Lecture Notes for
Machine Learning in Python

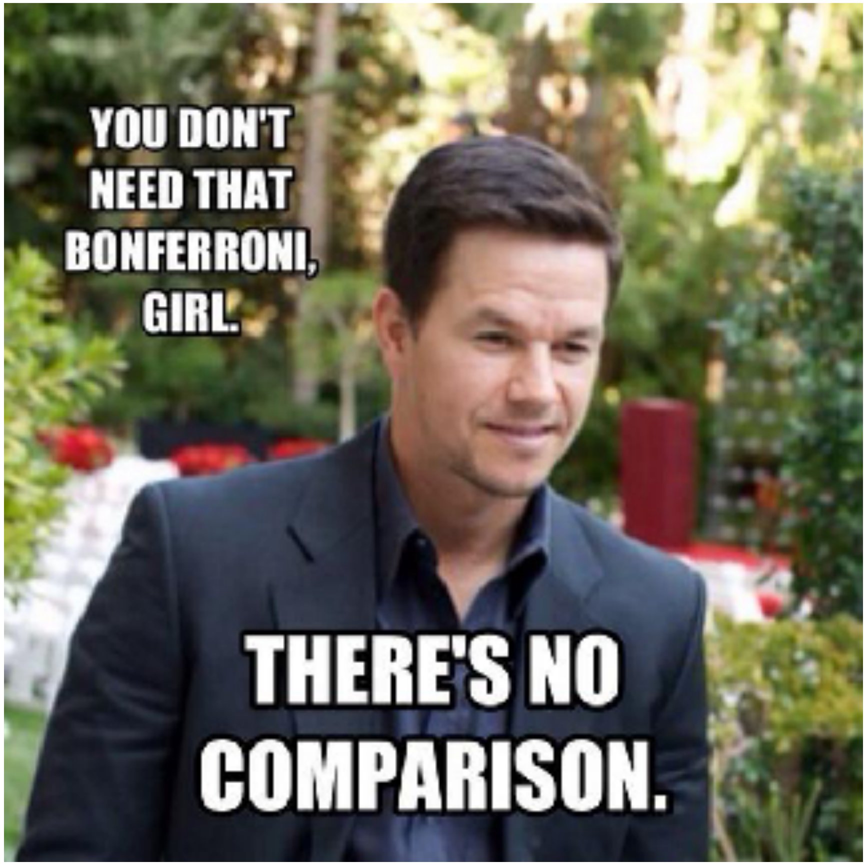
Professor Eric Larson

Week Nine A

Class Logistics and Agenda

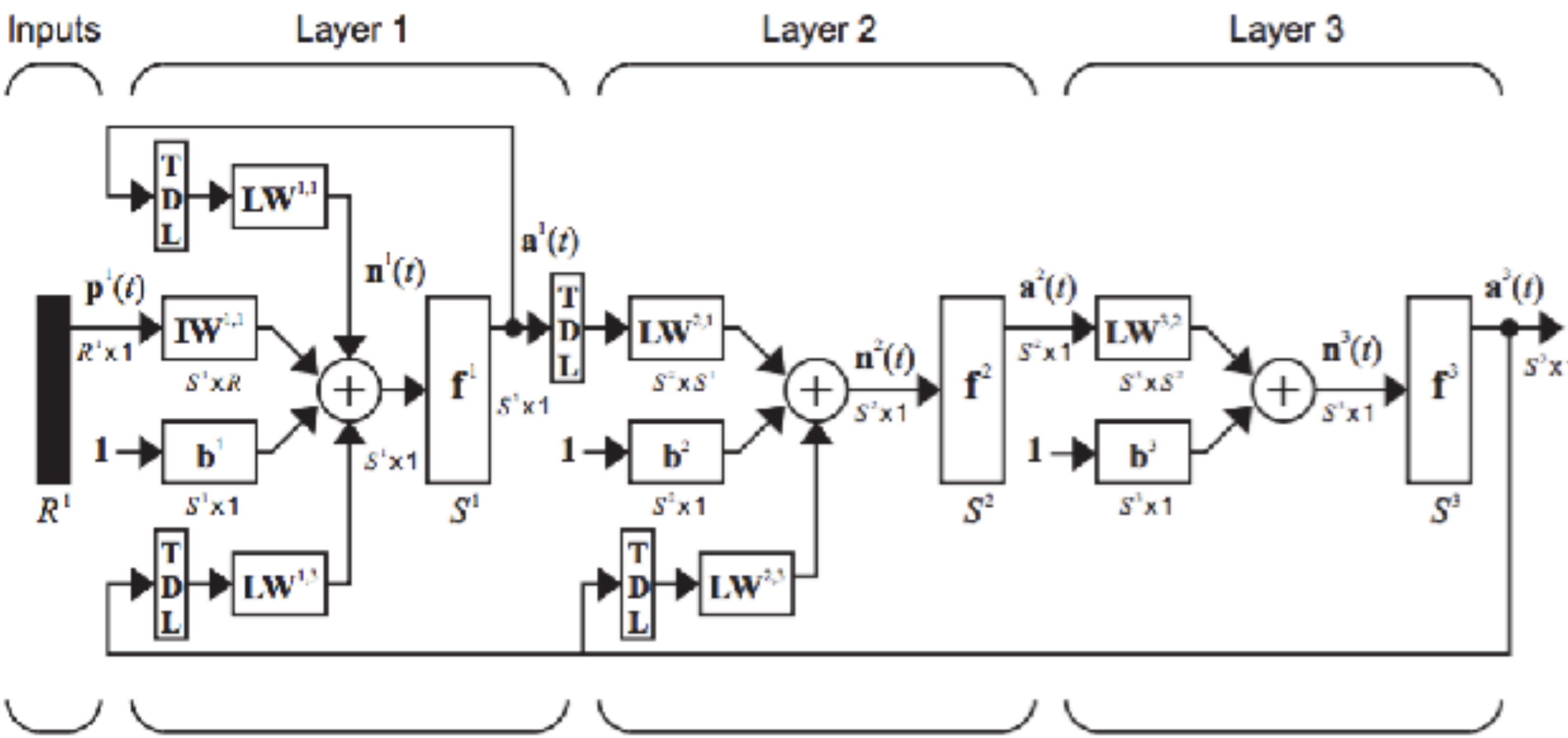
- Grades Coming Soon, but slowly
- **Next week**: project work week
- Agenda:
  - More advanced Neural Network Architectures
  - Ensemble methods
- **Next Time**: in-class assignment

Even More Advanced Architectures



More Advanced Architectures

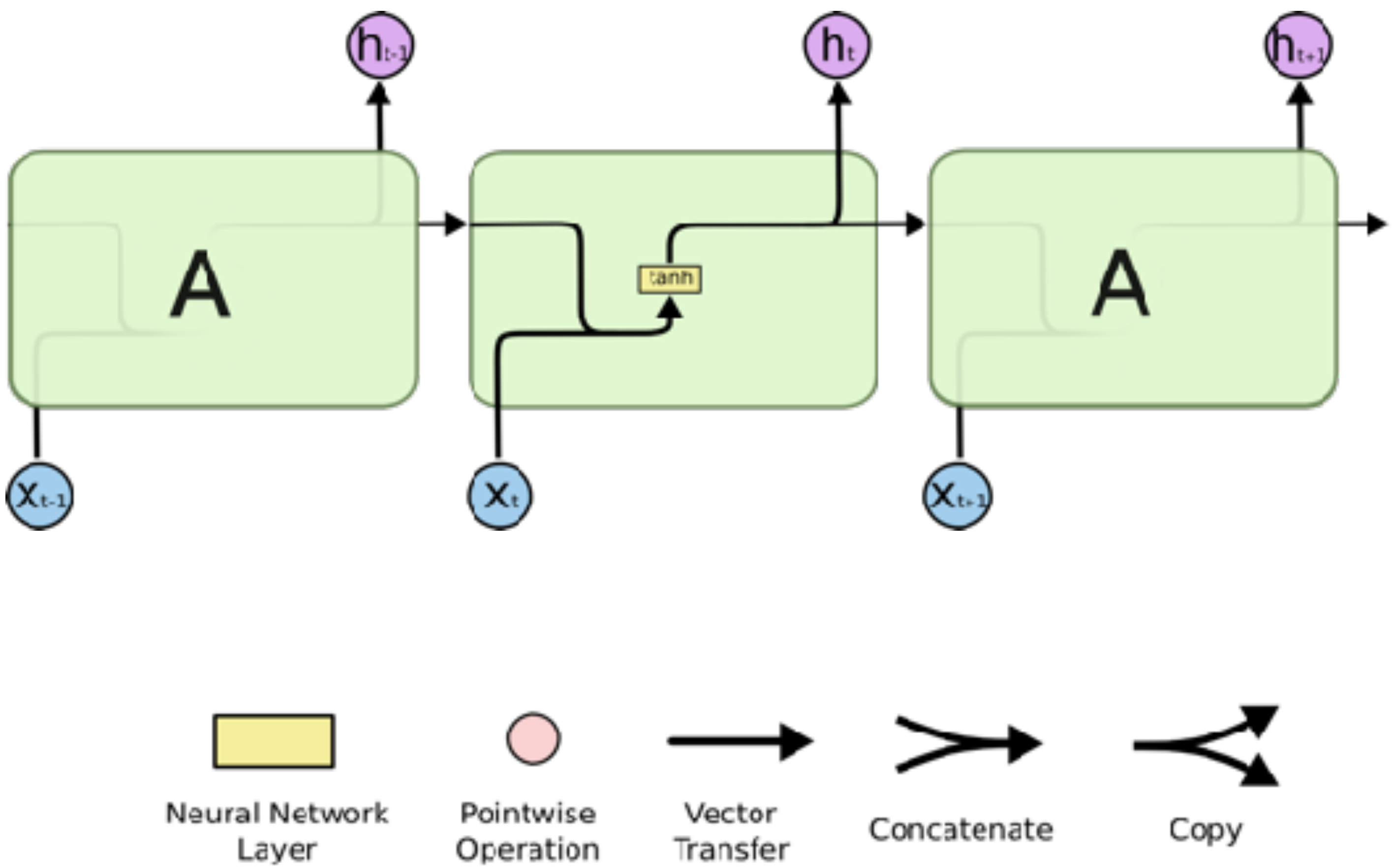
- Dynamic Networks (recurrent networks)
  - can use current and previous inputs, in time
  - still popular, but ultimately extremely hard to train
  - **highly successful variant**: long short term memory, **LSTM**



Neural Network Design, Hagan, Demuth, Beale, and De Jesus

More Advanced Architectures

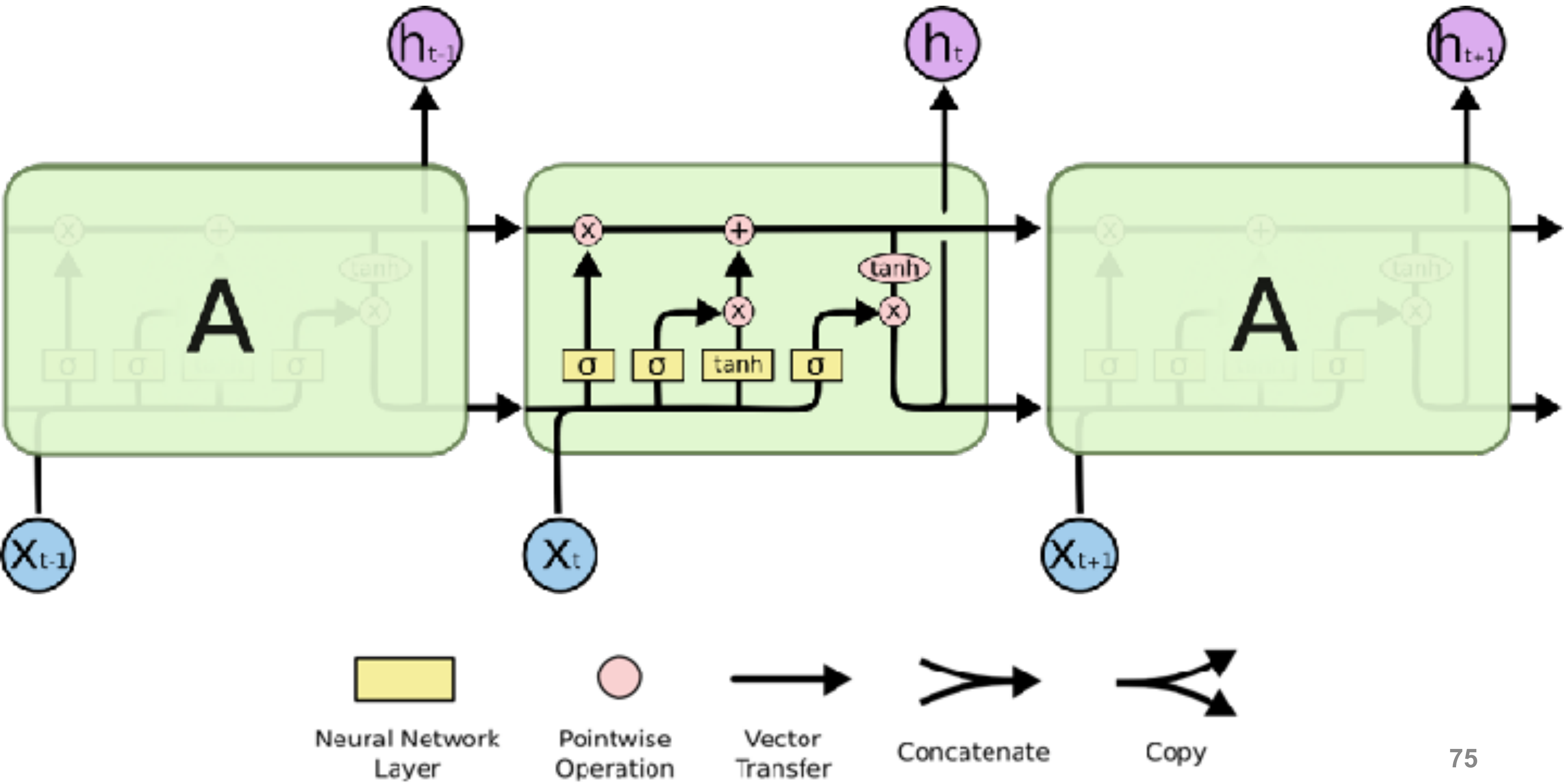
- **LSTM key idea**: limit how past data can affect output



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

More Advanced Architectures

- **LSTM key idea**: limit how past data can affect output

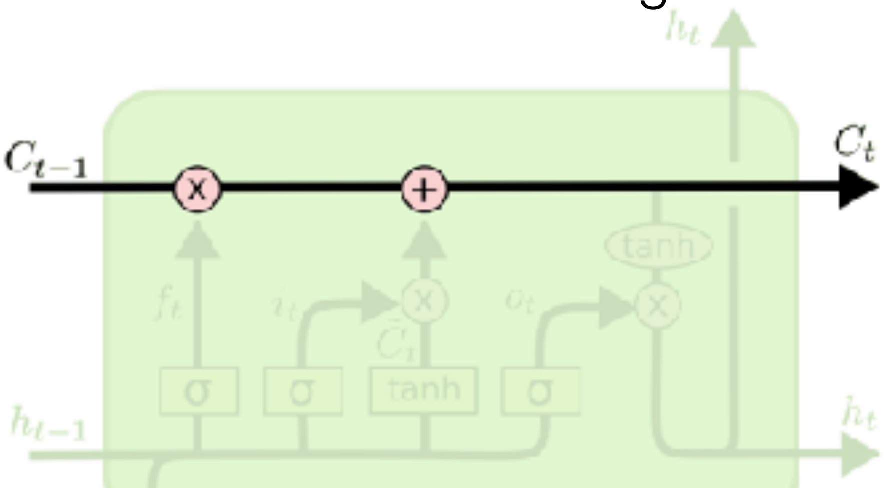


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

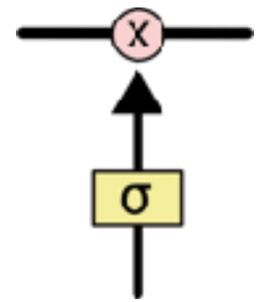
More Advanced Architectures

- **LSTM key idea**: limit how past data can affect output

let **cell state** through

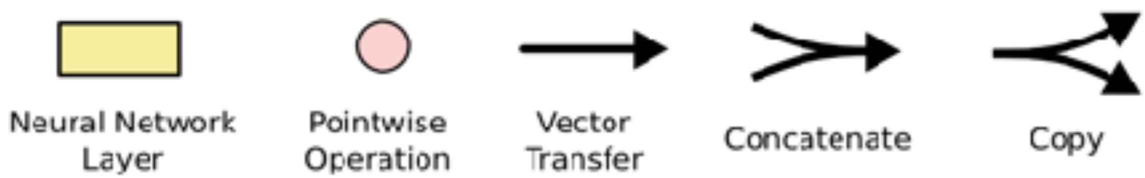


potentially **forget past** inputs via “gate” σ



we will return to this architecture later, for now:

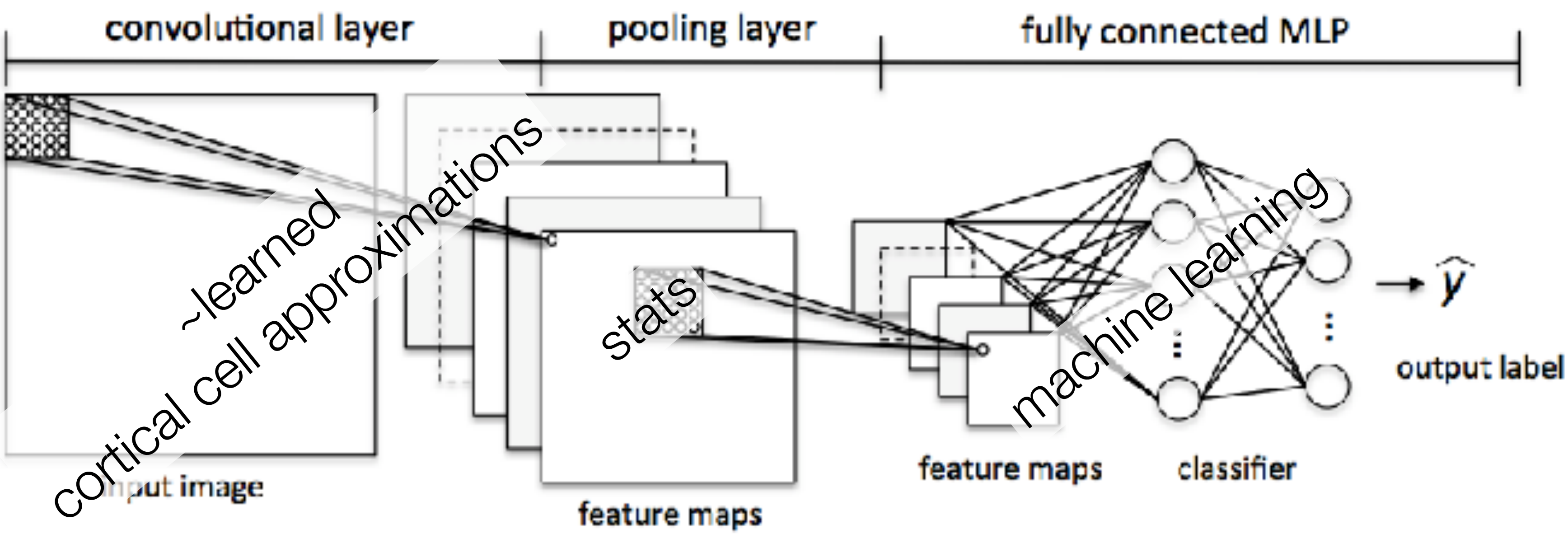
put it in long term memory 😂



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

More Advanced Architectures

- Convolutional Neural Networks
  - image processing operations



we will return to this architecture later, for now:

put it in long term memory

Problems with these Advanced Architectures

- These architectures have been around for 30 years
- And solved some amazingly hard problems
- but they had **big training problems** that back propagation could not solve readily:
  - unstable gradients (vanishing/exploding)
  - **extremely** non-convex space
    - more layers==many more local optima to get stuck
  - sometimes **gradient optimization is too computational** for weight updates
    - might need better optimization strategy than SGD
- The solution to these problems came from having large amounts of training data, better setup of the optimization
  - eventually was termed **deep learning**

End of Neural Networks Introduction

- Briefly step away from Neural Networks
- **Next time**: **In class assignment**:
    - evaluation and cross validation
- **Next Next time**: Ensembles

these will relate
back to neural networks

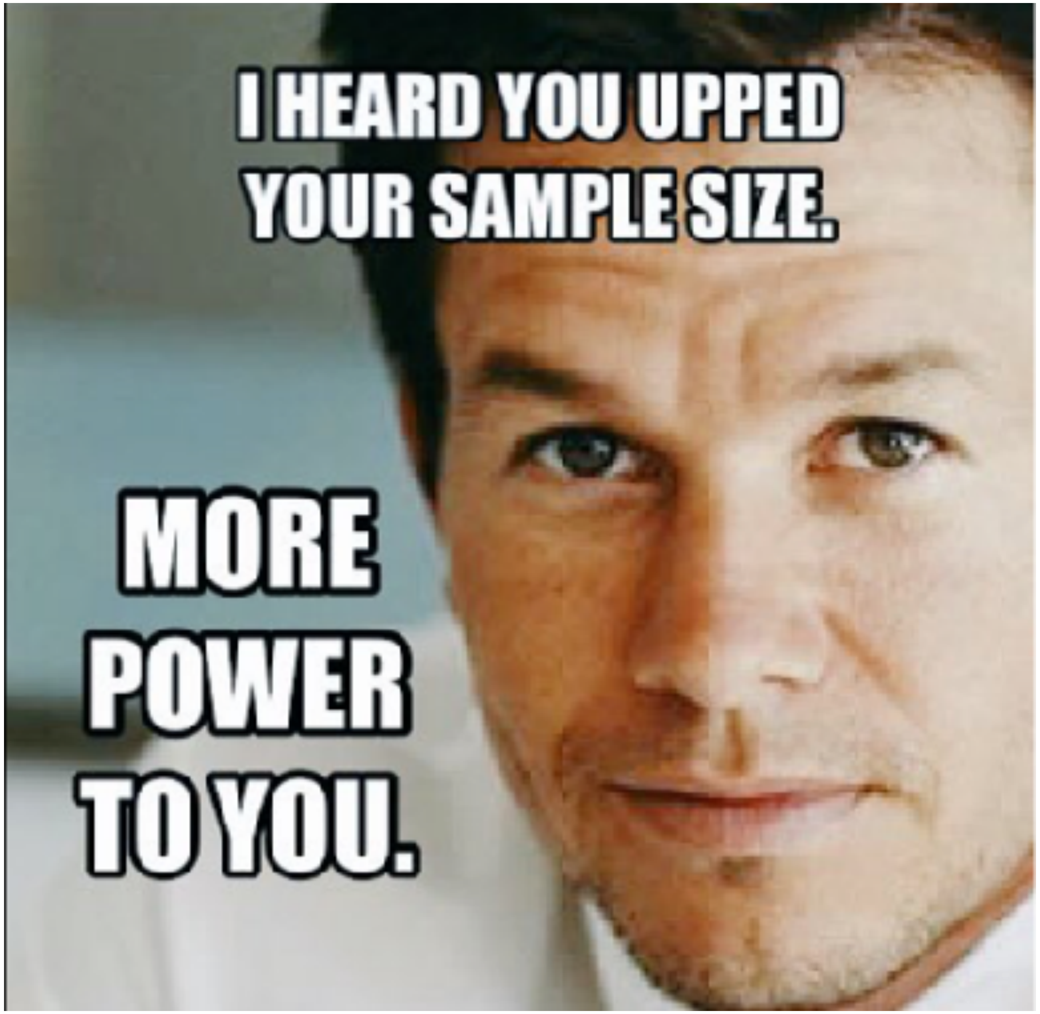
More help on neural networks:

Sebastian Raschka
https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch12/ch12.ipynb

Martin Hagan
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwioprvn27fPAhWMx4MKHYbwDIwQFggeMAA&url=http%3A%2F%2Fhagan.okstate.edu%2FNNDesign.pdf&usg=AFQjCNG5YbM4xSMm6K5HNsG-4Q8TvOu_Lw&sig2=bgT3k-5ZDDTPZ07Qu8Oreg

Michael Nielsen
http://neuralnetworksanddeeplearning.com

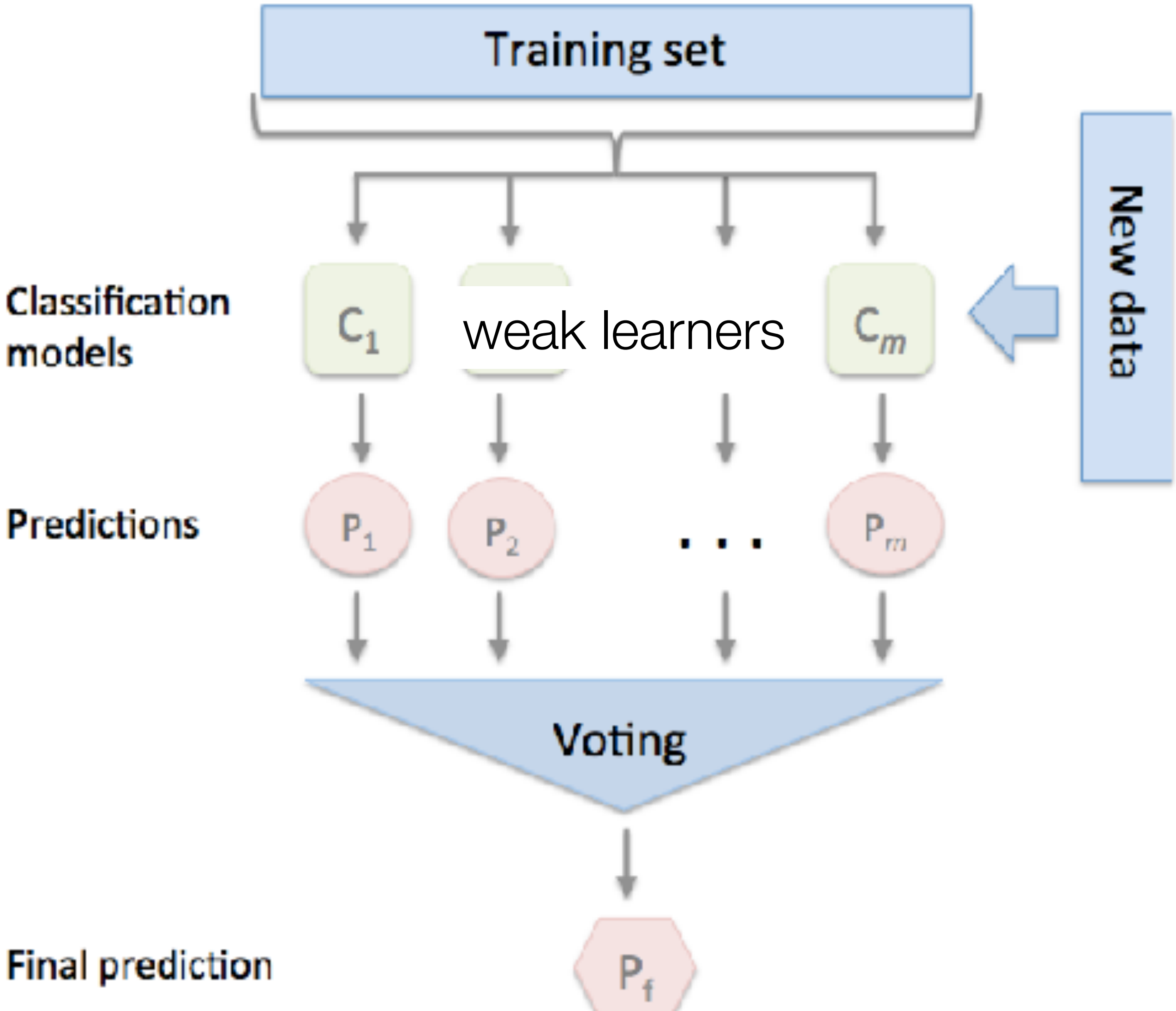
Classification: Ensemble Methods



Ensemble Methods

- Construct a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers
- Could be multiple Neural Networks

General Idea



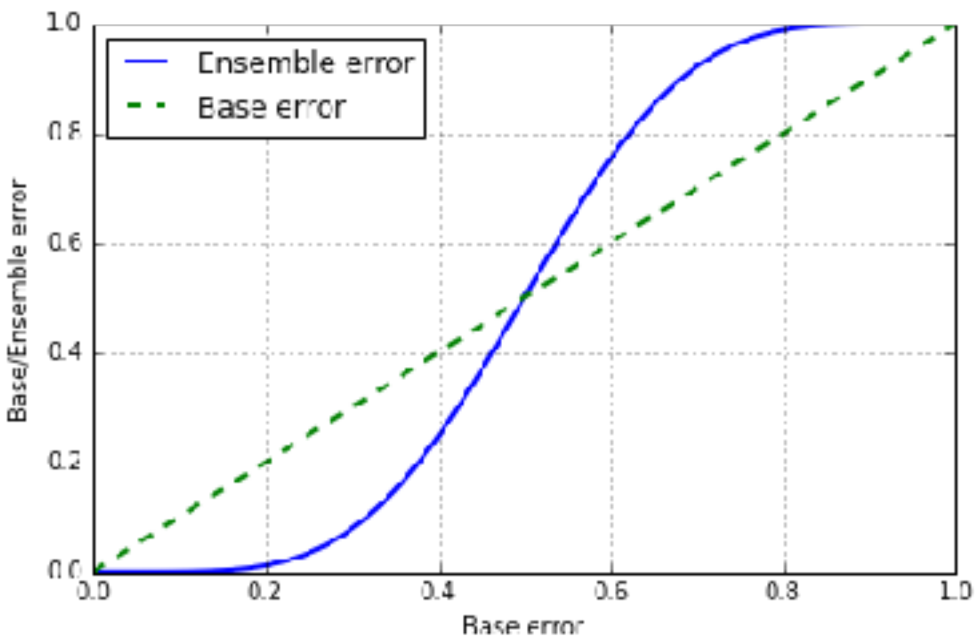
Weak and Strong

- **Weak learner**: a learner with better than chance accuracy (error < 50% for two class problem)
  - *i.e.,* classifier has high bias
  - like logistic regression
- **Strong learner**: arbitrarily small error rate
  - like MLP or kernel SVM
- Need to Ensemble many weak learners

Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent, so they make errors on different samples from dataset
  - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$



- But in practice, our classifiers are correlated, so it **does not work this well**

Why does it work?

- How much does this horse weigh?
  - Average of the guesses from many people is close to the true value
  - Average of *many* people is better than an expert's guess

Self Test:

A. 250 lbs
B. 750 lbs
C. 1200 lbs
D. 5000 lbs

Ensembles of Different Classifiers

- Step one: train m classifiers from dataset, $C_m(\boldsymbol{x})$
- Step two: combine outputs
  - majority vote:

$$\underset{\mathrm{i}}{\arg\max} \sum_j w_j [C_j(\boldsymbol{x}) = \mathrm{i}]$$

trust in classifier   classifier selected i

  - majority probabilistic vote:

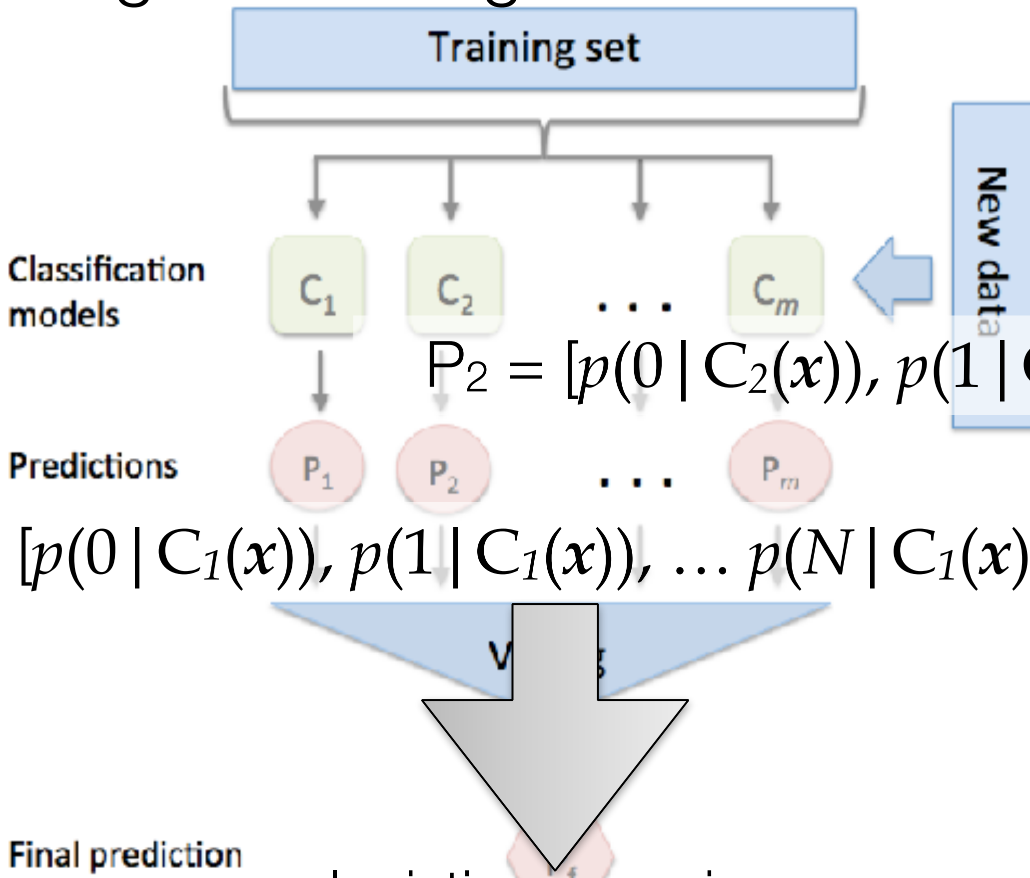
$$\underset{\mathrm{i}}{\arg\max} \sum_j w_j\, p(\mathrm{i} \mid C_j(\boldsymbol{x}))$$

trust in classifier

predict_proba i

Examples of Ensemble Methods

- Stacking/Cascading



$$P_2 = [p(0 \mid C_2(\boldsymbol{x})), p(1 \mid C_2(\boldsymbol{x})), \ldots p(N \mid C_2(\boldsymbol{x}))]$$

$$P_1 = [p(0 \mid C_1(\boldsymbol{x})), p(1 \mid C_1(\boldsymbol{x})), \ldots p(N \mid C_1(\boldsymbol{x}))]$$

logistic regression

$$\arg\max_{\mathrm{i}} \sum_j w_j \, p(\mathrm{i} \mid C_j(\boldsymbol{x}))$$ basically a way of getting w_j

Examples of Ensemble Methods

- Training set sampling methods:
    - Bagging
    - Boosting

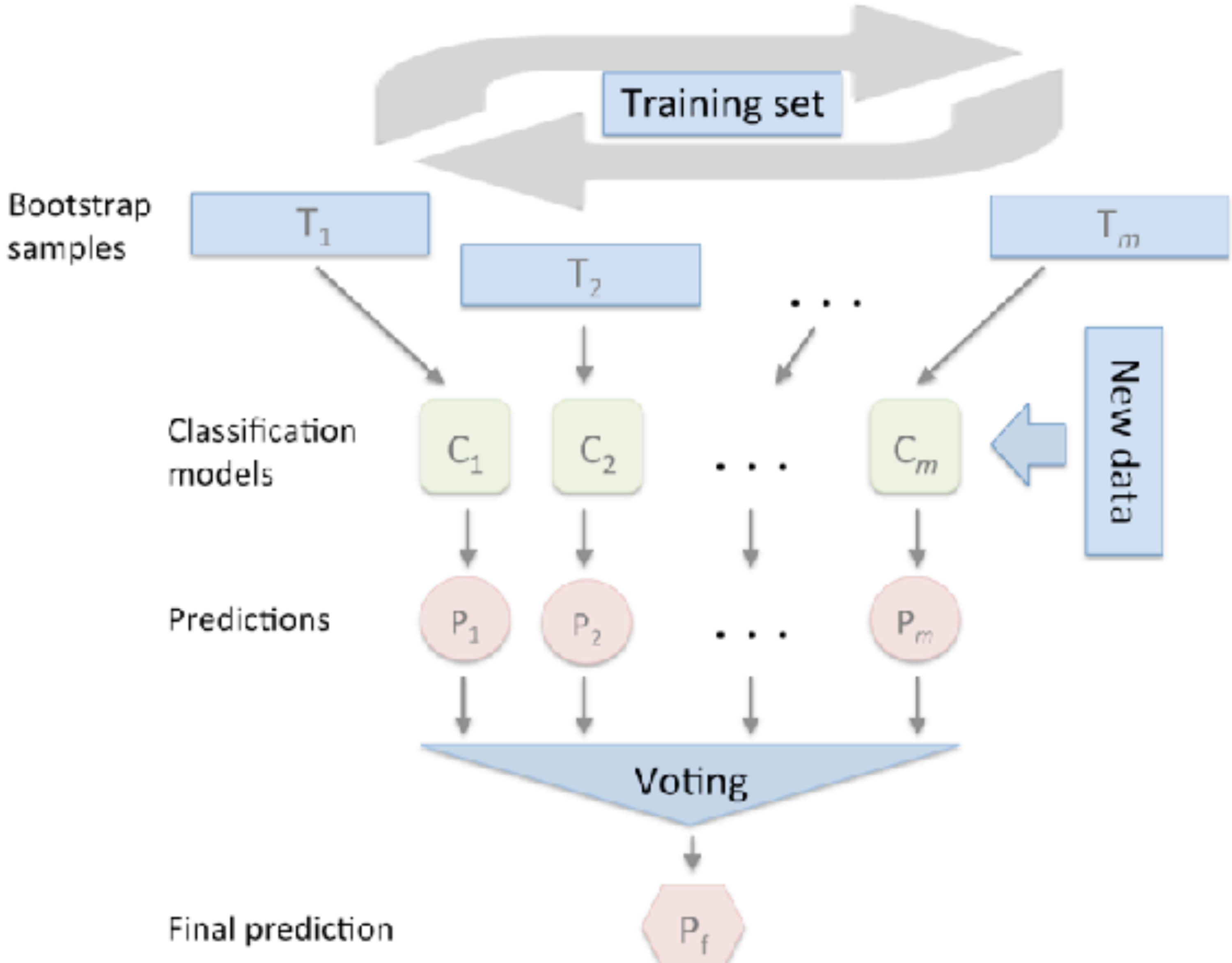
Bagging

- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifiers from subsamples of data
  - could be smart or just completely random (uniform)
  - could sample from instances (rows) or from features (columns)
- Combine the resulting classifiers as before
  - majority vote
  - argmax probability

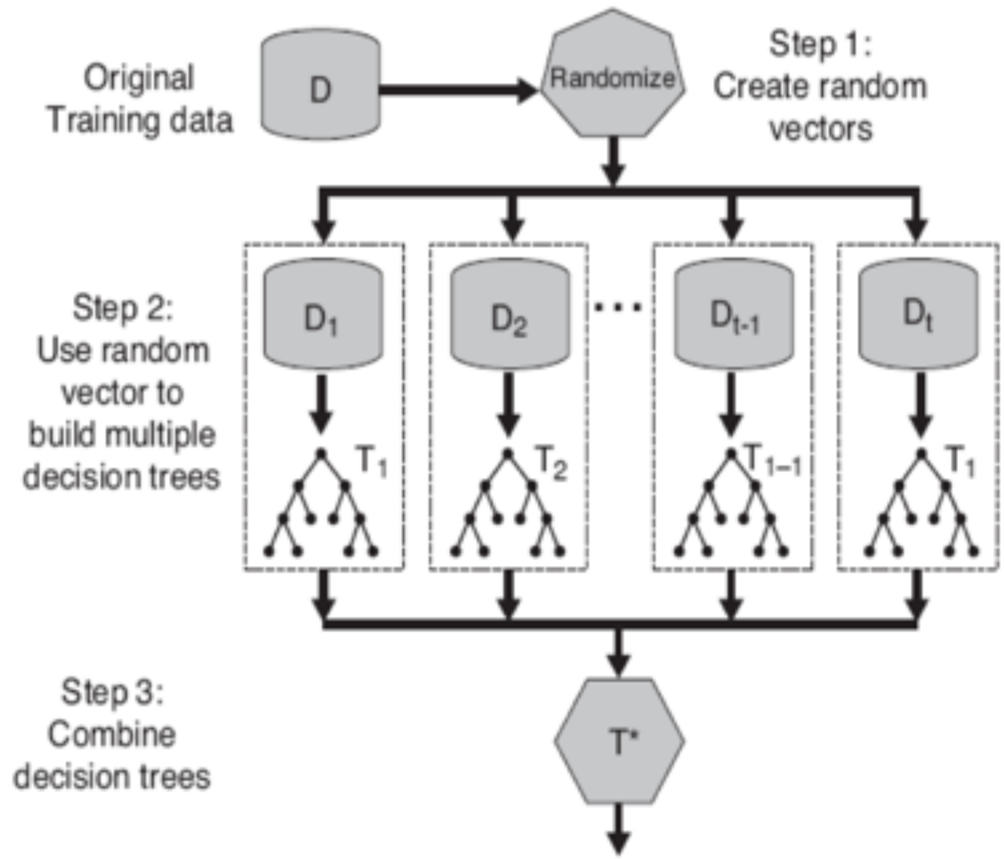
Bagging



Training set
Bootstrap samples
T1
T2
Tm
Classification models
C1
C2
Cm
New data
Predictions
P1
P2
Pm
Voting
Final prediction
Pf

The most famous bagging classifier

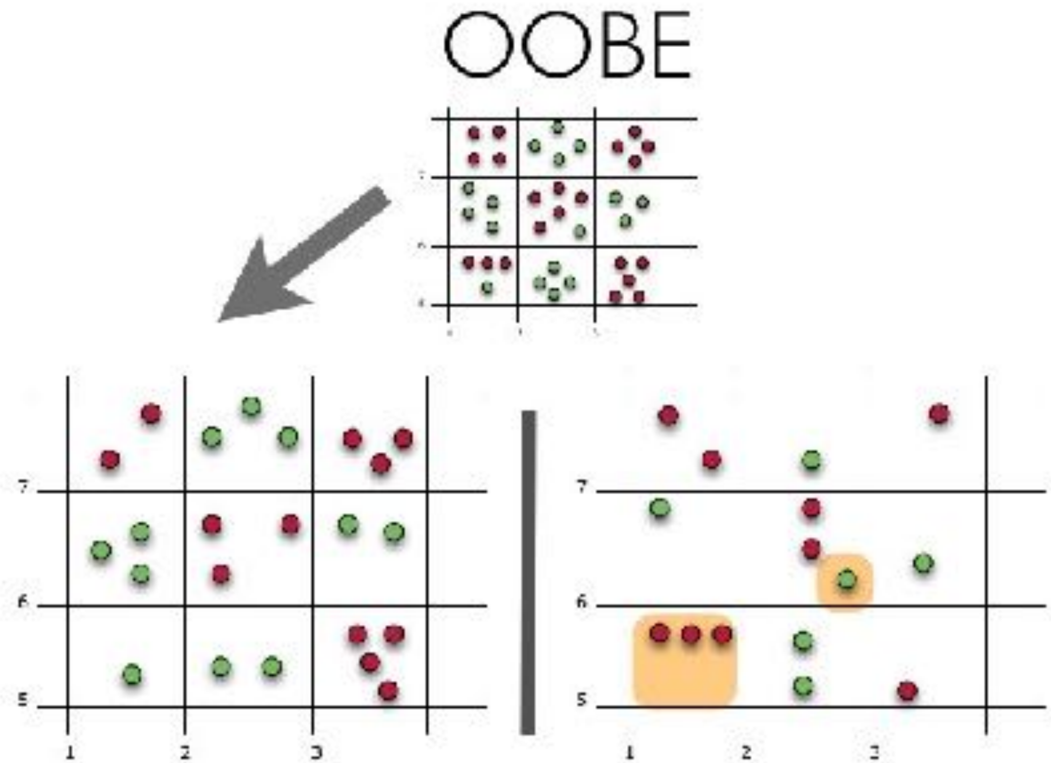
- Random Forests
    - Select random subset of samples
    - Select random subset of the features
    - build a tree
    - build many trees
    - actually a whole forest of trees



Random Forest

- Random Forests
  - Decision trees are built
  - But at each stage, a random subset of the features is selected (random subspace)
    - if "*f*" features, look at "*np.sqrt(f)*" features at each iteration
  - Generalization built in: Out-of-bag
  - Variable importance:
    - random feature permutation
    - look at out-of-bag samples
    - randomly permute the values of n^{th} feature
    - see how performance degrades

Random Forest



- One can use the training data to get an error estimate ("out of bag error" or OOBE)
- Validate each tree on complement of training data

http://www.slideshare.net/0xdata/jan-vitek-distributedrandomforest522013

Features of Random Forests:

- produce high accuracy on many real world datasets
- run efficiently on large databases (each tree is an easy prediction, easily extensible to map reduce)
- can handle thousands of input variables without variable deletion
- give estimates of what variables are important in the classification
- generate an internal unbiased estimate of the generalization error as the forest building progresses
- have effective method for estimating missing data
- have methods for balancing error in class population for unbalanced data sets

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#varimp

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of boosting round
  - Samples with a higher weight are more likely to be chosen

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Example: AdaBoost

- Base classifiers: $C_1, C_2, \ldots, C_T$

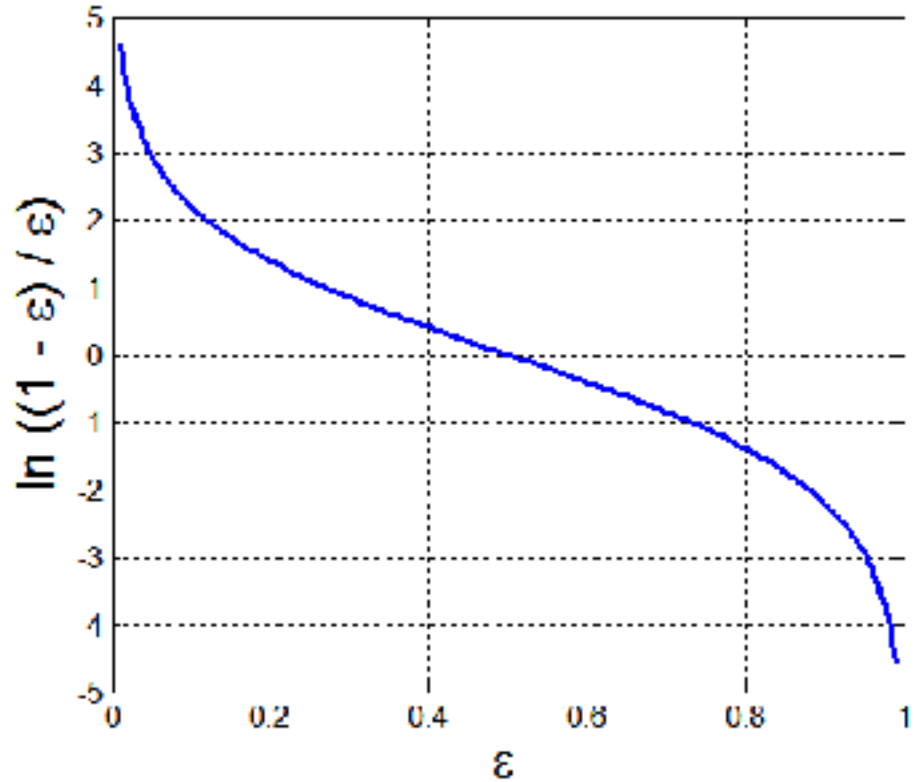
- Weighted Error rate of C_i is:

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

j^{th} instance weight   indicator

- Importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Example: AdaBoost

- Weight update:

Decrease weight

$$w_j \leftarrow w_j \times \begin{cases} 1 & \text{if } C_i(x_j) = y_i \\ (1-\epsilon_i)/\epsilon_i & \text{if } C_i(x_j) \neq y_i \end{cases}$$

Increase weight

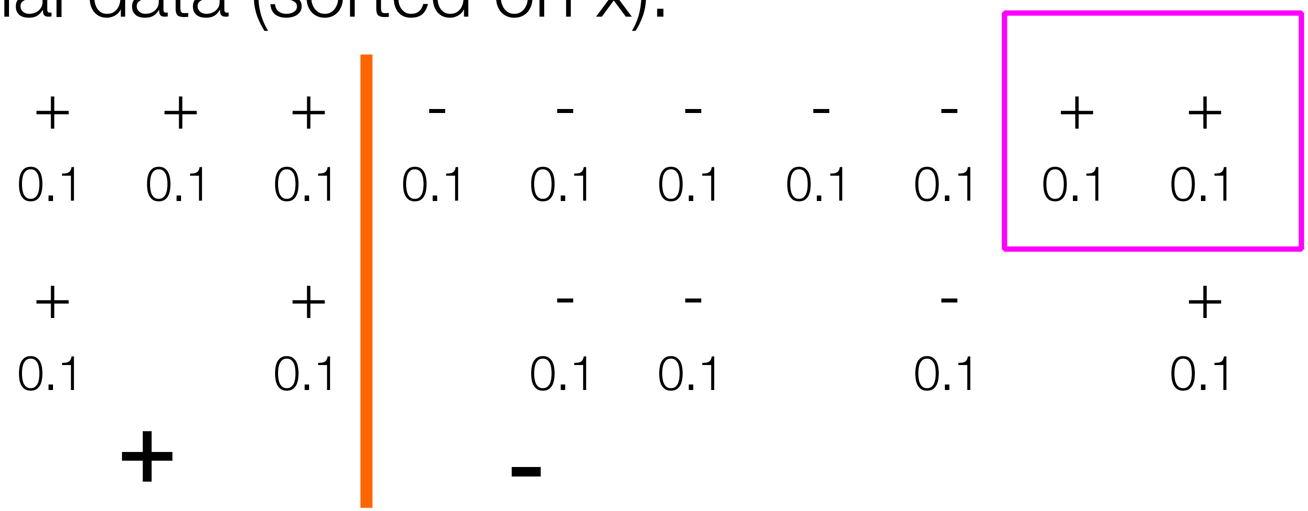
- Rescale weights to sum to one
- Classification:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\big(C_j(x) = y\big)$$

because we take arg max,
the 1/2 constant in α is not needed

Illustrating AdaBoost

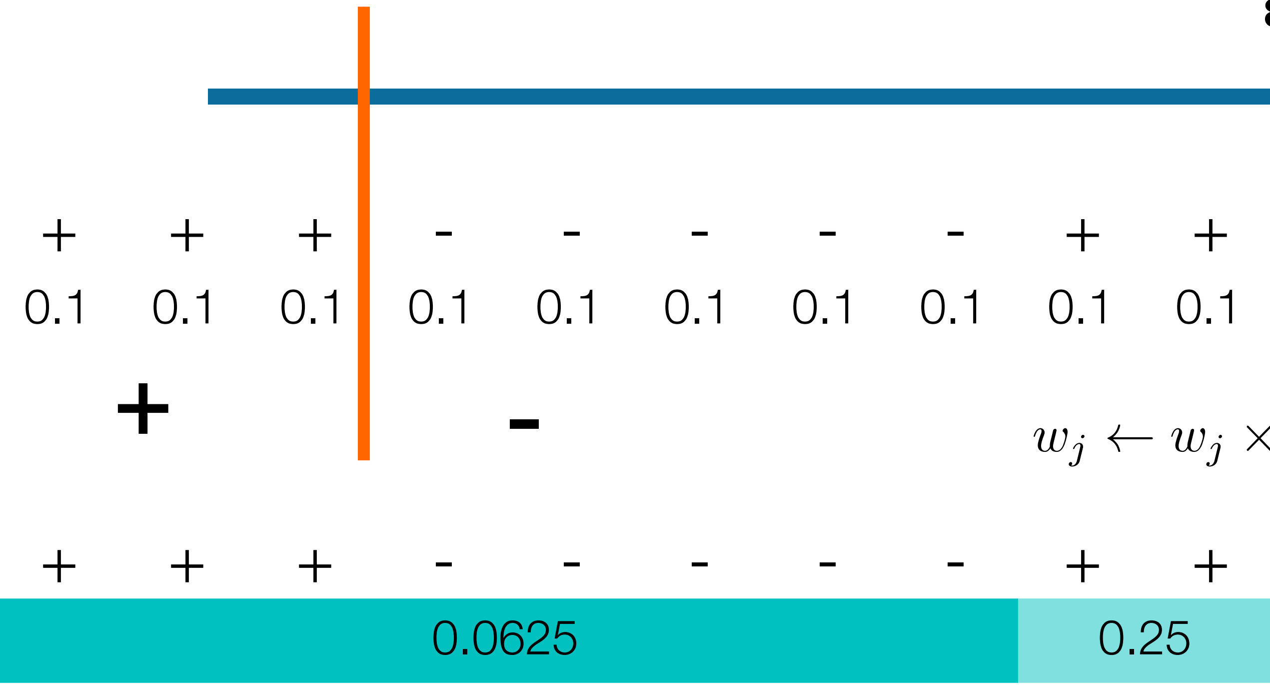
- Original data (sorted on x):



- We have 10 data points, so each data point gets initial weight 1/10.
- Suppose we sample *six* points
- Then train a “decision stump” classifier
- Which makes two errors with weight 0.1

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

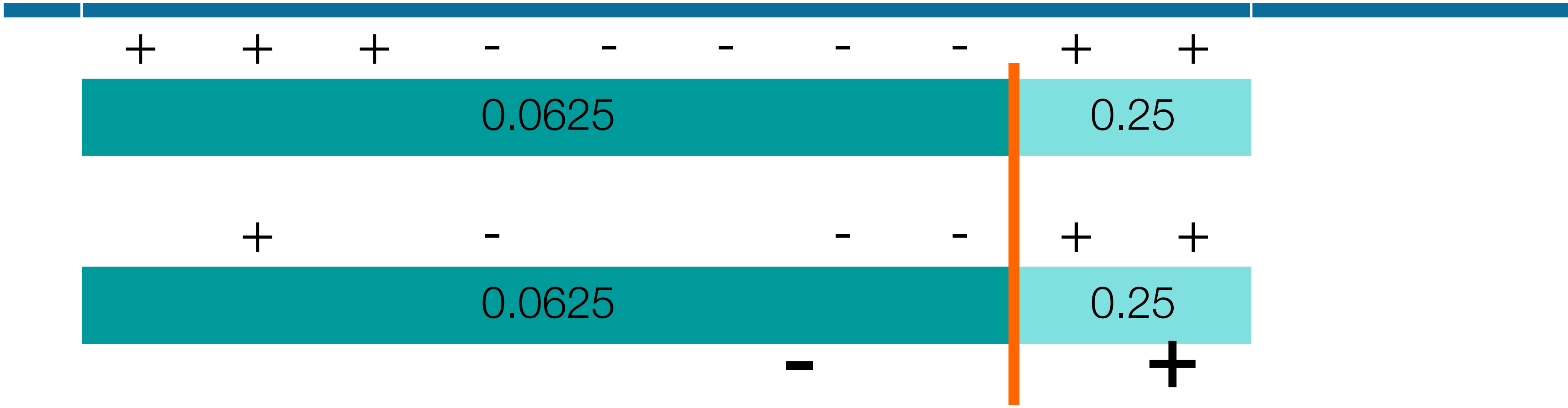
$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



$$w_j \leftarrow w_j \times \begin{cases} 1 & \text{if } C_i(x_j) = y_i \\ (1-\epsilon_i)/\epsilon_i & \text{if } C_i(x_j) \neq y_i \end{cases}$$

- Which makes two errors with weight 0.1
- So ε = 2x0.1=0.2, α = ln[(1-0.2)/0.2] = ln 4 ~ 1.38
- So weights of incorrect answers get multiplied by 4
- Then weights are rescaled to sum to one

Illustrating AdaBoost



- Sample six new samples, train stump
- Resulting in 3 errors with weights 0.0625
- So ε = 3x0.0625=0.1875,
- α = ln (1-0.1875)/0.1875 = ln 4.33 ~ 1.47
- Update weights

$$w_j \leftarrow w_j \times \begin{cases} 1 & \text{if } C_i(x_j) = y_i \\ (1-\epsilon_i)/\epsilon_i & \text{if } C_i(x_j) \neq y_i \end{cases}$$

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Illustrating AdaBoost

- New weights are:

+ + + - - - - - + +

0.17 0.039 0.15

- Chosen samples, round three

A + + + B - C + + D

0.17 0.039 0.15

- **Self test**: where is my new decision stump?

Self test: Illustrating AdaBoost

- New weights are:

$+ \quad + \quad + \quad - \quad - \quad - \quad - \quad - \quad + \quad +$

0.17 0.039 0.15

- Chosen samples, round three

$+ \quad + \quad + \qquad\qquad\qquad - \qquad + \quad +$

0.17 0.039 0.15

$+ \quad | \quad -$

  - So $\varepsilon = 5 \times 0.039 = 0.195$,
  - $\alpha = \ln (1-0.195)/0.195 = \ln 4.13 \sim 1.42$

Illustrating AdaBoost

- Combined classifiers:
- C_1, α=1.38
- C_2, α=1.47
- C_3, α=1.42

C_1	+	+	+	-	-	-	-	-	-	-
C_2	-	-	-	-	-	-	-	-	+	+
C_3	+	+	+	+	+	+	+	+	+	+

- C*

C*	+	+	+	-	-	-	-	-	+	+

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

A final thought on boosting and bagging

- Boosting is what won the Netflix prize
- But was never implemented
  - *"…additional accuracy gains that we measured did not seem to justify the engineering effort to bring them into a production environment."*

Watch videos before class!

- Next time is an in-class-assignment
  - cross validation!

synchronous