Ensemble Learning

Dr. Ha Nguyen
Ambyint - Canada

Week 4 - Agenda

1. Introduction
2. Basic Ensemble Learning
3. Advanced Ensemble Learning
4. Algorithms based on Ensemble Learning

Netflix Prize

From Wikipedia, the free encyclopedia

The **Netflix Prize** was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users or the films being identified except by numbers assigned for the contest.

The competition was held by Netflix, an online DVD-rental and video streaming service, and was open to anyone who is neither connected with Netflix (current and former employees, agents, close relatives of Netflix employees, etc.) nor a resident of certain blocked countries (such as Cuba or North Korea).[1] On September 21, 2009, the grand prize of US$1,000,000 was given to the BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%.[2]

Contents [hide]

Recommender systems

Concepts
Collective intelligence · Relevance · Star ratings · Long tail

Methods and challenges
Cold start · Collaborative filtering · Dimensionality reduction · Implicit data collection · Item-item collaborative filtering · Matrix factorization · Preference elicitation · Similarity search · Social loafing

Implementations
Collaborative search engine · Content discovery platform · Decision support system · Music Genome Project · Product finder

Research
GroupLens Research · MovieLens · **Netflix Prize**

V · T · E

Problem and data sets [edit]

Netflix provided a *training* data set of 100,480,507 ratings that 480,189 users gave to 17,770 movies. Each training rating is a quadruplet of the form `<user, movie, date of grade, grade>`. The user and movie fields are integer IDs, while grades are from 1 to 5 (integral) stars.[3]

Ensemble methods

Machine learning competition with a $1 million prize

Leaderboard

Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	The Ensemble	0.8553	10.10	2009-07-26 18:38:22
2	BellKor's Pragmatic Chaos	0.8554	10.09	2009-07-26 18:18:28
	Grand Prize - RMSE <= 0.8563			
3	Grand Prize Team	0.8571	9.91	2009-07-24 13:07:49
4	Opera Solutions and Vandelay United	0.8573	9.89	2009-07-25 20:05:52
5	Vandelay Industries !	0.8579	9.83	2009-07-26 02:49:53
6	PragmaticTheory	0.8582	9.80	2009-07-12 15:09:53
7	BellKor in BigChaos	0.8590	9.71	2009-07-26 12:57:25
8	Dace	0.8603	9.58	2009-07-24 17:18:43
9	Opera Solutions	0.8611	9.49	2009-07-26 18:02:08
10	BellKor	0.8612	9.48	2009-07-26 17:19:11
11	BigChaos	0.8613	9.47	2009-06-23 23:06:52
12	Feeds2	0.8613	9.47	2009-07-24 20:06:46
	Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos			
13	xiangliang	0.8633	9.26	2009-07-21 02:04:40
14	Gravity	0.8634	9.25	2009-07-26 15:58:34
15	Ces	0.8642	9.17	2009-07-25 17:42:38
16	Invisible Ideas	0.8644	9.14	2009-07-20 03:26:12
17	Just a guy in a garage	0.8650	9.08	2009-07-22 14:10:42
18	Craig Carmichael	0.8656	9.02	2009-07-25 16:00:54
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	acmehill	0.8659	8.99	2009-04-16 06:29:35
	Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell			
	Cinematch score on quiz subset - RMSE = 0.9514			

	users					
movies	1		?	3	5	?
	?	1				2
		4		4	5	?

Basic Ensemble Learning

1. Max Voting
2. Averaging
3. Weighted Averaging

Basic Ensemble Learning

1. Max Voting

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x train,y train)
model2.fit(x train,y train)
model3.fit(x_train,y_train)

pred1=model1.predict(x test)
pred2=model2.predict(x test)
pred3=model3.predict(x_test)

final pred = np.array([])
for i in range(0,len(x test)):
    final_pred = np.append(final_pred, mode([pred1[i], pred2[i], pred3[i]]))
```

Or using `from sklearn.ensemble import VotingClassifier`

Basic Ensemble Learning

2. Averaging

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x train,y train)
model2.fit(x train,y train)
model3.fit(x_train,y_train)

pred1=model1.predict proba(x test)
pred2=model2.predict proba(x test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1+pred2+pred3)/3
```

Basic Ensemble Learning

3. Weighted Average

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x train,y train)
model2.fit(x train,y train)
model3.fit(x_train,y_train)

pred1=model1.predict proba(x test)
pred2=model2.predict proba(x test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```

Ensemble methods: Netflix

Clear demonstration of the power of ensemble methods

Original progress prize winner (BellKor) was ensemble of 107 models!

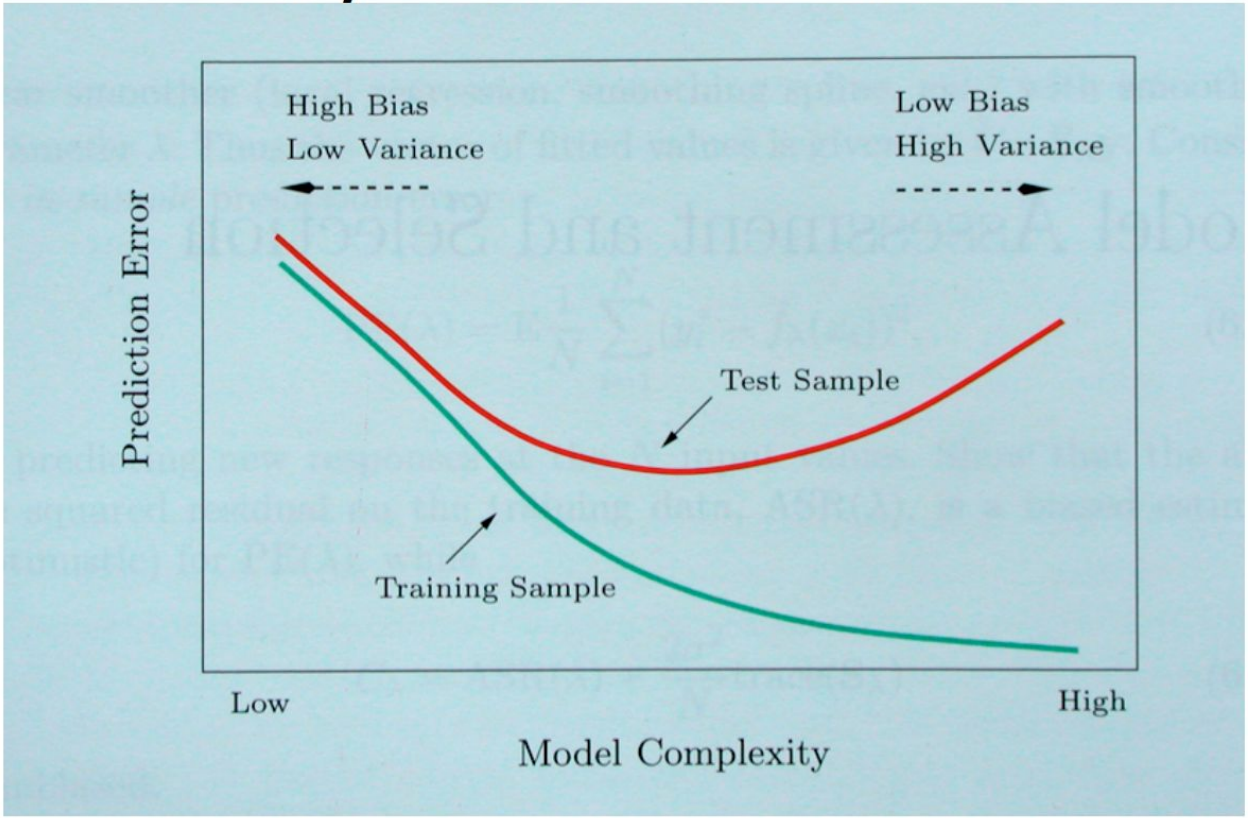
> *"Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a simple technique."*

> *"We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different complementing aspects of the data. Experience shows that this is very different than optimizing the accuracy of each individual predictor."*

Ensemble Learning

- IDEA:
  - do not learn a *single* classifier but learn a *set of classifiers*
  - *combine the predictions* of multiple classifiers
- MOTIVATION:
  - reduce variance: results are less dependent on peculiarities of a single training set
  - reduce bias: a combination of multiple classifiers may learn a more expressive concept class than a single classifier
- KEY STEP:
  - formation of an ensemble of *diverse* classifiers from a single training set

Bias/Variance Tradeoff



Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

Reduce Variance Without Increasing Bias

- Averaging reduces variance:

$$Var(\bar{X}) \quad \frac{Var(X)}{N}$$

(when predictions are **independent**)

Average models to reduce model variance

One problem:

only one training set

where do multiple models come from?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
    - i.e., probability that a classifier makes a mistake does not depend on whether other classifiers made a mistake
    - **Note:** in practice they are not independent!
- Probability that the ensemble classifier makes a wrong prediction
  - The ensemble makes a wrong prediction if the majority of the classifiers makes a wrong prediction
  - The probability that 13 or more classifiers err is

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

Bagging: Bootstrap Aggregation

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set *D*.
- *Bootstrap sampling*: Given set *D* containing *N* training examples, create *D'* by drawing *N* examples at random with replacement from *D*.

- Bagging:
  - Create *k* bootstrap samples $D_1 \ldots D_k$.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

Bagging

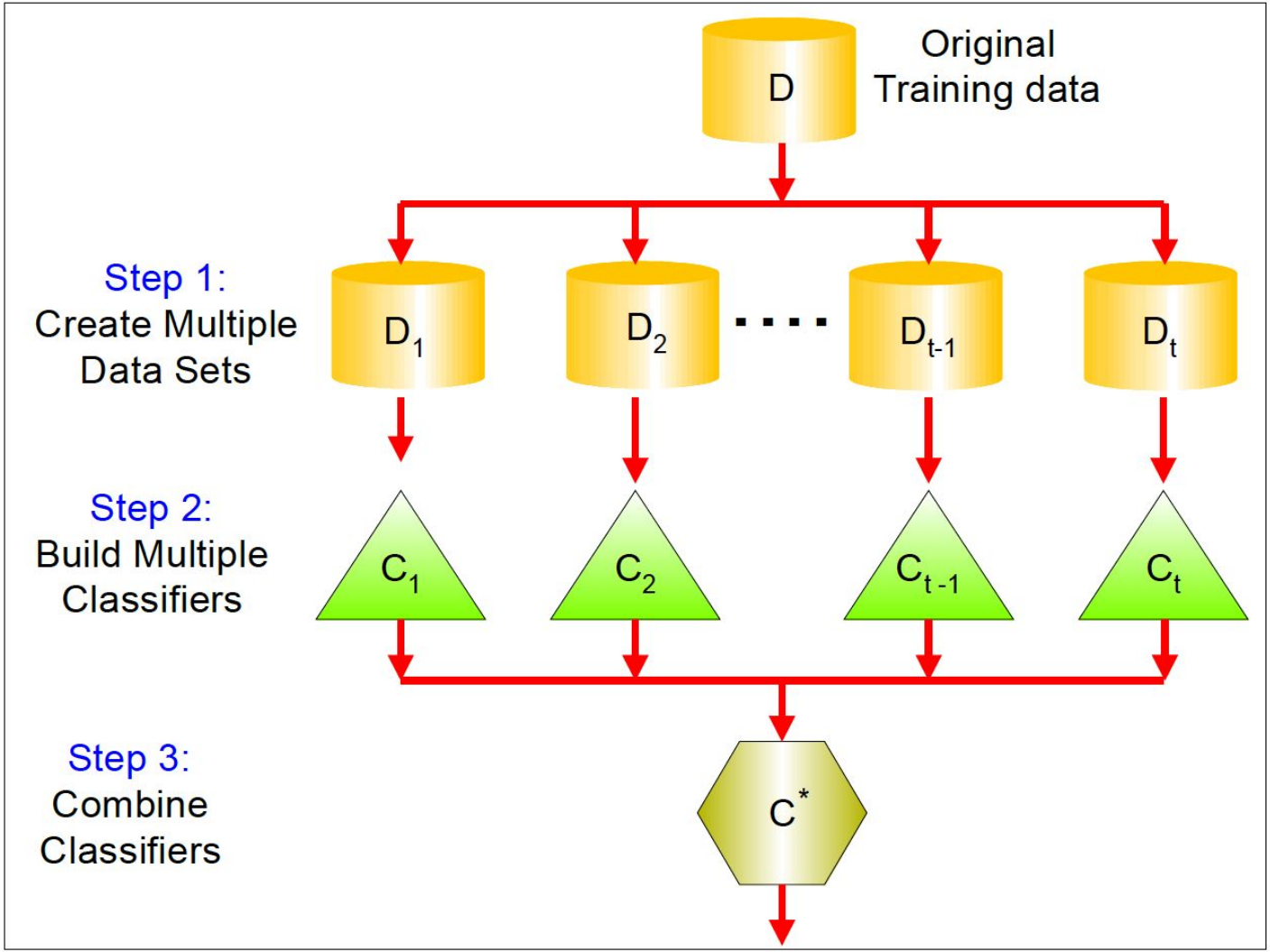
- Best case:

$$Var(Bagging(L(x, D))) \quad \frac{Variance(L(x, D))}{N}$$

In practice:

models are correlated, so reduction is smaller than 1/N

variance of models trained on fewer training cases usually somewhat larger


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2:
Build Multiple
Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3:
Combine
Classifiers
C^*

- Generate new training sets using 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

  - some examples may appear in more than one set
  - for each set, the probability that a given example appears in it is

$$\Pr(x \in D_i) = (1 - \frac{1}{n})^n \rightarrow 0.3678$$

  - i.e., less than 2/3 of the examples appear in one bootstrap sample

1. for $m = 1$ to M // M ... number of iterations
   a) draw (with replacement) a bootstrap sample D_m of the data
   b) learn a classifier C_m from D_m
2. for each test example
   a) try all classifiers C_m
   b) predict the class that receives the highest number of votes

- variations are possible
  - e.g., size of subset, sampling w/o replacement, etc.
- many related variants
  - sampling of features, not instances
  - learn a set of classifiers with different algorithms

Reduce Bias^2 and Decrease Variance?

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average *and* reduce bias?
- Yes:

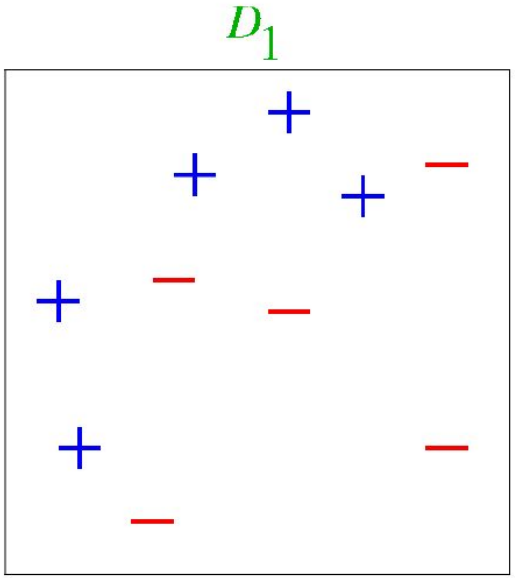
- Boosting

- Basic Idea:
  - later classifiers focus on examples that were misclassified by earlier classifiers
  - weight the predictions of the classifiers with their error
- Realization
  - perform multiple iterations
    - each time using different example weights
  - weight update between iterations
    - increase the weight of incorrectly classified examples
    - this ensures that they will become more important in the next iterations
      (misclassification errors for these examples count more heavily)
  - combine results of all iterations
    - weighted by their respective error measures

1. initialize example weights $w_i = 1/N \quad (i = 1..N)$
2. for $m = 1$ to M // M ... number of iterations
   a) learn a classifier C_m using the current example weights
   b) compute a weighted error estimate $err_m = \frac{\sum w_i \, of \, all \, incorrectly \, classified \, e_i}{\sum_{i=1}^{N} w_i}$ ← $= 1$ because weights are normalized
   c) compute a classifier weight $\alpha_m = \frac{1}{2}\log\left(\frac{1-err_m}{err_m}\right)$
   d) for all correctly classified examples e_i : $w_i \leftarrow w_i e^{-\alpha_m}$
   e) for all incorrectly classified examples e_i : $w_i \leftarrow w_i e^{\alpha_m}$
   f) normalize the weights w_i so that they sum to 1

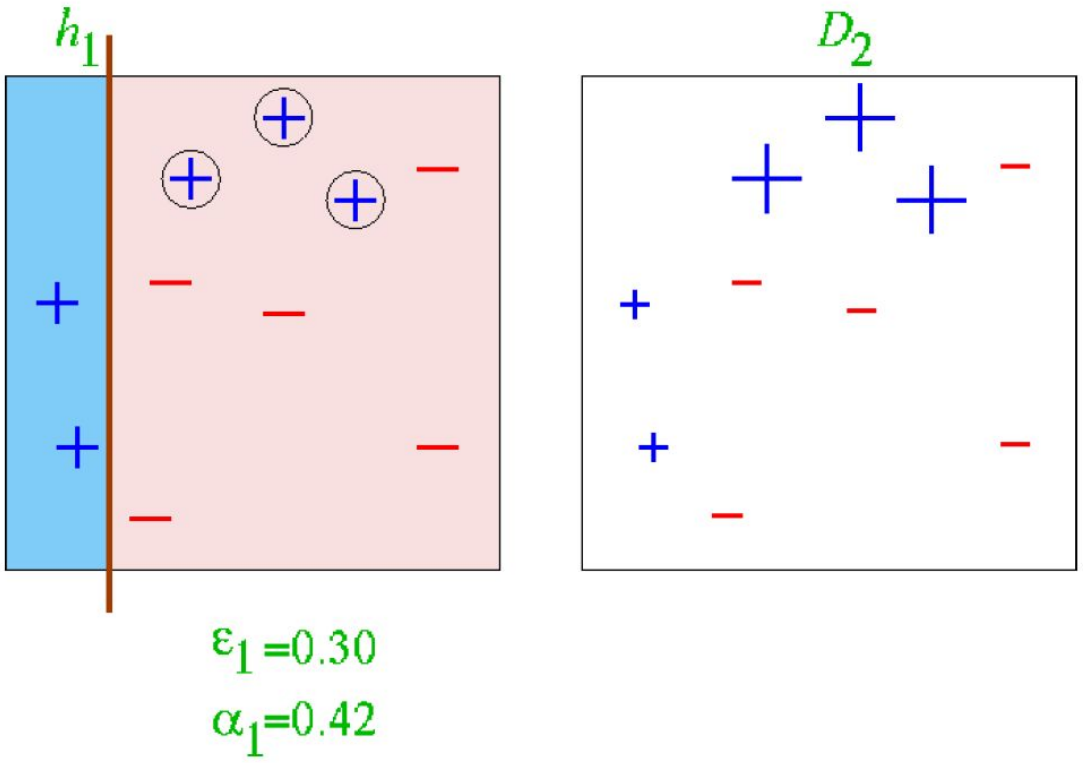
   update weights so that sum of correctly classified examples equals sum of incorrectly classified examples
3. for each test example
   a) try all classifiers C_m
   b) predict the class that receives the highest sum of weights α_m

Example

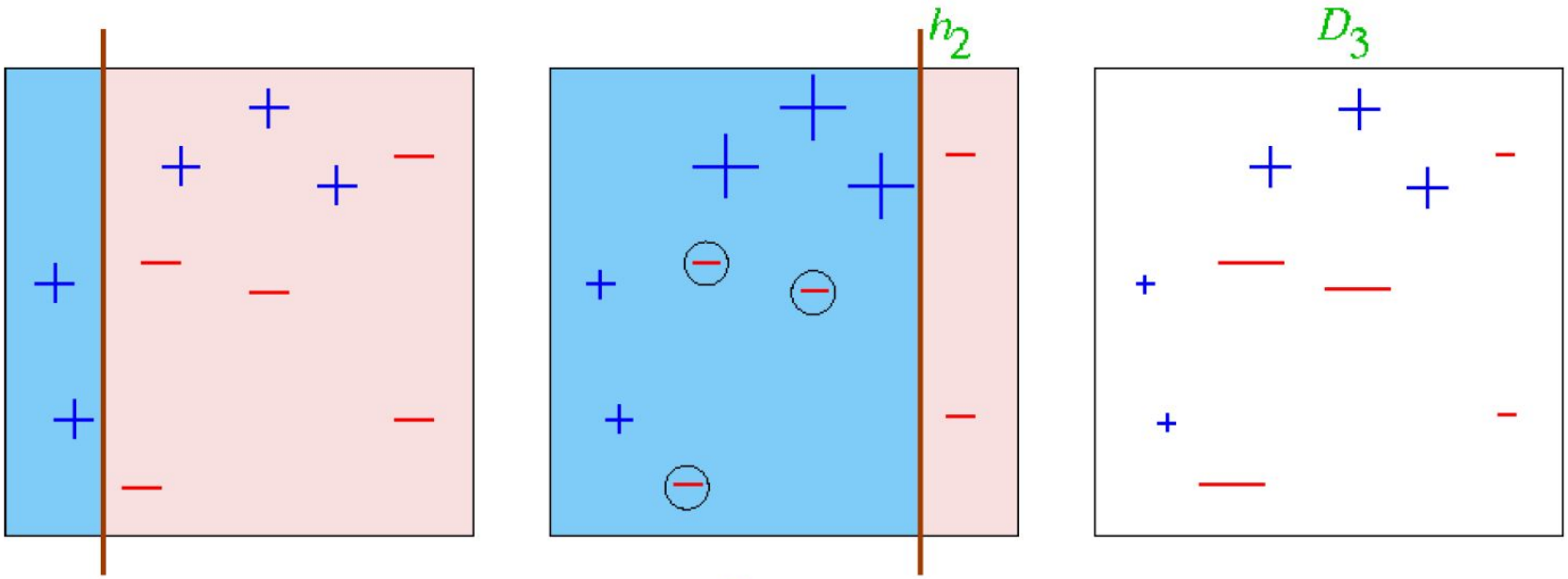


(taken from Verma & Thrun, Slides to CALD Course CMU 15-781, Machine Learning, Fall 2000)

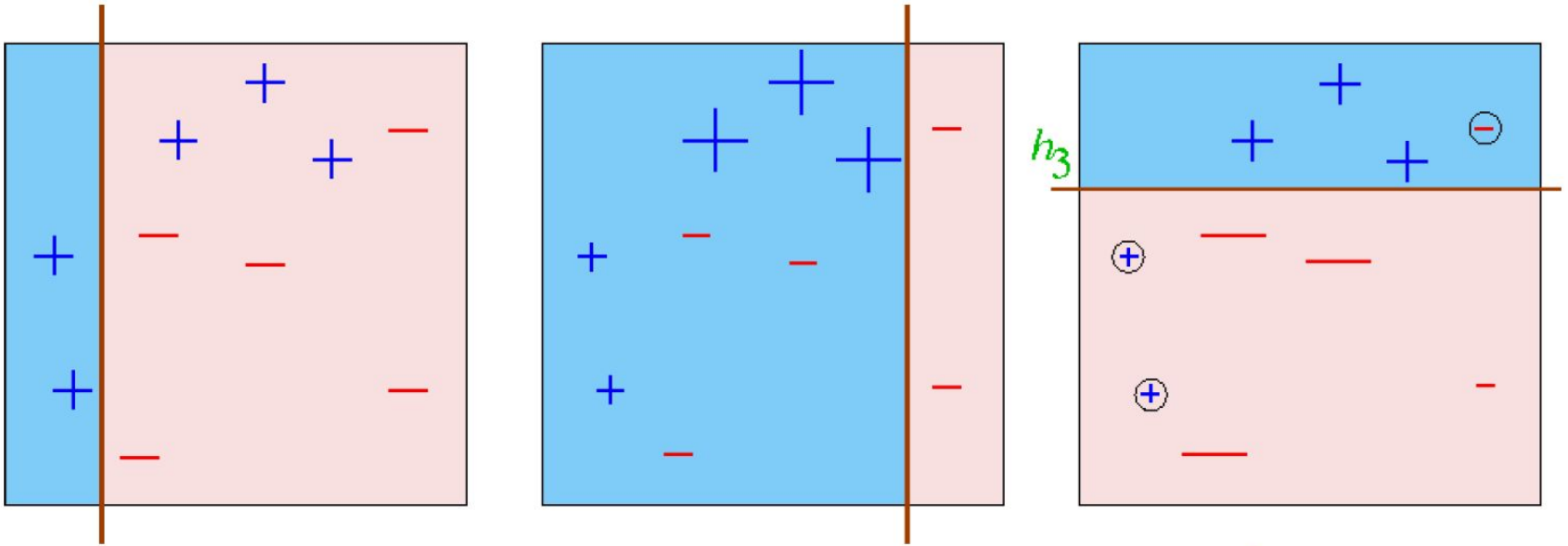
Round 1



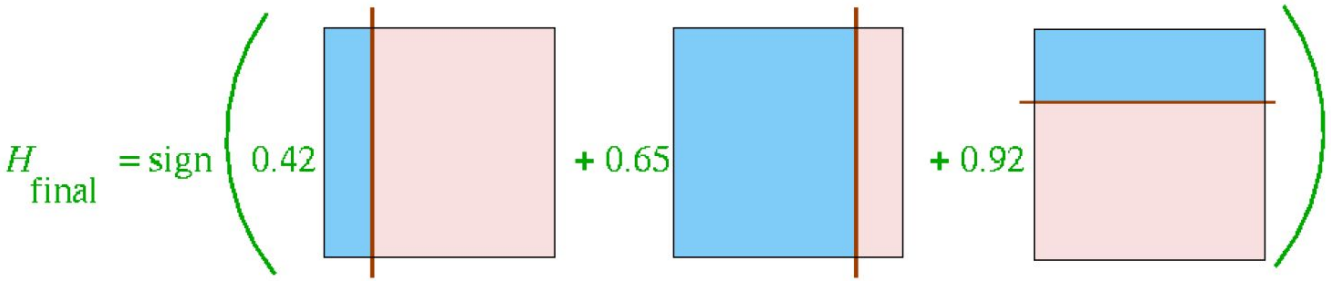
Round 2

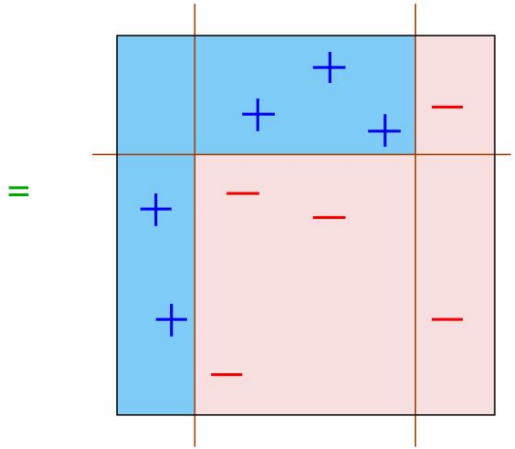


Round 3



Final Round





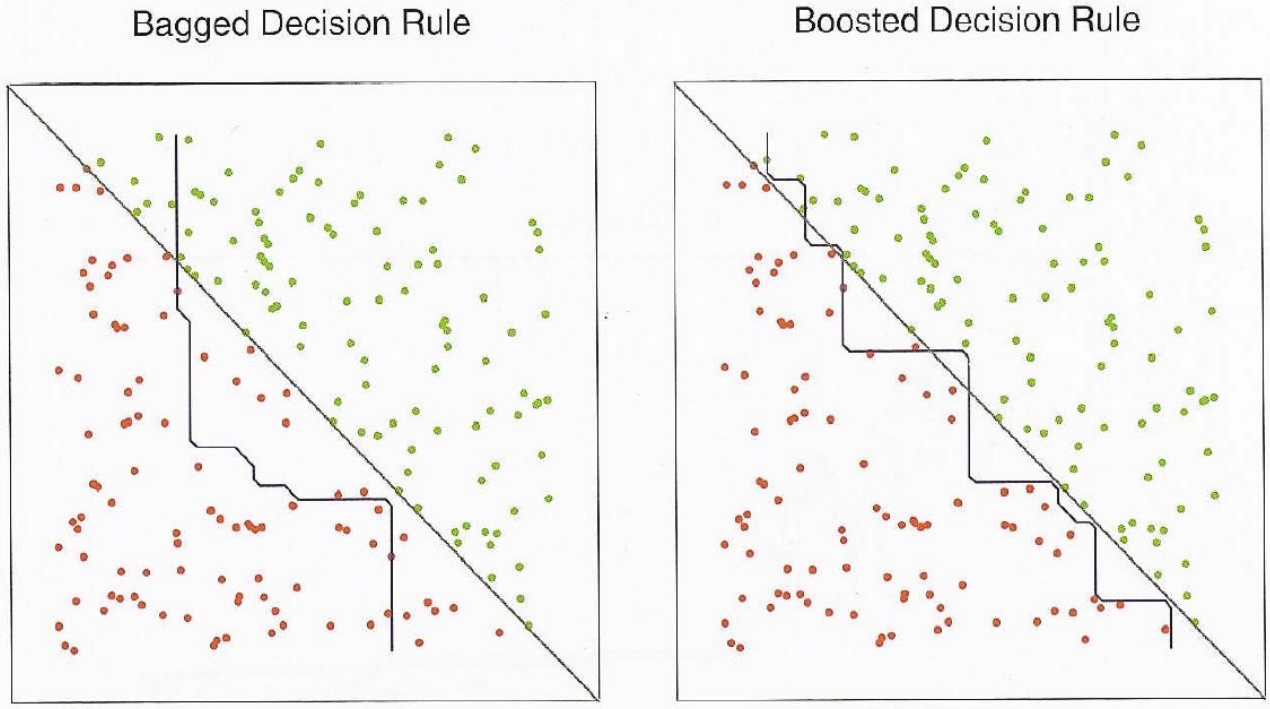


FIGURE 8.11. *Data with two features and two classes, separated by a linear boundary. Left panel: decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. Right panel: decision boundary from boosting the decision rule of the same classifier. The test error rates are* 0.166, *and* 0.065 *respectively. Boosting is described in Chapter 10.*

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001

Bagging vs Boosting

- Bagging
  - noise-tolerant
  - produces better class probability estimates
  - not so accurate
  - statistical basis
  - related to random sampling
- Boosting
  - very susceptible to noise in the data
  - produces rather bad class probability estimates
  - if it works, it works really well
  - based on learning theory (statistical interpretations are possible)
  - related to windowing

Combining Prediction

- voting
  - each ensemble member votes for one of the classes
  - predict the class with the highest number of vote (e.g., bagging)
- weighted voting
  - make a *weighted* sum of the votes of the ensemble members
  - weights typically depend
    - on the classifiers confidence in its prediction (e.g., the estimated probability of the predicted class)
    - on error estimates of the classifier (e.g., boosting)
- stacking
  - Why not use a classifier for making the final decision?
  - training material are the class labels of the training data and the (cross-validated) predictions of the ensemble members

Stacking

- Basic Idea:
  - learn a function that combines the predictions of the individual classifiers
- Algorithm:
  - train n different classifiers $C_1 \ldots C_n$ (the *base classifiers*)
  - obtain predictions of the classifiers for the training examples
    - better do this with a cross-validation!
  - form a new data set (the *meta data*)
    - **classes**
      - the same as the original dataset
    - **attributes**
      - one attribute for each base classifier
      - value is the prediction of this classifier on the example
  - train a separate classifier M (the *meta classifier*)

Stacking

- Example:

Attributes			Class
x_{11}	...	x_{1n_a}	t
x_{21}	...	x_{2n_a}	f
...	...	...	...
x_{n_e1}	...	$x_{n_en_a}$	t

training set

C_1	C_2	...	C_{n_c}
t	t	...	f
f	t	...	t
...	...	...	...
f	f	...	t

predictions of the classfiers

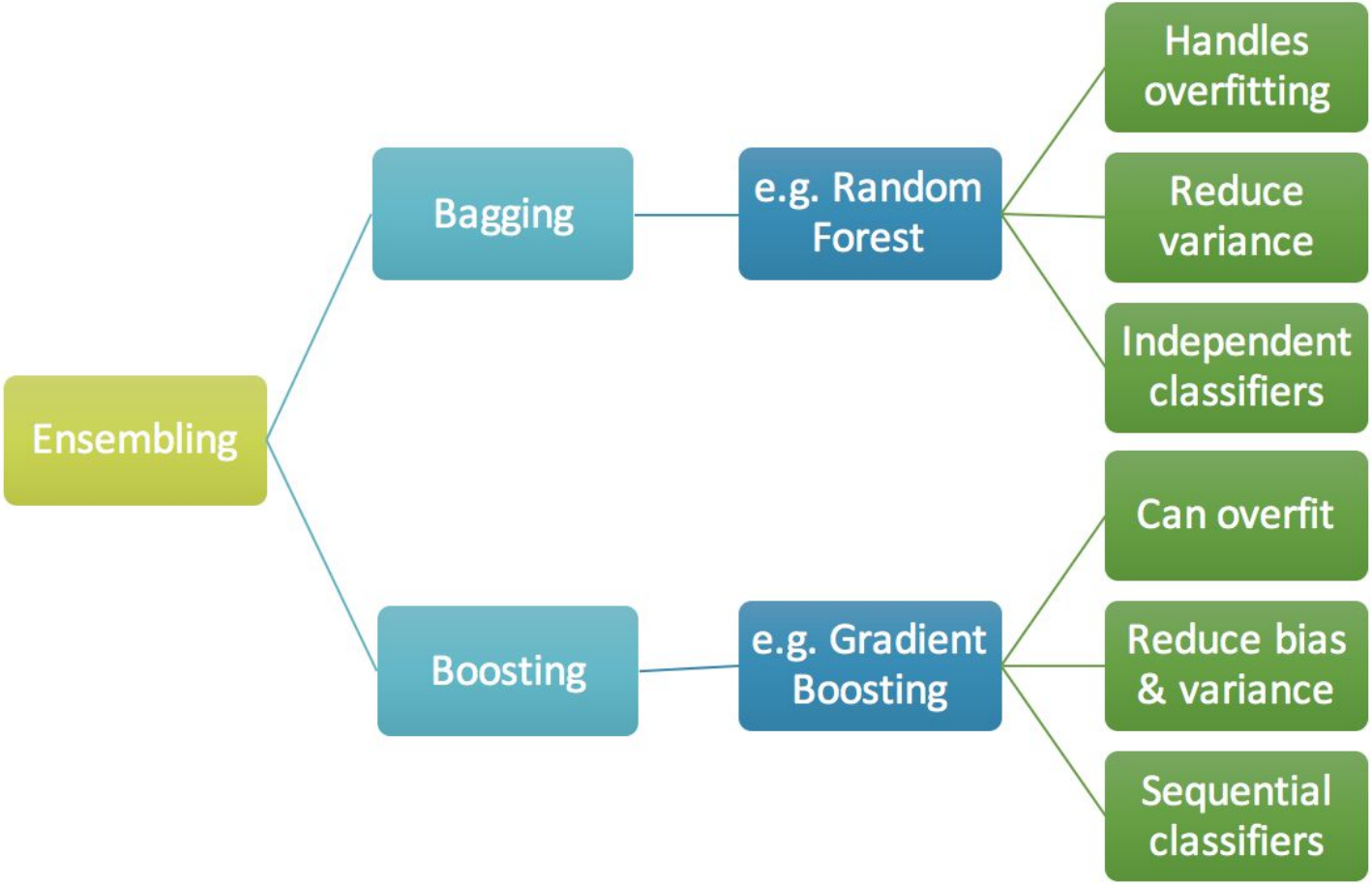
C_1	C_2	...	C_{n_c}	Class
t	t	...	f	t
f	t	...	t	f
...	...	...	...	...
f	f	...	t	t

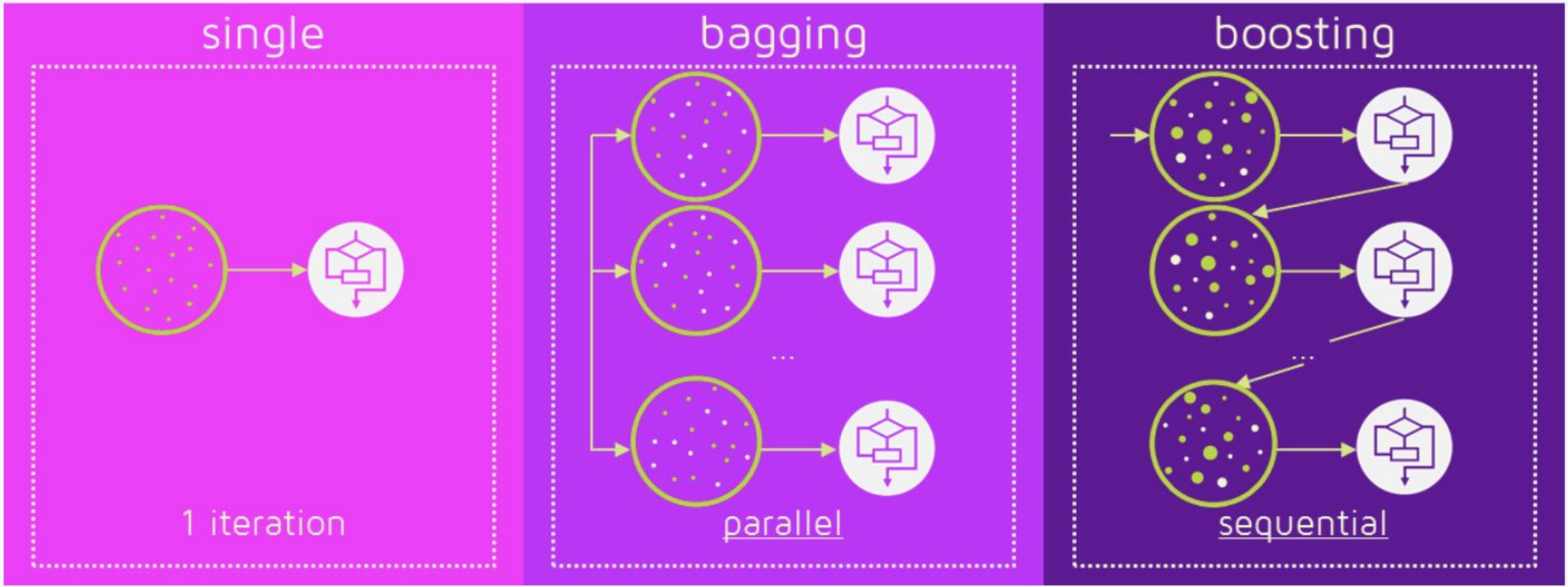
training set for stacking

- Using a stacked classifier:
  - try each of the classifiers $C_1 ... C_n$
  - form a feature vector consisting of their predictions
  - submit this feature vectors to the meta classifier M

Stacking

- Modifying the data
  - Subsampling
    - bagging
    - boosting
  - feature subsets
    - randomly feature samples
- Modifying the learning task
  - pairwise classification / round robin learning
  - error-correcting output codes
- Exploiting the algorithm characterisitics
  - algorithms with random components
    - neural networks
  - randomizing algorithms
    - randomized decision trees
  - use multiple algorithms with different characteristics
- Exploiting problem characteristics
  - e.g., hyperlink ensembles


Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduce variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduce bias & variance
Sequential classifiers


single
1 iteration
bagging
parallel
boosting
sequential

Gradient boosting

- Make a set of predictions $\hat{y}[i]$
- The "error" in our predictions is $J(y,\hat{y})$
  - For MSE: $J(.) = \sum (y[i] - \hat{y}[i])^2$
- We can "adjust" $\hat{y}$ to try to reduce the error
  - $\hat{y}[i] = \hat{y}[i] + \text{alpha } f[i]$
  - $f[i] \approx \nabla J(y, \hat{y})$ $= (y[i]-\hat{y}[i])$ for MSE
- Each learner is estimating the gradient of the loss f'n
- Gradient descent: take sequence of steps to reduce J
  - Sum of predictors, weighted by step size alpha

https://youtu.be/sRktKszFmSk

Ensemble Learning Algorithms

Bagging algorithms:

- Bagging meta-estimator
- Random forest

Boosting algorithms:

- AdaBoost
- GBM
- XGBM
- Light GBM
- CatBoost

XGBOOST

https://xgboost.readthedocs.io/en/latest/

Reference

1. Data Mining: Practical Machine Learning Tools and Techniques
2. http://people.csail.mit.edu/dsontag/courses/ml12/slides/lecture12.pdf
3. https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d
4. http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/
5. http://scikit-learn.org/stable/modules/ensemble.html
6. https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
7. https://github.com/rasbt/python-machine-learning-book-2nd-edition

Thank you