Improving weak learners

Artificial Intelligence and Machine Learning for SupTech – Lecture 5

Iman van Lelyveld – Michiel Nijhuis

VU Amsterdam

Improving weak learners

1. How to grow a decision tree? How to split?
   - Decision trees, purity measures
2. Can Ensemble Classifiers improve weak learners?
   - Bagging, boosting, AdaBoost, XGBoost
3. Can we use forests in other ways?
   - Isolation forests

Outline

Improving weak learners

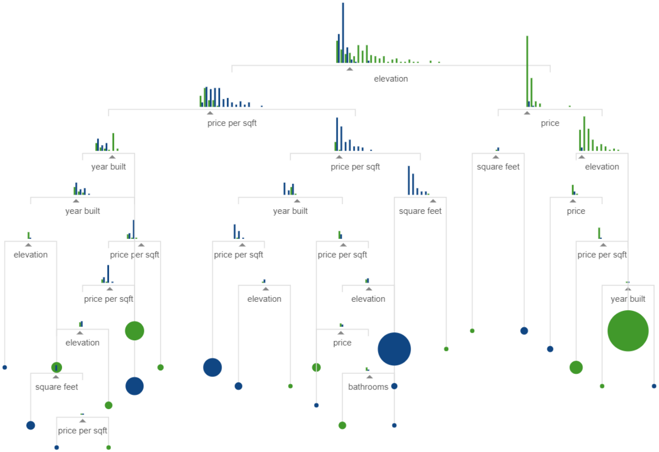
- Trees
- Forests – Ensemble Classifiers
- Bootstrapping aka Bagging
- Boosting
- Isolation forest

Knowledge clips

- Decision Tree Classifier from Scratch (Josh Gordon) (link)
- ADAboost clearly explained (Josh Starmer) (link)
- XGBoost from Start to Finish (Josh Starmer) (link)

  An extensive webinar where the XGBoost discussion starts at 36.32. The intro is nice to see how to handle data (labeling, missing values, ...)

Decision tree



Source: r2d3

Decision trees

- A decision tree is a learning algorithm that constructs a set of decisions based on training data.
- Decision trees are popular because:
  - They are naturally non-linear, so you can use them to solve complex problems
  - They are easy to visualise
  - How they work is easily explained
  - They can be used for regression (predict a number) and classification (predict a class)
- Drawback: classification at each step only use ’local’ information

How to get maximum information gain at each junction?

- At each split we create as much clarity (ie information) as possible. The split should make classifying the resulting 'children' easier. Formally (for two-way splits):

$$\mathit{InformationGain}(D_{\mathbf{par}\mathit{ent}}, \mathbf{f}\mathit{eature}) = I(D_{par}) - \frac{N_{left}}{N_{par}} I(D_{left}) - \frac{N_{right}}{N_{par}} I(D_{right})$$

- What could be the **Impurity** measure?

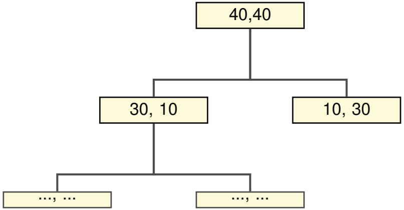
Gini	$I_{\mathbf{Gini}}$	$= 1 - \sum_{i=1}^{c} p(i\|t)^2$
Entropy	$I_{\mathbf{ent}\mathit{ropy}}$	$= - \sum_{i=1}^{c} p(i\|t) log_2 p(i\|t)$
Classification error	$I_{\mathbf{clas}\mathit{sification}}$	$= 1 - max\{p(i\|t)\}$

where $p(i|t)$ is the probability that you get it right in 'child' t

- $I_{\mathbf{Gini}}$ and $I_{\mathbf{ent}\mathit{ropy}}$ are minimal if homogeneous and maximal if perfectly mixed

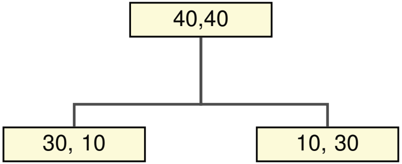
Splitting with Classification Error: $1 - max\{p(i|t)\}$

Split scenario A (class 0/1)



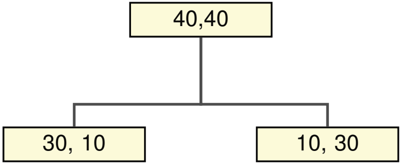
Splitting with Classification Error: $1 - max\{p(i|t)\}$

Split scenario A (class 0/1)



Splitting with Classification Error: $1 - max\{p(i|t)\}$

Split scenario A (class 0/1)



40,40

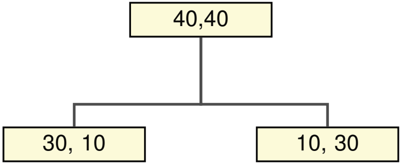
30, 10

10, 30

$$
\begin{aligned}
I_{\mathbf{clas}}(D_{par}) &= 1 - 0.5 && = 0.5 \\
I_{\mathbf{clas}}(D_{left}) &= 1 - \tfrac{30}{40} && = 0.25 \\
I_{\mathbf{clas}}(D_{right}) &= 1 - \tfrac{30}{40} && = 0.25 \\
IG_{\mathbf{clas}} &= 0.5 - \tfrac{40}{80}0.25 - \tfrac{40}{80}0.25 && = 0.25
\end{aligned}
$$

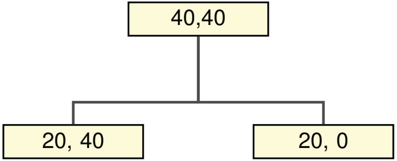
Splitting with Classification Error: $1 - max\{p(i|t)\}$

Split scenario A (class 0/1)



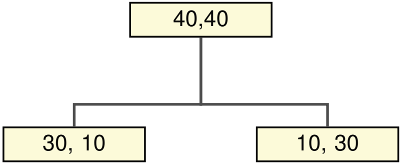
$$\begin{aligned}
I_{\mathbf{clas}}(D_{par}) &= 1 - 0.5 &&= 0.5\\
I_{\mathbf{clas}}(D_{left}) &= 1 - \tfrac{30}{40} &&= 0.25\\
I_{\mathbf{clas}}(D_{right}) &= 1 - \tfrac{30}{40} &&= 0.25\\
IG_{\mathbf{clas}} &= 0.5 - \tfrac{40}{80}0.25 - \tfrac{40}{80}0.25 &&= 0.25
\end{aligned}$$

Split scenario B



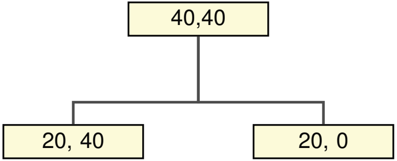
Splitting with Classification Error: $1 - max\{p(i|t)\}$

Split scenario A (class 0/1)



$$\begin{aligned}
I_{\textbf{clas}}(D_{par}) &= 1 - 0.5 && = 0.5 \\
I_{\textbf{clas}}(D_{left}) &= 1 - \tfrac{30}{40} && = 0.25 \\
I_{\textbf{clas}}(D_{right}) &= 1 - \tfrac{30}{40} && = 0.25 \\
IG_{\textbf{clas}} &= 0.5 - \tfrac{40}{80} 0.25 - \tfrac{40}{80} 0.25 && = 0.25
\end{aligned}$$

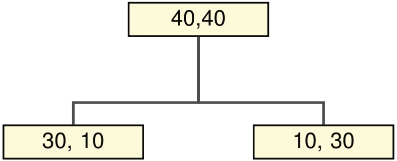
Split scenario B



$$\begin{aligned}
I_{\textbf{clas}}(D_{left}) &= 1 - \tfrac{40}{60} && = \tfrac{20}{60} \\
I_{\textbf{clas}}(D_{right}) &= 1 - \tfrac{20}{20} && = 0 \\
IG_{\textbf{clas}} &= 0.5 - \tfrac{60}{80} \tfrac{20}{60} - 0 && = 0.25
\end{aligned}$$

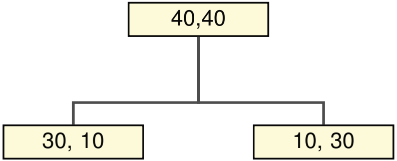
Splitting with Gini: $1 - \sum_{i=1}^{c} p(i|t)^2$

Split scenario A (class 0/1)



Splitting with Gini: $1 - \sum_{i=1}^{c} p(i|t)^2$

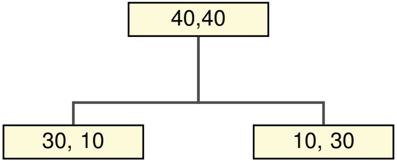
Split scenario A (class 0/1)



$$
\begin{aligned}
I_{\textbf{Gini}}(D_{par}) &= 1 \text{ - } (0.5^2 + 0.5^2) && = 0.5 \\
I_{\textbf{Gini}}(D_{left}) &= 1 \text{ - } (\tfrac{10}{40}^2 + \tfrac{30}{40}^2) && = \tfrac{30}{80} \\
I_{\textbf{Gini}}(D_{right}) &= 1 \text{ - } (\tfrac{30}{40}^2 + \tfrac{10}{40}^2) && = \tfrac{30}{80} \\
IG_{\textbf{Gini}} &= 0.5 \text{ - } \tfrac{40}{80}\tfrac{30}{80} \text{ - } \tfrac{40}{80}\tfrac{30}{80} && = 0.125
\end{aligned}
$$

Splitting with Gini: $1 - \sum_{i=1}^{c} p(i|t)^2$

Split scenario A (class 0/1)



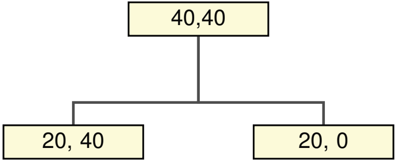
40,40

30, 10

10, 30

$$
\begin{aligned}
I_{\textbf{Gini}}(D_{par}) &= 1 - (0.5^2 + 0.5^2) && = 0.5 \\
I_{\textbf{Gini}}(D_{left}) &= 1 - ({\tfrac{10}{40}}^2 + {\tfrac{30}{40}}^2) && = \tfrac{30}{80} \\
I_{\textbf{Gini}}(D_{right}) &= 1 - ({\tfrac{30}{40}}^2 + {\tfrac{10}{40}}^2) && = \tfrac{30}{80} \\
IG_{\textbf{Gini}} &= 0.5 - \tfrac{40}{80}\tfrac{30}{80} - \tfrac{40}{80}\tfrac{30}{80} && = 0.125
\end{aligned}
$$

Split scenario B



40,40

20, 40

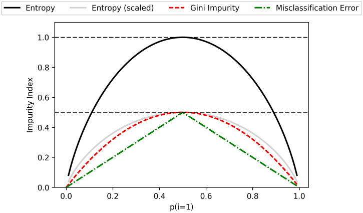
20, 0

$$
\begin{aligned}
I_{\textbf{Gini}}(D_{left}) &= 1 - ({\tfrac{20}{60}}^2 + {\tfrac{40}{60}}^2) && = \tfrac{4}{9} \\
I_{\textbf{Gini}}(D_{right}) &= 1 - (1^2 - 0) && = 0 \\
IG_{\textbf{Gini}} &= 0.5 - \tfrac{60}{80}\tfrac{4}{9} - 0 && = 0.16
\end{aligned}
$$

Information gain – wrap up

- The Entropy criterion also prefers Scenario B over A
- Differences between the I criteria are small

  Gini often faster but tends to put most frequent class in 1 branch
- Choice of stopping criteria much more important
  - e.g. until no leaf contains more than x observations
- Overfitting can be reduced by Cost complexity pruning: adding a cost to leafy tree. Not implemented in SKLearn library.
- Alternatives:
  - Set maximum depth of tree
  - Set maximum number of leaf nodes
  - Set minimum number of samples per split



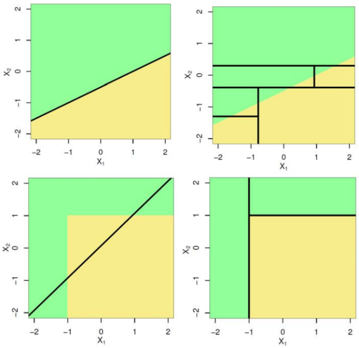
Is a linear classifier better than a tree?

Case 1: linear Data Generating Process (DGP)

- A linear regression (left) is a better fit than a tree (right).

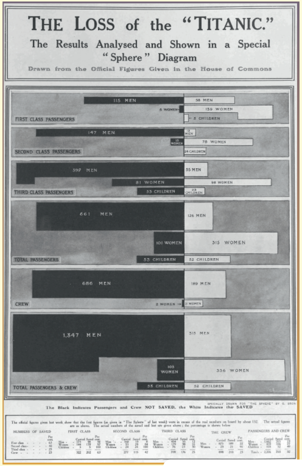
Case 2: non-linear DGP

- A tree (right) is a better fit than a linear regression (left).



Exploring classification with Titanic data

- Following the sinking of the Titanic there was an inquest that documented many features on the passengers
- G.Bron's chart of "The Loss of the 'Titanic"', from The Sphere, 4 May 1912, is the first attempt to show proportions of features (Friendly et al. (2019))
- The data has led to numerous ML flavoured analyses (See here, here,here, and here). These are valuable resources to get to grips with classification trees
- A nice example using temperature data is here
- See Jake van der Plas's Notebook which can be found here



Outline

Improving weak learners

- Trees
- Forests – Ensemble Classifiers
- Bootstrapping aka Bagging
- Boosting
- Isolation forest

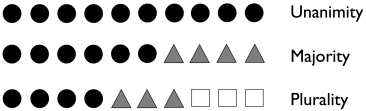
Learning with ensembles

... the interests of truth require a diversity of opinions. *(Mill (1859))*

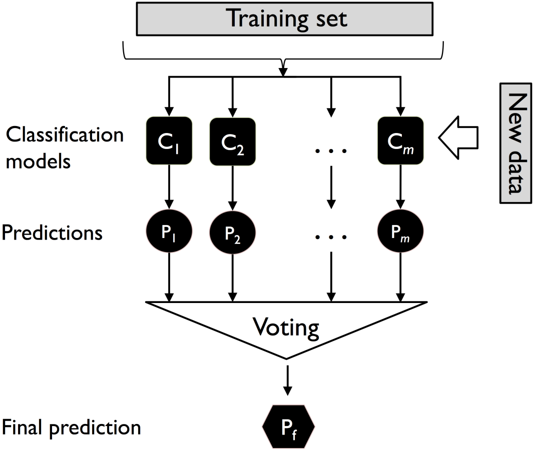
- What if we can learn from the wisdom of crowds? That is, can we combine multiple classifiers?
- In a sense we get a "mixture of experts"
- Predictions more accurate and robust
- Simplest approach: majority voting
- We will discuss the intuition of why this might work

Majority voting

- Majority voting refers to binary setting but can easily generalize to multi-class (e.g. plurality voting)
- Select class label that receives the most votes (mode)
- Train m classifiers $C_1, \ldots, C_m$
- Options:
  1. Build ensemble using different classification algorithms (e.g. SVM, logistic regression, etc.) but same data
  2. Use the same algorithm but fit different subsets of the training set (e.g. random forest)



General approach



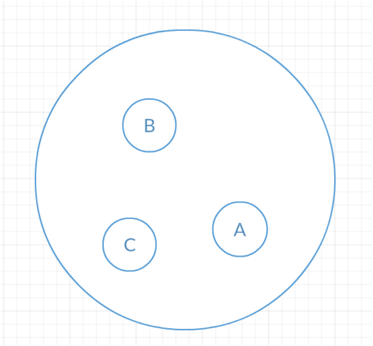
Combining predictions via majority voting

We have predictions of individual classifiers C_j and need to select the final class label $\hat{y}$

$$\hat{y} = mode\{C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_m(\mathbf{x})\}$$

For example, in a binary classification task where $class_1 = -1$ and $class_2 = +1$, we can write the majority vote prediction as follows:

$$C(\mathbf{x}) = sign\left[\sum_j^m C_j(\mathbf{x})\right] = \begin{cases} 1 & \text{if } \sum_j C_j(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



Rohith Ghandi

Intuition why ensembles can work better

- If errors ϵ are more or less independent a combination will reduce the variance
- Assume that all n base classifiers have the same error rate ϵ
- The probability of an error of an ensemble can be expressed as a probability mass function of a binomial distribution:

$$P(y \geq k) = \sum_{k}^{n} \binom{n}{k} \epsilon^{k} (1 - \epsilon)^{n-k} = \epsilon_{\text{ensemble}}$$

- Here, $\binom{n}{k}$ is the binomial coefficient and the distribution computes what the probability is of k successes in n trials. In other words, we compute the probability that the prediction of the ensemble is wrong.

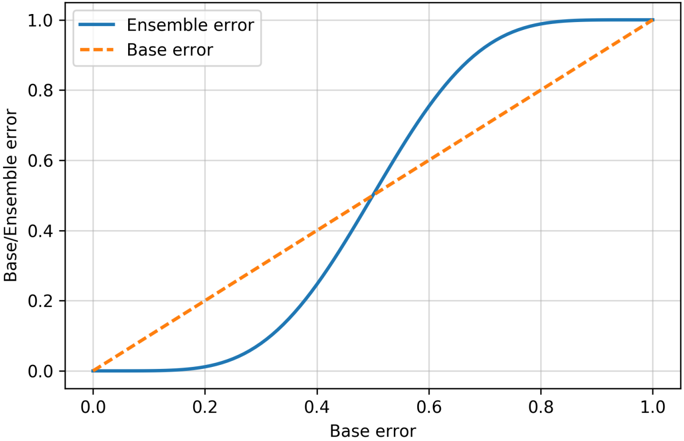
Example

Imagine we have 11 base classifiers ($n = 11$) with individual error rates ϵ of 0.25 and:

$$P(y \geq k) = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1 - 0.25)^{11-k} = 0.034$$

So the error rate of the ensemble of $n = 11$ classifiers (0.034) is much lower than the error rate of the individual classifiers (0.25).

Same reasoning applied to a wider range of error rates



Voting classifier in scikit-learn

- Simply instantiate several classifiers and make a list
- Pass to sklearn.ensemble.VotingClassifier(...)

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)]
ens_clf = VotingClassifier(estimators)
ens_clf = eclf1.fit(X, y)
```

- If all classifiers can estimate class probabilities, then using soft voting might help. In contrast with hard voting, confident estimators are given more weight

▸ sklearn API link

Outline

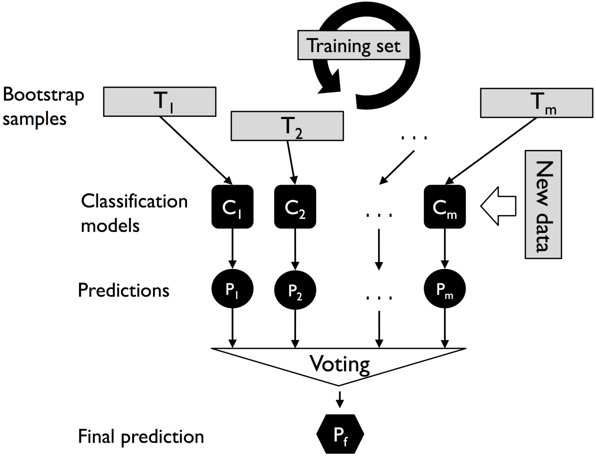
Improving weak learners

- Trees
- Forests – Ensemble Classifiers
- Bootstrapping aka Bagging
- Boosting
- Isolation forest

Bootstrap aggregation (bagging)

- We used the entire training set for the majority vote classifier
- Here we draw random bootstrap samples
  - In statistics, bootstrapping is any test or metric that relies on random sampling with replacement
  - bootstrapping often used as an alternative to statistical inference based on the assumption of a parametric model when that assumption is in doubt
  - Alternatively we can sample without replacement: pasting
- The basic idea of bootstrapping is that inference about a population from sample data, can be modeled by resampling with replacement the sample data and performing inference about a sample from resampled data

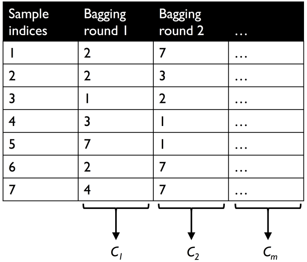
Bagging



Training set
Bootstrap samples
T_1
T_2
. . .
T_m
Classification models
C_1
C_2
. . .
C_m
New data
Predictions
P_1
P_2
. . .
P_m
Voting
Final prediction
P_f

Bagging example

- Seven training examples
- Sample randomly with replacement
- Use each bootstrap sample to train a classifier C_j
- C_j is typically a decision tree
- Random Forests: also use random feature subsets
  - Instantiate a decision tree classifier
  - Make a bagging classifier with decision trees
  - Check that the accuracy is higher for the bagging classifier
- Asses model on out-of-bag (oob) observations (`oob_score=True`).



Sample indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…

▸ iPython notebook on github

Outline

Improving weak learners

- Trees
- Forests – Ensemble Classifiers
- Bootstrapping aka Bagging
- Boosting
- Isolation forest

Boosting

- Basic idea: start with weak learners that have only a slight performance advantage over random guessing (e.g. a decision tree ’stump’) and try to boost their performance by focusing on training samples that are hard to classify
- Very simple base classifiers learn from misclassified training examples
- The original boosting algorithm was formulated by Robert Schapire in 1990
- It was later refined into AdaBoost – short for Adaptive Boosting
- AdaBoost used to be the common implementation of boosting. Lately XGBoost is becoming very popular.

Original boosting algorithm

1. Draw a random subset of training samples d_1 without replacement from the training set D to train a weak learner C_1
2. Draw second random training subset d_2 without replacement from the training set and add 50 percent of the samples that were previously misclassified to train a weak learner C_2
3. Find the training samples d_3 in the training set D on which C_1 and C_2 disagree to train a third weak learner C_3
4. Combine the weak learners C_1, C_2, and C_3 via majority voting

AdaBoost

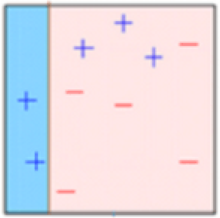
- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

slide at the end

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

Learner D1

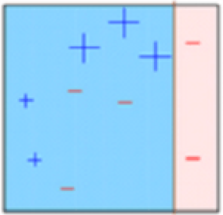


slide at the end

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

Learner D2

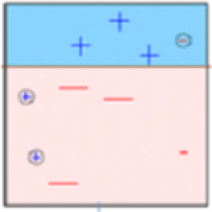


slide at the end

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

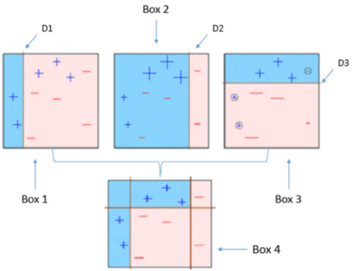
Learner D3



slide at the end

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

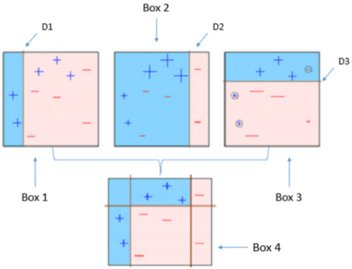


slide at the end

AdaBoost

- In contrast, AdaBoost uses the complete training set to train the weak learners
- Training samples are reweighted in each iteration to build a strong classifier
- End goal is to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble

```
# AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifierclf = AdaBoostClassifier()
# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)
clf.fit(x_train,y_train)
clf.predict(x_test)
```



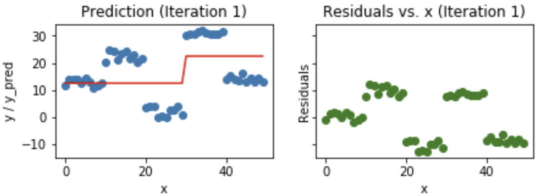
Gradient boosting

- View boosting problem as an optimisation problem, i.e we take up a loss function and try to optimise it
- New weak learners are added to concentrate on the areas where the existing learners are doing poorly

Gradient boosting

- View boosting problem as an optimisation problem, i.e we take up a loss function and try to optimise it
- New weak learners are added to concentrate on the areas where the existing learners are doing poorly

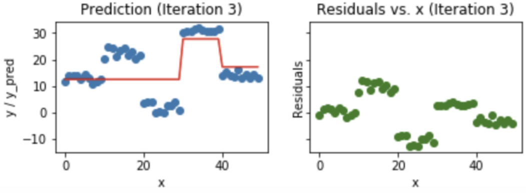
Iteration 1



Gradient boosting

- View boosting problem as an optimisation problem, i.e we take up a loss function and try to optimise it
- New weak learners are added to concentrate on the areas where the existing learners are doing poorly

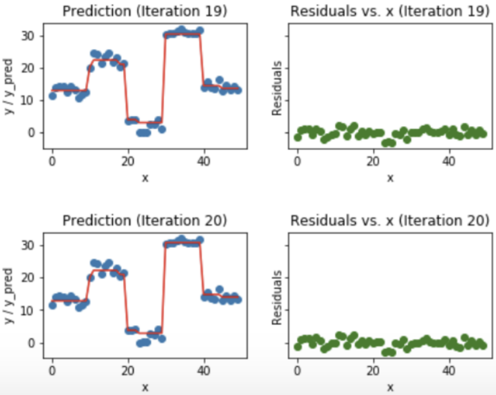
Iteration 2



Gradient boosting

- View boosting problem as an optimisation problem, i.e we take up a loss function and try to optimise it
- New weak learners are added to concentrate on the areas where the existing learners are doing poorly

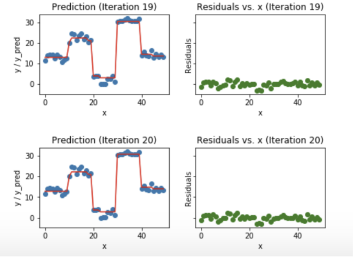
... last round



Gradient boosting

- View boosting problem as an optimisation problem, i.e we take up a loss function and try to optimise it
- New weak learners are added to concentrate on the areas where the existing learners are performing poorly

```
# Gradient Boosting
from sklearn.ensemble import
    GradientBoostingClassifier
clf = GradientBoostingClassifier()
# n_estimators = 100 (default)
# loss function = deviance(default) used
    in Logistic Regression
clf.fit(x_train,y_train)
clf.predict(x_test)
```



Best kid on the block: XGBoost

- XGBoost is similar to gradient boosting algorithm but it has a few smart features
  - Clever penalisation of Trees
  - A proportional shrinking of leaf nodes
  - Newton Boosting
  - Extra randomisation parameter to reduce correlation between learners

```python
# XGBoost
from xgboost import XGBClassifier
clf = XGBClassifier()
# n_estimators = 100 (default)
# max_depth = 3 (default)
clf.fit(x_train,y_train)
clf.predict(x_test)
```

- See the in-depth Webinar by Josh Starmer for a walk through. XGBoost starts at 36.32 but the intro is nice to see how to handle data.

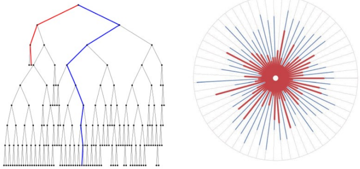
Outline

Improving weak learners

- Trees
- Forests – Ensemble Classifiers
- Bootstrapping aka Bagging
- Boosting
- Isolation forest

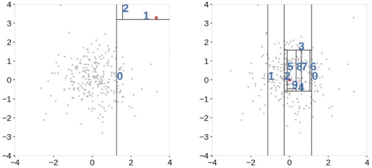
How does an Isolation Forest work?

- ‘Isolates’ observations by randomly selecting a feature and then randomly selecting a split value (min. $<$ split $<$ max.)



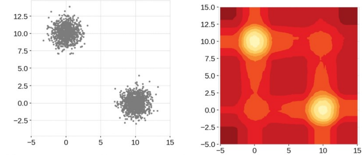
How does an Isolation Forest work?

- ‘Isolates’ observations by randomly selecting a feature and then randomly selecting a split value (min. < split < max.)
- Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node
- Path length (averaged over a forest of random trees) is a measure of normality and our decision function. Shorter path lengths for particular samples, they are likely to be anomalies



How does an Isolation Forest work?

- ‘Isolates’ observations by randomly selecting a feature and then randomly selecting a split value (min. $<$ split $<$ max.)
- Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node
- Path length (averaged over a forest of random trees) is a measure of normality and our decision function. Shorter path lengths for particular samples, they are likely to be anomalies
- Can be expressed as an ’outlier score’

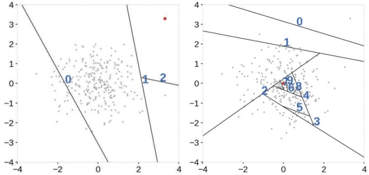


(a) Two normally distributed clusters

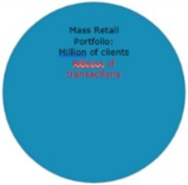
(b) Anomaly Score Map

Extending the Isolation Forest

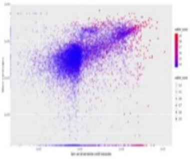
- Extended isolation forest selects random intercept and random slope (**haririExtendedIsolationForest2021**)
- Needs to store less information → faster, especially with many dimensions



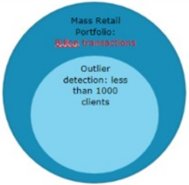
How DNB applies the outlier detection model for KYC



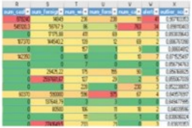
1
Large volume of client and transaction data received from superverised bank



2
Data parsed by algorithm and this provides list of outliers

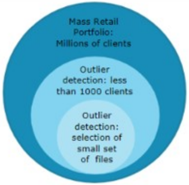


3
First selection



R	S	T	U	V	W	X
num_cas	num_foreig	num_wi	num_forei	num_ca	alert	outlier_sc
970240	14049	238	238	11	41	[illegible]
549320.3	56767.9	86	9	763	34	[illegible]
0	7775.88	48	69	17	3	[illegible]
107370	146402	139	12	69	2	[illegible]
0	0	157	1	3	0	[illegible]
162390	0	10	0	10	7	[illegible]
0	0	7	0	0	0	[illegible]
0	29425.22	175	155	90	0	[illegible]
0	2597607.67	127	29	2	5	[illegible]
0	0	229	0	230	3	[illegible]
60370	590000	596	975	67	4	[illegible]
0	137648.79	3	3	0	0	[illegible]
0	10560	106	11	18	2	[illegible]
0	0	11	5	1	0	[illegible]
0	29496495	733	71	0	3	[illegible]

4
Output is a list of alle dossiers above a particular outlier score



5
A selected few client files are checked

Summary

In this lecture we covered:

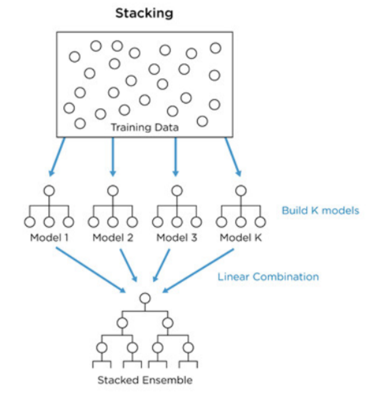
1. How to define a tree, specifically how to split at each node
2. How to combine classifiers: ensemble learning a.k.a. “forests”
3. How ensembles can combine weak learners to come to much stronger learners. In particular AdaBoost and XGBoost
4. How we can use isolation forests for outlier detection

AdaBoost algorithm

1. Set weight vector $\mathbf{w}$ to uniform weights where $\sum_i w_i = 1$
2. For j in m boosting rounds, do the following:
    1. Train a weighted weak learner $C_j = \textit{train}(\mathbf{X}, \mathbf{y}, \mathbf{w})$
    2. Predict class labels $\hat{y} = \textit{predict}(C_j, \mathbf{X})$
    3. Compute the weighted error rate $\epsilon = \mathbf{w} \cdot (\hat{\mathbf{y}} \neq \mathbf{y})$
    4. Compute the coefficient α_j $\alpha_j = 0.5 \log \frac{1-\epsilon}{\epsilon}$
    5. Update the weights $\mathbf{w} := \mathbf{w} \times \exp\big(-\alpha_j \times \hat{\mathbf{y}} \times \mathbf{y}\big)$
    6. Normalize weights to sum to 1 $\mathbf{w} := \mathbf{w} / \sum_i w_i$
3. Compute the final prediction: $\hat{\mathbf{y}} = \big(\sum_{j=1}^{m} \big(\alpha_j \times \textit{predict}(C_j, \mathbf{X})\big) > 0\big)$

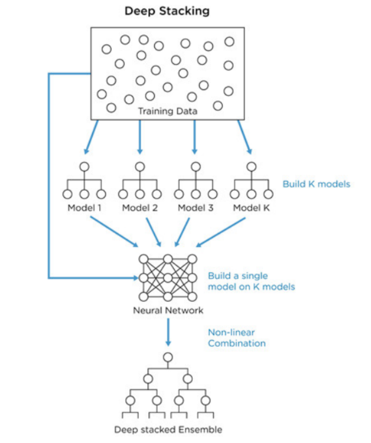
Advanced topic: Stacking and deep stacking

- Stacking combines the outputs from multiple base models into a single score. The base-level models are trained based on a complete dataset, and then their outputs are used as input features to train an ensemble function
- Usually, the ensemble function is a simple linear combination of the base model scores.



Advanced topic: Stacking and deep stacking

- Stacking combines the outputs from multiple base models into a single score. The base-level models are trained based on a complete dataset, and then their outputs are used as input features to train an ensemble function
- Usually, the ensemble function is a simple linear combination of the base model scores.
- Deep stacking feeds the ensemble outcome(s) into a neural net + the original data



References I

- Friendly, M., Symanzik, J., & Onder, O. (2019). Visualising the Titanic disaster. *Significance*, *16*(1), 14–19.
- Mill, J. S. (1859). *On Liberty*. Cambridge University Press.