Random Forests and Ferns

David Capel

The Multi-class Classification Problem

• A classifier is a mapping $H$ from feature vectors $f$ to discrete class labels $C$

$$f = (f_1, f_2, \ldots, f_n)$$

$$C \in \{c_1, c_2, \ldots, c_M\}$$

$H : f \rightarrow C$

• Numerous choices of feature space $f$ are possible, e.g.,

- Raw pixel values
- Text histograms
- Color histograms
- Oriented filter banks

The Multi-class Classification Problem

• We have a (large) database of labelled exemplars

$$D^m = (f^m, C^m) \quad \text{for } m = 1 \ldots M$$

• Problem: Given such training data, learn the mapping $H$

$$H : f \rightarrow C$$

• Even better: learn the posterior distribution over class label conditioned on the features:

$$P(C = c_i | f_1, f_2, \ldots, f_n)$$

...and obtain classifier $H$ as the mode of the posterior:

$$H(f) = \arg\max_i P(C = c_i | f_1, f_2, \ldots, f_n)$$

How do we represent and learn the mapping?

There are numerous ways to build multi-class classifiers. See, for example, Bishop’s PRML book.

• Direct, non-parametric: learning of class posterior (Bag of Words)
• K-Nearest Neighbours
• Fisher Linear Discriminants
• Relevance Vector Machines
• Multi-class SVMs

The Multi-class Classification Problem

Training: Labelled exemplars representing multiple classes

Classifying: to which class does this new example belong?

$S$ ?

$?$.?
Binary Decision Trees

- Decision trees classify features by a series of Yes/No questions
- At each node, the feature space is split according to the outcome of some binary decision criterion
- The leaves are labelled with the class C corresponding the feature reached via that path through the tree

Binary Decision Trees: Training

- To train, recursively partition the training data into subsets according to some Yes/No tests on the feature vectors
- Partitioning continues until each subset contains features of a single class
Note, different choices of partitions could lead to simpler trees.

Common decision rules

- **Axis-aligned splitting**
  - Threshold on a single feature at each node
  - Very fast to evaluate
  - \( f_\theta > T \)

- **General plane splitting**
  - Threshold on a linear combination of features
  - More expensive but produces smaller trees
  - \( w^T f > T \)
Choosing the right split

- The split location may be chosen to optimize some measure of classification performance of the child subsets.
- Encourage child subsets with lower entropy (confusion).

\[ f_s = \text{fraction of each class } p_i \]

\[ f_s > T ? \]

- **Gini impurity**
  \[ I = \sum_i p_i(1 - p_i) + \sum_i p_i(1 - p_i) \]

- **Information gain (entropy reduction)**
  \[ J = -\sum_i p_i \log p_i - \sum_i p_i \log p_i \]

Related concept: Ensemble learning

### Pros and Cons of Decision Trees

**Advantages**

- Training can be fast and easy to implement.
- Easily handles a large number of input variables.

**Drawbacks**

- Clearly, it is always possible to construct a decision tree that scores 100% classification accuracy on the training set.
- But they tend to over-fit and do not generalize very well.
- Hence, performance on the testing set will be far less impressive.

So, what can be done to overcome these problems?

How to introduce randomness?

- **Bagging**
  Generate randomized training sets by sampling with replacement from the full training set (bootstrap sampling).
  
  - Full training set \[ \{D, D, D, D, D, D, D, D, D, D, D\} \]
  
  - Random “bag” \[ \{D, D, D, D, D, D, D, D, D, D, D\} \]

- **Feature subset selection**
  Choose different random subsets of the full feature vector to generate each tree.

\[ f_s, f_t, f_r, f_s, f_t, f_r, f_s, f_t, f_r \]

Related idea: choose one feature at random to threshold on at each node in the tree.

Random Forests


**Basic idea**

- Somehow introduce randomness into the tree-learning process.
- Build multiple, independent trees based on the training set.
- When classifying an input, each tree votes for a class label.
- The forest output is the consensus of all the tree votes.
- If the trees really are independent, the performance should improve with more trees.

**Related concept:** Ensemble learning
Each tree votes for which class. Combining those votes and normalizing gives a posterior on class label.

Performance comparison

- Results from Hoi 1995 (This is an old idea, by the way)
- Handwritten digits (10 classes)
- 20x20 pixel binary images
- 60000 training, 10000 testing samples
- Feature set f1: raw pixel values (400 features)
- Feature set f2: f1 plus gradients (852 features in total)
- Uses full training set for every tree (no bagging)
- Compares different features subset sizes (100 and 200 resp)

Revision: Naive Bayes Classifiers

- We would like to evaluate the posterior probability over class label
\[
\arg\max_k P(C_k) \prod_{x=1}^{N} P(f_x|C_k)
\]

- Bayes’ rules tells us that this is equivalent to
\[
\arg\max_k \frac{P(C_k)}{P(f_x)} \prod_{x=1}^{N} P(f_x|C_k)
\]

- But learning the joint likelihood distributions over all features is most likely intractable!

- Naïve Bayes makes the simplifying assumption that features are conditionally independent given the class label
\[
P(f_1, f_2, \ldots, f_N|C_k) = \prod_{x=1}^{N} P(f_x|C_k)
\]

**Ferns:** “Semi-Naïve” Bayes


- Group of features into L small sets of size S (called Ferns)
\[
F_l = \{f_{l1}, f_{l2}, \ldots, f_{lS}\}
\]
where features \(f_l\) are the outcome of a binary test on the input vector, such that \(f_l : \{0, 1\}\)

- Assume groups are conditionally independent, hence
\[
P(f_{l1}, f_{l2}, \ldots, f_{lS}|C_k) = \prod_{x=1}^{S} P(f_{lx}|C_k)
\]

- Learn the class-conditional distributions for each group and apply Bayes rule to obtain posterior,
\[
\text{Class}(f) = \arg\max_k P(C_k) \prod_{x=1}^{S} P(f_{lx}|C_k)
\]

Revision: Naive Bayes Classifiers

\[
\text{Class}(f) = \arg\max_k P(C_k) \prod_{x=1}^{N} P(f_x|C_k)
\]

- This independence assumption is usually false!
- The resulting approximation tends to grossly underestimate the true posterior probabilities

**However...**

- It is usually easy to learn the 1-d conditional densities \(P(f_x|C_k)\)
- It often works rather well in practice!
- *Can we do better without sacrificing simplicity?*
Naive vs Semi-Naive Bayes

- Full joint class-conditional distribution: \( P(f_1, f_2, ..., f_k | C_k) \)
  - Intractable to estimate

- Naive approximation: \( P(f_1, f_2, ..., f_k | C_k) \approx \prod_{i=1}^{k} P(f_i | C_k) \)
  - Too simplistic
  - Poor approximation to true posterior

- Semi-Naive: \( P(f_1, f_2, ..., f_k | C_k) = \prod_{i} P(f_i | C_k) \)
  - Balance of complexity and tractability
  - Trade complexity/performance by choice of fern size (S), Numferns (L)

So how does a single Fern work?

- The output of a fern when applied to a large number of input vectors of the same class is a multinomial distribution.

\[
\begin{array}{cccccc}
5 & 5 & 5 & \cdots & 5 & 5
\end{array}
\]

\[
p(F|C) \uparrow
\]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \cdots & 2^L
\end{array}
\]

Training a Fern

- Apply the fern to each labelled training example \( D_{train}(l, y_i, C_k) \) and compute its output \( F(D_{train}) \)
- Learn multinomial densities \( p(F|C_k) \) as histograms of fern output for each class

Classifying using a single Fern

- Given a test input, simply apply the fern and “look-up” the posterior distribution over class label

\[
\begin{array}{cccccc}
\text{Input: } x & \rightarrow & F(D_{test}) & \rightarrow & p(F|C_k) & \rightarrow & p(C_k|F) = \frac{p(F|C_k)p(C_k)}{\sum_k p(F|C_k)p(C_k)}
\end{array}
\]

Adding randomness: an ensemble of ferns

- A single fern does not give great classification performance
- But, we can build an ensemble of “independent” ferns by randomly choosing different subsets of features

\[
\begin{array}{cccccc}
\text{Class}(\mathbf{f}) = \arg\max_k P(C_k) \prod_{l=1}^{L} P(F_l | C_k)
\end{array}
\]

- Finally, combine their outputs using Semi-Naive Bayes:
Note: Fern can be interpreted as a decision tree with the same feature used for all decisions at one level of the tree.

In my opinion, for some applications, (e.g. pedestrian/face tracking) the extreme rotations are overkill.

Classification Rate vs Number of Ferns

- 500 classes were used
- Also compares to Random Trees of equivalent size
Conclusions:

- Random Ferns are easy to understand and easy to train
- Very fast to perform classification once trained
- Provide a probabilistic output (how accurate though?)
- Appear to outperform Random Forests
- Can be very memory hungry!

Drawback: Memory requirements

- Fern classifiers can be very memory hungry, e.g.

Fern size = 2
Number of ferns = 50
Number of classes = 1000

\[
\text{RAM} = 2^\text{size} \times \text{sizeeff} \times \text{NumFerns} \times \text{NumClasses} \\
= 2048 \times 4 \times 50 \times 1000 \\
= 400 \text{ MBytes!}
\]