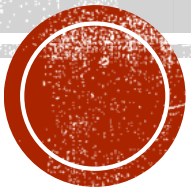
PRACTICAL ISSUES



CS 412 Introduction to Machine Learning

Prof. Zheleva

February 6, 2018

Reading Assignment: CIML: 5, ISL: 5.1

LAST LECTURE: PERCEPTRON

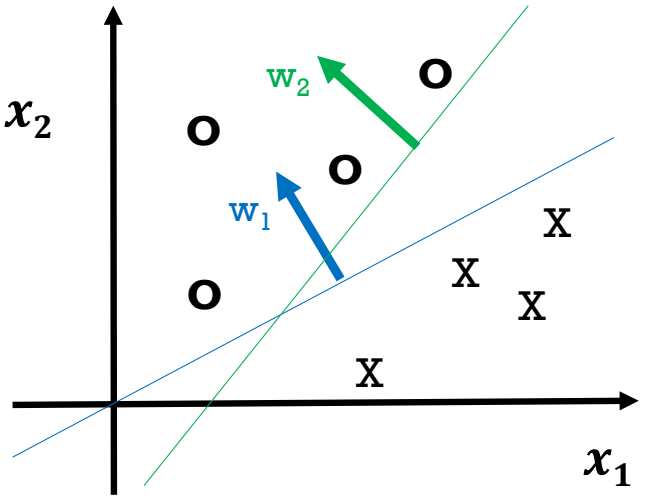
- Perceptron concepts
  - Training/prediction algorithms (standard, voted, averaged)
  - Convergence theorem and what practical guarantees it gives us
  - How to describe the decision boundary of a perceptron classifier
- Fundamental ML concepts
  - Determine whether a data set is linearly separable and define its margin
  - Error-driven algorithms, online vs. batch algorithms

FUNCTION APPROXIMATION WITH PERCEPTRON

- Problem setting
  - **X** – set of possible instances
    - Each instance $x \in X$ is a feature vector $x = [x_1, \ldots, x_D]$
  - Unknown target function f: **X** $\rightarrow$ Y
    - Classification: Y is binary valued
  - Set of function hypotheses H={h | h: **X** $\rightarrow$ Y}
    - Each hypothesis h is a hyperplane in D-dimensional space

- Input
  - Training examples $\{(x^{(1)}, y^{(1)}), \ldots (x^{(N)}, y^{(N)})\}$ of unknown distribution

- Output
  - Hypothesis $h \in H$ that best approximates target function f



$$a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$$

TODAY: PRACTICAL ISSUES

- Finish up with Perceptron
- Practical issues in machine learning
  - Feature representation
  - Evaluation metrics
  - Revisit bias-variance tradeoff

CONVERGENCE OF PERCEPTRON

Theorem (Block & Novikoff, 1962):

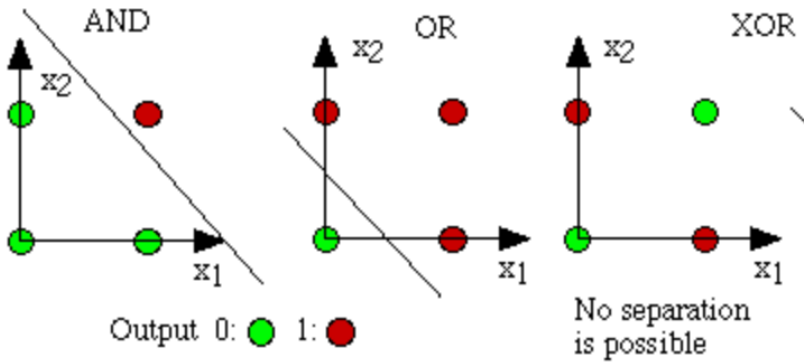
If the training data $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots (\boldsymbol{x}^{(N)}, y^{(N)})\}$ is linearly separable with margin γ by a unit norm hyperplane w_*, assuming $\left|\left|\boldsymbol{x}^{(i)}\right|\right| \leq R$ for all i and b=0, then perceptron converges after $\frac{R^2}{\gamma^2}$ errors during training.

What does it mean?

- Perceptron converges quickly when margin is large, slowly when it is small
- Bound does not depend on number of training examples N, nor on number of features
- Proof guarantees that perceptron converges but not necessarily to the max margin separator

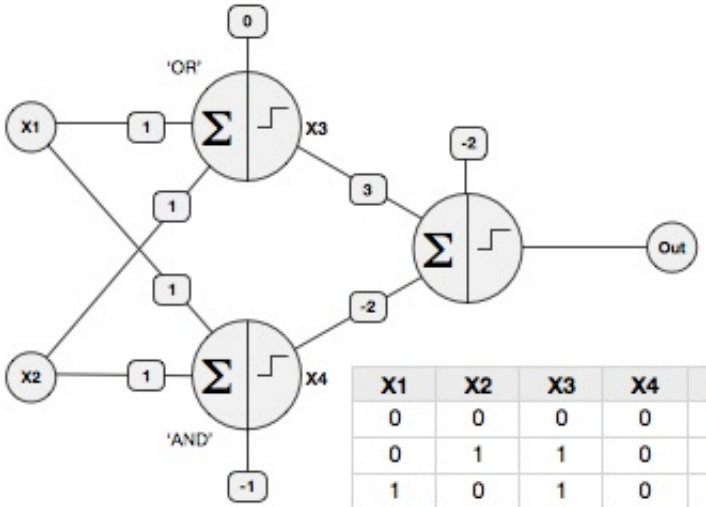
PERCEPTRON FALLS OUT OF FAVOR

- Minsky and Papert (1969) show that (single layer) neuron model is very limited
  - Requires linearly separable data
  - Cannot learn "XOR" function
  - Many falsely believe this limitation is for all neural networks (not just single layer)
  - Interest in neural networks greatly diminishes



PERCEPTRON FALLS OUT OF FAVOR

- One way around the XOR is feature combinations
  - Combinatorial feature explosion
    - From D to $O(D^2)$ dimensions
- Beyond linear separation
  - Kernels
  - Neural networks



X1	X2	X3	X4	Out
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

ML APPLICATION DESIGN

1	real world goal	increase revenue
2	real world mechanism	better ad display
3	learning problem	classify click-through
4	data collection	interaction w/ current system
5	collected data	query, ad, click
6	data representation	bow^2, $\pm$ click
7	select model family	decision trees, depth 20
8	select training data	subset from april'16
9	train model & hyperparams	final decision tree
10	predict on test data	subset from may'16
11	evaluate error	zero/one loss for $\pm$ click
12	deploy!	(hope we achieve our goal)

PRACTICAL ISSUES

- "Garbage in, garbage out"
  - Learning algorithms can't compensate for useless training examples
    - E.g., if all features are irrelevant
  - Feature design can have bigger impact on performance than tweaking the learning algorithm

PICKING GOOD FEATURES

Figure 5.1: object recognition in pixels

Figure 5.2: object recognition in patches

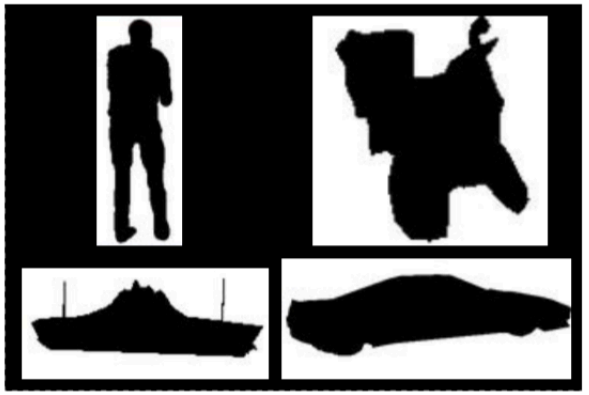


Figure 5.3: object recognition in shapes

Figure 5.15: object recognition with full information

EXPECTATION AND VARIANCE

- **Expectation**: what should happen <u>on average</u>

$$E[f(X)] = \sum P(x)\, f(x)$$

  - Linearity properties

$$E[f_1(X) + f_2(X)] = E[f_1(X)] + E[f_2(X)]$$

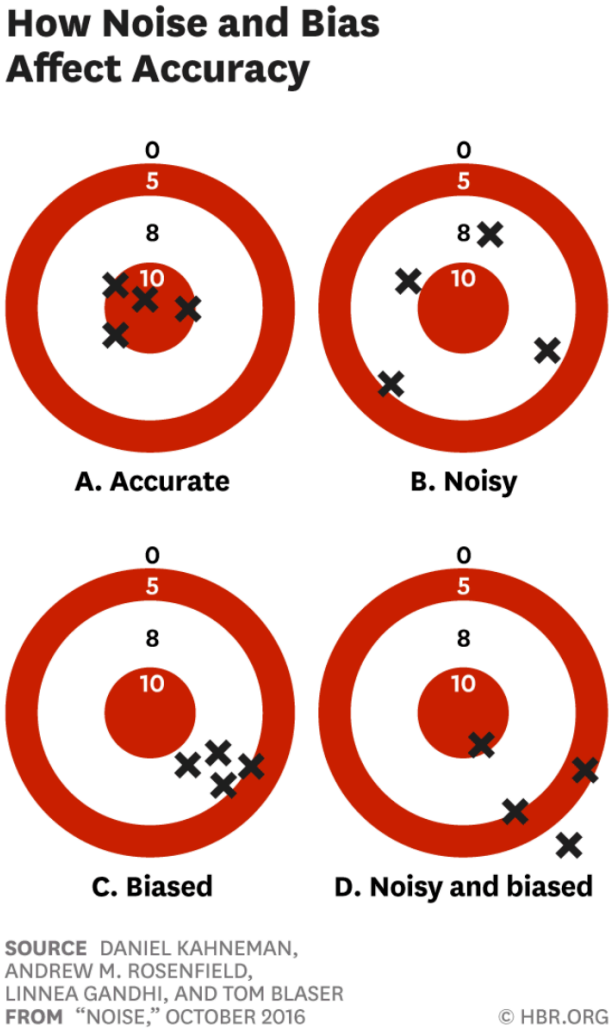
$$E[\alpha\, f(X)] = \alpha\, E[f(X)]$$

- **Bias**: difference between expected and true values

$$\text{Bias}(f_D(x)) = E[f_D(x)] - f(x)$$

- **Variance**: measure of the "spread" of a distribution

$$\text{Var}(f_D(x)) = E[(f_D(x) - E[f_D(x)])^2] = E[f_D(x)^2] - (E[f_D(x)])^2$$



IMPROVING INPUT REPRESENTATION

- Feature pruning
  - Irrelevant features
    - Remove features with low variance
  - Redundant features
    - Good: DT
    - Not good: kNN, Perceptron
  - Dealing with combinatorial feature explosion
    - Can use DT to decide on useful combinations, then use features in perceptron

IMPROVING INPUT REPRESENTATION

- Feature normalization: one feature at a time
  - If significant difference in scale
  - If sparse data with binary features: not a good idea

$$\text{Centering:} \qquad x_{n,d} \leftarrow x_{n,d} - \mu_d \tag{5.1}$$

$$\text{Variance Scaling:} \qquad x_{n,d} \leftarrow x_{n,d} / \sigma_d \tag{5.2}$$

$$\text{Absolute Scaling:} \qquad x_{n,d} \leftarrow x_{n,d} / r_d \tag{5.3}$$

where:

$$\mu_d = \frac{1}{N} \sum_n x_{n,d} \tag{5.4}$$

$$\sigma_d = \sqrt{\frac{1}{N-1} \sum_n (x_{n,d} - \mu_d)^2} \tag{5.5}$$

$$r_d = \max_n \left| x_{n,d} \right| \tag{5.6}$$

IMPROVING INPUT REPRESENTATION

- Feature logarithmic transformation
  - "every time this feature doubles, I'm equally more like to predict a positive label"

$$x_d \mapsto \log_2(|x_d| + 1)\text{sign}(x_d)$$

- Example normalization: one example at a time

$$\boldsymbol{x}_n \leftarrow \boldsymbol{x}_n / \, ||\boldsymbol{x}_n||$$

  - Might be a good idea when merging datasets
  - When can it be a bad idea?

PRACTICAL ISSUES: EVALUATION

- So far we've measured classification performance using **accuracy**
- But this is not a good metric when some errors matter more than others
  - Given medical record, predict whether patient has cancer or not
  - Given a document collection and a query, find documents in collection that are relevant to query
- What to do if one class is more important than the other?

THE 2-BY-2 CONTINGENCY TABLE

- Imagine we are addressing a document retrieval task for a given query where +1 means that the document is relevant and -1 means that the document is not relevant
- For any classifier results, we can categorize each document as
  - True/False Positive
  - True/False Negative

	True label = +1	**True label = -1**
Predicted label = +1	True positives (TP)	False positives (FP)
True label = -1	False negatives (FN)	True negatives (TN)

PRECISION AND RECALL

- Precision: % positive predictions that are correct

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

- Recall: % positive true labels that are predicted

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

	True label = +1	**True label = -1**
Predicted label = +1	True positives (TP)	False positives (FP)
True label = -1	False negatives (FN)	True negatives (TN)

PRECISION/RECALL CURVE

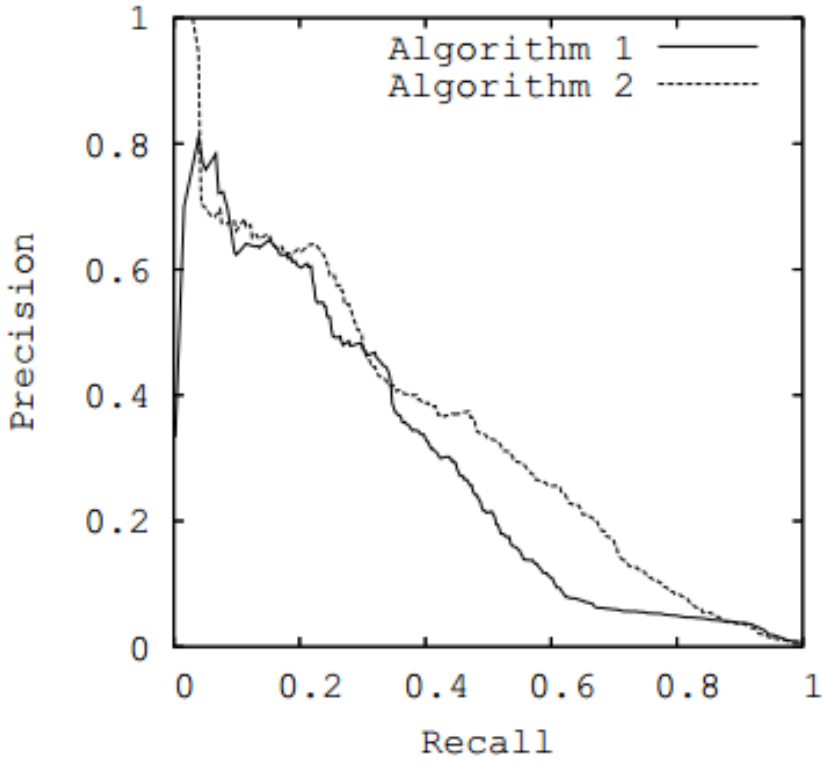
- Precision: % positive predictions that are correct

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

- Recall: % positive true labels that are predicted

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

- Different thresholds lead to different precision-recall tradeoff



F-SCORE

- Precision: % positive predictions that are correct

$$Pr = \frac{TP}{TP + FP}$$

- Recall: % positive true labels that are predicted

$$Re = \frac{TP}{TP + FN}$$

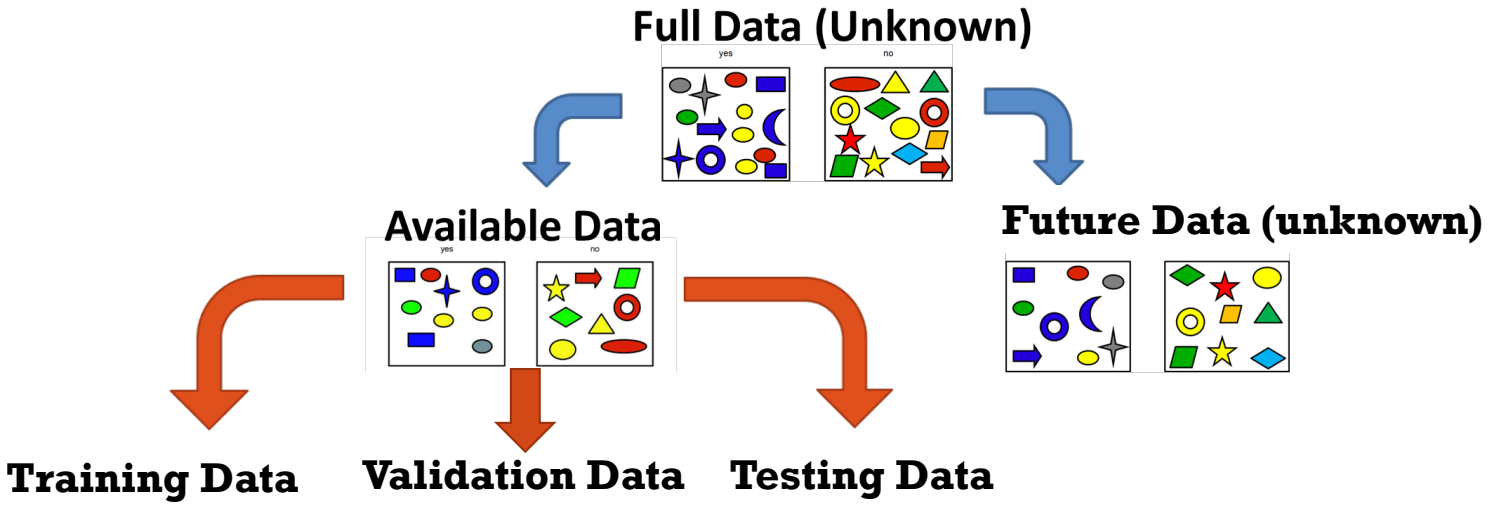
- F-score: harmonic mean

$$F = \frac{2 * Pr * Re}{Pr + Re}$$

- Weighted f-score: if precision is more important than recall

$$F = \frac{(1 + \beta^2) * Pr * Re}{\beta^2 (Pr + Re)}$$

SUPERVISED LEARNING



We use testing data for evaluation

CROSS-VALIDATION

Algorithm 8 CROSSVALIDATE(*LearningAlgorithm*, *Data*, K)

```
1:  ê ← ∞                                          // store lowest error encountered so far
2:  α̂ ← unknown                                    // store the hyperparameter setting that yielded it
3:  for all hyperparameter settings α do
4:      err ← [ ]                                  // keep track of the K-many error estimates
5:      for k = 1 to K do
6:          train ← {(x_n, y_n) ∈ Data : n mod K ≠ k − 1}
7:          test ← {(x_n, y_n) ∈ Data : n mod K = k − 1}   // test every Kth example
8:          model ← Run LearningAlgorithm on train
9:          err ← err ⊕ error of model on test     // add current error to list of errors
10:     end for
11:     avgErr ← mean of set err
12:     if avgErr < ê then
13:         ê ← avgErr                             // remember these settings
14:         α̂ ← α                                  // because they're the best so far
15:     end if
16: end for
```

SUMMARY

- Learning algorithm is only one of many steps in designing a ML application
- Many things can go wrong, but there are practical strategies for
  - Improving inputs
  - Evaluating
  - Tuning
  - Debugging (left for next time)
- Fundamental ML concepts: estimation vs. approximation error
  - Revisit bias-variance tradeoff (left for next time)

ANNOUNCEMENTS

- My office hours tomorrow, February 7, will be 9-11am instead of the usual 2-4pm

ACKNOWLEDGEMENTS

- These slides use materials by Marine Carpuat, Brian Ziebart