COMPSCI762: Introduction to Machine Learning

Ensembles

Jörg Simon Wicker

The University of Auckland

SCIENCE
SCHOOL OF COMPUTER SCIENCE
MACHINE LEARNING

Admin

- Any questions on the assignment?
- Plan for the remaining weeks

Week	Topic
Week 8	Instance-based Learning & SVMs
Week 9	**Ensembles**
Week 10	Clustering
Week 11	Association Rules
Week 12	Outlier Detection

Motivation

Motivation

This week we will cover..

Motivation

Ensembles

- Averaging
- Random Forests
- AdaBoost
- XGBoost

Partially based on Slides from University of British Columbia

Ensembles

Ensembles

- Ensembles are classifiers that have classifiers as input
- With great names:
  - Averaging
  - Boosting
  - Bootstrapping
  - Bagging
  - Cascading
  - Random Forests
  - Stacking
- Ensemble methods often have higher accuracy than the input classifiers
- How would you build an ensemble classifier?

Ensembles

- Remember the fundamental trade-off
  1. E_{train}: How small you can make the training error
     vs.
  2. E_{approx}: How well training error approximates the test error
- Goal of ensemble methods is that the ensemble classifier
  - Does much better on one of these than the individual classifiers
  - Doesn't do too much worse on the other
- This suggests two types of ensemble methods
  1. **Boosting**: Improves training error of classifiers with high E_{train}
  2. **Averaging**: Improves approximation error of classifiers with high E_{approx}

Averaging

Averaging

- Input to averaging is the predictions of a set of models, for example
  - Decision trees make one prediction
  - Naïve Bayes makes another prediction
  - KNN makes another prediction
- Simple model averaging:
  - Take the mode of the predictions (or average probabilities if probabilistic)

$$
\begin{tikzcd}
 & \textit{decistion tree} \arrow[r] & \texttt{not spam} \arrow[dr] & \\
x_i \arrow[ur] \arrow[r] \arrow[dr] & \textit{naïve Bayes} \arrow[r] & \texttt{spam} \arrow[r] & \texttt{spam} \\
 & \textit{k-nearest neighbour} \arrow[r] & \texttt{spam} \arrow[ur] &
\end{tikzcd}
$$

Stacking

- A common variation of averaging is stacking
  - Fit another classifier that use the predictions as features

$$\begin{tikzcd}
 & \textit{decistion tree} \arrow[r] & \texttt{not spam} \arrow[dr] & & \\
x_i \arrow[ur] \arrow[r] \arrow[dr] & \textit{naïve Bayes} \arrow[r] & \texttt{spam} \arrow[r] & \textit{decision tree \#2} \arrow[r] & \texttt{spam} \\
 & \textit{k-nearest neighbour} \arrow[r] & \texttt{spam} \arrow[ur] & &
\end{tikzcd}$$

- How does the training set of *decision tree #2* look like?
- Averaging or stacking often performs better than individual models
  - Typically used by Kaggle winners
  - E.g., the Netflix \$1M user-rating competition winner was a stacked classifier

Why does averaging work?

- Consider 3 binary classifiers, each independently correct with probability 0.8
- With simple averaging, the ensemble is correct if we have *at least 2 right*:

$$
\begin{aligned}
P(\text{all 3 right}) &= 0.8^3 && = 0.512 \\
P(\text{2 right, 1 wrong}) &= 3 * 0.8^2(1-0.8) && = 0.384 \\
P(\text{all 3 right}) &= 3 * (1-0.8)^2 0.8 && = 0.096 \\
P(\text{all 3 right}) &= (1-0.8)^3 && = 0.512
\end{aligned}
$$

- So the ensemble is right with a probability of 0.896 (=0.512+0.384)
- Note:
  - For averaging to work, classifiers need to be at least somewhat independent
  - You also want the probability of being right to be > 0.5, otherwise it will do much worse
  - Probabilities also shouldn't be to different (otherwise, it might be better to take most accurate)

Why does averaging work?

- Consider a set of classifiers that makes these predictions:
  - Classifier 1: `spam`
  - Classifier 2: `spam`
  - Classifier 3: `spam`
  - Classifier 4: `not spam`
  - Classifier 5: `spam`
  - Classifier 6: `not spam`
  - Classifier 7: `spam`
  - Classifier 8: `spam`
  - Classifier 9: `spam`
  - Classifier 10: `spam`
- I these independently get 80% accuracy, the mode will be close to 100%
  - In practice errors won't be completely independent due to noise in the labels

Why can averaging work?

- Why can averaging lead to better results?
- Consider classifiers that overfit (like deep decision trees)
  - If they all overfit in exactly the same way, averaging does nothing – Why?
- But if they make **independent** errors
  - Probability that *average* is wrong can be lower than for each classifier
  - Less attention to specific overfitting of each classifier

Random Forests

Random Forests

- Random forests average a set of deep decision trees
  - Tend to be one of the best *out of the box* classifiers
  - Often close to the best performance of any method on the first run
  - And predictions are very fast
- Does averaging work if you use trees with the same parameters?
- Do deep decision trees make independent errors?
  - No: with the same training data you'll get the same decision tree
- Two key ingredients in random forests:
  - Bootstrapping
  - Random trees

Bootstrap Sampling

- Start with a standard deck of 52 cards
  1. Sample a random card, put it back and re-shuffle
  2. Sample a random card, put it back and re-shuffle
  3. Sample a random card, put it back and re-shuffle

     ...

  52 Sample a random card, put it back and re-shuffle
- Make a new deck of the 52 samples

Bootstrap Sampling

- The new 52-card deck is called a **bootstrap sample**
- Some cards will be missing, and some cards will be duplicated
  - So calculations on the bootstrap sample will give different results than original data
- However, the bootstrap sample roughly maintains trends:
  - Roughly 25% of the cards will be diamonds
  - Roughly 3/13 of the cards will be *face* cards
  - There will be roughly four 10 cards
- Common use: compute a statistic based on several bootstrap samples
  - Gives you an idea of how the statistic varies as you vary the data

Random Forest Ingredient 1: Bootstrap

SCIENCE
SCHOOL OF COMPUTER SCIENCE
MACHINE LEARNING

- Bootstrap sample of a list of n examples
  - A new set of size n chosen independently with replacement

forall $i \in 1, \ldots, n$ **do**
  $j = rand(1 : n)$; `// pick a random number from` $\{1, 2, \ldots, n\}$
  $X_{bootstrap}[i, :] = X[j, :]$ `// use the random sample`
end

  - Gives new dataset of n examples, with some duplicated and some missing
    - For large n, approximately 63% of original examples are included (see next slide)
- Bagging: using bootstrap samples for ensemble learning – **B**ootstrap **Agg**regat**ing**
  - Generate several bootstrap samples of the examples (x_i, y_i)
  - Fit a classifier to each bootstrap sample
  - At test time, average the predictions

0.632 Bootstrapping

- Probability of an arbitrary x_i being selected in a bootstrap sample

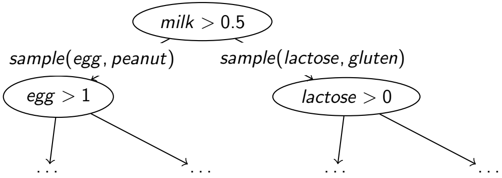
$$
\begin{aligned}
& p(\text{selected at least once in } n \text{ trials}) \\
& \quad = 1 - p(\text{not selected in any of } n \text{ trials}) \\
& \quad = 1 - (p(\text{not selected in one trial})^n \qquad //\text{trials are independent} \\
& \quad = 1 - (1 - \frac{1}{n})^n \qquad //prob = \frac{n-1}{n} \text{ for choosing any of the } n-1 \text{ other samples} \\
& \quad \approx 1 - \frac{1}{e} \qquad //((1 - \frac{1}{n})^n \rightarrow e^{-1} \text{ as } n \rightarrow \inf \\
& \quad \approx 0.632
\end{aligned}
$$

Random Forest Ingredient 2: Random Trees

- For **each split** in a random tree model
  - Randomly sample a small number of possible features (typically $\sqrt{d}$)
  - Only consider these random features when searching for the optimal rule

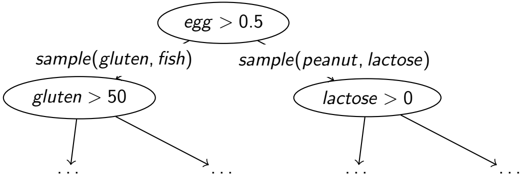
- Random tree 1:

$$sample(milk, oranges)$$



- Random tree 1:

$$sample(egg, lactose)$$



Random Forest Ingredient 2: Random Trees

- Splits will tend to use different features in different trees
  - They will still overfit, but hopefully errors will be more independent
- So the average tends to have a much lower test error
- Empirically, random forests are one of the *best* classifiers
- Fernandez-Delgado et al. [2014]
  - Compared 179 classifiers on 121 datasets
  - Random forests are most likely to be the best classifier

AdaBoost

AdaBoost: Classic Boosting Algorithm

- A classic boosting algorithm for binary classification is AdaBoost
- AdaBoost assumes we have a *base* binary classifier that
  - Is simple enough that it does not overfit much
  - Can obtain $> 50\%$ weighted accuracy on any dataset.
- Example: decision stumps or low-depth decision trees
  - Easy to modify stumps/trees to use weighted accuracy as score

AdaBoost: Classic Boosting Algorithm

- Overview of AdaBoost:
  1. Fit a classifier on the training data
  2. Give a higher weight to examples that the classifier got wrong
  3. Fit a classifier on the weighted training data
  4. Go back to 2
     - Weight gets exponentially larger each time you are wrong.
- Final prediction: weighted vote of individual classifier predictions
  - Trees with higher (weighted) accuracy get higher weight

AdaBoost: Classic Boosting Algorithm

- Are decision stumps a good base classifier?
  - They tend not to overfit
  - Easy to get $> 50\%$ weighted accuracy
- Base classifiers that don't work:
  - Deep decision trees (no errors to boost)
  - Decision stumps with infogain (does not guarantee $> 50\%$ weighted accuracy)
  - Weighted logistic regression (does not guarantee $> 50\%$ weighted accuracy)

AdaBoost: Classic Boosting Algorithm

- AdaBoost with shallow decision trees gives fast/accurate classifiers
  - Classically viewed as one of the best *off the shelf* classifiers
  - Procedure originally came from ideas in learning theory
- Many attempts to extend theory beyond binary case
  - Led to *gradient boosting*, which is like *gradient descent with trees*
- Modern boosting methods:
  - Look like AdaBoost, but don't necessarily have it as a special case

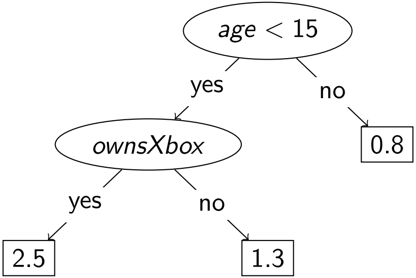
XGBoost

XGBoost: Modern Boosting Algorithm

- Boosting has seen a recent resurgence, partially due to XGBoost:
  - A boosting implementation that allows huge datasets
  - Has been part of many recent winners of Kaggle competitions
- As base classifier, XGBoost uses *regularized regression trees*

Regression Trees

- Regression trees used in XGBoost
  - Each split is based on 1 feature
  - Each leaf gives a real-valued prediction
  - Example: *How many hours of video games per day?*



(Sorry for this example!)

  - We would predict `2.5 hours` for a 14-year-old who owns an Xbox

Regression Trees

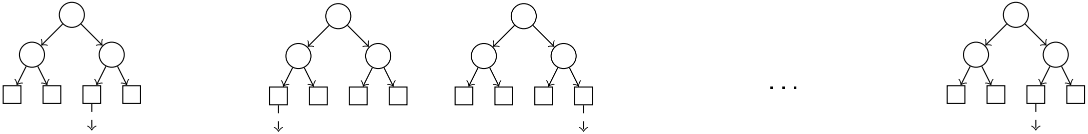
- How can we fit a regression tree?
- Simple approach:
  - Predict: at each leaf, predict mean of the training y_i assigned to the leaf
    - Weight w_L at leaf L is set to $mean(y_i)$ among y_i at the leaf node
  - Train: set the $w - L$ values by minimizing the squared error

$$f(w_1, w_2, \ldots) = \sum_{i=1}^{n} (w_{L_i} - y_i)^2$$

  - Same speed as fitting decision trees from earlier in the semester
    - Use mean instead of mode, and use squared error instead of accuracy / infogain
  - Use greedy strategy for growing tree, as earlier

Boosted Regression Trees: Prediction

- Consider an ensemble of regression trees
  - For an example i, they each make a continuous prediction



$\hat{y}_{i_1} = 0.2$ $\quad \hat{y}_{i_2} = -0.1$ $\quad \hat{y}_{i_3} = 0.1$ $\quad \ldots \quad \hat{y}_{i_4} = -0.01$

- In XGBoost, final prediction is the sum of individual predictions

$$\begin{aligned}\hat{y}_i &= \hat{y}_{i_1} + \hat{y}_{i_2} + \hat{y}_{i_3} + \ldots + \hat{y}_{i_k} \\ &= 0.2 + (-0.1) + 0.1 + \ldots + (-0.01)\end{aligned}$$

Boosted Regression Trees: Prediction

- Notice we are **not using the mean** as we would with random forests
  - What we do instead?
  - In boosting, each tree is **not individually trying to predict the true** y_i value (we assume they underfit)
  - Instead, **each new tree tries to fix the prediction made by the old trees**, so that sum is y_i

Boosted Regression Trees: Training

- Consider the following *gradient tree boosting* procedure:
  - $Tree[1] = fit(X, y)$
  - $\hat{y} = Tree[1].predict(X)$
  - $Tree[2] = fit(X, y - \hat{y})$
  - $\hat{y} = \hat{y} + Tree[2].predict(X)$
  - $Tree[3] = fit(X, y - \hat{y})$
  - $\hat{y} = \hat{y} + Tree[3].predict(X)$
  - $Tree[4] = fit(X, y - \hat{y})$
  - $\hat{y} = \hat{y} + Tree[4].predict(X)$
  - $\ldots$
- Each tree is trying to predict *residuals* $(\hat{y}_i - y_i)$ of current prediction
  - "True label is 0.9, old prediction is 0.8, so I can improve $\hat{y}_i$ by predicting 0.1

Regularized Regression Trees

- Procedure monotonically decreases the training error
  - As long as not all $w_L = 0$, each tree decreases training error
- Can it overfit?
  - It can overfit if trees are too deep or you have too many trees
    - To restrict depth, add L0-regularization (stop splitting if $w_L = 0$)

$$f(w_1, w_1, \ldots) = \sum_{i=1}^{n} (w_{L_i} - r_i) + \lambda_0 ||w||_0$$

    - Only split if you decrease squared error by λ_0.
  - To further fight overfitting, XGBoost also adds L2-regularization of w

$$f(w_1, w_1, \ldots) = \sum_{i=1}^{n} (w_{L_i} - r_i) + \lambda_0 ||w||_0 + \lambda_2 ||w||^2$$

XGBoost Discussion

- Instead of pruning trees if score does not improve, grows full trees
  - And then prunes parts that don't improve score with L0-regularizer added
- Cost of fitting trees in XGBoostis same as usual decision tree cost
  - XGBoost includes a lot of tricks to make this efficient
  - But cannot be done in parallel like random forest – Why?
- In XGBoost, it's the residuals that act like the *weights* in AdaBoost
  - Focuses on decreasing error in examples with large residuals
- How do you maintain efficiency if not using squared error?
  - For non-quadratic losses like logistic, there is no closed-form solution
  - Approximates non-quadratic losses with second-order Taylor expansion
    - Maintains least squares efficiency for other losses (by approximating with quadratic)

Summary

- Ensembles combine predictions of base classifiers
- Boosting: ensemble methods that improve training error
- XGBoost: modern boosting method based on regression trees
  - Each tree modifies the prediction made by the previous trees

Discussion

- In which cases would you **not** use Ensembles?
- How would you handle preprocessing in ensembles?
  - How about feature selection / importance?

Literature

SCIENCE
SCHOOL OF COMPUTER SCIENCE
MACHINE LEARNING

- Chapter 14 in Bishop
- Chapter 11 in Peter Flach's *Machine Learning* (short)

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SCIENCE
SCHOOL OF COMPUTER SCIENCE
MACHINE LEARNING

Thank you for your attention!

https://ml.auckland.ac.nz