The effectiveness of gradual learning

ENSEMBLE METHODS IN PYTHON

Román de las Heras
Data Scientist, SAP / Agile Solutions

Collective vs gradual learning

Collective Learning

- **Principle:** wisdom of the crowd
- Independent estimators
- Learning the same task for the same goal
- Parallel building

Gradual Learning

- **Principle:** iterative learning
- Dependent estimators
- Learning different tasks for the same goal
- Sequential building

Gradual learning

Possible steps in gradual learning:

1. First attempt (initial model)
2. Feedback (model evaluation)
3. Correct errors (subsequent model)

Fitting to noise

White noise

- Uncorrelated errors
- Unbiased errors and with constant variance

Improvement tolerance

- If *Performance difference* < *improvement threshold*:
  - Stop training

It's your turn!

ENSEMBLE METHODS IN PYTHON

Adaptive boosting: award winning model

ENSEMBLE METHODS IN PYTHON

Román de las Heras
Data Scientist, SAP / Agile Solutions

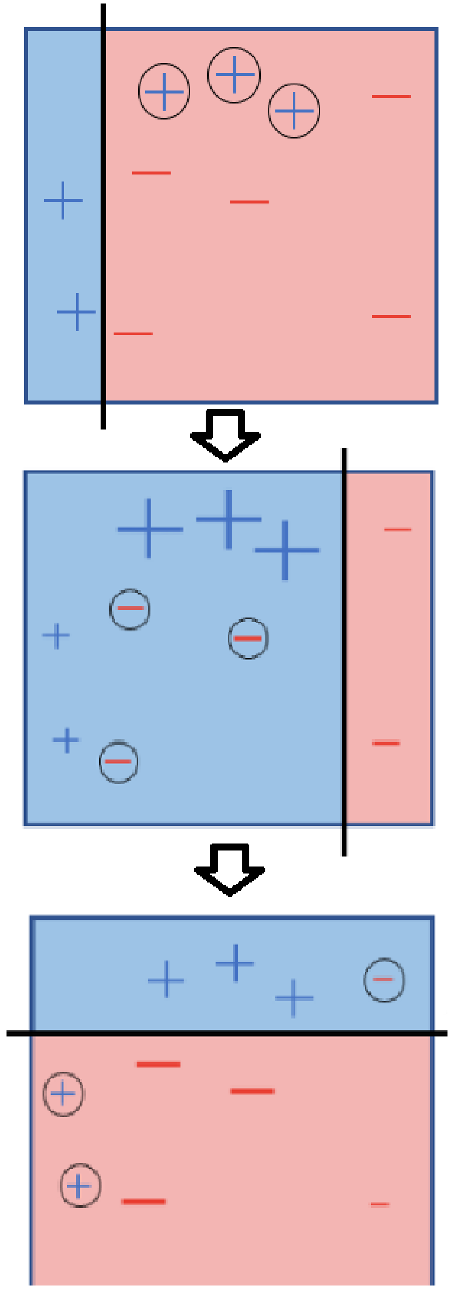
Award winning model

About AdaBoost:

- Proposed by Yoav Freund and Robert Schapire (1997)
- Winner of the Gödel Prize in (2003)
- The first practical boosting algorithm
- Highly used and well known ensemble method

AdaBoost properties

1. Instances are drawn using a sample distribution
    - Difficult instances have higher weights
    - Initialized to be uniform
2. Estimators are combined with a weighted majority voting
    - Good estimators are given higher weights
3. Guaranteed to improve
4. Classification and Regression



AdaBoost classifier with scikit-learn

AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
```

```
clf_ada = AdaBoostClassifier(
    base_estimator,
    n_estimators,
    learning_rate
)
```

Parameters

- `base_estimator`
  - Default: Decision Tree (max_depth=1)
- `n_estimators`
  - Default: 50
- `learning_rate`
  - Default: 1.0
  - Trade-off between `n_estimators` and `learning_rate`

AdaBoost regressor with scikit-learn

AdaBoostRegressor

```
from sklearn.ensemble import AdaBoostRegressor
```

```
reg_ada = AdaBoostRegressor(
    base_estimator,
    n_estimators,
    learning_rate,
    loss
)
```

Parameters

- `base_estimator`
  - Default: Decision Tree (max_depth=3)
- `loss`
  - linear (default)
  - square
  - exponential

Let's practice!

ENSEMBLE METHODS IN PYTHON

Gradient boosting

ENSEMBLE METHODS IN PYTHON

Román de las Heras
Data Scientist, SAP / Agile Solutions

Intro to gradient boosting machine

Objective: $y = f(X)$

1. Initial model (weak estimator): $y \sim f_1(X)$
2. New model fits to residuals: $y - f_1(X) \sim f_2(X)$
3. New **additive** model: $y \sim f_1(X) + f_2(X)$
4. Repeat n times or until error is small enough
5. Final **additive** model: $$y \sim f_1(X) + f_2(X) + \cdots + f_n(X) = \sum_{i=1}^{n} f_i(X)$$

Equivalence to gradient descent

$$\textbf{Residuals}: y - F_i(X)$$

Gradient Descent:

$$\textbf{Loss}: \frac{(F_i(X) - y)^2}{2}$$

$$\textbf{Gradient}: \frac{\partial Loss}{\partial F_i(X)} = F_i(X) - y$$

Residuals = Negative Gradient

$$y - F_i(X) = -\frac{\partial Loss}{\partial F_i(X)}$$

Gradient boosting classifier

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf_gbm = GradientBoostingClassifier(
   n_estimators=100,
   learning_rate=0.1,
   max_depth=3,
   min_samples_split,
   min_samples_leaf,
   max_features
)
```

- `n_estimators`
  - Default: 100
- `learning_rate`
  - Default: 0.1
- `max_depth`
  - Default: 3
- `min_samples_split`
- `min_samples_leaf`
- `max_features`

Gradient boosting regressor

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
reg_gbm = GradientBoostingRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    min_samples_split,
    min_samples_leaf,
    max_features
)
```

Time to boost!

ENSEMBLE METHODS IN PYTHON

Gradient boosting flavors

ENSEMBLE METHODS IN PYTHON

Román de las Heras
Data Scientist, SAP / Agile Solutions

Variations of gradient boosting

Gradient Boosting Algorithm

- Extreme Gradient Boosting
- Light Gradient Boosting Machine
- Categorical Boosting

Implementation

- XGBoost
- LightGBM
- CatBoost

Extreme gradient boosting (XGBoost)

Some properties:

- Optimized for distributed computing
- Parallel training by nature
- Scalable, portable, and accurate

```
import xgboost as xgb

clf_xgb = xgb.XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    random_state
)
```

```
clg_xgb.fit(X_train, y_train)
pred = clf_xgb.predict(X_test)
```

Light gradient boosting machine

Some properties:

- Released by Microsoft (2017)
- Faster training and more efficient
- Lighter in terms of space
- Optimized for parallel and GPU processing
- Useful for problems with big datasets and constraints of speed or memory

```
import lightgbm as lgb

clf_lgb = lgb.LGBMClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=-1,
    random_state
)
```

```
clf_lgb.fit(X_train, y_train)
pred = clf_lgb.predict(X_test)
```

Categorical boosting

Some properties:

- Open sourced by Yandex (April 2017)
- Built-in handling of categorical features
- Accurate and robust
- Fast and scalable
- User-friendly API

```
import catboost as cb

clf_cat = cb.CatBoostClassifier(
    n_estimators=1000,
    learning_rate=0.03,
    max_depth=6,
    random_state
)
```

```
clf_cat.fit(X_train, y_train)
pred = clf_cat.predict(X_test)
```

It's your turn!

ENSEMBLE METHODS IN PYTHON