Lecture Notes for
Machine Learning in Python

Professor Eric Larson

Grid Searches and Ensemble Methods

Class Logistics and Agenda

- Logistics
  - **Next time**: project work day
    - Lab due at end of week
  - **Next Next Time**: altering schedule, no class
  - **Next Week**: Deep Learning History
- Agenda:
  - Grid Searching
  - Ensemble methods

Grid Searching

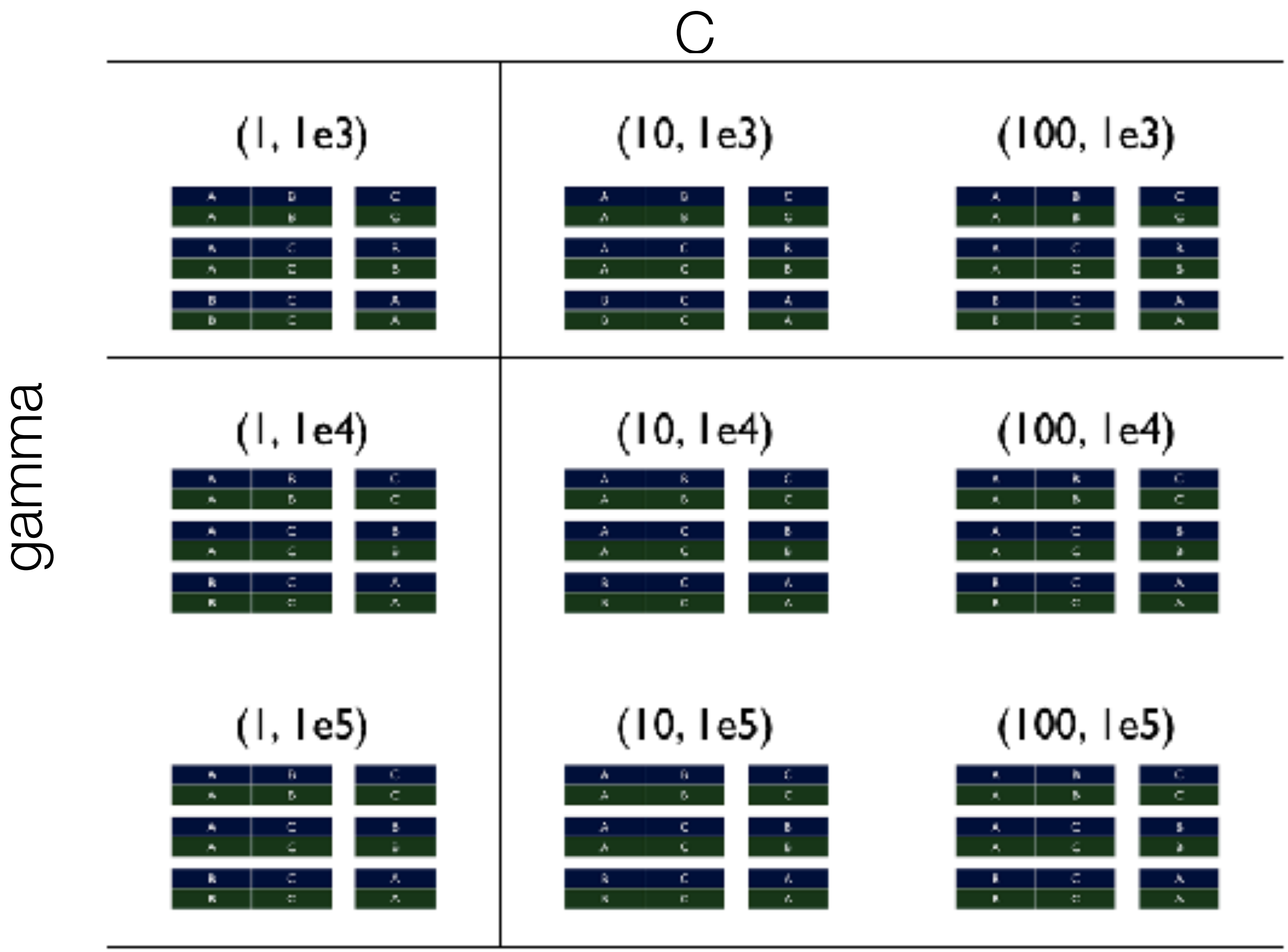
- Trying to find the best parameters
  - `SVM: C=[1, 10, 100] gamma=[1e3,1e4,1e5]`

C

	(1, 1e3)	(10, 1e3)	(100, 1e3)
gamma	(1, 1e4)	(10, 1e4)	(100, 1e4)
	(1, 1e5)	(10, 1e5)	(100, 1e5)

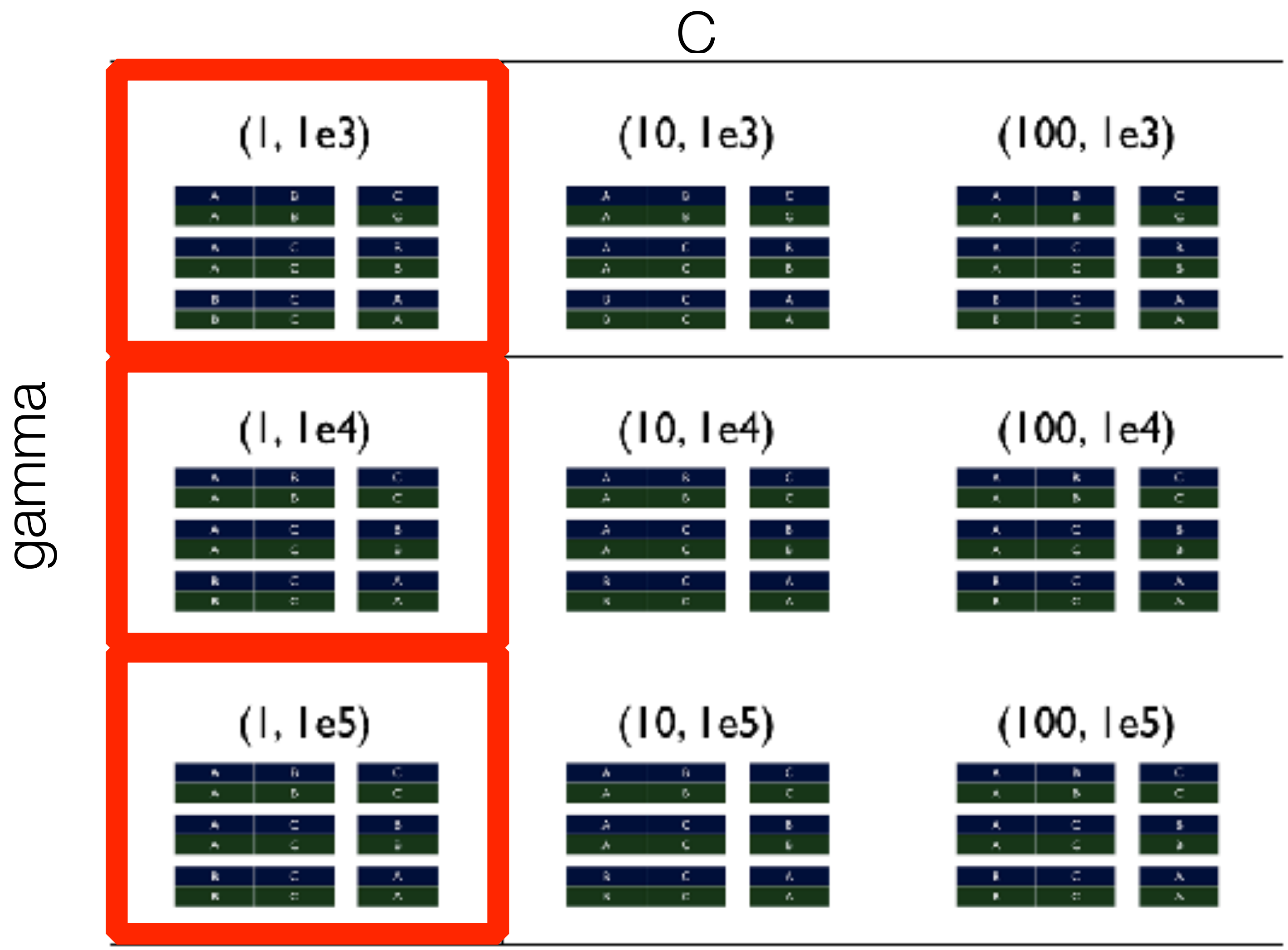
Grid Searching

- For each value, want to run cross validation…



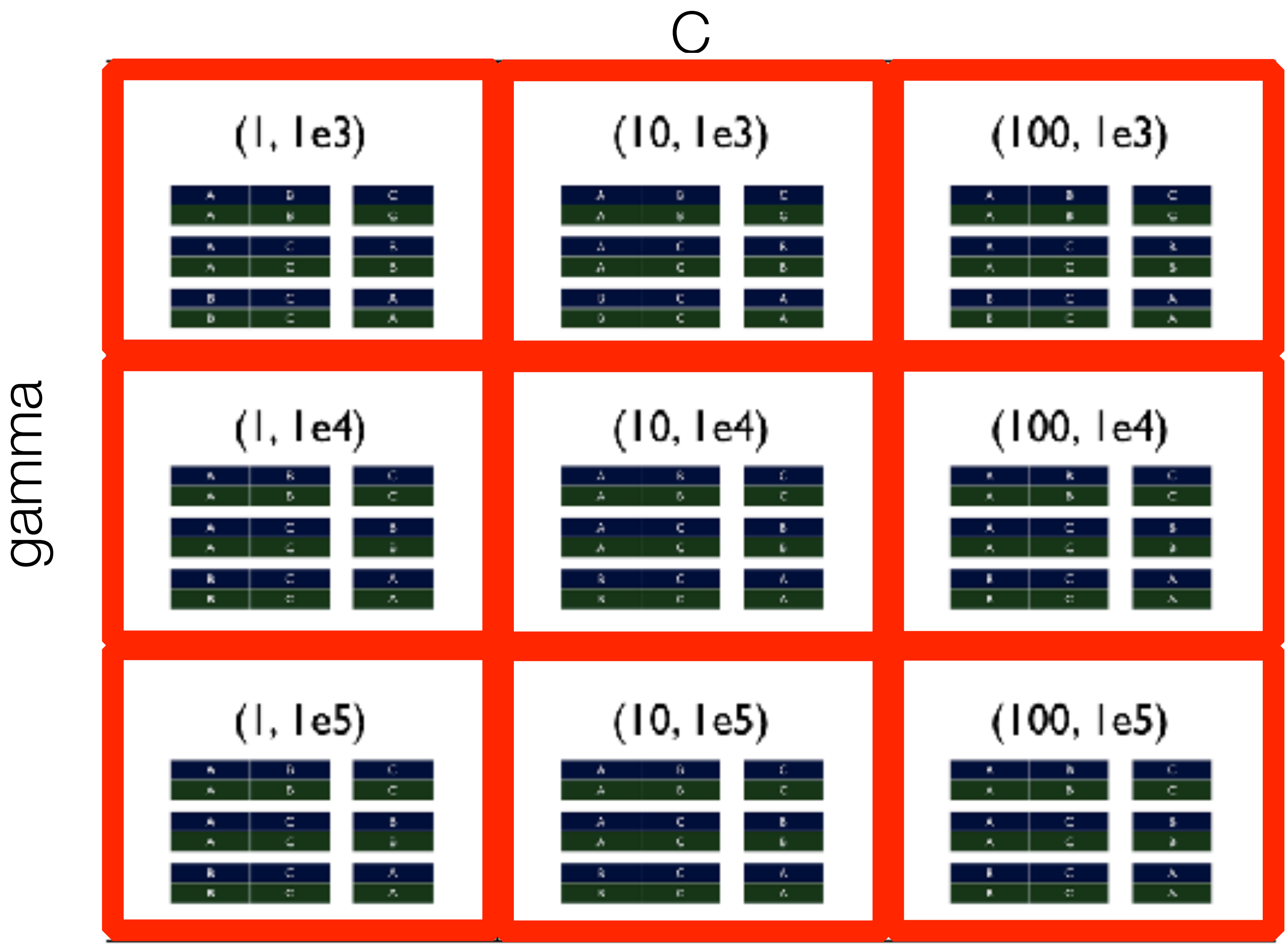
Grid Searching

- Could perform iteratively

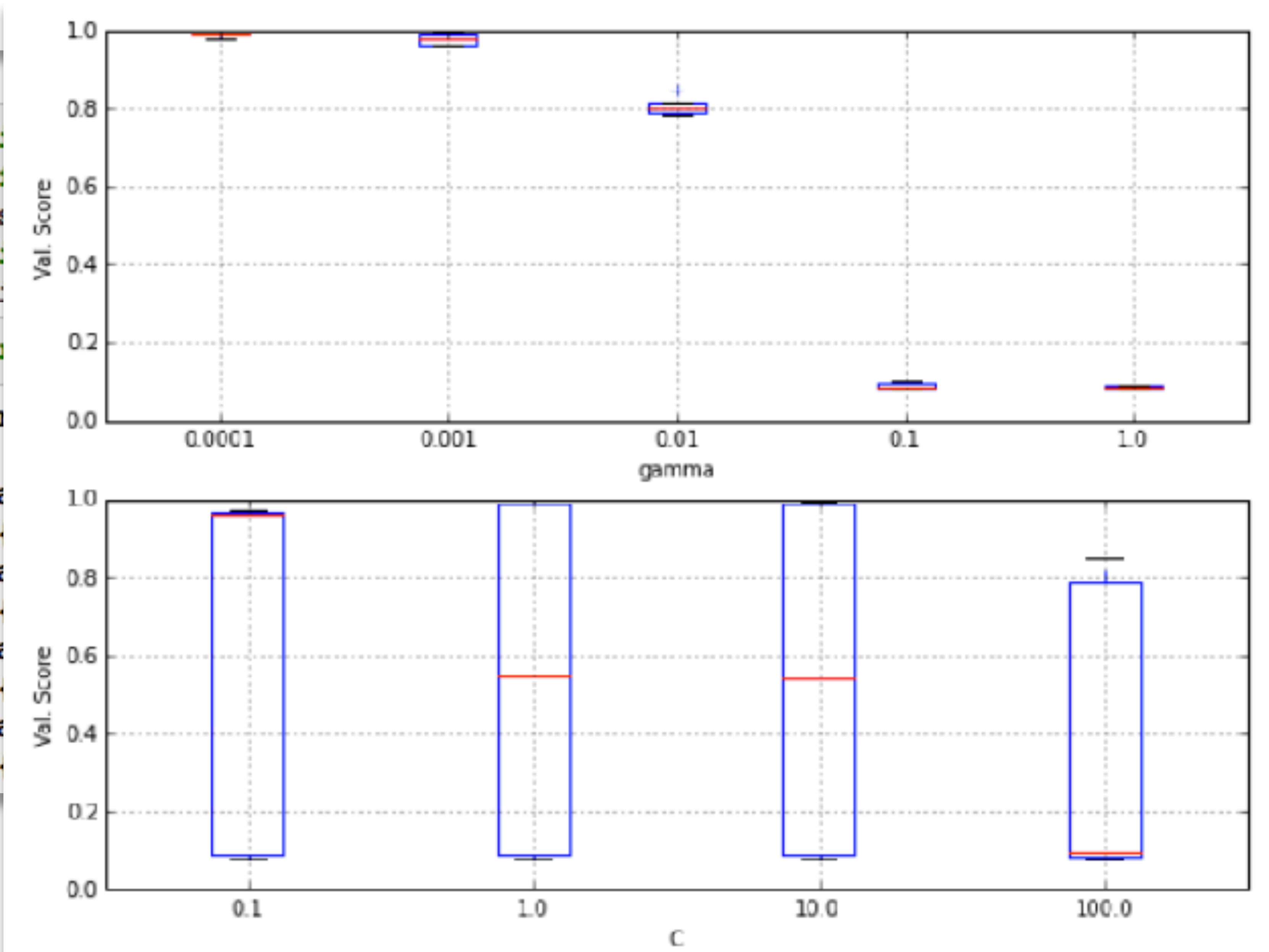


Grid Searching

- or at random…



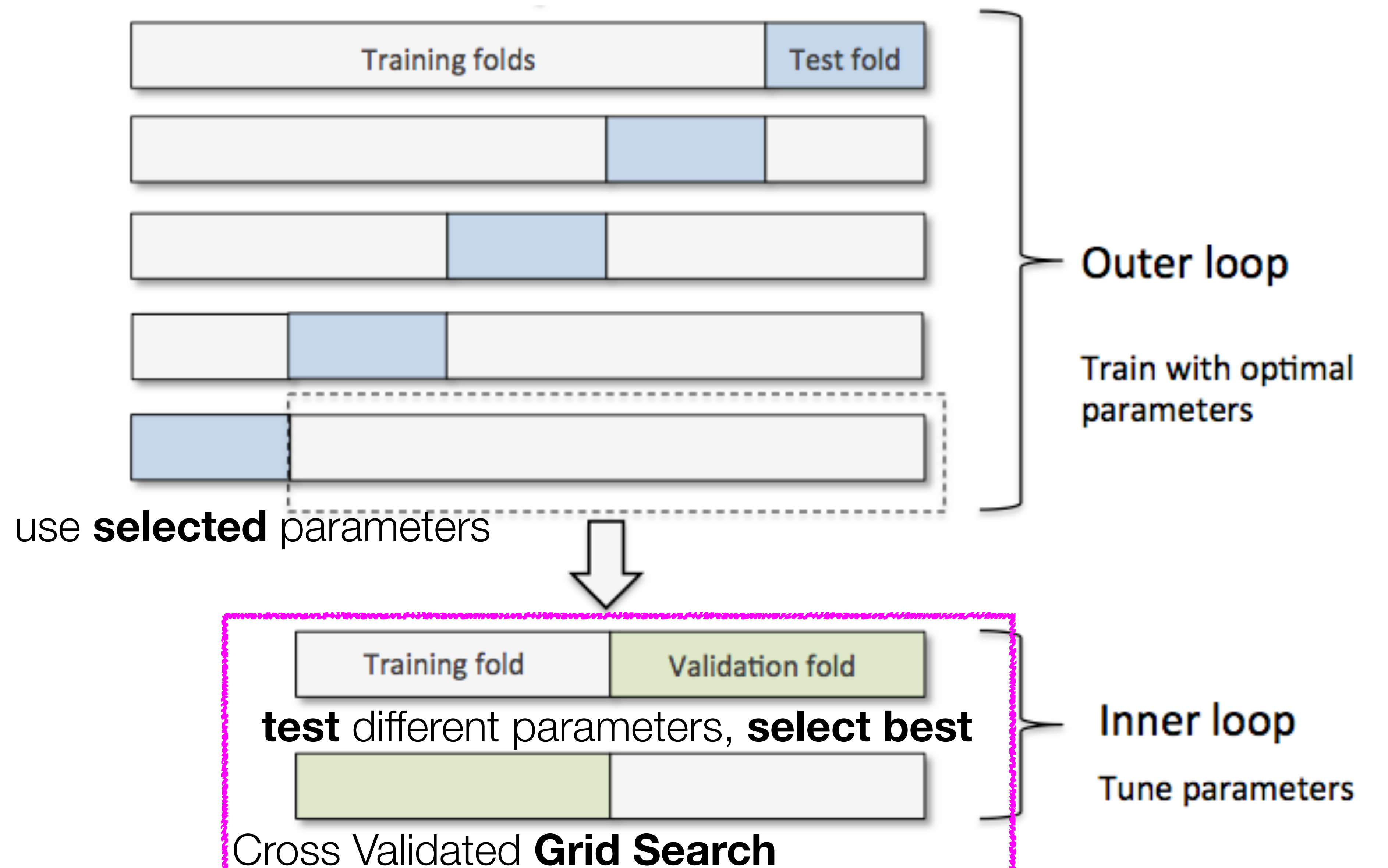
```
print(search.report())
search.boxplot_parameters(display_train=False)
```



The Problem

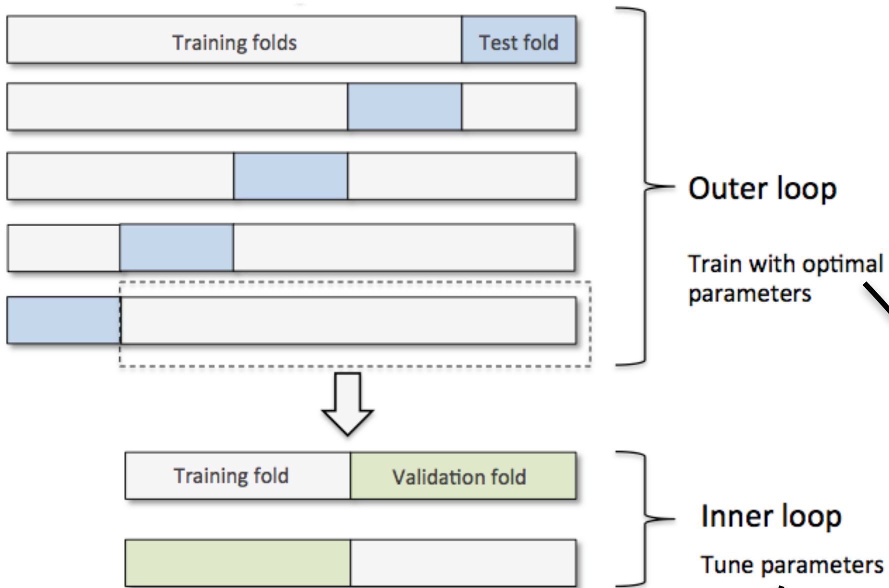
- Using the grid search parameters and testing on the same set…
  - the **performance on the dataset** will now be **biased**
  - cannot determine the **expected performance** on new data

Solution: Nested Cross Validation



https://github.com/rasbt/python-machine-learning-book/

Nested Cross Validation: Hyper-parameters




```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)

# Note: Optionally, you could use cv=2
# in the GridSearchCV above to produce
# the 5 x 2 nested CV that is shown in the figure.

scores = cross_val_score(gs, X_train, y_train, scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

https://github.com/rasbt/python-machine-learning-book/

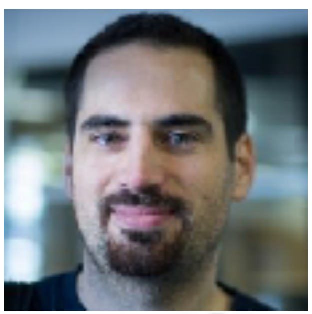
Self Test

- **What is the end goal of nested cross-validation?**
  - A. To determine hyper parameters
  - B. To estimate generalization performance
  - C. To estimate generalization performance when performing hyper parameter tuning
  - D. To estimate the variation in tuned hyper parameters

Demo

Grid Searching and Nested Cross Validation

Other tutorials:



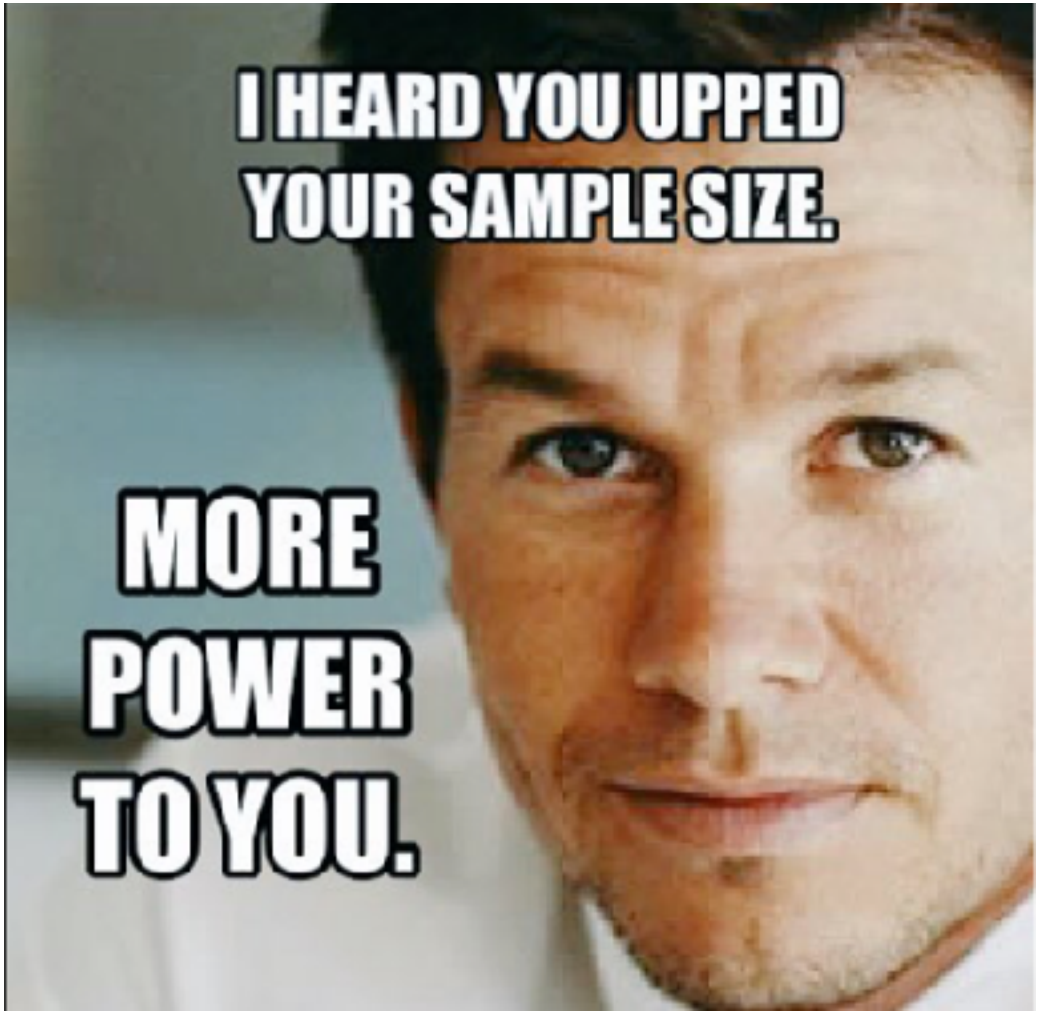
Olivier Grisel's Tutorial:

https://www.youtube.com/watch?v=iFkRt3BCctg

~3 hours

https://github.com/ogrisel/parallel_ml_tutorial/blob/master/rendered_notebooks/06%20-%20Distributed%20Model%20Selection%20and%20Assessment.ipynb

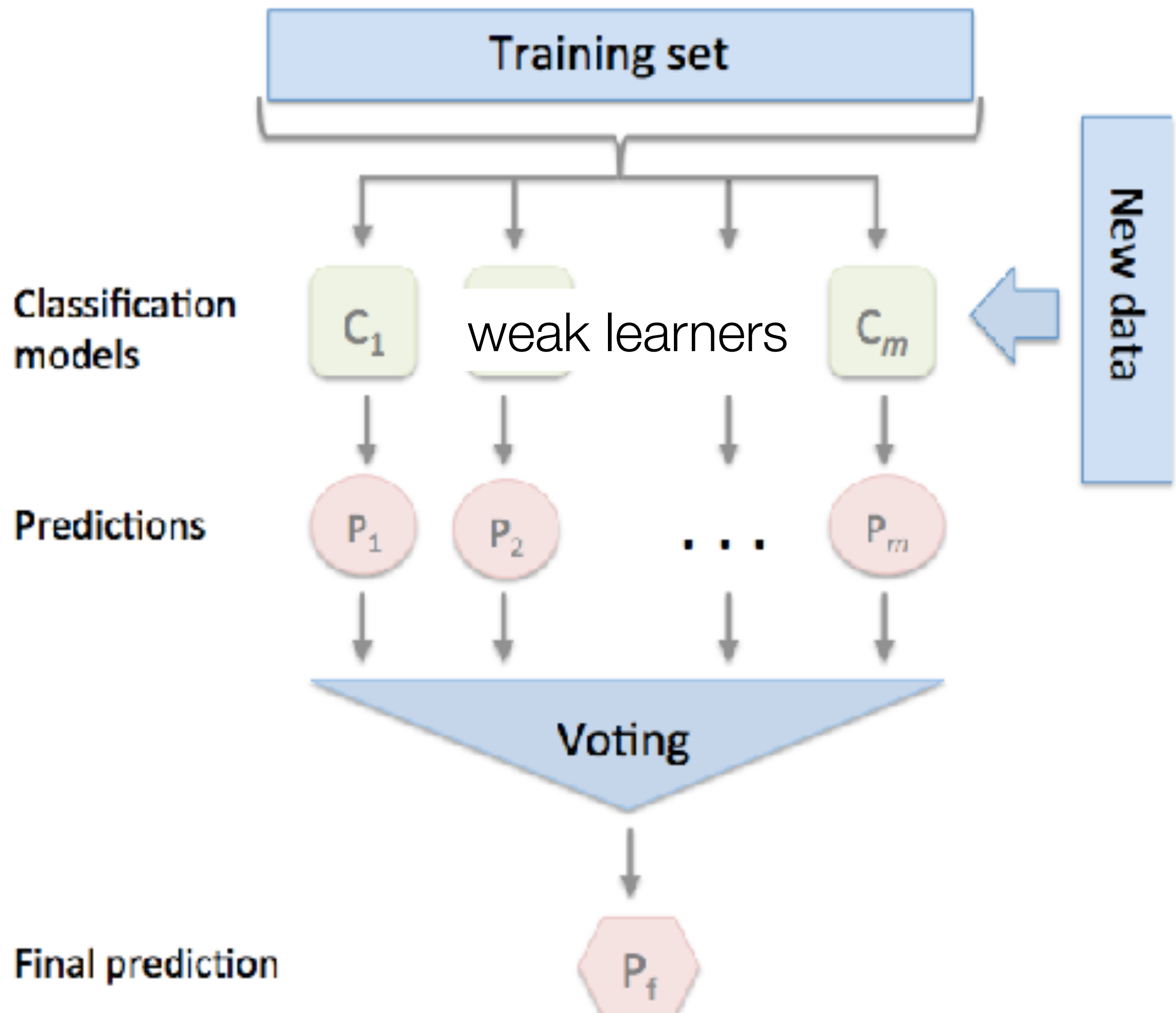
Classification: Ensemble Methods



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



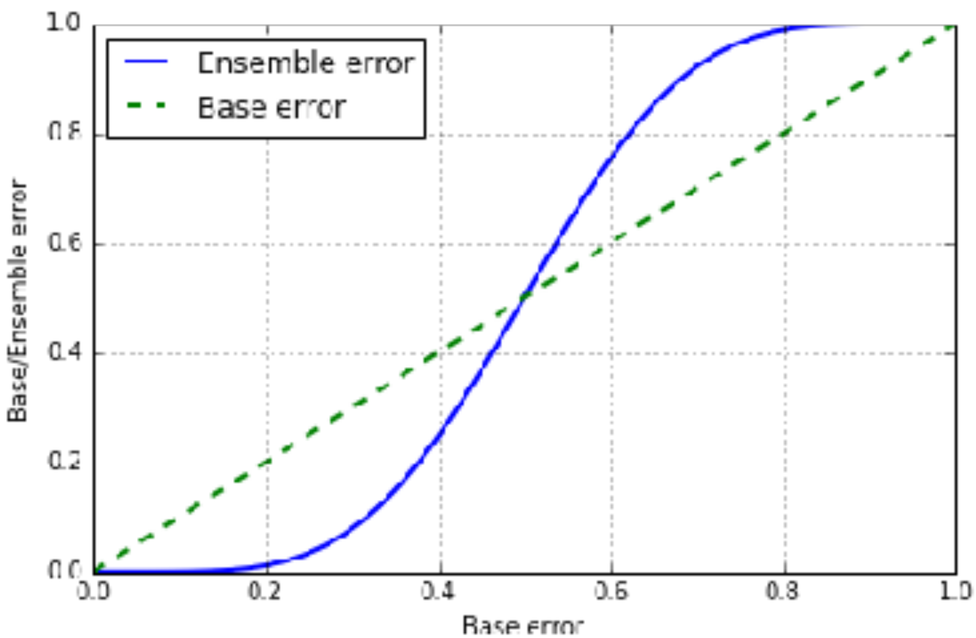
Weak and Strong

- **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:

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

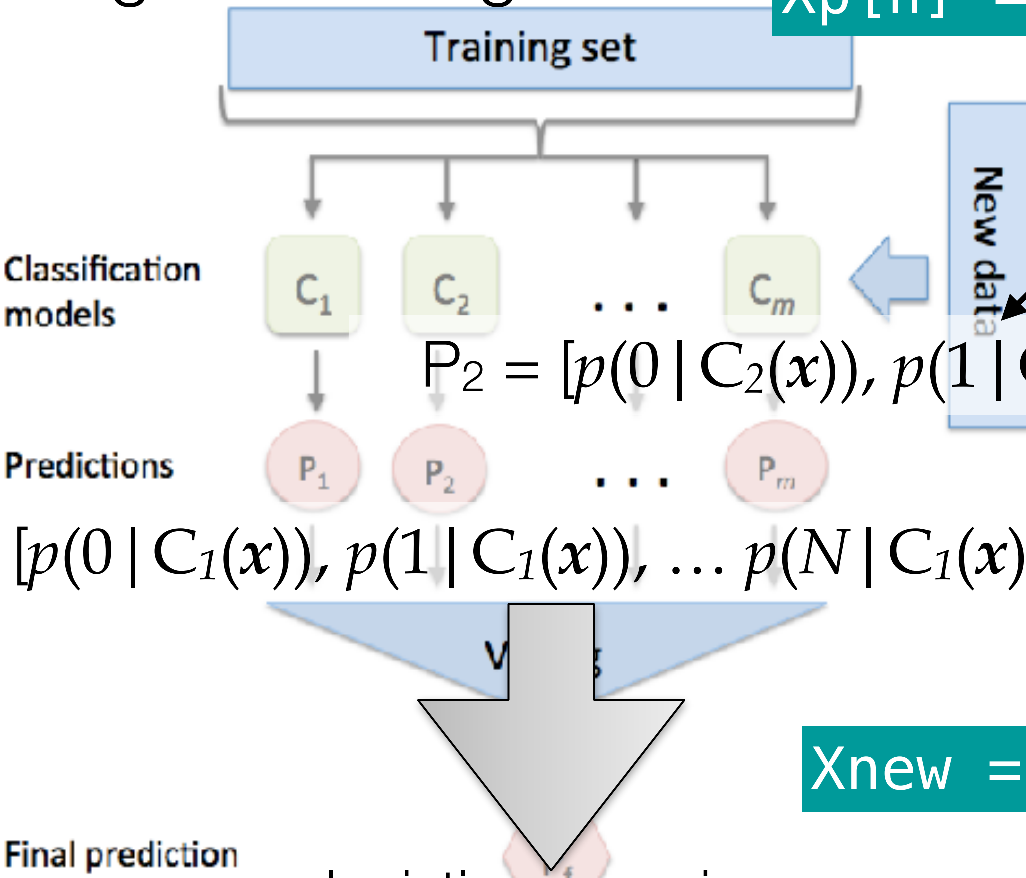
trust in classifier

predict_proba i

Examples of Ensemble Methods

- Stacking/Cascading

```
Xp[n] = predict_proba(X)
```



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

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

```
Xnew = np.hstack(Xp)
```

logistic regression

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

```
LogisticRegression().fit(Xnew,y)
```

Examples of Ensemble Methods

- Training set sampling methods:
    - Bagging
    - Boosting

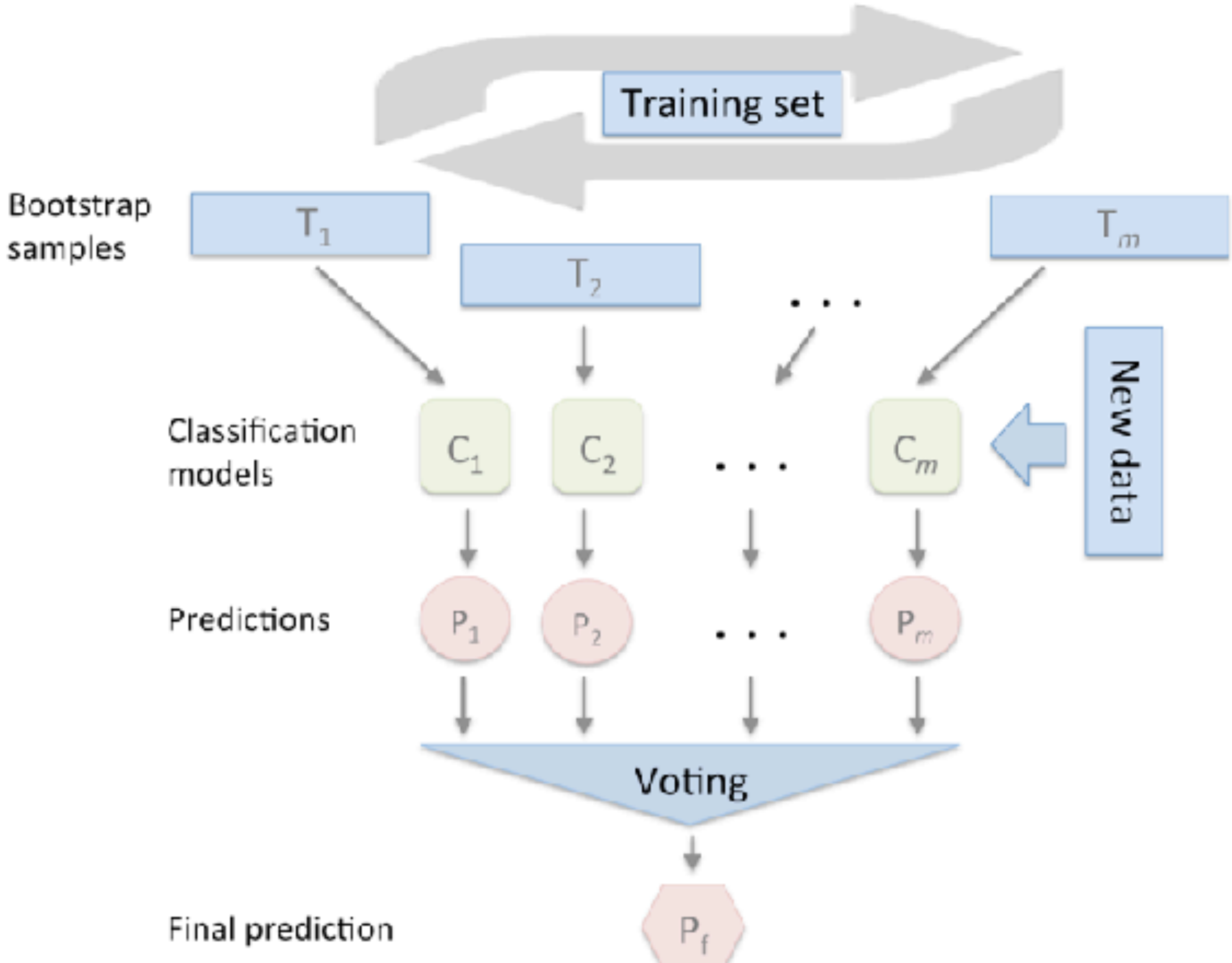
Bagging

- 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

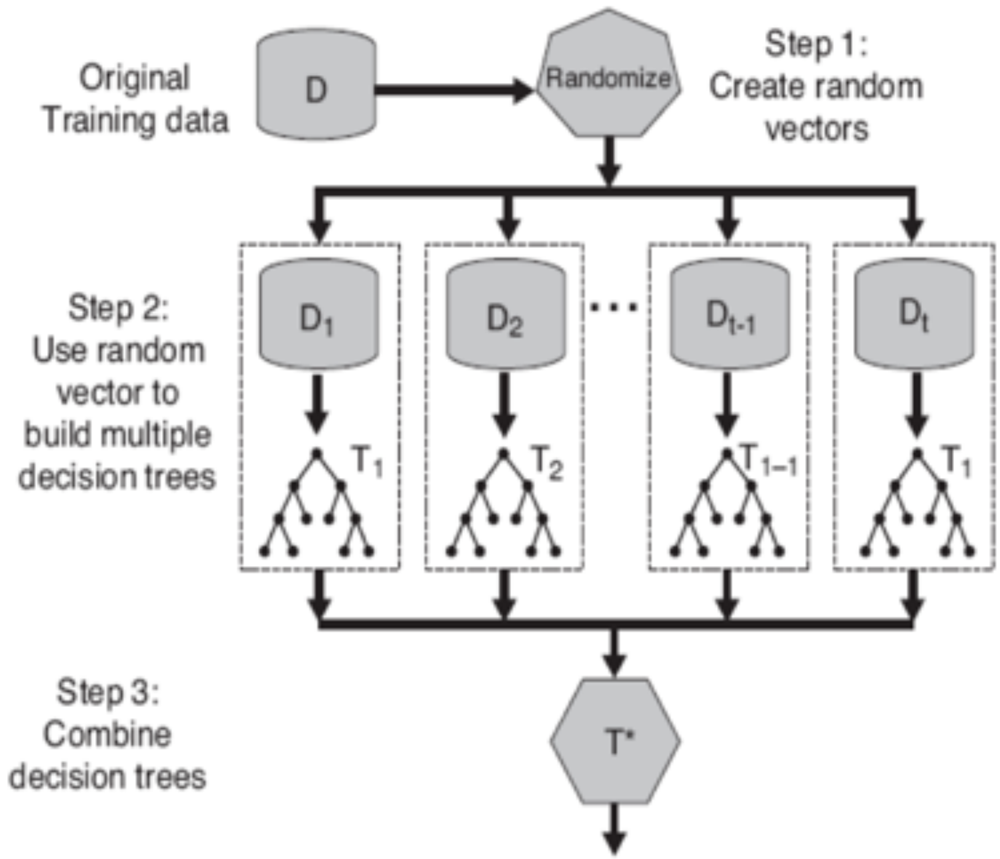
Bagging



Training set
Bootstrap samples
T_1
T_2
T_m
Classification models
C_1
C_2
C_m
New data
Predictions
P_1
P_2
P_m
Voting
Final prediction
P_f

The most famous bagging classifier

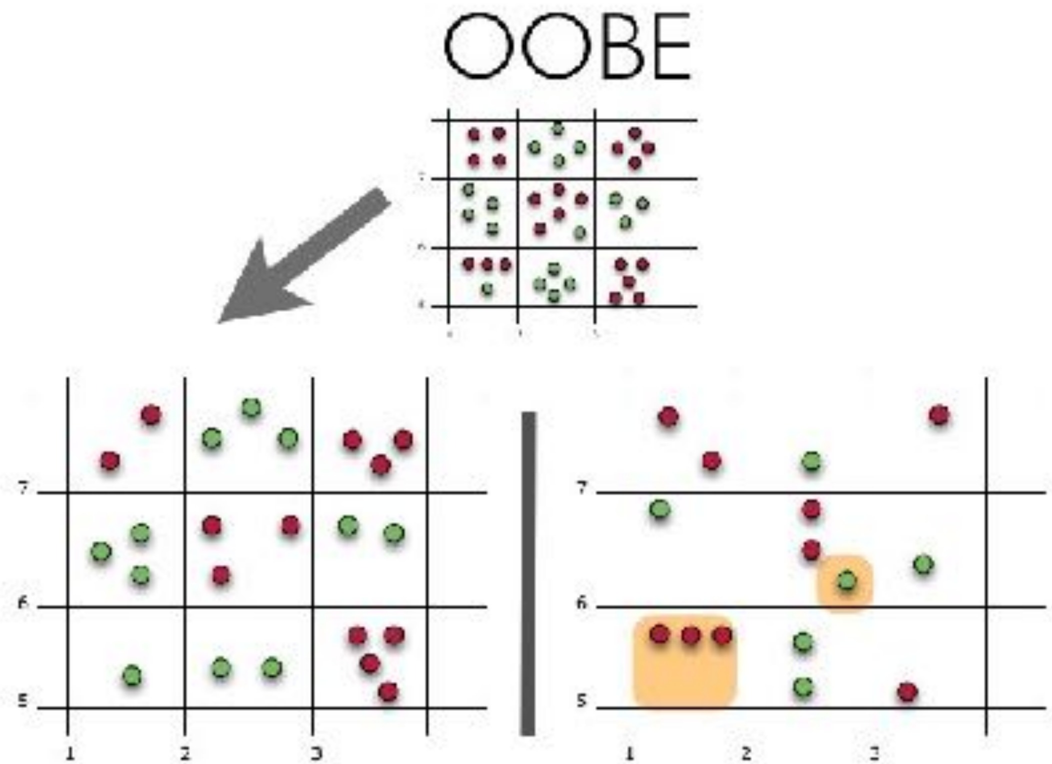
- Random Forests
  - Select random subset of samples
  - Select random subset of the features
  - build a tree
  - build many trees
  - actually a whole forest of trees



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

Characteristics 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

Overview: AdaBoost

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

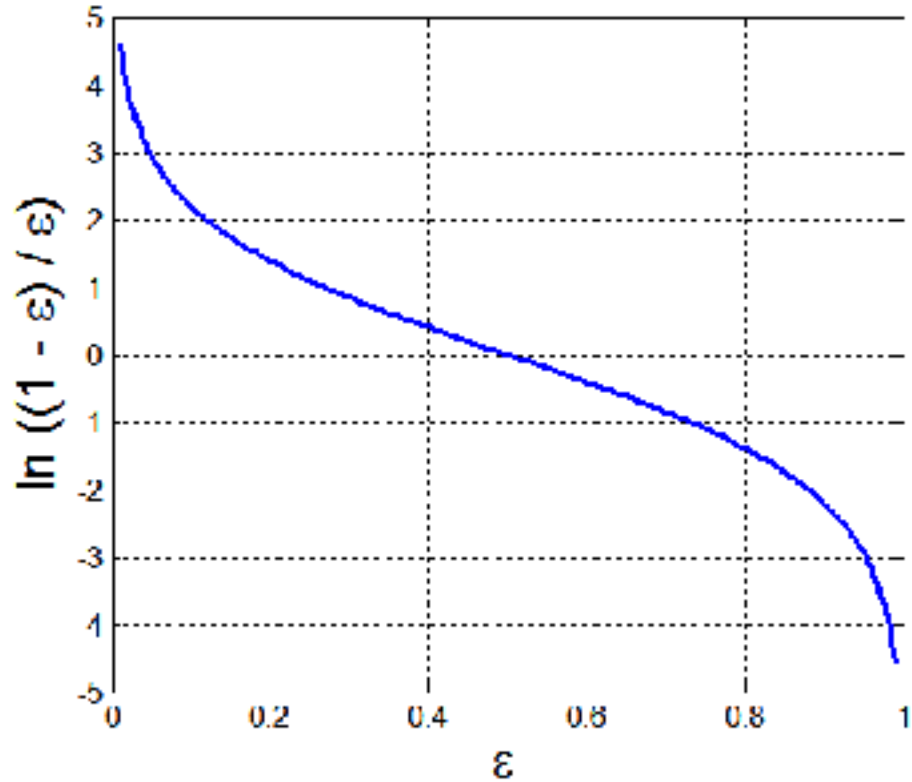
- Weighted Error rate of C_i is:

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

j^{th} instance weight indicator

- Importance of a classifier:

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



Overview: 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:

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

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

Illustrating AdaBoost

- Original data (sorted on x):

+	+	+	-	-	-	-	-	+	+
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
+		+		-	-		-		+
0.1		0.1		0.1	0.1		0.1		0.1
	+			-					

- 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

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

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

+	+	+	-	-	-	-	-	+	+
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

+ **-**

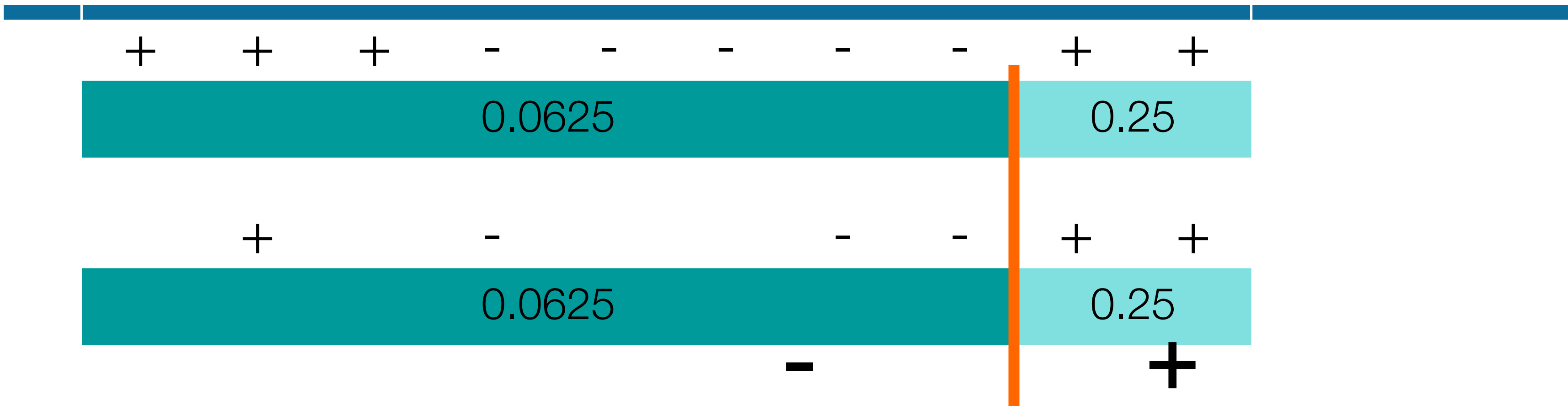
$$w_j \leftarrow w_j \times \begin{cases} 1 & \text{if } C_i(x_j) = y_i \\ (1-\epsilon_i)/\epsilon_i & \text{if } C_i(x_j) \neq y_i \end{cases}$$

+	+	+	-	-	-	-	-	+	+

0.0625 0.25

- 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

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

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right) \qquad \alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Illustrating AdaBoost

- New weights are:

+ + + - - - - - + +

0.17 0.039 0.15

- Chosen samples, round three

$\boxed{A}$ + + + $\boxed{B}$ - $\boxed{C}$ + + $\boxed{D}$

0.17 0.039 0.15

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

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

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

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

Illustrating AdaBoost

- New weights are:

+ + + - - - - - + +

0.17 0.039 0.15

- Chosen samples, round three

+ + + - + +

0.17 0.039 0.15

+ **-**

- So ε = 5x0.039=0.195,
- α = ln (1-0.195)/0.195 = ln 4.13 ~ 1.42

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right) \qquad \alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right) \qquad w_j \leftarrow w_j \times \begin{cases} 1 \\ (1-\epsilon_i)/\epsilon_i \end{cases}$$

Illustrating AdaBoost

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

C_1, α=1.38	+	+	+	-	-	-	-	-	-	-
C_2, α=1.47	-	-	-	-	-	-	-	-	+	+
C_3, α=1.42	+	+	+	+	+	+	+	+	+	+
C*	+	+	+	-	-	-	-	-	+	+

- C*

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

Gradient Boosting from 10,000 feet

- **Adaboost**: weight different classifiers by their performance
- **Gradient Boosting Regression**: use ensemble to fit errors of your classifier,
  - weak learner, `L`
  - current model, $F_i(X)$
  - $F_{i+1}(X) = F_i(X) +$
    `L.fit( X, y−` F_i`(X).predict(X) )`
- For **classification**, same procedure but use:
  - `y_one_hot −` F_i`(X).predict_proba(X)`

want more? http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf

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."*

- But… gradient boosting has become quite accessible, sklearn and XGBoost
  - Keep these in mind for exceptional credit!!

Next Time

- Next time, Thursday: **No Class**
- Next Next Time, Tuesday: **Also No Class**