Lecture 2: Ensemble Learning - AdaBoost and Random Forests

COSC470

Brendan McCane

Department of Computer Science, University of Otago

Ensemble Learning

In all the following, a 2-class problem is assumed, with $\mathbf{Y}_i \in \{-1, 1\}$.

The idea of ensemble learning is simple. Given some number of classifiers, $f_1, f_2, \ldots, f_m$, create a new classifier that uses the results of the f_i. In this case, $f_i : \mathbb{R}^d \to \{-1, 1\}$.

So, f is some function of the f_i:

$$f(\mathbf{x}) = F(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \tag{1}$$

for some function F.

Majority Vote

The simplest form of F is a majority vote. Choose the label that most of the f_i choose.

This works pretty well if the individual classifiers are independent and also if they happen to focus on different aspects of the problem.

Bootstrap aggregation (bagging) is a particular example of this sort of classifier. Random Forests are a particular type of bagging classifier.

Can we do better than majority vote?

Let's assume we have just two classifiers that we want to combine linearly to make a better classifier:

$$f(\mathbf{x}) = af_1(\mathbf{x}) + bf_2(\mathbf{x}), \tag{2}$$

for some constants a and b, as yet unspecified.

Let the classification rate for f_i be α_i. Ideally the classification rate would be the actual classification rate, but the best we can do is get the classification rate from some validation set.

Loss function

How do we choose a and b? Usually, we specify some function to minimise. We call this sort of function a *loss function*. In this case, let's use Euclidean distance from the label:

$$
\begin{aligned}
L &= \sum_{i=1}^{n} (f(\mathbf{X}_i) - \mathbf{Y}_i)^2 \\
&= \sum_{i=1}^{n} (a f_1(\mathbf{X}_i) + b f_2(\mathbf{X}_i) - \mathbf{Y}_i)^2 \\
&= \sum_{i=1}^{n} a^2 f_1^2 + 2ab f_1 f_2 - 2a f_1 \mathbf{Y}_i - 2b f_2 \mathbf{Y}_i + b^2 f_2^2 + \mathbf{Y}_i^2
\end{aligned}
$$

Some simplifications and calculus

Because the f_i and $\mathbf{Y}_i$ are either -1 or 1 we can do some simplifying.

In particular $\sum f_i^2 = n$ and $\sum \mathbf{Y}_i^2 = n$.

But also, $f_1 \mathbf{Y}_i$ will be 1 if f_1 is correct, and -1 if f_1 is incorrect. This means that $\sum f_j \mathbf{Y}_i = 2n(\alpha_j - 0.5)$.

So we get the following:

$$L = na^2 + nb^2 + n - 4an(\alpha_1 - 0.5) \sum_{i=1}^{n} 2abf_1 f_2 - 2af_1 \mathbf{Y}_i - 2bf_2 \mathbf{Y}_i$$

$$\frac{\partial L}{\partial a} = 2na - 4n(\alpha_1 - 0.5) - 2b \sum_{i=1}^{n} f_1 f_2 = 0$$

$$\frac{\partial L}{\partial b} = 2nb - 4n(\alpha_2 - 0.5) - 2a \sum_{i=1}^{n} f_1 f_2 = 0$$

Solving for a and b

We get:

$$a = \frac{n^2(2\alpha_1 - 1) + n(2\alpha_2 - 1)\sum f_1 f_2}{n^2 - (\sum f_1 f_2)^2}$$
$$b = \frac{n^2(2\alpha_2 - 1) + n(2\alpha_1 - 1)\sum f_1 f_2}{n^2 - (\sum f_1 f_2)^2}$$

Which is a bit complicated, but $\sum f_1 f_2$ is essentially the correlation between f_1 and f_2. If they are uncorrelated, then that term is 0, leaving:

$$a = 2\alpha_1 - 1$$
$$b = 2\alpha_2 - 1$$

Limitations

- Doing an ensemble of two classifiers is actually pointless. Why?
- The Euclidean distance is actually not what we want to optimise. What is?
- The assumption that f_1 and f_2 are not correlated is extremely unrealistic.

The Important Ideas

But the exercise highlights the usual steps in developing these sorts of classifiers:

1. Identify the idea (combine multiple classifiers to form a better one).
2. Identify an appropriate loss function. This is usually a proxy for what we want to optimise and is chosen for convenience.
3. Identify assumptions or approximations that make solving the proxy problem feasible.
4. Optimise the loss function.
5. Evaluate the new method (we left that bit out).

A Bayesian Interpretation

We can recast the problem of classification in a Bayesian manner. Given training data $\mathbf{X}, \mathbf{Y}$ and a new data point $\mathbf{x}$, we'd like to compute:

$$P(f(\mathbf{x}) = y | \mathbf{X}, \mathbf{Y}) \tag{3}$$

for each possible label y. And if we needed a concrete label, then we would just choose the label that maximises the equation.

In most cases, we do not know what f is, so we can't evaluate the probability directly. Can we estimate it?

Hypothesis spaces

If we have a particular type of classifier (e.g. a decision tree), then we may think about the space of all such classifiers limited to a particular problem.

Call the space of all such classifiers the hypothesis space, $\mathcal{H}$. This space is just a set, but it has a bit more structure than a basic set because there are relationships between different classifiers.

A particular classifier, f_i, then is a point in this abstract hypothesis space: $f_i \in \mathcal{H}$.

We'd like to use $\mathcal{H}$ to find a good approximation of $f(x)$. This might be a single classifier, f_1, or a combination of classifiers all sampled from $\mathcal{H}$ somehow.

Some probabilities

$$P(f(\mathbf{x}) = y|\mathbf{X}, \mathbf{Y}) = \sum_{f_i \in \mathcal{H}} P(f(\mathbf{x}) = y, f_i|\mathbf{X}, \mathbf{Y})$$
$$= \sum_{f_i \in \mathcal{H}} P(f(\mathbf{x}) = y|f_i, \mathbf{X}, \mathbf{Y}) P(f_i|\mathbf{X}, \mathbf{Y})$$

The hypothesis terms should be read as "given f_i is the correct or best hypothesis in the hypothesis space".

So the last term, $P(f_i|\mathbf{X}, \mathbf{Y})$ should be read as: what is the probability that f_i is the best hypothesis given this set of training data.

Unfortunately, neither of these terms are easy to compute. But we can approximate them.

$P(f(\mathbf{x}) = y | f_i, \mathbf{X}, \mathbf{Y})$

The first term is really just the probability that the underlying function and our learned classifer give the same answer. The easiest thing to do there is assume the errors from f_i are uniformly distributed, in which case:

$$P(f(x) = y | f_i, \mathbf{X}, \mathbf{Y}) = \begin{cases} \alpha_i & \text{if } f_i(\mathbf{x}) = y \\ 1 - \alpha_i & \text{otherwise} \end{cases} \tag{4}$$

where α_i is classification accuracy as before.

We can do a bit better by using accuracies associated with the given classes.

We could do a bit better than that if we could assign accuracies to different regions of the input space, but I don't know of any method that does that.

$P(f_i|\mathbf{X}, \mathbf{Y})$

This term is a bit harder to compute because how can we know what that probability is?

We can make use of Bayes' rule:

$$P(f_i|\mathbf{X}, \mathbf{Y}) \propto P(\mathbf{X}, \mathbf{Y}|f_i)P(f_i) \tag{5}$$

The $P(f_i)$ term is called the prior and it encodes our knowledge of the problem. In other words, with no other information, what is the probability that f_i is the best hypothesis? Normally we have no idea what the prior is, so in practice either a uniform or Gaussian prior is chosen.

$P(\mathbf{X}, \mathbf{Y}|f_i)$

This term means: given that f_i is the correct hypothesis, what is the probability that the training data would have been generated?

Again, it's not clear how to compute this probability and it depends on the underlying classifier.

For a decision tree, we could say: given the proportion of classes in the leaf are the true proportions, if we randomly draw n_m samples from that distribution, what is the probability we would see the proportions we did see.

The details aren't that important - it is often easiest just to assume this probability is 1.

Finally: the simplest Bayesian Ensemble

If we take all of that analysis, and make the simplest choices at each stage, we end up with:

$$P(f(\mathbf{x}) = y) = \sum_{f_i \in \mathcal{H}} \alpha_i I(f_i(\mathbf{x}) = y) + (1 - \alpha_i) I(f_i(\mathbf{x})! = y) \tag{6}$$

which is similar in form to our linear combination classifier.

The problem

Aside from all the unrealistic assumptions, what's the problem?

Usually the hypothesis space of classifiers is very large and often infinite.

Therefore there is no hope of evaluating all possible classifiers.

Our only option then is to sample from the space somehow. We could sample randomly, but then we'd likely get a lot of classifiers with very low α_i's.

The two most common sampling methods are: bagging and boosting.

Bootstrap aggregation (bagging)

Bagging is pretty simple:

For some number of rounds:

1. randomly sample from the training data with replacement
2. learn a classifier using the sampled data

combine the resulting classifiers.

Random Forests

Random forests are a particular type of bagging classifer:

For some number of rounds:

1. randomly sample from the training data with replacement
2. learn a randomised decision tree classifier using the sampled data

combine the resulting classifiers using voting (original method) or the α_i's (sklearn method).

Randomised decision tree

There are a couple of variants, but the simplest one is:

1. choose a subset of the features to split on
2. choose the best split amongst these

Rather than pruning, usually some other stopping criterion is used (max depth, min samples per leaf, etc).

Boosting

Boosting uses a different method.

Rather than sampling from the training set, it seeks to weight the samples based on importance.

Importance is updated during training so that harder to classify samples get heavier weighting as training progresses.

AbaBoost (Adaptive Boosting), provides an effective method for doing this adaptively. It tries to change the weightings of the samples, so that subsequently trained classifiers are uncorrelated. Why is this important?

AdaBoost algorithm from Freund and Schapire 1997

Algorithm AdaBoost

Input: sequence of N labeled examples $\langle (x_1, y_1), ..., (x_N, y_N) \rangle$
distribution D over the N examples
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, ..., N$.

Do for $t = 1, 2, ..., T$

1. Set

$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$$

2. Call **WeakLearn**, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t: X \to [0, 1]$.
3. Calculate the error of h_t: $\varepsilon_t = \sum_{i=1}^{N} p_i^t \, |h_t(x_i) - y_i|$.
4. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.
5. Set the new weights vector to be

$$w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$$

Note: this algorithm assumes the weak classifier outputs 0 or 1.

Final Classifier

The final classifier output is given by:

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} (\log 1/\beta_t) h_t(\mathbf{x}) \geq \frac{1}{2} \sum_{t=1}^{T} \log 1/\beta_t \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Why does AdaBoost work?

Theorem

Suppose WeakLearn generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$, then the error of the final hypothesis output by AdaBoost is:

$$\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \tag{8}$$

Notes:

- this is the training error
- generally we want $\epsilon_t < 0.5$, but this will work as long as $\epsilon_t ! = 0.5$ for all t.

Homework

1. Implement the random forests algorithm.
2. Implement AdaBoost using decision stumps as the weak learner.
3. Compare.

Decision stumps are decision trees of height 1 (that is, just a threshold on a single feature).