

# Boosting

Adam Richards

July 25, 2016

# Objectives

- 1 Discuss model averaging and other ensemble methods
- 2 Introduce an example
- 3 Adaboost
- 4 Gradient boosting
- 5 Summarize the important points with room for discussion

# Machine learning algorithms

There are always advantages and disadvantages

Algorithm	Strengths	Challenges
<b>GLMs</b>	fast, interpretable	non-linearity, high-dimensionality
<b>KNNs</b>	simple, non-linear	does not scale well, distance function
<b>Decision trees</b>	fast, robust to noise	not competitive in terms of accuracy
<b>Neural Networks</b>	very flexible	prone to overfit, black-box, runtime
<b>SVM's</b>	scales well	parameter and kernel choices
...	...	...
...	...	...
<b>Bayesian networks</b>	probabilistic, priors	not always tractable
<b>Genetic algorithms</b>	mimics natural selection	unsuited for discrete decisions

**There is no free lunch** in statistics...

Or we could say that no one method outperforms all others given all possible data sets

see <http://www.no-free-lunch.org> for more on the theorems

**TABLE 10.1.** Some characteristics of different learning methods. Key:  $\blacktriangle$  = good,  $\blacklozenge$  = fair, and  $\blacktriangledown$  = poor.

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of “mixed” type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large $N$ )	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

[Hastie et al., 2009]

# Combining models

- **Committees** - train some number of models and use, for example, the average of the predictions
- **Boosting** - a variant on committees, where models are **trained in sequence** and the **error function** for a given model depends on the previous one
- **Decision trees** - instead of averaging let the model choice be a function of the input variables
- **Mixture of experts** - instead of hard partitioning of the input space we can move to a **probabilistic framework** for partitioning
- **Model stacking** - a method to combine models of different types

To learn more about model stacking see the [Kaggle guide](#)

# Model averaging

In the **mixture of experts** we combine  $K$  models as follows

$$p(t|\mathbf{x}) = \sum_{k=1}^K \pi_k(\mathbf{x}) p(t|\mathbf{x}, k) \quad (1)$$

where  $\mathbf{x}$  are the inputs,  $t$  are the targets and  $\pi_k(\mathbf{x}) = p(k|\mathbf{x})$

An example of **model combination** is the density of a Gaussian mixture model

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \text{ where } p(\mathbf{X}) = \prod_{n=1}^N p(\mathbf{x}_n) \quad (2)$$

**Bayesian model averaging** has a similar form

$$p(\mathbf{X}) = \sum_{k=1}^K p(\mathbf{X}|k) p(k) \quad (3)$$

except each data point in the former has a weight ( $\pi_k$ ) where each model has weight ( $p(k)$ ) in the latter.

# Model averaging

## Ensemble learning or committees

Use multiple predictive models and combine the predictions

- **Bagging** or Bootstrap aggregation [Breiman, 1996]
  - Bootstrap the training data and grow a tree from each bootstrap sample
  - Average the bootstrapped trees → reduces variance
  - Pruning adds bias so do not prune the trees just average them
- **Random Forests** [Breiman et al., 1999]
  - Bagging except we randomly select predictors at each split
  - Decorrelates the trees
- **Boosting** [Freund and Schapire, 1996]
  - Each subsequent tree is grown based on a reweighted version of the training data
  - Decorrelates the trees

# But why do we want to do this again?

If we use simple linear regression as an example model and imagine we knew the true regression function  $h(\mathbf{x})$ . From bagging we had

$$y_b(\mathbf{x}) = h(\mathbf{x}) + \epsilon_b(\mathbf{x}) \quad (4)$$

we could compute the average sum of squares error as

$$E_{\text{AV}} = \frac{1}{B} \sum_{b=1}^B \mathbb{E}_{\mathbf{x}} [\{y_b(\mathbf{x}) - h(\mathbf{x})\}^2] \quad (5)$$

$$E_{\text{COM}} = \mathbb{E}_{\mathbf{x}} \left[ \left\{ \frac{1}{B} \sum_{b=1}^B y_b(\mathbf{x}) - h(\mathbf{x}) \right\}^2 \right] \quad (6)$$

If we assume that the errors have zero mean and they are uncorrelated

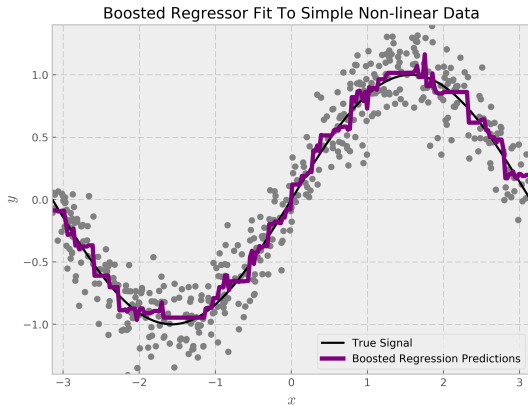
$$E_{\text{COM}} = \frac{1}{B} E_{\text{AV}} \quad (7)$$

So in a perfect world the average error is reduced by a factor of  $B$

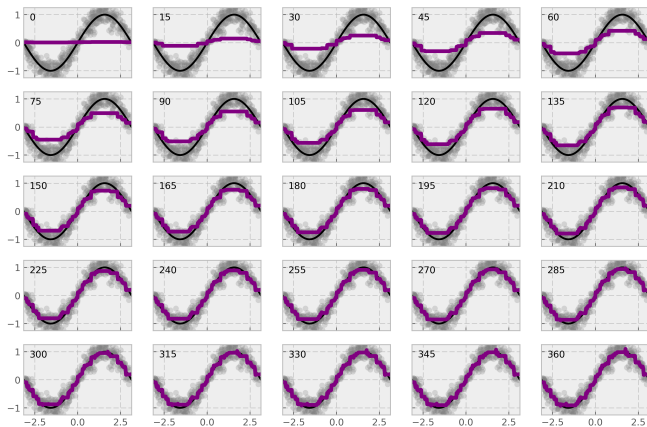
see [Bishop, 2006][Chapter 14] for more details



# A simple example

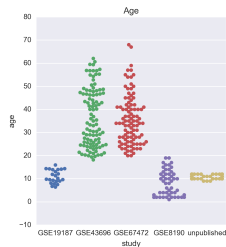
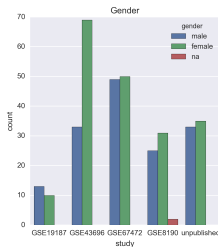
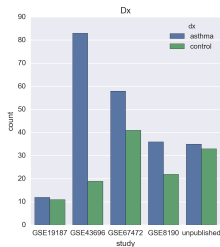


## Boosting Stages Over Time



# Predicting asthma with gene expression data

dx	study	age	gender	gene-1	gene-2	...	...	gene- <i>N</i>
asthma	study1	10	male	0.24381	-0.14274	...	...	-0.04381
control	study3	8	female	0.17981	0.42480	...	...	-0.05456
...	...	...	...	...	...	...	...	...
control	study2	14	female	0.19933	-0.57384	...	...	0.403921



350 samples, filtered by variance from 11523 to 200 genes

# A return to random forests

```
import numpy as np
from sklearn.cross_validation import train_test_split

data = train_test_split(X, y, test_size=0.20, random_state=1)
X_train, X_test, y_train, y_test = data
X_train = X_train[:, np.argsort(X_train.var(axis=0))[:, :-1][:200]]
```

## RandomForestClassifier/Regressor

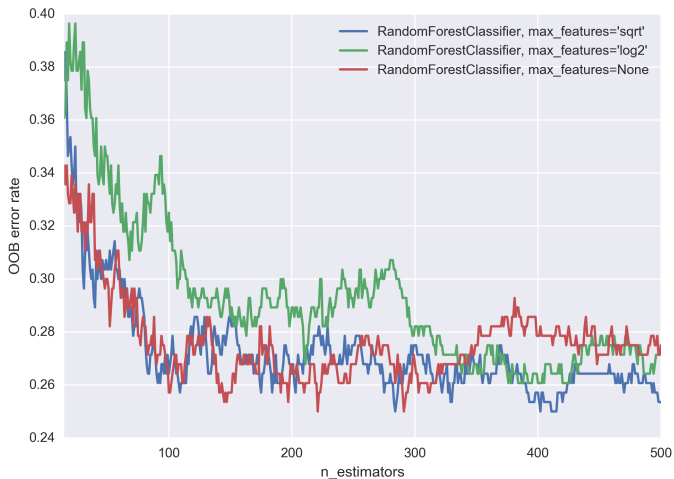
- `sklearn.ensemble`
- `n_estimators` - the trees
- `max_features` - modulates  $m$
- `oob_score`

The sklearn implementation averages the probabilistic prediction (not votes)

## OOB

out-of-bag observations - those not used by a given bootstrapped tree

- Convenient to estimate test error
- For ea. observation predict the response for all trees where it was OOB
- Essentially, a free version of leave one out cross validation



[see the sklearn example](#)

# sklearn.model\_selection.cross\_val\_score

```
from sklearn.model_selection import cross_val_score
clf = RandomForestClassifier(n_estimators=100,
                             max_features='sqrt',
                             random_state=42)
scores = cross_val_score(clf, X, y, cv=5, scoring=None)
print("Accuracy: %0.2f (+/- %0.2f)"%(scores.mean(), scores.std()*2))

scores = cross_val_score(clf, X, y, cv=5,
                           scoring='f1_weighted')
print("F1_weighted: %0.2f (+/- %0.2f)"%(scores.mean(), scores.std()*2))
```

```
Accuracy: 0.71 (+/- 0.11)
F1_weighted: 0.68 (+/- 0.14)
```

There are iterators and other convenience classes for CV i.e. [KFold](#)

# sklearn.model\_selection.grid\_search.GridSearchCV

```
from sklearn.model_selection import GridSearchCV

random_forest_grid = {'max_depth': [3, None],
                      'max_features': ['sqrt', 'log2', None],
                      'min_samples_split': [1, 2, 4],
                      'min_samples_leaf': [1, 2, 4],
                      'bootstrap': [True, False],
                      'n_estimators': [20, 40, 60, 80, 100, 120],
                      'random_state': [42]}

rf_gridsearch = GridSearchCV(RandomForestClassifier(),
                              random_forest_grid,
                              n_jobs=-1, verbose=True,
                              scoring='f1_weighted')
rf_gridsearch.fit(X_train, y_train)
print "best parameters:", rf_gridsearch.best_params_
```

```
best parameters: {'bootstrap': True, 'min_samples_leaf': 1, '
n_estimators': 100, 'min_samples_split': 1, 'random_state': 42, '
max_features': 'sqrt', 'max_depth': None}
```

Adaboost [[Freund and Schapire, 1996](#)]

### Adaptive boosting

- Strives to create a *strong* classifier out of multiple *weak* ones
- Weak learners are often decision trees in practice
- Works on both classification and regression problems
- Classifiers are trained in sequence
- Training occurs on **weighted** data dependent on the previous classifier
- With each iteration the next classifier is more focused on the problematic data
- Predictions are made with a weighted majority vote



- 1 Initialize data weighting coefficients  $\{w_n\}$  by setting them equal to  $1/N$
- 2 For each classifier:
  - Fit the classifier to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \quad (8)$$

- Then evaluate

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}} \quad (9)$$

- to get

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \quad (10)$$

- Update the weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \quad (11)$$

- 3 Make predictions using the final model

$$Y_M(\mathbf{x}) = \text{sign} \left[ \sum_{m=1}^M \alpha_m C_m(\mathbf{x}) \right] \quad (12)$$

# sklearn.ensemble.AdaBoostClassifier

Parameter	Default	Comment
base_estimator	DecisionTreeClassifier	By far the most commonly used
n_estimators	50	Plotting OOB error can help
learning_rate	1.0	Shrinks the contribution of each classifier
max_depth	3	Maximum depth limit for number of nodes in tree
loss	'ls'	Loss function to be optimized (regressor only)

```

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

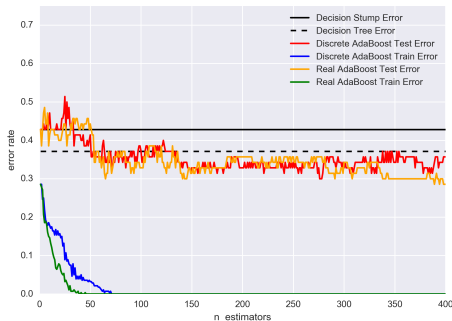
clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
                        n_estimators=100)

scores = cross_validation.cross_val_score(clf,X,y,cv=5,
                                          scoring='f1_weighted')
print("F1_weighted:%0.2f(+/- %0.2f)"%(scores.mean(),scores.std()*2))

```

```
F1_weighted: 0.73 (+/- 0.08)
```

```
dt_stump = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
ada_discrete = AdaBoostClassifier(base_estimator=dt_stump,
                                  n_estimators=n_estimators,
                                  algorithm="SAMME")
ada_real = AdaBoostClassifier(base_estimator=dt_stump,
                              n_estimators=n_estimators,
                              algorithm="SAMME.R")
```



# Gradient boosting

It is a generalization of boosting to arbitrary differentiable loss functions

- Typically uses fixed trees
- Adaboost uses a exponential loss function
- Loss function can be used with both regression and classification
- It uses parameter optimization function **Gradient Decent**

Gradient boosting creates learners as a process. Subsequent learners use the previous step to (i.e. residuals) to weight observations.

Adaboost requires users to specify a set of weak learners where observation weights are continually updated.

# sklearn.ensemble.GradientBoostingClassifier

Parameter	Default	Comment
base_estimator	DecisionTreeClassifier	By far the most commonly used
n_estimators	100	Plotting OOB error can help
learning_rate	0.1	Shrinks the contribution of each classifier
max_depth	3	Maximum depth limit for number of nodes in tree
loss	'ls'	Loss function to be optimized

```
params = {'n_estimators': 400, 'max_depth': 4, 'min_samples_split': 1,
          'learning_rate': 0.01}
```

```
clf = ensemble.GradientBoostingClassifier(**params)
```

```
clf.fit(X_train, y_train)
```

```
scores = cross_validation.cross_val_score(clf, X, y, cv=5, scoring='f1_weighted')
```

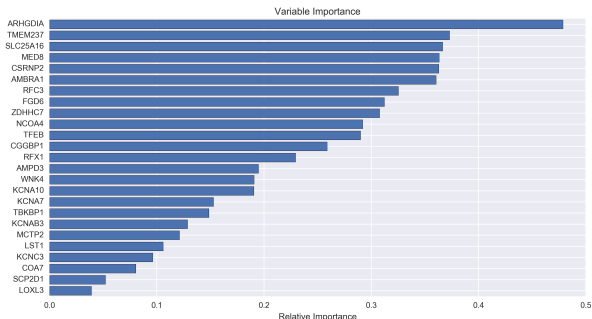
```
print("F1_weighted: %0.2f (+/- %0.2f)"%(scores.mean(),scores.std()*2))
```

```
F1_weighted: 0.65 (+/- 0.14)
```

## Feature importance

- In regression - Total amount of RSS decreased due to splits for a given predictor
- In classification - Add the total amount the Gini index is decreased due to splits for a given predictor

```
feature_importance = clf.feature_importances_  
feature_importance = 100.0*(feature_importance/feature_importance.max())
```



Modified from [gradient boosting regression example](#)

# Summary

- Bagging provided a major methodological step forward
- Bayesian model averaging in that conceptually it is a single model
- Random Forests reduce the correlation among trees
- Boosting also reduces the correlation among trees
- Bagging and RFs are ensemble methods that use **averaging**
- Boosting is a sequential method **averaging**

- Bagging is improved by using more complex base estimators (i.e. trees)
- Boosting on the other hand works best with simple ones
- Random Forests reduce the correlation among trees
- Boosting also reduces the correlation among trees
- Bagging and RFs are ensemble methods that use **averaging**
- Boosting is a sequential method **averaging**

# Resources

## Videos

- [Hastie and Tibshirani ISLR book video](#)
- [Patrick Winston's chalk talk](#)

## Other

- [A well-written paper about Bayesian model averaging in density estimation](#)
- [The new elements of statistical learning book](#)
- [Presentation on gradient boosting](#)





Bishop, C. M. (2006).  
*Pattern Recognition and Machine Learning*.  
Springer.



Breiman, L. (1996).  
Bagging predictors.  
*Machine Learning*, 24(2):123–140.



Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1999).  
*Classification and Regression Trees*.  
CRC Press, New York.



Freund, Y. and Schapire, R. E. (1996).  
Experiments with a new boosting algorithm.  
In Saitta, L., editor, *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, pages 148–156. Morgan Kaufmann.



Hastie, T., Tibshirani, R., and Friedman, J. (2009).  
*The elements of statistical learning: data mining, inference and prediction*.  
Springer, 2 edition.