Meta Learning

Seoul AI Meetup, September 16

Martin Kersner, m.kersner@gmail.com

References

- Ensembling
  - https://mlwave.com/kaggle-ensembling-guide/
  - https://github.com/ageron/handson-ml/blob/master/07_ensemble_learning_and_random_forests.ipynb
- AdaBoost
  - http://www.robots.ox.ac.uk/~az/lectures/cv/adaboost_matas.pdf
  - http://www.cs.princeton.edu/courses/archive/spr08/cos424/readings/Schapire2(
- Netflix Prize
  - http://www.netflixprize.com/
  - https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-sta part-1-55838468f429
- Kaggle competitions
  - https://www.kaggle.com/

Mathematical Notation

- X data, input space
- Y labels, output space
- x_i is the feature vector of the i-th example
- y_i is label (i.e., class) for x_i
- m number of training examples
- n number of features
- $D_j(i)$ weight of i-th training example for j-th base learner (AdaBoost)
- E erorr function

Technical Terms

base learner = weak learner

Content

- Prerequsities
    - Supervised Learning
    - Classification, Regression
    - Data Splitting
    - Bias-Variance Tradeoff, Irreducible error
    - Underfitting, Overfitting
- Ensembles
    - Voting Ensemble
    - Ranking
- Meta Learning
    - Bagging
    - Boosting
    - Stacking/Blending
- Nexar Challenge

Supervised Learning

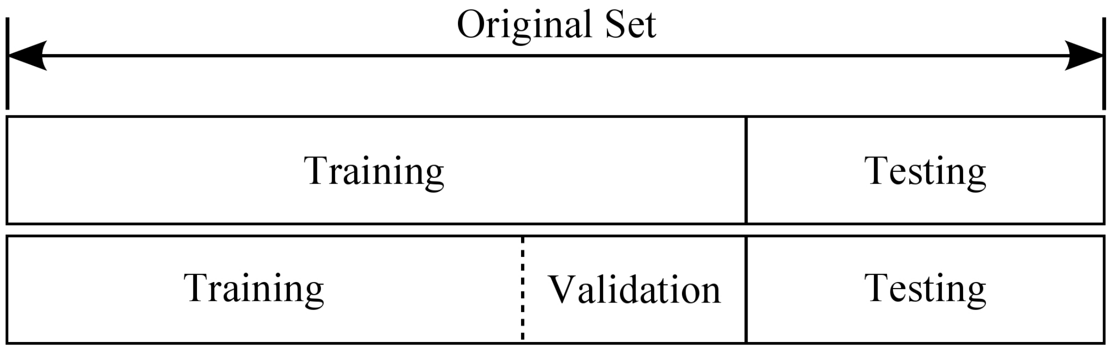
https://en.wikipedia.org/wiki/Supervised_learning

- N training examples of the form $\{\{x_1, y_1\}, \ldots \{x_n, y_n\}\}$
- Searching for a function $g : X \to Y$

Classification vs Regression

- Regression
  - Output variable takes **continuous values**.
  - E.g. Price prediction of certain stock.
- Classification
  - Output variable takes **class labels**.
  - E.g. Prediction of what object is in image.

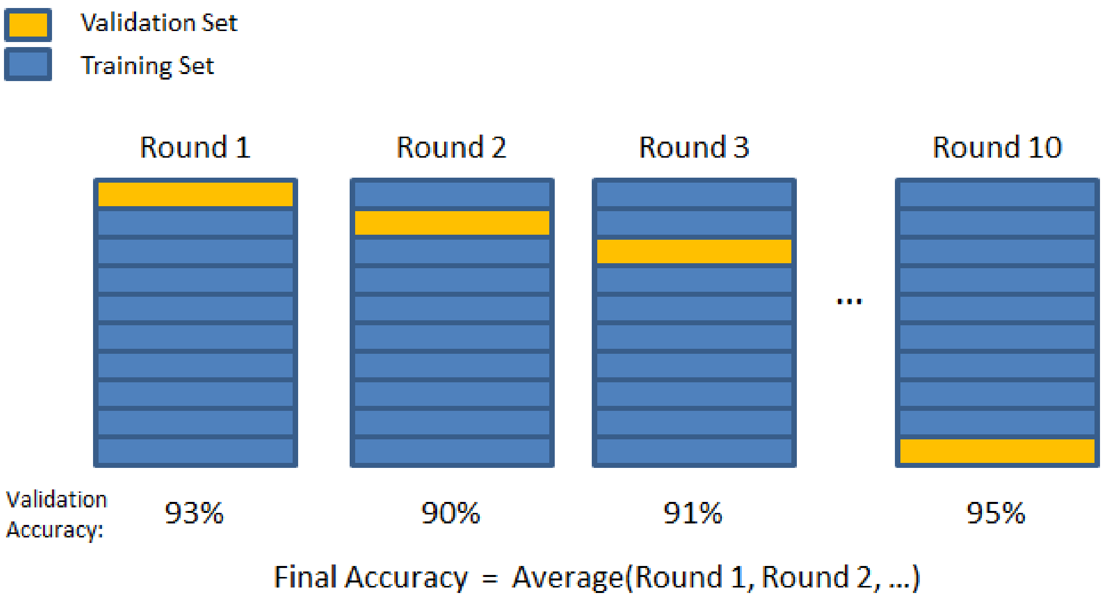
Data Splitting



- **Training** dataset is used for training.
- **Validation** dataset is used for model evaluation.
- **Testing** data imitate real unseen data.

Data Splitting

Cross-validation



Generalization Error

https://en.wikipedia.org/wiki/Generalization_error

- Generalization error is measure of how accurately an algorithm is able to predict outcome values for previously **unseen data**.
- Generalization error is composed of three parts:
    - Bias
    - Variance
    - Irreducible Error
        - Due to noisiness of the data.
        - Can be reduced by cleaning the data (not using wrong/inaccurate data points).

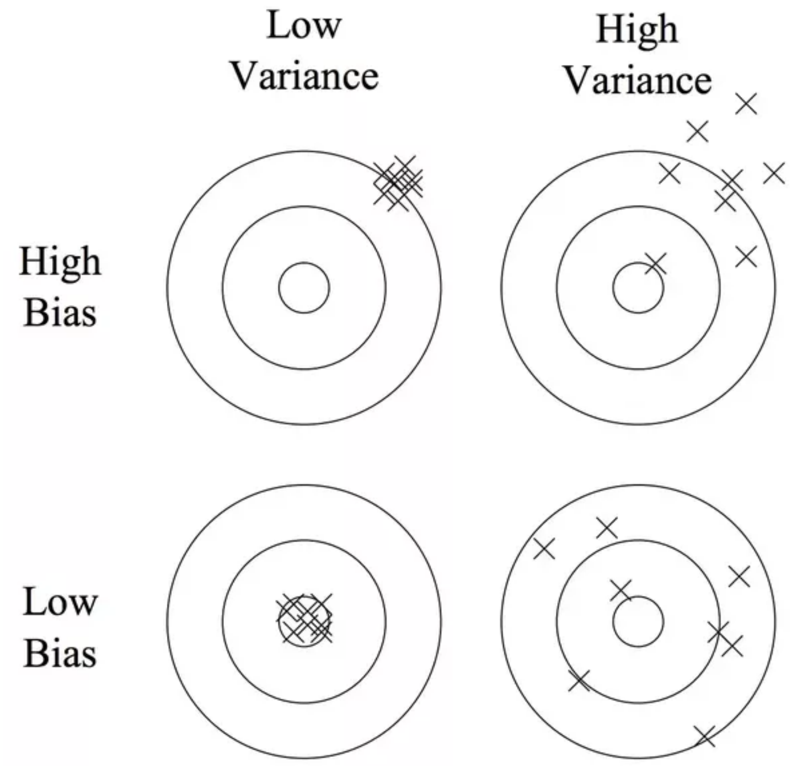
Bias, Variance

https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

- Bias
  - Error from **wrong assumptions** in the learning algorithm.
  - Can cause **underfitting**.

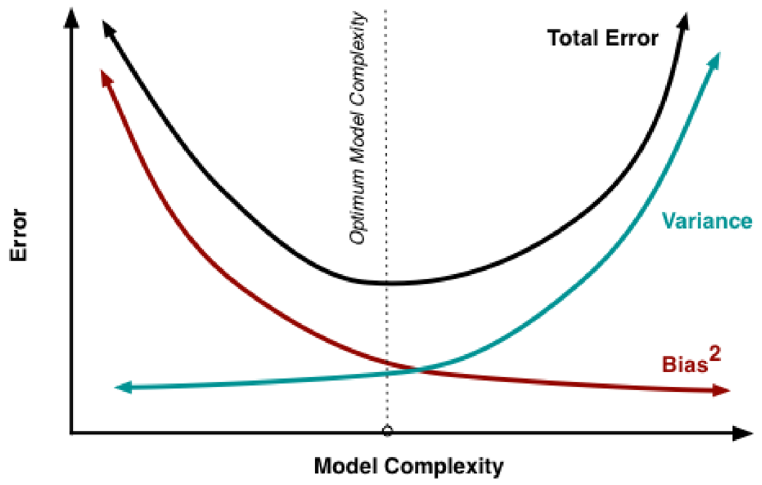
- Variance
  - Error from **sensitivity to small variations** in the training set.
  - Can cause **overfitting**.

Bias, Variance

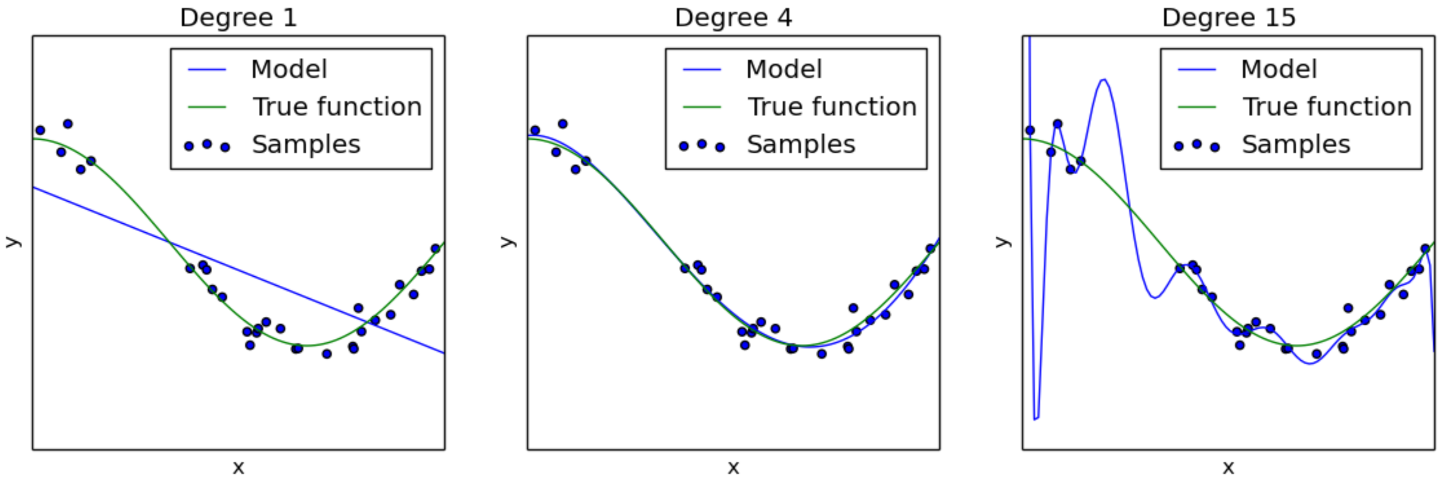


Bias-Variance Tradeoff

- **Increasing model complexity** lead to **increase of variance** and **reduction of bias**.
- **Reducing model complexity** lead to **increase of bias** and **reduction of variance**.



Underfitting, Overfitting



Ensembles and Meta Learning

https://en.wikipedia.org/wiki/Meta_learning_(computer_science)

*"Meta learning is a subfield of Machine learning where **automatic learning algorithms** are applied on meta-data about machine learning experiments."*

- Ensembles
- Meta-Algorithms

Ensembles

- Majority Voting Ensembles
- Weighted Voting Ensemble
- Rank Averaging

Voting Ensembles

https://mlwave.com/kaggle-ensembling-guide/

- Works better with **low-correlated** model predictions.
- Good for **hard predictions** (e.g. multiclass classification accuracy)
- Final class is selected based on (weighted) majority voting.

Voting Ensembles

Approaches

- Majority Voting Ensemble
    - Hard Voting
    - Soft Voting
        - Predict class with the highest class probability, averaged over individual classifiers.
        - In *scikit-learn* all weak classifiers need to have implemented `predict_proba()` method and `voting` parameter of models set to `True`.
- Weighted Voting Ensemble

Majority Voting Ensemble

Example

3 independent binary classification models (A, B, C) with accuracy 70 %.

- 70 % of time correct prediction.
- 30 % of time wrong prediction.

At least two predictions (out of three) have to be correct.

Voting Mechanism	• *A: 1* • *B: 1* • *C: 0*	• *Final classification: 1*

Majority Voting Ensemble

All three are correct

In [2]:
```
P3 = 0.7 * 0.7 * 0.7
print(P3)
```

```
0.3429999999999999
```

Two are correct

In [3]:
```
P2 = 3 * (0.7 * 0.7 * 0.3)
print(P2)
```

```
0.4409999999999999
```

Majority Voting Ensemble

One is correct

In [4]:
```
P1 = 3 * (0.3 * 0.3 * 0.7)
print(P1)
```

```
0.189
```

None is correct

In [5]:
```
P0 = 0.3 * 0.3 * 0.3
print(P0)
```

```
0.027
```

Majority Voting Ensemble

Result

Most of the time (P2 ~ 44 %) the majority vote corrects an error.

Prediction accuracy of majority ensembling mode will be **78.38 %** (P3 + P2) which is higher than when using models individually.

Using **5** independent binary models with accuracy 70 %, accuracy of majority voting raises to **83.69 %**.

Correlation

- **0** no correlation
- **+1** positive correlation
- **-1** negative correlation

Pearson Correlation

https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

- **Linear** correlation between **two** variables

Spearman's rank correlation coefficient

https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

- **Monotonic** correlation between **two** variables

Open-source

https://github.com/MLWave/Kaggle-Ensemble-Guide/blob/master/correlations.py

Correlated Models

In [7]:
```
GT = np.array([1,1,1,1,1,1,1,1,1,1])

A  = np.array([1,1,1,1,1,1,1,1,0,0]) # 80 % accuracy
B  = np.array([1,1,1,1,1,1,1,1,0,0]) # 80 % accuracy
C  = np.array([1,0,1,1,1,1,1,1,0,0]) # 70 % accuracy
```

In [8]:
```
sum(A+B+C >= 2)/len(A)
```

Out[8]:
```
0.8000000000000004
```

Accuracy with voting ensembles is still **only 80 %**!

For highly correlated models, majority voting enembles don't help much or not at all.

Non-correlated Models

In [9]:
```
A = np.array([1,1,1,1,1,1,1,1,0,0]) # 80 % accuracy
B = np.array([0,1,1,1,0,1,1,1,0,1]) # 70 % accuracy
C = np.array([1,0,0,0,1,0,1,1,1,1]) # 60 % accuracy
```

Using **highly uncorrelated models**, accuracy raised to **90 %**.

In [10]:
```
sum(A+B+C >= 2)/len(A)
```

Out[10]:
```
0.9000000000000002
```

Majority Voting Ensemble

scikit-learn

sklearn.ensemble.VotingClassifier

In [12]:

```
# source: Hands-on Machine Learning with Scikit-Learn & Tensorflow, Chapter 7
log_clf = LogisticRegression(random_state=random_state)
rnd_clf = RandomForestClassifier(random_state=random_state)
svm_clf = SVC(random_state=random_state)

voting_clf = VotingClassifier(estimators=[('lr', log_clf),
                                          ('rf', rnd_clf),
                                          ('svc', svm_clf)],
                              voting='hard') # or soft

voting_clf.fit(X_train, y_train)
```

Out[12]:

```
VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=Non
e, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=42, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)), ('rf', RandomFor...f',
  max_iter=-1, probability=False, random_state=42, shrinking=True,
  tol=0.001, verbose=False))],
         flatten_transform=None, n_jobs=1, voting='hard', weights=None)
```

Weighted Voting Ensemble

- Weights of individual models in ensemble can differ.
- The main purpose is to give more weight to a better model.
- E.g. Model with better performance should have larger impact. Low performing models have to overrule (same prediction) high performing model, otherwise their classification result will be ignored.

Weighted Voting Ensemble

Approaches to Weight Selection

One of the most common challenge with ensemble modeling is to find optimal weights to ensemble base models.

- Same weights for each model
- Heuristical approach
- Use cross-validation score of base models to estimate weights
- Explore Kaggle winning solutions

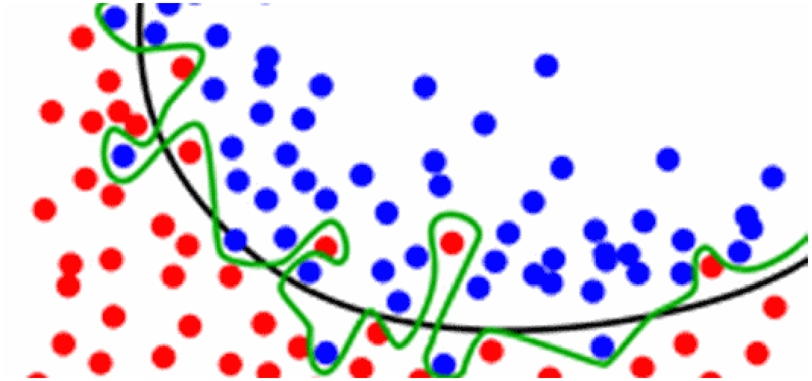
Averaging

- Works well for a wide range of problems (classification/regression) and metrics (AUC, squared error or logaritmic loss).
- Often **reduces overfit** (smoothens separation between classes).

- Arithmetic Mean
  - https://github.com/MLWave/Kaggle-Ensemble-Guide/blob/master/kaggle_avg.py
- Geometric Mean
  - $\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$
  - Good when comparing values with different numeric ranges.
  - https://github.com/MLWave/Kaggle-Ensemble-Guide/blob/master/kaggle_geomean.py

Averaging

Why it works?



Rank Averaging

https://github.com/MLWave/Kaggle-Ensemble-Guide/blob/master/kaggle_rankavg.py

https://www.kaggle.com/cbourguignat/why-calibration-works

- Good for uncalibrated predictors.
    - Probability predictions aren't spread over whole range (0.0 - 1.0)
- Works well on evaluation metric as ranking or threshold based like AUC.

Computation

1. Turn the predictions into ranks (`np.argmin()`).
2. Average these ranks.
3. Compute ranks of averages and normalize them to 0 - 1 range.

Rank Averaging

Example

In [13]:
```
A = np.array([0.57,  # 1
              0.04,  # 0
              0.96,  # 2
              0.99]) # 3

B = np.array([0.35000056,  # 1
              0.35000002,  # 0
              0.35000098,  # 2
              0.35000111]) # 3

C = np.array([0.350000]*4)
```

When averaging model with uncorrelated model added information is only minimal

```
A = np.array([0.57, 0.04, 0.96, 0.99])

B = np.array([0.35000056, 0.35000002, 0.35000098, 0.35000111])
```

In [14]:
```
# Arithmetic Mean
A_B = (A + B)/2
print(A_B)
```

```
[ 0.46000028  0.19500001  0.65500049  0.67000055]
```

In [15]:
```
A_C = (A + C)/2
print(A_B-A_C)
```

```
[  2.80000000e-07   1.00000000e-08   4.90000000e-07   5.55000000e-07]
```

In [16]:
```
# Rank Averaging
R_AB = (np.argsort(A)+np.argsort(B))/2
print(R_AB / np.max(R_AB))
```

```
[ 0.33333333  0.          0.66666667  1.        ]
```

How To Select Base Models?

- Forward Selection of base models
- Model selection with replacement
- Meta-algorithms

Meta-Algorithms

- Bagging
- Boosting
- Stacking/Blending

- Every algorithm consists of two steps (stats.stackexchange.com):

  1. Producing a distribution of **simple models** on **subsets** of the original data.
  2. Combining the distribution of simple models into one **aggregated** model.

Meta-Algorithms

Pros

- Better prediction
- More stable model

Cons

- Slower
- Models are non-human readeable
- Can cause overfitting

Bagging

https://en.wikipedia.org/wiki/Bootstrap_aggregating

1. Create **random samples** (sampling uniformly and **with replacement**) of the training data set.
2. Train a model **from each sample**.
3. **Combine** results of these multiple classifiers using **average** (regression) or **majority voting** (classification).

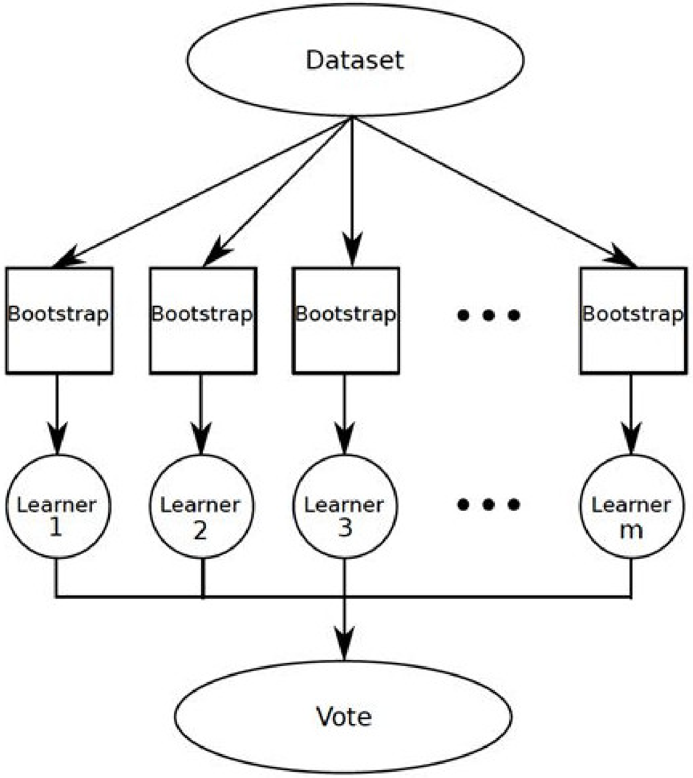
- Bagging helps to reduce the variance error.
- Models are trained **independently**.

Bagging

Pasting

- Same method as bagging, however training samples are sampled **without replacement**.
- Set `bootstrap=False` in sklearn.ensemble.BaggingClassifier or sklearn.ensemble.BaggingRegressor.

Bagging



Dataset
Bootstrap
Bootstrap
Bootstrap
Bootstrap
Learner 1
Learner 2
Learner 3
Learner m
Vote

Bagging

scikit-learn

- sklearn.ensemble.BaggingClassifier
- sklearn.ensemble.BaggingRegressor

In [17]:
```
# source: Hands-on Machine Learning with Scikit-Learn & Tensorflow, Chapter 7
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(DecisionTreeClassifier(random_state=random_state),
                            # 500 base models (Decision Trees)
                            n_estimators=500,
                            # if True, features are randomly selected with repla
cement
                            bootstrap_features=False,
                            # if False, then using all data
                            bootstrap=True,
                            random_state=random_state)

bag_clf.fit(X_train, y_train)
```

Out[17]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, cri
terion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=500, n_jobs=1, oob_score=False,
         random_state=42, verbose=0, warm_start=False)
```

Random Forest

https://en.wikipedia.org/wiki/Random_forest

- Bagging algorithm
- Base learners are Decision Trees.
- Classification, Regression
- De-correlation by **random sampling** (both data and features).
- An **optimal number of trees** can be found using **cross-validation** or by observing the **out-of-bag error**.

Out-Of-Bag Error

- The mean prediction error on each training sample X_i, using only the trees that did not have X_i in their bootstrap sample.
- `oob_score_` attribute in sklearn.ensemble.RandomForestClassifier when trained with `oob_score=True`

Random Forest

Training procedure

1. Select a random sample (from training data) with replacement.
2. At each node split, utilize only random subset of the features (= "feature bagging").
   - If `max_features=auto` in sklearn.ensemble.RandomForestClassifier then $size\_of\_subset = \sqrt{number\ of\ features}$
3. Repeat 1 and 2 steps until you obtain desired number of weak learners.
4. Combine base learners for final prediction using **mode** (classification) or **mean** (regression).

Random Forest

scikit-learn

- sklearn.ensemble.RandomForestClassifier
- sklearn.ensemble.RandomForestRegressor

In [18]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100,
                             max_depth=2,
                             random_state=random_state)

clf.fit(X_train, y_train)
```

Out[18]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=2, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Extremely Randomized Trees

Same as Random Forest but **nodes are NOT split based on the most discriminative threshold**, thresholds are drawn at **random** for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.

- Decrease variance even more.
- Bias slightly increase.

Extremely Randomized Trees

scikit-learn

[sklearn.ensemble.ExtraTreesClassifier]()

In [19]:
```
from sklearn.ensemble import ExtraTreesClassifier

clf = ExtraTreesClassifier(n_estimators=100,
                           max_depth=2,
                           random_state=0)

clf.fit(X, y)
```

Out[19]:
```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=2, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Boosting

https://en.wikipedia.org/wiki/Boosting_(machine_learning)
http://www.cs.princeton.edu/courses/archive/spr08/cos424/readings/Schapire2003.pdf

Boosting is a method of turning a sequence of **weak learners** to one **strong learner**.

- Weak learner
  - Classifier/Regressor which can label testing examples **better than random guessing**.
- Strong learner
  - Classifier/Regressor that is **arbitrarily well-correlated** with the true label.

Boosting

Properties

- Models are trained **sequentally**.
- Unlike bagging, **data subset creation is not random** and depends upon the performance of the previous models.
- When weak learners are put together, they are typically weighted in some way.

- Boosting is primarily **reducing bias**.
- Tends to overfit the training data.

Boosting

Training Procedure

1. Train a weak learner on whole training dataset.
2. Train another weak learner that will try to improve classification/regression results performed by previous weak learners.
3. Combine all weak learners together and evaluate.
4. Repeat steps 2-3 until you achieve desired accuracy or reach the maximum number of weak learners.

AdaBoost

https://en.wikipedia.org/wiki/AdaBoost

Properties

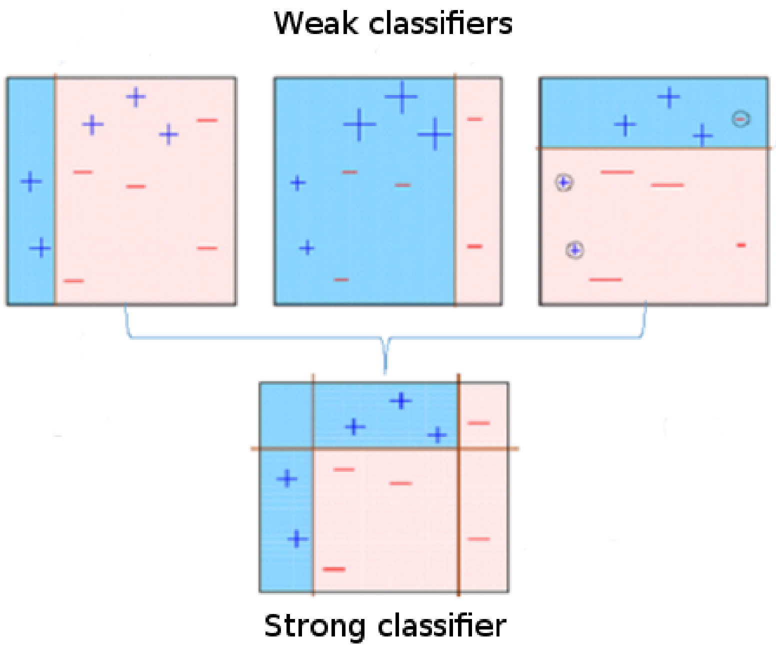
- Any weak learner can be used (often used decision stumps).
- Sensitive to noisy data and outliers.

Training Procedure

1. Assign weight (same for each example; $D_1(i) = \frac{1}{m}$) to each training example.
2. Train weak learner on whole training dataset.
3. Evaluate weak learner and reweight data accordingly.
    - Misclassified examples **gain** weight.
    - Correctly clasified examples **lose** weight.
4. Train another weak learner that focuses on examples that were misclassified by previous weak learner.
5. Evaluate weak learner and update weights appropriately (as in step 3).
6. Combine all weak learners using **weighted sum** and evaluate.
7. Repeat steps 4 - 6 until you achieve desired accuracy or reach the maximum number of weak learners.

AdaBoost

Reweighting



AdaBoost

Binary Classifier

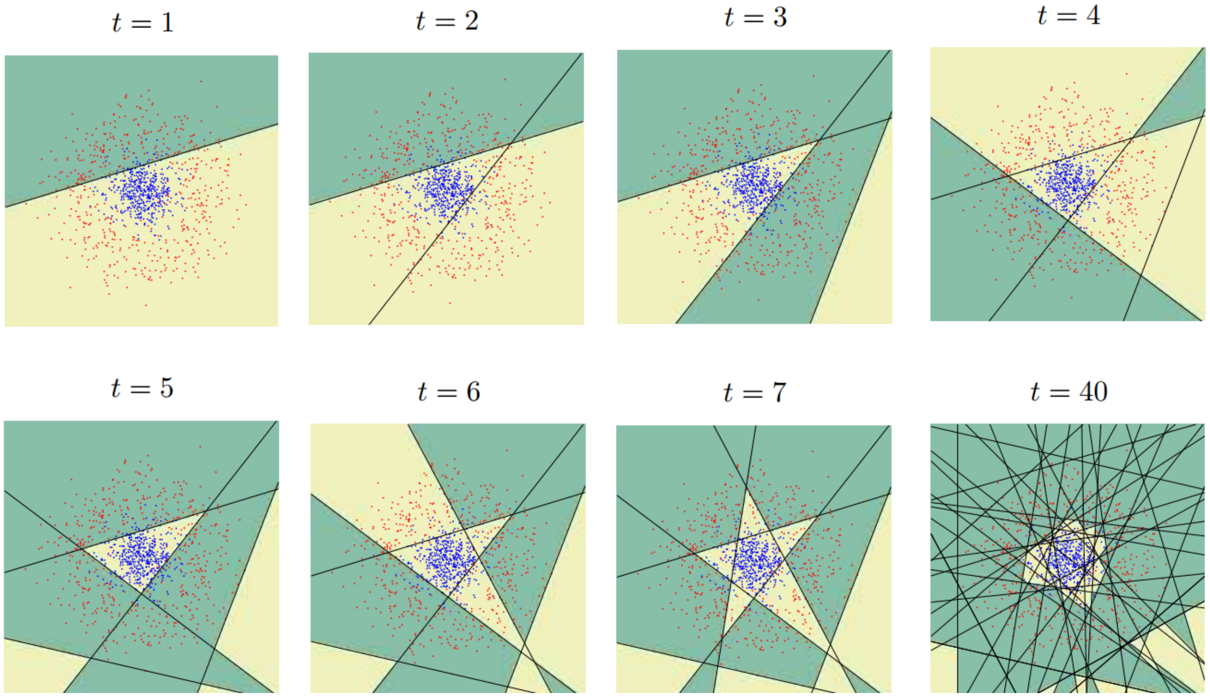
- $WeakLearner = f_t(x) = \alpha_t h_t(x)$
- $AdaBoost_T(x) = sign(\sum_{t=1}^{T} f_t(x))$

Minimizing error of AdaBoost classifier at t-th iteration:

- $E_t = \sum_i E[AdaBoost_{t-1}(x_i) + \alpha_t h_t(x_i)]$, where E represents error function

AdaBoost

Visualization of training



AdaBoost

scikit-learn

- `base_estimator` defines a weak learner (requires `sample_weight` parameter in `fit()` method).
- `n_estimator` represents number of weak learners.

AdaBoostClassifier

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

```
sklearn.ensemble.AdaBoostClassifier(base_estimator=None,
                                    n_estimators=50,
                                    learning_rate=1.0,
                                    algorithm='SAMME.R',
                                    random_state=None)
```

Gradient Boosting

- Trained **sequentually**
- Classification / Regression
- Add models to an ensemble; each model is correcting its predecessor.
- Fit new model to the **residual errors** made by previous model.
- Early stopping
    - Technique to estimate number of base models.

```python
from sklearn.tree import DecisionTreeRegressor

# The first tree
t1 = DecisionTreeRegressor().fit(X, y)

# The second tree
y2 = y - t1.predict(X)
t2 = DecisionTreeRegressor().fit(X, y2)

# The third tree
y3 = y2 - t2.predict(X)
t3 = DecisionTreeRegressor().fit(X, y3)

# Final prediction
t = sum(t1.predict(X_new) + t2.predict(X_new) + t3.predict(X_new))
```

Gradient Boosting

Stochastic Gradient Boosting

If `subsample` parameter is less than 1.0 sample only part of training dataset sklearn.ensemble.GradientBoostingRegressor.

Gradient Boosting

scikit-learn

sklearn.ensemble.GradientBoostingRegressor

Other open-source implementations

- https://github.com/dmlc/xgboost
- https://github.com/catboost/catboost
- https://github.com/Microsoft/LightGBM

Stacking (Stacked Generalization)

Training a model to combine the predictions of several other models.

1. Split dataset to n folds.
2. Train independently on each fold and predict for the others.
3. Aggregate predictions from different folds and use them as input to another layer.
4. If there are more layers, predictions are split and trained on n folds independently again.

- Because each layer uses the "same" dataset, due to incorrect data manipulation **information leak** could happen.

Blending

- Similar to stacking, but **uses less data**.

1. Split dataset to n parts, where n represents number of layers.
2. Train model(s) on the first part of data and predict on the second part.
3. Train another layer of model(s) using predictions from previous layer.
4. Repeat step 3 until n is reached.

Open-source implementation for stacking/blending

https://github.com/viisar/brew

- Ensembling
- Stacking
- Blending
- Ensemble Generation
- Ensemble Pruning
- Dynamic Classifier Selection
- Dynamic Ensemble Selection

Nexar

Model 1 Model 2 Final model