Random Forests and Ferns

David Capel
The Multi-class Classification Problem

Training: Labelled exemplars representing multiple classes

<table>
<thead>
<tr>
<th>Handwritten digits</th>
<th>Planes</th>
<th>Faces</th>
<th>Cars</th>
<th>Cats</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>5</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>6</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>7</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
<tr>
<td>9</td>
<td><img src="image" alt="Plane" /></td>
<td><img src="image" alt="Face" /></td>
<td><img src="image" alt="Car" /></td>
<td><img src="image" alt="Cat" /></td>
</tr>
</tbody>
</table>

Classifying: to which class does this new example belong?

5 ? [Plane] ?
The Multi-class Classification Problem

- A classifier is a mapping \( H \) from feature vectors \( f \) to discrete class labels \( C \)

\[
f = (f_1, f_2, \ldots, f_N)
C \in \{c_1, c_2, \ldots, c_K\}
H : f \rightarrow C
\]

- Numerous choices of feature space \( F \) are possible, e.g.

- Raw pixel values
- Texton histograms
- Color histograms
- Oriented filter banks
The Multi-class Classification Problem

• We have a (large) database of labelled exemplars

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

• Problem: Given such training data, learn the mapping \( H \)

\[ H : \mathbf{f} \rightarrow C \]

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

\[ P(C = c_k | f_1, f_2, \ldots, f_N) \]

..and obtain classifier \( \mathbf{H} \) as the mode of the posterior:

\[ H(\mathbf{f}) = \arg\max_k P(C = c_k | f_1, f_2, \ldots, f_N) \]
How do we represent and learn the mapping?

In Bishop’s book, we saw numerous ways to build multi-class classifiers, e.g.

- Direct, non-parametric learning of class posterior (histograms)
- K-Nearest Neighbours
- Fisher Linear Discriminants
- Relevance Vector Machines
- Multi-class SVMs
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**

- 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

- 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

- 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 to the feature reached via that path through the tree.

![Binary Decision Tree Diagram]
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.
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
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.
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.
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.
Common decision rules

• **Axis-aligned splitting**
  
  Threshold on a single feature at each node
  
  Very fast to evaluate
  
  \[ f_n > T \]

• **General plane splitting**
  
  Threshold on a linear combination of features
  
  More expensive but produces smaller trees
  
  \[ \mathbf{w}^T \mathbf{f} > T \]
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)

$$f_n > T ?$$

- Gini impurity

$$I_G = \sum_k p_k^{\text{left}} (1 - p_k^{\text{left}}) + \sum_k p_k^{\text{right}} (1 - p_k^{\text{right}})$$

- Information gain (entropy reduction)

$$I_E = - \sum_k p_k^{\text{left}} \log p_k^{\text{left}} - \sum_k p_k^{\text{right}} \log p_k^{\text{right}}$$
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?*
Random Forests

Breiman, Leo (2001) "Random Forests" Machine Learning

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*
How to introduce randomness?

- **Bagging**
  Generate randomized training sets by sampling with replacement from the full training set (bootstrap sampling)

  - Full training set: $D_1 \ D_2 \ D_3 \ D_4 \ D_5 \ D_6 \ D_7 \ D_8 \ D_9 \ D_{10} \ D_{11} \ D_{12}$
  - Random “bag”: $D_4 \ D_9 \ D_3 \ D_4 \ D_{12} \ D_{10} \ D_{10} \ D_7 \ D_3 \ D_1 \ D_6 \ D_1$

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

  - Full feature vector: $f_1 \ f_2 \ f_3 \ f_4 \ f_5 \ f_6 \ f_7 \ f_8 \ f_9 \ f_{10}$
  - Feature subset: $f_4 \ f_6 \ f_7 \ f_9 \ f_{10}$
Random Forests

Input feature vector $\mathbf{f}$

Tree 1

$c_1$ $c_2$ $c_1$ $c_1$

$D_2^1$ $D_3^1$ $D_1^1$

Tree 2

$c_2$ $c_1$ $c_1$ $c_2$

$D_2^2$ $D_3^2$ $D_1^2$

Tree L

$c_1$ $c_2$ $c_2$ $c_1$

$D_2^L$ $D_3^L$ $D_1^L$

Class 1

Class 2

Votes
Performance comparison

- Results from Ho 1995
- Handwritten digits (10 classes)
- 20x20 pixel binary images
- 60000 training, 10000 testing samples

- Feature set $f_1$: raw pixel values (400 features)
- Feature set $f_2$: $f_1$ plus gradients (852 features in total)

- Uses full training set for every tree (no bagging)
- Compares different features subset sizes (100 and 200 resp)
Performance comparison

![Graph showing performance comparison with different forest sizes and parameters.

- `cap (f2), 200d`
- `cap (f1), 200d`
- `cap (f2), 100d`
- `cap (f1), 100d`]
Revision: Naive Bayes Classifiers

- We would like to evaluate the posterior probability over class label
  \[
  \arg\max_k P(C_k|f_1, f_2, ..., f_N)
  \]

- Bayes’ rules tells us that this is equivalent to
  \[
  \arg\max_k P(f_1, f_2, ..., f_N|C_k)P(C_k) \quad \text{(likelihood x prior)}
  \]

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

- Naive Bayes makes the simplifying assumption that features are conditionally independent given the class label
  \[
  P(f_1, f_2, ..., f_N|C_k) = \prod_{i=1}^{N} P(f_i|C_k)
  \]
Revision: Naive Bayes Classifiers

\[
\text{Class}(\mathbf{f}) \equiv \arg\max_k P(C_k) \prod_{n=1}^{N} P(f_n|C_k)
\]

- 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 \( P(f_i|C_k) \)
- It often works rather well in practice!
- \textit{Can we do better without sacrificing simplicity?}
Ferns: "Semi-Naive" Bayes


• Group of features into L small sets of size S (called Ferns)
  \[ F_l = \{ f_{l,1}, f_{l,2}, \ldots, f_{l,S} \} \]
  where features \( f_n \) are the outcome of a binary test on the input vector, such that \( f_n : \{0,1\} \)

• Assume groups are conditionally independent, hence
  \[ P(f_1, f_2, \ldots, f_N | C_k) = \prod_{l=1}^{L} P(F_l | C_k) \]

• Learn the class-conditional distributions for each group and apply Bayes rule to obtain posterior,
  \[ \text{Class}(\mathbf{f}) \equiv \arg\max_{k} P(C_k) \prod_{l=1}^{L} P(F_l | C_k) \]
Naive vs Semi-Naive Bayes

• Full joint class-conditional distribution: \( P(f_1, f_2, \ldots, f_N|C_k) \)
  Intractable to estimate

• Naive approximation: \( P(f_1, f_2, \ldots, f_N|C_k) = \prod_{i=1}^{N} P(f_i|C_k) \)
  Too simplistic
  Poor approximation to true posterior

• Semi-Naive: \( P(f_1, f_2, \ldots, f_N|C_k) = \prod_{l=1}^{L} P(F_l|C_k) \)
  Balance of complexity and tractability
  Trade complexity/performance by choice of Fern size (S), NumFerns (L)
So how does a single Fern work?

- A Fern applies a series of $S$ binary tests to the input vector $\mathbf{I}$

  e.g relative intensities of a pair of pixels:
  \[
  f_1(\mathbf{I}) = I(x_a, y_a) > I(x_b, y_b) \rightarrow \text{true} \\
  f_2(\mathbf{I}) = I(x_c, y_c) > I(x_d, y_d) \rightarrow \text{false}
  \]

- This gives an $S$-digit binary code for the feature which may be interpreted as an integer in the range $[0 \ldots 2^S-1]$

- It’s really just a simple hashing scheme that drops input vectors into one of $2^S$ “buckets”
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**.
Training a Fern

- Apply the fern to each labelled training example \( D_m = (l_m, c_m) \) and compute its output \( F(D_m) \)
- Learn multinomial densities \( p(F|C_k) \) as histograms of fern output for each class

\[
\begin{align*}
\text{p}(F|C_0) & \quad 0 \quad 2^5 \\
\text{p}(F|C_1) & \quad 0 \quad 2^5 \\
\text{p}(F|C_2) & \quad 0 \quad 2^5 \\
\vdots & \quad \cdot \\
\text{p}(F|C_K) & \quad 0 \quad 2^5
\end{align*}
\]
Classifying using a single Fern

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

\[
p(C_k|F) = \frac{p(F|C_k)}{\sum_k p(F|C_k)}
\]
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

  e.g. \( F_1 = \{ f_2, f_7, f_{22}, f_5, f_9 \} \)
  \( F_2 = \{ f_4, f_1, f_{11}, f_8, f_3 \} \)
  \( F_3 = \{ f_6, f_{31}, f_{28}, f_{11}, f_2 \} \)

• Finally, combine their outputs using Semi-Naive Bayes:

\[
\text{Class}(f) \equiv \arg\max_k P(C_k) \prod_{l=1}^L P(F_l|C_k)
\]
Classifying using Random Ferns

- Having randomly selected and trained a collection of ferns, classifying new inputs involves only simple look-up operations.
One small subtlety...

- Even for moderate size ferns, the output range can be large
e.g. fern size $S=10$, output range = \([0 \ldots 2^{10}] = [0 \ldots 1024]\)
- Even with large training set, many “buckets” may see no samples

\[
p(F|C_k) = \frac{N(F = z|C_k) + 1}{\sum_{z=0}^{2^S-1} (N(F = z|C_k) + 1)}\]

where $N(F=z|C_k)$ is the number of times we observe fern output equal to $z$ in the training set for class $C_k$

---

**These zero-probabilities will be problematic for our Semi-Naive Bayes!**
Comparing Forests and Ferns

**Forests**
- Decision trees directly learn the posterior $P(C_k|F)$
- Applies different sequence of tests in each child node
- Training time grows exponentially with tree depth
- Combine tree hypotheses by averaging

**Ferns**
- Learn class-conditional distributions $P(F|C_k)$
- Applies the same sequence of tests to every input vector
- Training time grows linearly with fern size $S$
- Combine hypothesis using Bayes rule (multiplication)
Application: Fast Keypoint Matching

- Ozuysal et al. use Random Ferns for keypoint recognition
- Similar to typical application of SIFT descriptors
- Very robust to affine viewing deformations
- Very fast to evaluate (13.5 microsec per keypoint)
Training

- Each keypoint to be recognized is a separate class
- Training sets are generated by synthesizing random affine deformations of the image patch (10000 samples)
- Features are pairwise intensity comparisons: \( f_n(I) = I(x_a,y_a) > I(x_b,y_b) \)
- Fern size \( S=10 \) (randomly selected pixel pairs)
- Ensembles of 5 to 50 ferns are tested

Synthesized viewpoint variation

Fig. 6. Warped patches from the images of Figure 5 show the range of affine deformations we considered. In each line, the leftmost patch is the original one and the others are deformed versions of it. (a) Sample patches from the City image. (b) Sample patches from the Flowers image. (c) Sample patches from the Museum image.
Figure 4. The percentage of correctly classified image patches is shown against different classifier parameters for the fern and tree based methods. The independence assumption between ferns allows the joint utilization of the features resulting in a higher classification rate. The error bars show the 95% confidence interval assuming a Gaussian distribution.

Figure 5. Classification using ferns can handle many more classes than Random Tree based methods. For both figures ferns with size 10 and trees with depth 10 are used.

4.1. Ferns Outperform Trees

We evaluate the performance of the proposed fern based approach by comparing to the results of a Random Tree based implementation. The number of tests in the ferns and the depth of the trees are taken to be equal, and we compare the classification rate when using the same number of structures, that is of ferns or Random Trees. In particular, the same number of tests is performed on each keypoint, and the same number of joint probabilities has to be stored.

We do our tests on the images presented with 500 classes and calculate the average classification rate on randomly generated test images while eliminating false matches using object geometry. Since the feature selection is random, we repeat the test 10 times and calculate the mean and variance of the classification rate and we perform the test on the two images.

As depicted by Figure 4, despite the inaccuracy of the independence assumptions the fern based classifier outperforms the combination of trees. Furthermore as the number of ferns is increased the random selection method does not cause large variations on the classifier performance.

We also investigate the behavior of the classification rate as the number of classes increases. Figure 5 shows that a larger number of classes does not affect the performance of ferns much, while tree based methods can not cope with many classes. In both experiments we have trained the classifiers using classes from three different images up to 700 classes for each image.

4.2. Linking the Two Approaches

Here we show that the two approaches are equivalent when the training set is small and give some insights into why the ferns perform better when it is large.

Recall that we evaluate $P(F_k|C=c_i)$ of Eq. (3) $P(F_k|C=c_i) = P_e P(\Theta(F_k)) + \mu (1 - P_e)$, where $P_e$ is the empirical probability and $\mu = 1/H$. Since computing the product of such terms is the same as sum-
Figure 4. The percentage of correctly classified image patches is shown against different classifier parameters for the fern and tree based methods. The independence assumption between ferns allows the joint utilization of the features resulting in a higher classification rate. The error bars show the 95% confidence interval assuming a Gaussian distribution.

Figure 5. Classification using ferns can handle many more classes than Random Tree based methods. For both figures ferns with size 10 and trees with depth 10 are used.

4.1. Ferns Outperform Trees

We evaluate the performance of the proposed fern based approach by comparing to the results of a Random Tree based implementation. The number of tests in the ferns and the depth of the trees are taken to be equal, and we compare the classification rate when using the same number of structures, that is of ferns or Random Trees. In particular, the same number of tests is performed on each keypoint, and the same number of joint probabilities has to be stored. We do our tests on the images presented with 500 classes and calculate the average classification rate on randomly generated test images while eliminating false matches using object geometry. Since the feature selection is random, we repeat the test 10 times and calculate the mean and variance of the classification rate and we perform the test on the two images.

As depicted by Figure 4, despite the inaccuracy of the independence assumptions the fern based classifier outperforms the combination of trees. Furthermore as the number of ferns is increased the random selection method does not cause large variations on the classifier performance.

We also investigate the behavior of the classification rate as the number of classes increases. Figure 5 shows that a larger number of classes does not affect the performance of ferns much, while tree based methods can not cope with many classes. In both experiments we have trained the classifiers using classes from three different images up to 700 classes for each image.

4.2. Linking the Two Approaches

Here we show that the two approaches are equivalent when the training set is small and give some insights into why the ferns perform better when it is large.

Recall that we evaluate $P(F_k|C_i=c)$ of Eq. (3)

$$P(F_k|C_i=c) = P_e P(\Theta(F_k)) + \mu (1 - P(\Theta(F_k)))$$

where $P_e$ is the empirical probability and $\mu = \frac{1}{H}$. Since computing the product of such terms is the same as summing...

• 30 random ferns of size 10 were used
Drawback: Memory requirements

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

Fern size = 11
Number of ferns = 50
Number of classes = 1000

$$\text{RAM} = 2^{\text{Fern size}} \times \text{sizeof(float)} \times \text{NumFerns} \times \text{NumClasses}$$

$$= 2048 \times 4 \times 50 \times 1000$$

$$= 400 \text{ MBytes}!$$
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!