1 Module 20: Ensemble Systems

Contents

- Bagging: Random trees
- Boosting: AdaBoost, Gradient-boosted trees

Individual models must be independent for the ensemble system to work.

$$\text{Variance[ensemble]} = \frac{\text{Variance[individual]}}{N}$$

1.1 Aggregating predictors

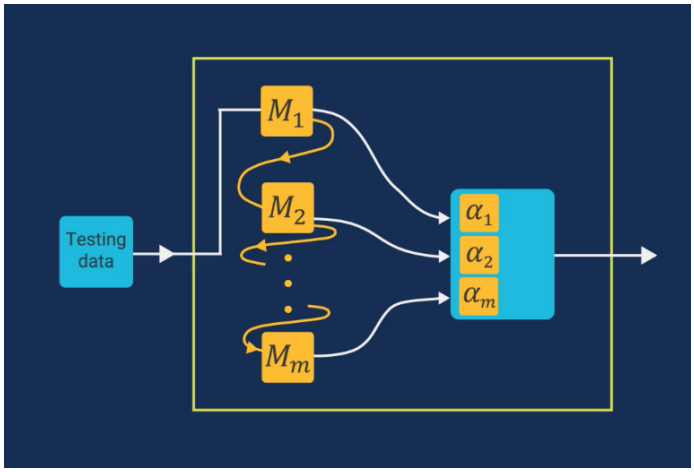
Metamodel = ensemble system

→ assign weights α_1,... ,α_m to models and fit their influences on final decision.

We find that using the same model in many different variations performs better than multiple different types of classifiers.

- Aggregation for classification
  - Hard voting: Majority vote
  - Soft voting: average the predicted class distributions and choose maximum
- "" for regression
  - Simple average

Boosting: predictors are trained sequentially

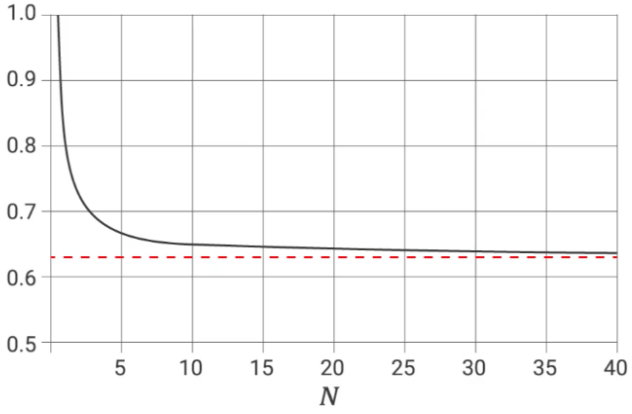


1.2 Bootstrapping & Bagging

High variance → too myopically focused on training data.

Bootstrapping (decorrelating models' data) Assume that training data samples from a population → sample with replacement (a sampled data point isn't removed from the dataset).

```
indices = np.random.choice( range(ntrain), ntrain )
X_bs = X_train[ind,:]
y_bs = y_train[ind,:]
```



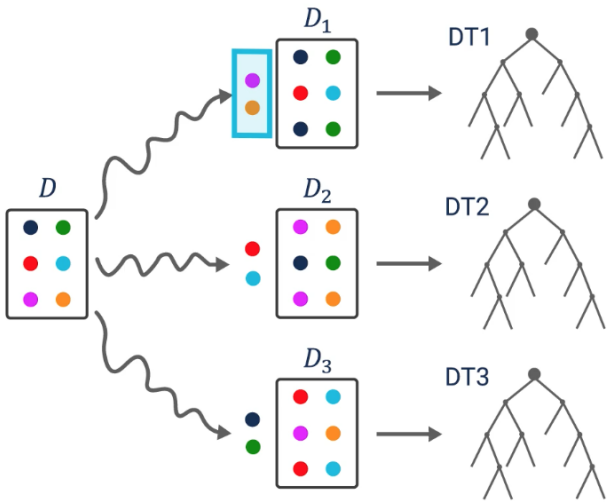
$$\lim_{N\to\infty}\left(1-\left(1-\frac{1}{N}\right)^{N}\right)=1-\frac{1}{e}\cong 0.632$$

We can expect in the lim of $\ll$ training data, we expect for each sample set to be 63% new data and 37% repeated data.

Bagging: "boostrap aggregation" Performed by our choice of aggregation technique.

```
y_ensemble = scipy.stats.mode(y_pred).mode[0]
```

Out-of-bag evaluation: withhold a subset of data per trained ensemble classifier, which is used as the testing set.




```
model = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators = 500, # number of models in ensemble
    oob_score = True # Out of bag score
)
model.fit(X,y)
model.oob_score_
```

1.3 Random Forests

Ordinarily, DTrees choose to split on features that produce the greatest change in entropy. We can simply require that the `max_features` > 1 such that we search along otherwise neglected branches. $\rightarrow$ increase diversity of ensemble

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier( n_estimators = 30, max_features = 10, oob_score =
↪  True )
model.fit( X,y )
model.oob_score_
```

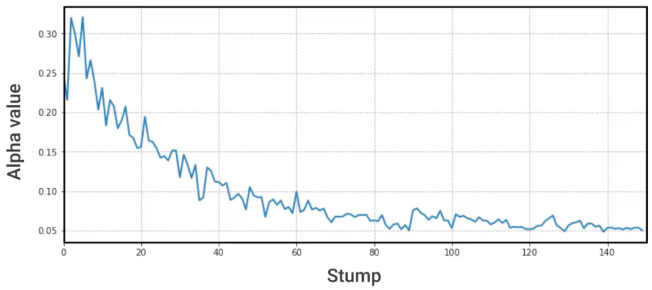
1.4 Boosting

Reduces bias of a set of weak learners (without increasing variance as much) → Combine a large number of weak models in a clever way to create a strong model.

Typically employ very shallow (one-node) decision trees (decision stumps) i.e. cut the dataset with a single slice.

Adaboost algorithm

- Initialize, s=0
  - for all samples i = 1, ..., N and stumps s: $w_s^i = \frac{1}{N}$
  - == the effort a stump should put into correctly classifying i
  - to begin with, all points are weighted equally
- Loop as log derived:
  - Create a stump to correctly classify samples for at least half of the total weight $S_s \cdot \text{fit}(x_1 y_1, W_S)$; if <0.5, just flip the inequality
  - misclassification score $\epsilon_S = \sum_{\text{misclassified}} W_S^i < 0.5$ as the system is at least better than random chance
  - influence coefficient $\alpha_S = \frac{1}{2}\log\frac{1-\epsilon_S}{\epsilon_S}$ use misclassification score to determine influence of particular stump on the ensemble
- Allow influence coefficient to update weights
  - $w_{S+1}^i = \left\{ \begin{array}{cc} w_S^i e^{\alpha_S} & i \text{ was misclassified} \\ w_S^i e^{-\alpha_S} & \text{otherwise} \end{array} \right\}$

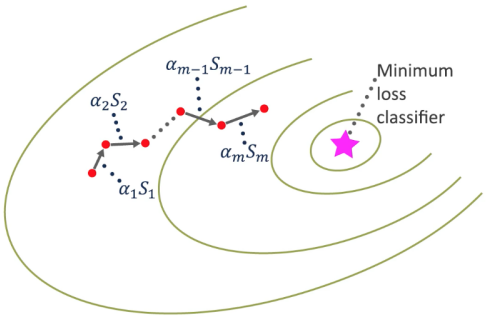


```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1))
model.fit(X,y)
```

Algorithm is based on **gradient boosting**. Boosting algorithms are not easily overfitted.

GRADIENT BOOSTING



Gradient descent:

$$X_n = \sum_{i=1}^{n-1} \gamma_i(-\nabla f(x_i))$$

Gradient boosting (AdaBoost):

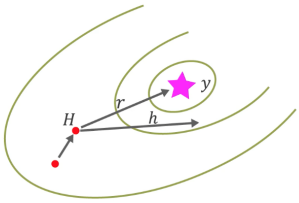
$$\text{Ensemble model} = \sum_{i=1}^{s} \alpha_s \left(\begin{matrix} weak \\ learner \end{matrix}\right)$$

Gradient Boosting Trees Use trees for base model BUT: use squared loss as loss fcn.

- H = 0
- Loop as log
  - $r_i = y_i - H(X_i)$
  - h = Shallow regression tree (depth ≈ 4) trained on r $\rightarrow h(x_i) \approx r(x_i)$
  - H $\leftarrow$ H + αh

- $H = 0$
- Loop as log as desired
  - $r_i = y_i - H(x_i)$
  - h = Shallow regression tree trained on r
  - $H \leftarrow H + \alpha h$

$h(x_i) \cong r(x_i)$




```
from sklearn.ensemble import GradientBoostingRegressor
GradientBoostingRegressor(n_estimators = 1000, max_depth = 1)
```