*^T)\boldsymbol{x}_* = \boldsymbol{w}^T(t)\boldsymbol{x}_* - \boldsymbol{x}_*^T\boldsymbol{x}_*$$
$$< \boldsymbol{w}^T(t)\boldsymbol{x}_*$$

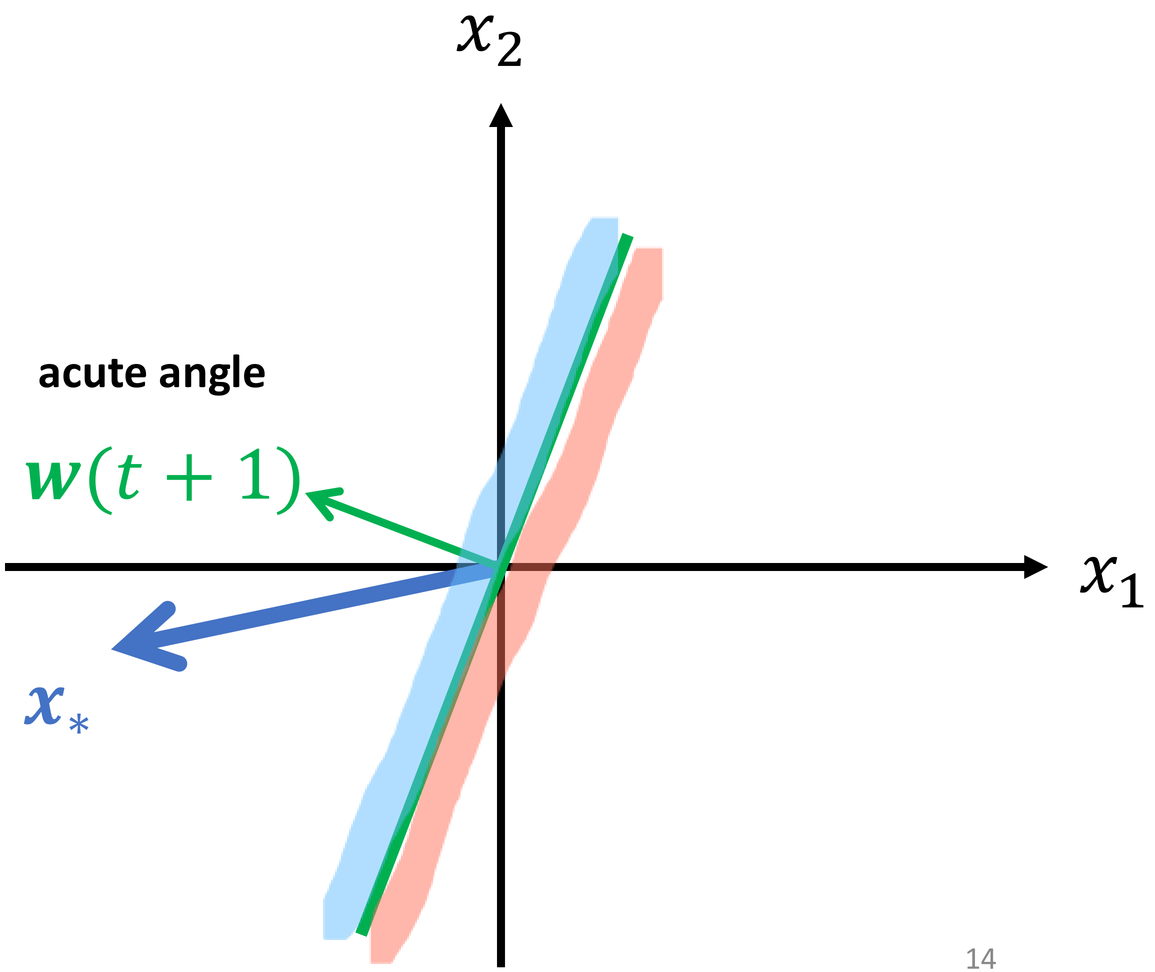
$$sign(w_1x_1 + w_2x_2 + \cdots + w_dx_d - threshold)$$
$$sign(w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_0)$$

The Perceptron Update Rule

$\boldsymbol{y}_* = +\mathbf{1}$
and
$\boldsymbol{w}^T(t)\boldsymbol{x}_* < 0$

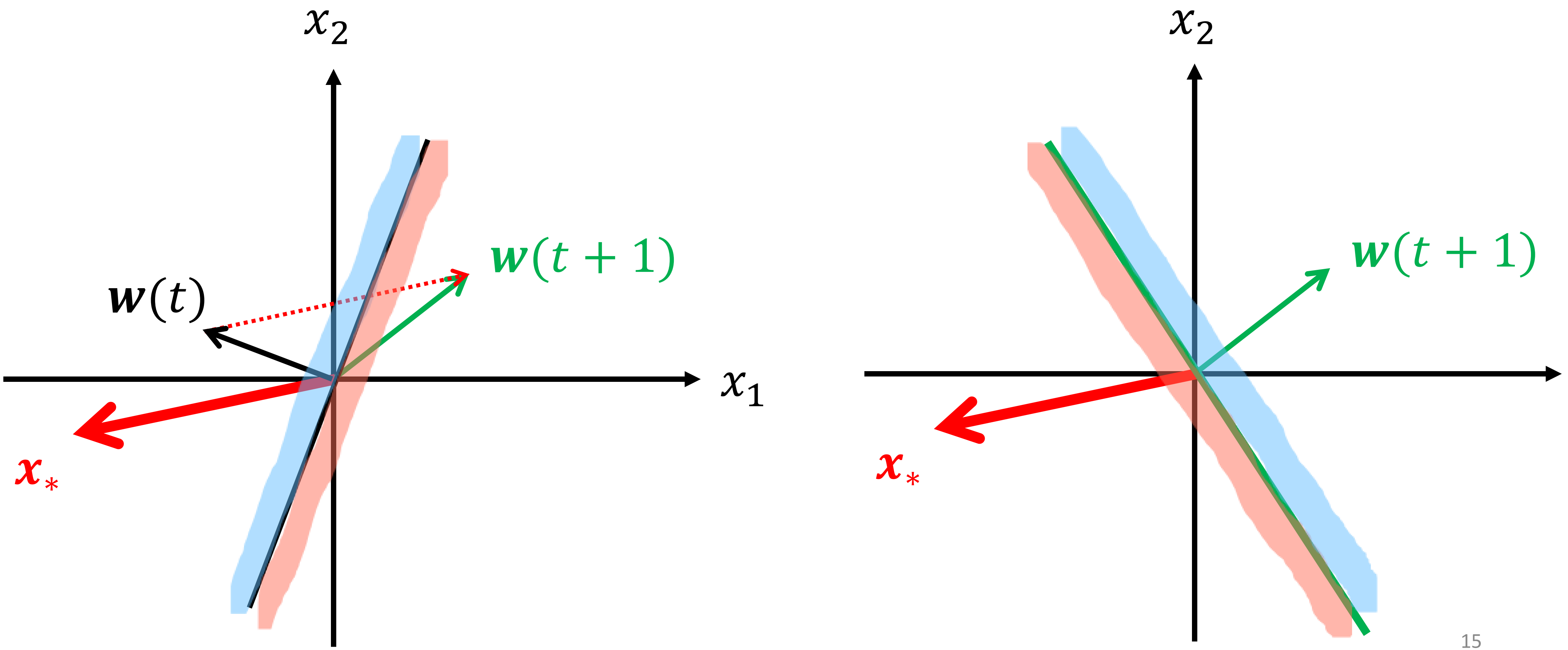


obtuse angle



The Perceptron Update Rule

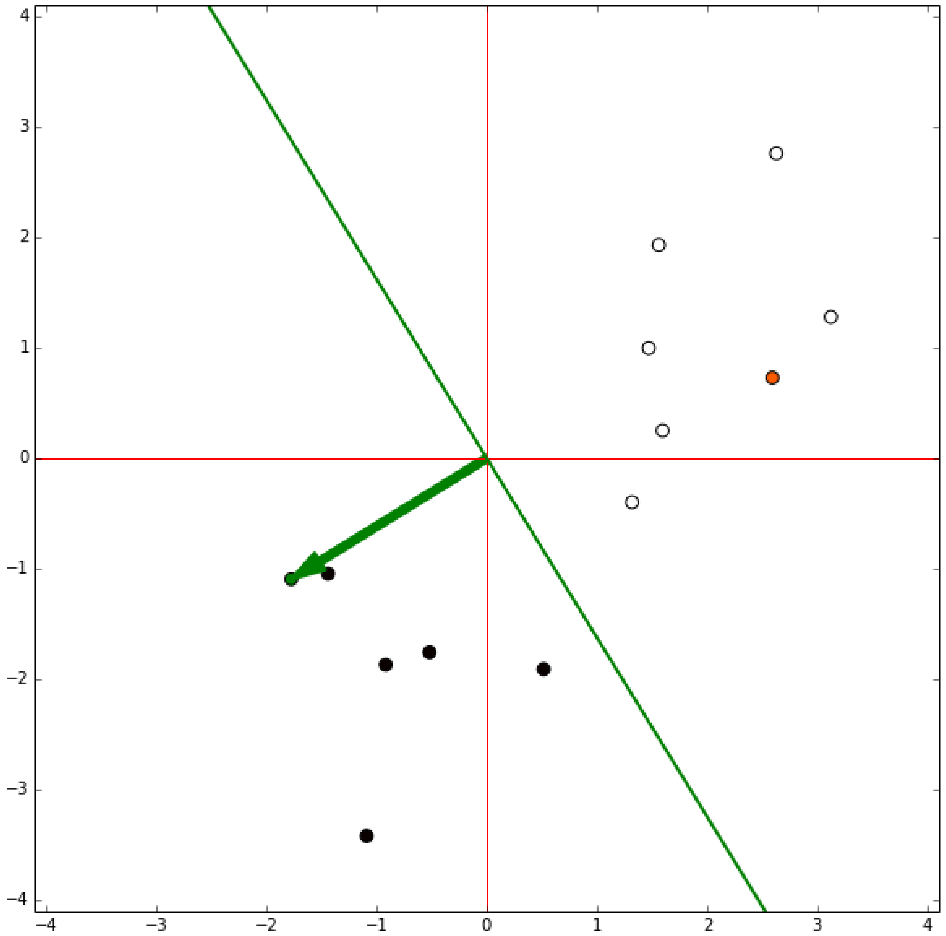
$\boldsymbol{y_* = -1}$
and
$\boldsymbol{w}^T(t)\boldsymbol{x}_* > 0$



Does PLA Work?

Theorem: If the data can be fit by a linear separator,

then after some ***finite*** number of steps, PLA will find one



Pocket Algorithm (for Linear Classification)

The Pocket Algorithm:

- Run the Perceptron Learning Algorithm $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{x}_* y_*$

Here, x_* is any data point misclassified due to $\boldsymbol{w}(t)$

- In each round, record E_{in} (and $\boldsymbol{w}$) if it is the best E_{in} observed so far

Other approaches:

- Linear regression (coming soon)
- Logistic regression (coming soon)
- Linear Programming

$$\min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} 1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n)$$

Evaluating Classifiers

Class:

Bat first= ‘yes’

Bat first = ‘no’

Outlook	Temperature	Humidity	Wind	Bat First
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No
Rainy	Hot	High	False	No

Classifier Evaluation Metrics: Confusion Matrix

- **Confusion Matrix** (CM)**:**

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

- In a confusion matrix with m classes, $CM_{i,j}$ indicates # of tuples in class i that were labeled by the classifier as class j
  - May have extra rows/columns to provide totals
- **Example of Confusion Matrix:**

Actual class\Predicted class	Bat first = yes	Bat first = no	Total
Bat first = yes	**6954**	**46**	7000
Bat first = no	**412**	**2588**	3000
Total	7366	2634	10000

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Real-world truth

Predictions

- **Classifier accuracy,** or recognition rate
  - Percentage of test set tuples that are correctly classified

    Accuracy = (TP + TN)/All
- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

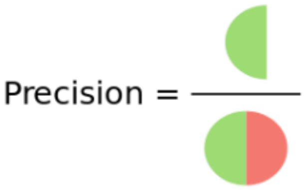
- **Class imbalance problem**
  - One class may be *rare*
    - E.g., fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - Handling the class imbalance problem
    - **Sensitivity** (recall): True positive recognition rate
      - **Sensitivity = TP/P**
    - **Specificity**: True negative recognition rate
      - **Specificity = TN/N**

Classifier Evaluation Metrics: Precision and Recall, and F-measures

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

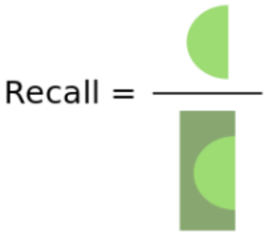
- **Precision** (exactness): what % of tuples that the classifier labeled as positive are actually positive?

$$P = \text{Precision} = \frac{TP}{TP + FP}$$

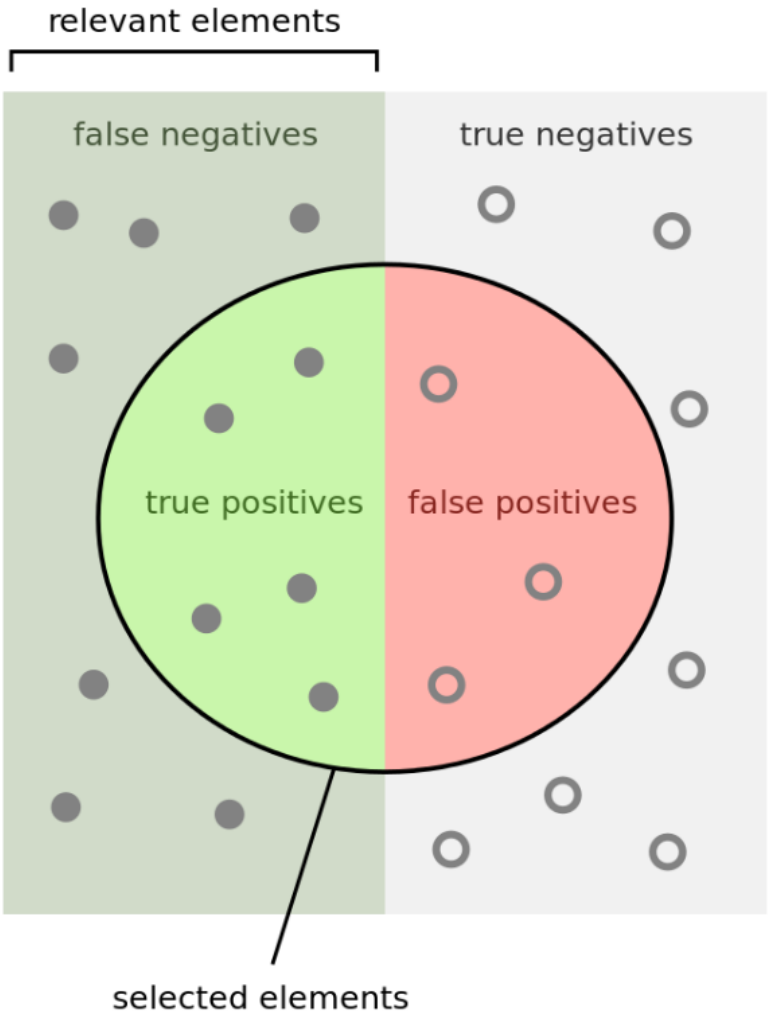


- **Recall** (completeness): what % of positive tuples did the classifier label as positive?

$$R = \text{Recall} = \frac{TP}{TP + FN}$$



  - Range: [0, 1]



https://en.wikipedia.org/wiki/Precision_and_recall

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- The "inverse" relationship between precision & recall
- ***We want one number to say if a classifier is good or not***
- ***F* measure** (or ***F*-score**): harmonic mean of precision and recall
  - In general, it is the weighted measure of precision & recall

$$F_\beta = \frac{1}{\alpha \cdot \frac{1}{P} + (1 - \alpha) \cdot \frac{1}{R}} = \frac{(\beta^2 + 1)P * R}{\beta^2 P + R}$$

Assigning β times as much weight to recall as to precision

  - ***F1-measure (balanced F-measure)***
    - That is, when β = 1,

$$F_1 = \frac{2P * R}{P + R}$$

Classifier Evaluation Metrics: Example

- Use the same confusion matrix, calculate the measure just introduced

Actual Class\Predicted class	cancer = yes	cancer = no	Total
cancer = yes	**90**	**210**	300
cancer = no	**140**	**9560**	9700
Total	230	9770	10000

- Sensitivity =
- Specificity =
- Accuracy =
- Error rate =
- Precision =
- Recall =
- F1 =

Quiz 2, Problem 4

(3.5 points)