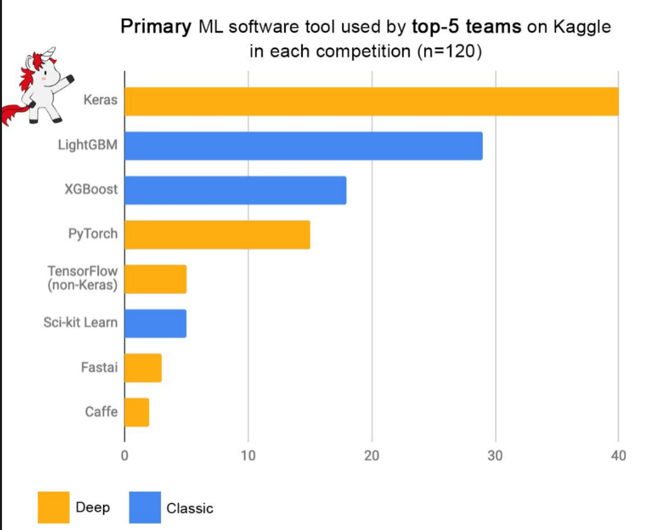
Ensemble learning

Daniel Lawson University of Bristol

Lecture 05.2 (v2.0.1)

https://www.tcw.com/Insights/2023/2023-06-23-Agency-MBS-Fe



What wins at Kaggle? From a 2019 Twitter post

Signposting

- In 05.1 we introduced key baseline classification methods
- Other core methods based on trees and forests and in Block 06.
- This lecture is not about specific methods, but how they can be combined using **ensemble learning** (also called **meta-methods**).

Questions

- If we have two good models, can we combine them into a better model?
- If we have many bad models, can they be combined into a good model?
- Can we apply cross-validation and other tricks to improve performance?

Theory and Practice

- All of the practical work is done in the Workshop.
- **Activity 1** of the Workshop is about making a standardized interface for classifiers.
- This is an essential first step for using classifiers as building blocks for more complex operations.
- It tends to be where the Python Package SciKit Learn shines over R, though there are still inconsistencies

Broad approaches to combining learners

- The goal is “Meta Learning”, i.e. combining multiple “learners”
  - learner = machine learning algorithms that consume data and make predictions
  - generalisation of a classifier
- There are two main approaches:
- **Parallel** approaches:
  - Run independently
  - Exploits independence structure between learners
- **Sequential** approaches:
  - Run dependently
  - Exploits dependence structure between learners
  - i.e. focus on what the previous set of learners are bad at

Core topics

- Bagging
- Boosting
- Stacking

Bagging

- The algorithm for bagging is straightforward - simply taking the average of bootstrapped learners
- In parallel, B times:
  - Form data sample $\{X\}_b$, e.g. by sampling with replacement, or leaving out a random subset of data
  - Learn a classifier $f_b(x)$
- Output: A "bagged" classifier $f(x) = \frac{1}{B} \sum_{b=1} f_m(x)$

Bagging comments

- Bagging reduces overfitting to the data, and therefore works well on complex classifiers
- Same rules for resampling apply as in statistics: e.g. it works well when you respect the correlation structure
- In theory under certain assumptions, the **distribution of bagged learners** give a distribution on:
  - "what I could have seen if I obtained new data"
  - From the same distribution I got my data
- Usually little reason not to try it in practice

Boosting

- The general idea of **Boosting** is:
  - Build a classifier, predict the data
  - Treat the residuals as "new data"
  - Repeat
- Boosting sounds like it should work for arbitrary classifiers, but because of the iterative nature it is applied to simple classifiers.
- There are many boosting algorithms, amongst which are:
  - Majority vote (Early and weak)[1]
  - Adaboost[2] (Adaptive boosting - first game-changer)
  - xgboost[3] (exploits sparsity and gradients)
  - LightGBM[4] (parallelizable and efficient)

---

[1] Kearns M and Valiant L (1989). Symp. Theor. Comp. ACM. 21: 433-444

[2] Freund and Schapire in 1996

[3] Chen T and Carlos G (2016) KDD 2016.

[4] Microsoft dev team

Boosted feature splitter

- A very simple way to use boosting is to allow classifiers only from **single features**:
- Initialise weights of each **data sample** (uniformly)
- For T iterations:
  - Normalise weights
  - Train a classifier on every feature individually
  - Choose the best classifier, i.e. feature
  - Update the data weights by upweighting correct decisions and downweighting wrong decisions
- The boosted classifier uses a weighted sum of the selected classifiers

Adaboost

- **Given**: N data $(x_1, y_1), \dots, (x_N, y_N); x_i \in \mathcal{X}, y_i \in \{-1, 1\}$
- Set data weights $D_{t=1}(n) = 1/N$. For $t = 1 \cdots T$:
  - **Train M "weak" classifiers** $h_{mt}(x_t) : \mathcal{X} \to \{-1, 1\} \in \mathcal{H}$
  - With weighted **prediction error** $\epsilon_{mt} = \sum_{i=1}^{N} D_t(i)(h_{mt}(x_i) - y_i)/2$
  - Choose the best classifier $h_t = \mathrm{argmin}_m \epsilon_{mt}(h_{mt})$ with error ϵ_t
  - Evaluate $\alpha_t = \log([1 - \epsilon_t]/\epsilon_t)$
  - Update the **weights**:

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(\alpha_t \mathcal{I}(y_i \neq h_t(x_i))\right)}{Z_t}$$

  - Where Z_t re-normalises weights D_{t+1} to sum to 1.
- **Output**: Boosted classifier:

$$H(x) = \mathrm{sign}\left(\sum_{m=1}^{M} \theta_m h_m(x)\right)$$

Boosting comments

- α grows (towards infinity) as ϵ shrinks (towards zero)
- The weighting process is chosen to ensure that the sign operation ensures correct classification
- Boosting is **computed** as a "decision tree" describing which classifier to use
- But outputs a mixture solution!
- All information about previous decisions is encoded into the weights
- When there is no residual error left for a data point, its weight is set to zero
- h can be thought of as "features" and $\mathcal{H} = \{h(x)\}$ can be large or infinite.
- Implementations in practice usually restrict weak classifiers h to a single, simple class (e.g. decision tree, perceptron)
- Weak classifiers are often generated by subsetting features, e.g. one at a time

Stacking

- Stacking is a different way to combine multiple weak learners. It is more appropriate to combining “good classifiers” to make a meta-classifier.
- In theory a stacked classifier will always outperform its constituents if implemented appropriately[5].
  - Cross-validation and asymptotics are required for this guarantee but in practice many approaches work.

[5] van der Laan, M, Polley E, Hubbard A, “Super Learner” (2007) Statistical Applications in Genetics and Molecular Biology, Volume 6.

Super learner

- **Set up** the ensemble:
  - Specify L base classifiers.
  - Specify a metalearning algorithm.
- **Train** the ensemble:
  - Train the L base algorithms on the N training data. Use k-fold cross-validation for these learners.
  - For the $N \times L$ matrix of predictions. Form the "level one" data with this matrix and the raw data.
  - Train the metalearning algorithm on the "level one" data.
- **Predict** on new data:
  - Generate predictions from the base classifiers.
  - Feed those predictions into the meta-learner to generate the ensemble prediction.

More Stacking

- Related approaches:
  - Run any number of classification algorithms
  - Use their predictions as features
  - Use the data in addition to the predictions
- Pass this new feature set to any classification algorithm
- In practice, the best algorithm will be the one that generalises best in the test dataset. Common techniques:
  - Majority vote: use the prediction that most classifiers choose
  - Regularisation
  - Boosting-like prediction combination

Wrapup

- Key to high prediction accuracy are:
  - **Complexity**: Non-linearity helps dramatically
  - **Bias control**: Don't overfit
  - **Meta-learning**: Boosting and stacking are essential for the final few percent.

Signposting

- Next Block: Random Forests and decision trees and more practice using classification.
- Next Lecture: The workshop Lecture going over Bagging, Stacking and Boosting in practice.
- **References**:
- Ensemble learning in general:
  - Vadim Smolyakov, MIT: ML-perspective on Ensemble Methods
  - Stacked Ensembles by H2O, a Commercial AI Company focussing on Deployable AI
  - StackExchange: Stacking vs Bagging vs Boosting
  - Super Learners: van der Laan, M, Polley E, Hubbard A, "Super Learner" (2007) Statistical Applications in Genetics and Molecular Biology, Volume 6.

Signposting (2)

- **More References**:
- Boosting:
  - AdaBoost paper: Experiments with a New Boosting Algorithm Freund and Schapire (1996).
  - Explaining AdaBoost, Rob Schapire, Empirical Inference (2013) pp 37-52.
  - xgboost Chen T and Carlos G (2016) KDD 2016.
  - xgboost explained, a blog post about Didrik Nilsen's paper Tree Boosting With XGBoost: Why Does XGBoost Win "Every" Machine Learning Competition?