Machine Learning – Lecture 11

Random Forests

23.11.2017

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de
leibe@vision.rwth-aachen.de
Course Outline

• Fundamentals
  ➢ Bayes Decision Theory
  ➢ Probability Density Estimation

• Classification Approaches
  ➢ Linear Discriminants
  ➢ Support Vector Machines
  ➢ Ensemble Methods & Boosting
  ➢ Random Forests

• Deep Learning
  ➢ Foundations
  ➢ Convolutional Neural Networks
  ➢ Recurrent Neural Networks

• Main idea  
  ➢ Instead of resampling, reweight misclassified training examples.  
    – Increase the chance of being selected in a sampled training set.  
    – Or increase the misclassification cost when training on the full set.  
  [Freund & Schapire, 1996]

• Components  
  ➢ $h_m(x)$: “weak” or base classifier  
    – Condition: <50% training error over any distribution  
  ➢ $H(x)$: “strong” or final classifier

• AdaBoost:  
  ➢ Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

$$H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)$$
Recap: AdaBoost – Algorithm

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.

2. For $m = 1, \ldots, M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimizing the weighted error function
   
   $$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)$$
   
   b) Estimate the weighted error of this classifier on $X$:
   
   $$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$
   
   c) Calculate a weighting coefficient for $h_m(x)$:
   
   $$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$
   
   d) Update the weighting coefficients:
   
   $$w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \}$$
Recap: AdaBoost – Error Functions

- "Cross-entropy error" used in Logistic Regression
  - Similar to exponential error for $z > 0$.
  - Only grows linearly with large negative values of $z$.

$E = - \sum t_n \ln y_n + (1 - t_n) \ln(1 - y_n)$

$z_n = t_n y(x_n)$
Topics of This Lecture

• Decision Trees

• Randomized Decision Trees
  ➢ Randomized attribute selection

• Random Forests
  ➢ Bootstrap sampling
  ➢ Ensemble of randomized trees
  ➢ Posterior sum combination
  ➢ Analysis
Topics of This Lecture

• Decision Trees

• Randomized Decision Trees
  ➢ Randomized attribute selection

• Random Forests
  ➢ Bootstrap sampling
  ➢ Ensemble of randomized trees
  ➢ Posterior sum combination
  ➢ Analysis
Decision Trees

• Very old technique
  - Origin in the 60s, might seem outdated.

• But…
  - Can be used for problems with nominal data
    - E.g. attributes color ∈ \{red, green, blue\} or weather ∈ \{sunny, rainy\}.
    - Discrete values, no notion of similarity or even ordering.
  - Interpretable results
    - Learned trees can be written as sets of if-then rules.
  - Methods developed for handling missing feature values.
  - Successfully applied to broad range of tasks
    - E.g. Medical diagnosis
    - E.g. Credit risk assessment of loan applicants
  - Some interesting novel developments building on top of them…
Decision Trees

- Example:
  - “Classify Saturday mornings according to whether they’re suitable for playing tennis.”
Decision Trees

- Elements
  - Each node specifies a test for some attribute.
  - Each branch corresponds to a possible value of the attribute.

Decision Trees

• **Assumption**
  - Links must be mutually distinct and exhaustive
  - I.e. one and only one link will be followed at each step.

• **Interpretability**
  - Information in a tree can then be rendered as logical expressions.
  - In our example:
    
    
    $$(\text{Outlook} = \text{Sunny} \land \text{Humidity} = \text{Normal})$$
    
    $$\lor (\text{Outlook} = \text{Overcast})$$
    
    $$\lor (\text{Outlook} = \text{Rain} \land \text{Wind} = \text{Weak})$$
Training Decision Trees

• Finding the optimal decision tree is NP-hard…

• Common procedure: Greedy top-down growing
  - Start at the root node.
  - Progressively split the training data into smaller and smaller subsets.
  - In each step, pick the best attribute to split the data.
  - If the resulting subsets are pure (only one label) or if no further attribute can be found that splits them, terminate the tree.
  - Else, recursively apply the procedure to the subsets.

• CART framework
  - Classification And Regression Trees (Breiman et al. 1993)
  - Formalization of the different design choices.
CART Framework

• Six general questions

  1. Binary or multi-valued problem?
     – I.e. how many splits should there be at each node?

  2. Which property should be tested at a node?
     – I.e. how to select the query attribute?

  3. When should a node be declared a leaf?
     – I.e. when to stop growing the tree?

  4. How can a grown tree be simplified or pruned?
     – Goal: reduce overfitting.

  5. How to deal with impure nodes?
     – I.e. when the data itself is ambiguous.

  6. How should missing attributes be handled?
CART – 1. Number of Splits

- Each multi-valued tree can be converted into an equivalent binary tree:

⇒ Only consider binary trees here…
CART – 2. Picking a Good Splitting Feature

• Goal
  - Want a tree that is as simple/small as possible (Occam’s razor).
  - But: Finding a minimal tree is an NP-hard optimization problem.

• Greedy top-down search
  - Efficient, but not guaranteed to find the smallest tree.
  - Seek a property \( T \) at each node \( s_j \) that makes the data in the child nodes as pure as possible.
  - For formal reasons more convenient to define impurity \( i(s_j) \).
  - Several possible definitions explored.
CART – Impurity Measures

- Misclassification impurity
  \[ i(s_j) = 1 - \max_k p(C_k | s_j) \]

“Fraction of the training patterns in category \( C_k \) that end up in node \( s_j \).”

Problem: discontinuous derivative!
CART – Impurity Measures

- Entropy impurity

\[
i(s_j) = - \sum_k p(C_k | s_j) \log_2 p(C_k | s_j)
\]

“Reduction in entropy = gain in information.”

CART – Impurity Measures

- Gini impurity (variance impurity)

\[ i(s_j) = \sum_{k \neq l} p(C_k | s_j) p(C_l | s_j) \]

\[ = \frac{1}{2} \left[ 1 - \sum_k p^2(C_k | s_j) \right] \]

"Expected error rate at node \( s_j \) if the category label is selected randomly."

CART – Impurity Measures

• Which impurity measure should we choose?
  - Some problems with misclassification impurity.
    - Discontinuous derivative.
    - Problems when searching over continuous parameter space.
    - Sometimes misclassification impurity does not decrease when Gini impurity would.
  - Both entropy impurity and Gini impurity perform well.
    - No big difference in terms of classifier performance.
    - In practice, stopping criterion and pruning method are often more important.
CART – 2. Picking a Good Splitting Feature

• Application
 ➢ Select the query that decreases impurity the most
  \[ \Delta i(s_j) = i(s_j) - P_L i(s_{j,L}) - (1 - P_L) i(s_{j,R}) \]

• Multiway generalization (gain ratio impurity):
  ➢ Maximize
  \[ \Delta i(s_j) = \frac{1}{Z} \left( i(s_j) - \sum_{m=1}^{M} P_m i(s_{j,m}) \right) \]
  ➢ where the normalization factor ensures that large \( K \) are not inherently favored:
  \[ Z = - \sum_{m=1}^{M} P_m \log_2 P_m \]

\( P_L = \) fraction of points at left child node \( s_{j,L} \)
CART – Picking a Good Splitting Feature

• For efficiency, splits are often based on a single feature
  ➢ “Monothetic decision trees”

• Evaluating candidate splits
  ➢ Nominal attributes: exhaustive search over all possibilities.
  ➢ Real-valued attributes: only need to consider changes in label.
    – Order all data points based on attribute $x_i$.
    – Only need to test candidate splits where $\text{label}(x_i) \neq \text{label}(x_{i+1})$. 
CART – 3. When to Stop Splitting

• Problem: Overfitting
  - Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - Reasons
    - Noise or errors in the training data.
    - Poor decisions towards the leaves of the tree that are based on very little data.

• Typical behavior

![Graph showing accuracy vs. hypothesis complexity for both training and test data](image-url)
CART – Overfitting Prevention (Pruning)

- Two basic approaches for decision trees
  - **Prepruning**: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
  - **Postpruning**: Grow the full tree, then remove subtrees that do not have sufficient evidence.
- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.

\[ C_N = \arg \max_k p(C_k | N) \]
Decision Trees – Computational Complexity

• Given
  ➢ Data points \{x_1,\ldots,x_N\}
  ➢ Dimensionality \(D\)

• Complexity
  ➢ Storage: \(O(N)\)
  ➢ Test runtime: \(O(\log N)\)
  ➢ Training runtime: \(O(DN^2 \log N)\)
    – Most expensive part.
    – Critical step: selecting the optimal splitting point.
    – Need to check \(D\) dimensions, for each need to sort \(N\) data points.
      \(O(DN \log N)\)
Summary: Decision Trees

• Properties
  - Simple learning procedure, fast evaluation.
  - Can be applied to metric, nominal, or mixed data.
  - Often yield interpretable results.
Summary: Decision Trees

- Limitations
  - Often produce noisy (bushy) or weak (stunted) classifiers.
  - Do not generalize too well.
  - Training data fragmentation:
    - As tree progresses, splits are selected based on less and less data.
  - Overtraining and undertraining:
    - Deep trees: fit the training data well, will not generalize well to new test data.
    - Shallow trees: not sufficiently refined.
  - Stability
    - Trees can be very sensitive to details of the training points.
    - If a single data point is only slightly shifted, a radically different tree may come out!
      ⇒ Result of discrete and greedy learning procedure.
  - Expensive learning step
    - Mostly due to costly selection of optimal split.
Topics of This Lecture

• Decision Trees

• Randomized Decision Trees
  - Randomized attribute selection

• Random Forests
  - Bootstrap sampling
  - Ensemble of randomized trees
  - Posterior sum combination
  - Analysis
Randomized Decision Trees (Amit & Geman 1997)

- Decision trees: main effort on finding good split
  - Training runtime: $O(DN^2 \log N)$
  - This is what takes most effort in practice.
  - Especially cumbersome with many attributes (large $D$).

- Idea: randomize attribute selection
  - No longer look for globally optimal split.
  - Instead randomly use subset of $K$ attributes on which to base the split.
  - Choose best splitting attribute e.g. by maximizing the information gain (= reducing entropy):
    \[
    \triangle E = \sum_{k=1}^{K} \frac{|S_k|}{|S|} \sum_{j=1}^{N} p_j \log_2(p_j)
    \]
Randomized Decision Trees

• Randomized splitting
  - Faster training: \( O(KN^2 \log N) \) \( K \ll D \)
  - Use very simple binary feature tests.
  - Typical choice
    - \( K = 10 \) for root node.
    - \( K = 100d \) for node at level \( d \).

• Effect of random split
  - Of course, the tree is no longer as powerful as a single classifier…
  - But we can compensate by building several trees.
Ensemble Combination

- Ensemble combination
  - Tree leaves \((l, \eta)\) store posterior probabilities of the target classes.
    
    \[ p_{l, \eta}(C|x) \]

  - Combine the output of several trees by averaging their posteriors (Bayesian model combination)
    
    \[ p(C|x) = \frac{1}{L} \sum_{l=1}^{L} p_{l, \eta}(C|x) \]

B. Leibe
Applications: Character Recognition

• Computer Vision: Optical character recognition
  - Classify small (14x20) images of hand-written characters/digits into one of 10 or 26 classes.

• Simple binary features
  - Tests for individual binary pixel values.
  - Organized in randomized tree.

Applications: Character Recognition

- **Image patches ("Tags")**
  - Randomly sampled $4 \times 4$ patches
  - Construct a randomized tree based on binary single-pixel tests
  - Each leaf node corresponds to a "patch class" and produces a tag

- **Representation of digits ("Queries")**
  - Specific spatial arrangements of tags
  - An image answers "yes" if any such structure is found anywhere

  - *How do we know which spatial arrangements to look for?*
Applications: Character Recognition

• Answer: Create a second-level decision tree!
  - Start with two tags connected by an arc
  - Search through extensions of confirmed queries (or rather through a subset of them, there are lots!)
  - Select query with best information gain
  - Recurse…

• Classification
  - Average estimated posterior distributions stored in the leaves.
Applications: Fast Keypoint Detection

• Computer Vision: fast keypoint detection
  ➢ Detect keypoints: small patches in the image used for matching
  ➢ Classify into one of ~200 categories (visual words)

• Extremely simple features
  ➢ E.g. pixel value in a color channel (CIELab)
  ➢ E.g. sum of two points in the patch
  ➢ E.g. difference of two points in the patch
  ➢ E.g. absolute difference of two points

• Create forest of randomized decision trees
  ➢ Each leaf node contains probability distribution over 200 classes
  ➢ Can be updated and re-normalized incrementally.
Application: Fast Keypoint Detection

Topics of This Lecture

• Decision Trees

• Randomized Decision Trees
  ➢ Randomized attribute selection

• Random Forests
  ➢ Bootstrap sampling
  ➢ Ensemble of randomized trees
  ➢ Posterior sum combination
  ➢ Analysis
Random Forests (Breiman 2001)

• General ensemble method
  - Idea: Create ensemble of many (very simple) trees.

• Empirically very good results
  - Often as good as SVMs (and sometimes better)!
  - Often as good as Boosting (and sometimes better)!

• Standard decision trees: main effort on finding good split
  - Random Forests trees put very little effort in this.
  - CART algorithm with Gini coefficient, no pruning.
  - Each split is only made based on a random subset of the available attributes.
  - Trees are grown fully (important!).

• Main secret
  - Injecting the “right kind of randomness”.

B. Leibe
Random Forests – Algorithmic Goals

- Create many trees (50 – 1,000)

- Inject randomness into trees such that
  - Each tree has maximal strength
    - I.e. a fairly good model on its own
  - Each tree has minimum correlation with the other trees.
    - I.e. the errors tend to cancel out.

- Ensemble of trees votes for final result
  - Simple majority vote for category.

  - Alternative (Friedman)
    - Optimally reweight the trees via regularized regression (lasso).
Random Forests – Injecting Randomness (1)

• Bootstrap sampling process
  - Select a training set by choosing $N$ times with replacement from all $N$ available training examples.
    - On average, each tree is grown on only ~63% of the original training data.
  - Remaining 37% “out-of-bag” (OOB) data used for validation.
    - Provides ongoing assessment of model performance in the current tree.
    - Allows fitting to small data sets without explicitly holding back any data for testing.
    - Error estimate is unbiased and behaves as if we had an independent test sample of the same size as the training sample.
Random Forests – Injecting Randomness (2)

• Random attribute selection
  - For each node, randomly choose subset of $K$ attributes on which the split is based (typically $K = \sqrt{N_f}$).
  => Faster training procedure
    - Need to test only few attributes.
  - Minimizes inter-tree dependence
    - Reduce correlation between different trees.

• Each tree is grown to maximal size and is left unpruned
  - Trees are deliberately overfit
  => Become some form of nearest-neighbor predictor.
Bet You’re Asking…

How can this possibly *ever* work???
A Graphical Interpretation

Different trees induce different partitions on the data.
A Graphical Interpretation

Different trees induce different partitions on the data.
A Graphical Interpretation

Different trees induce different partitions on the data.

By combining them, we obtain a finer subdivision of the feature space...
A Graphical Interpretation

Different trees induce different partitions on the data.

By combining them, we obtain a finer subdivision of the feature space...

...which at the same time also better reflects the uncertainty due to the bootstrapped sampling.

Slide credit: Vincent Lepetit
Summary: Random Forests

• Properties
  - Very simple algorithm.
  - Resistant to overfitting – generalizes well to new data.
  - Faster training
  - Extensions available for clustering, distance learning, etc.

• Limitations
  - Memory consumption
    - Decision tree construction uses much more memory.
  - Well-suited for problems with little training data
    - Little performance gain when training data is really large.
You Can Try It At Home…

• Free implementations available
  - Original RF implementation by Breiman & Cutler
    - Papers, documentation, and code…
    - …in Fortran 77.
  - But also newer version available in Fortran 90!
  - Fast Random Forest implementation for Java (Weka)

References and Further Reading

• More information on Decision Trees can be found in Chapters 8.2-8.4 of Duda & Hart.

R.O. Duda, P.E. Hart, D.G. Stork
Pattern Classification
2nd Ed., Wiley-Interscience, 2000

• The original papers for Randomized Trees

• The original paper for Random Forests:
References and Further Reading

• The original papers for Randomized Trees

• The original paper for Random Forests: