Quick exercise

Write down on the paper (don't write your name):
1) Something you're happy about right now
2) Something you're worried about right now

Fold the piece of paper

I'll collect them, redistribute them and we'll read them out loud

If you don't want to participate, just leave the paper blank

Ensemble learning

Basic idea: if one classifier works well, why not use multiple classifiers!
Ensemble learning

**Basic idea:** if one classifier works well, why not use multiple classifiers!

**Training**

- model 1
- model 2
- ...  
- model m

**Testing**

- prediction 1
- prediction 2
- ...  
- prediction m

**Benefits of ensemble learning**

Assume each classifier makes a mistake with some probability (e.g., 0.4, that is a 40% error rate)

Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e., error rate) with three classifiers for a binary classification problem?
Benefits of ensemble learning

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

<table>
<thead>
<tr>
<th>model 1</th>
<th>model 2</th>
<th>model 3</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>0.216</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>I</td>
<td>0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>C</td>
<td>0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>I</td>
<td>0.096</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>C</td>
<td>0.144</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>I</td>
<td>0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>C</td>
<td>0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>0.064</td>
</tr>
</tbody>
</table>

9

For 3 classifiers in general, for $r = $ probability of mistake for individual classifier:

$$ p(error) = 3r^2(1-r) + r^3 $$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$p(error)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.35</td>
</tr>
<tr>
<td>0.3</td>
<td>0.22</td>
</tr>
<tr>
<td>0.2</td>
<td>0.10</td>
</tr>
<tr>
<td>0.1</td>
<td>0.028</td>
</tr>
<tr>
<td>0.05</td>
<td>0.00873</td>
</tr>
</tbody>
</table>

11

For 5 classifiers in general, for $r = $ probability of mistake for individual classifier:

$$ p(error) = 10r^3(1-r)^2 + 5r^4(1-r) + r^5 $$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$p(error)$ 3 classifiers</th>
<th>$p(error)$ 5 classifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.35</td>
<td>0.32</td>
</tr>
<tr>
<td>0.3</td>
<td>0.22</td>
<td>0.16</td>
</tr>
<tr>
<td>0.2</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td>0.1</td>
<td>0.028</td>
<td>0.0086</td>
</tr>
<tr>
<td>0.05</td>
<td>0.0073</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

12
Benefits of ensemble learning

m classifiers in general, for \( r = \text{probability of mistake for individual classifier} \):

\[
p(\text{error}) = \sum_{i=(m+1)/2}^{m} \binom{m}{i} r^i (1-r)^{m-i}
\]

(cumulative probability distribution for the binomial distribution)

Given enough classifiers…

\[
p(\text{error}) = \sum_{i=(m+1)/2}^{m} \binom{m}{i} r^i (1-r)^{m-i} \quad r = 0.4
\]

What’s the catch?

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e. error rate) with three classifiers for a binary classification problem?

What’s the catch?

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e. error rate) with three classifiers for a binary classification problem?
Obtaining independent classifiers

Where do we get $m$ independent classifiers?

Idea 1: different learning methods

<table>
<thead>
<tr>
<th>Pros:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lots of existing classifiers already</td>
</tr>
<tr>
<td>Can work well for some problems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cons/concerns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Often, classifiers are not independent, that is, they make the same mistakes!</td>
</tr>
<tr>
<td>e.g. many of these classifiers are linear models</td>
</tr>
<tr>
<td>Voting won't help us if they're making the same mistakes</td>
</tr>
</tbody>
</table>

Idea 2: split up training data

Use the same learning algorithm, but train on different parts of the training data
Idea 2: split up training data

Pros:
- Learning from different data, so can’t overfit to same examples
- Easy to implement
- Fast

Cons/concerns:
- Each classifier is only training on a small amount of data
- Not clear why this would do any better than training on full data and using good regularization

Idea 3: bagging

Training Data 1

... 

Training Data m

Training alg

model 1

model m
bagging

Use training data as a proxy for the data generating distribution

sampling with replacements

pick a random example from the real training data

add it to the new “training” data
sampling with replacements

“Training” data 1

put it back (i.e. leave it) in the original training data

Training data

30

sampling with replacements

“Training” data 1

pick another random example

Training data

31

sampling with replacements

“Training” data 1

pick another random example

Training data

32
bagging

create m “new” training data sets by sampling with replacement from the original training data set (called m “bootstrap” samples)

train a classifier on each of these data sets

to classify, take the majority vote from the m classifiers

bagging concerns

Won’t these all be basically the same?

bagging concerns

For a data set of size n, what is the probability that a given example will NOT be select in a “new” training set sampled from the original?

\[1 - \frac{1}{n}\]

bagging concerns

What is the probability it isn’t chosen the first time?
What is the probability it isn’t chosen the any of the n times?

\[(1 - 1/n)^n\]

Each draw is independent and has the same probability

Converges very quickly to \(1 - 1/e \approx 37\%\)

Won’t these all be basically the same?

On average, a randomly sampled data set will only contain 63% of the examples in the original

Let’s say 10% of our examples are noisy (i.e. don’t provide good information)

For each of the “new” data set, what proportion of noisy examples will they have?

- They’ll still have \(-10\%\) of the examples as noisy
- However, these examples will only represent about two-thirds of the original noisy examples

For some classifiers that have trouble with noisy classifiers, this can help
When does bagging work

Bagging tends to reduce the variance of the classifier.

- By voting, the classifiers are more robust to noisy examples.
- Bagging is most useful for classifiers that are unstable: small changes in the training set produce very different models.
- Bagging is also useful for classifiers prone to overfitting.
- Often has a similar effect to regularization.

Idea 4: boosting

<table>
<thead>
<tr>
<th>Data</th>
<th>Label</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

“Strong” learner

- Given: a reasonable amount of training data, a target error rate \( \varepsilon \), and a failure probability \( p \).
- A strong learning algorithm will produce a classifier with error rate \( < \varepsilon \) with probability \( 1-p \).

“Weak” learner

- Given: a reasonable amount of training data, a failure probability \( p \).
- A weak learning algorithm will produce a classifier with error rate \( < 0.5 \) with probability \( 1-p \).

Weak learners are much easier to create!
weak learners for boosting

Data | Label | Weight
-----|-------|-------
0    | 0     | 0.2   
0    | 0     | 0.2   
1    | 0     | 0.2   
1    | 0     | 0.2   
0    | 0     | 0.2   

Need a weak learning algorithm that can handle weighted examples

boosting: basic algorithm

Training:
- start with equal example weights
- for some number of iterations:
  - learn a weak classifier and save
  - change the example weights

Classify:
- get prediction from all learned weak classifiers
- weighted vote based on how well the weak classifier did when it was trained (i.e., in relation to training error)

boosting basics

Start with equal weighted examples

Weights

Examples: E1, E2, E3, E4, E5

Learn a weak classifier:

We want to reweight the examples and then learn another weak classifier

How should we change the example weights?
Boosting

- decrease the weight for those we're getting correct
- increase the weight for those we're getting incorrect

Learn another weak classifier:

- decrease the weight for those we're getting correct
- increase the weight for those we're getting incorrect
Classifying

- weighted vote based on how well they classify the training data
- weak_2_vote > weak_1_vote since it got more right

Notation

- $x_i$ example $i$ in the training data
- $w_i$ weight for example $i$, we will enforce:
  - $w_i \geq 0$
  - $\sum_{i=1}^{n} w_i = 1$
- classifier_i(x) $+1/-1$ prediction of classifier $k$ example $i$

AdaBoost: train

for $k = 1$ to iterations:
- classifier_k = learn a weak classifier based on weights
- calculate weighted error for this classifier
  \[ \varepsilon_k = \sum_{i=1}^{n} w_i \cdot I[\text{label} \neq \text{classifier}_k(x_i)] \]
- calculate “score” for this classifier:
  \[ \alpha_k = \frac{1}{2} \log \left( \frac{1 - \varepsilon_k}{\varepsilon_k} \right) \]
- change the example weights
  \[ w_i = \frac{1}{Z} w_i \exp(-\alpha_k \cdot \text{label} \cdot \text{classifier}_k(x_i)) \]
**AdaBoost: train**

classifier\(_k\) = learn a weak classifier based on weights

weighted error for this classifier is:

\[
\varepsilon_k = \sum_{i=1}^{n} w_i \cdot \mathbb{I}[\text{label} \neq \text{classifier}(x_i)]
\]

What is the range of possible values?

- prediction
- did we get the example wrong
- weighted sum of the errors/mistakes

**AdaBoost: train**

classifier\(_k\) = learn a weak classifier based on weights

weighted error for this classifier is:

\[
\varepsilon_k = \sum_{i=1}^{n} w_i \cdot \mathbb{I}[\text{label} \neq \text{classifier}(x_i)]
\]

Between 0 (if we get all examples right) and 1 (if we get them all wrong)

- prediction
- did we get the example wrong
- weighted sum of the errors/mistakes

**AdaBoost: train**

classifier\(_k\) = learn a weak classifier based on weights

"score" or weight for this classifier is:

\[
\alpha_k = \frac{1}{2} \log \left( \frac{1 - \varepsilon_k}{\varepsilon_k} \right)
\]

What does this look like (specifically for errors between 0 and 1)?

**AdaBoost: train**

classifier\(_k\) = learn a weak classifier based on weights

"score" or weight for this classifier is:

\[
\alpha_k = \frac{1}{2} \log \left( \frac{1 - \varepsilon_k}{\varepsilon_k} \right)
\]

- ranges from \(\infty\) to \(-\infty\)
- for most reasonable values: ranges from 1 to -1
- errors of 50\% \(\equiv 0\)
- error < 50\% \(\equiv\) positive error > 50\% \(\equiv\) negative
AdaBoost: classify

\[ \text{classify}(x) = \text{sign} \left( \sum_{k=1}^{N_{\text{classifier}}} \alpha_k \cdot \text{classifier}_k(x) \right) \]

What does this do?

The weighted vote of the learned classifiers weighted by \( \alpha \) (remember \( \alpha \) generally varies from 1 to -1 training error)

What happens if a classifier has error >50%?

AdaBoost: train, updating the weights

update the example weights

\[ w_i = \frac{1}{Z} w_i \exp(-\alpha_k \cdot \text{label}_i \cdot \text{classifier}_k(x_i)) \]

Remember, we want to enforce:

\[ w_i \geq 0 \]
\[ \sum_{i=1}^{N_{\text{classifier}}} w_i = 1 \]

Z is called the normalizing constant. It is used to make sure that the weights sum to 1

What should it be?
AdaBoost: train

update the example weights

\[ w_i = \frac{1}{Z} w_i \exp(-\alpha_k \cdot \text{label}_k \cdot \text{classifier}(x_i)) \]

Remember, we want to enforce:

\[ w_i \geq 0 \]
\[ \sum w_i = 1 \]

normalizing constant (i.e. the sum of the “new” \( w_i \)):

\[ Z = \sum w_i \exp(-\alpha_k \cdot \text{label}_k \cdot \text{classifier}(x_i)) \]

What does this do?

Note: only change weights based on current classifier (not all previous classifiers)

correct \( \rightarrow \) positive
incorrect \( \rightarrow \) negative

correct \( \rightarrow \) small value
incorrect \( \rightarrow \) large value
AdaBoost: train

update the example weights

\[ w_i = \frac{1}{Z} w_i \exp(-\alpha_k \cdot \text{label}_k \cdot \text{classifier}_k(x_i)) \]

What does the \( \alpha \) do?

- If the classifier was good (<50% error) \( \alpha \) is positive: trust classifier output and move as normal
- If the classifier was bad (>50% error) \( \alpha \) is negative: classifier is so bad, consider opposite prediction of classifier

Exponential loss!

\[ l(y, y') = \exp(-yy') \]

AdaBoost turns out to be another approach for minimizing the exponential loss!
Other boosting variants

Loss
Correct

Mistakes

Brownboost

Logitboost

0-1 loss

Adaboost = $e^{-\gamma(y^*x)}$

Boosting example

Start with equal weighted data set

Boosting example

weak learner = line

What would be the best line learned on this data set?

$h \Rightarrow p(\text{error}) = 0.5$ it is at chance

Boosting example

This one seems to be the best

How should we reweight examples?

This is a ‘weak classifier’: It performs slightly better than chance.
What would be the best line learned on this data set?

How should we reweight examples?
The strong (non-linear) classifier is built as the combination of all the weak (linear) classifiers.

**AdaBoost: train**

for \( k = 1 \) to iterations:
- \( f_k \) = learn a weak classifier based on weights
- weighted error for this classifier is:
- “score” or weight for this classifier is:
- change the example weights

What can we use as a classifier?

- Anything that can train on weighted examples
- For most applications, must be fast!

Why?
One of the most common classifiers to use is a decision tree:
- can use a shallow (2-3 level tree)
- even more common is a 1-level tree
  - called a decision stump
  - asks a question about a single feature

What does the decision boundary look like for a decision stump?

Boosting in practice
Very successful on a wide range of problems

One of the keys is that boosting tends not to overfit, even for a large number of iterations

Using <10,000 training examples can fit >2,000,000 parameters!
Rapid Object Detection using a Boosted Cascade of Simple Features

Paul Viola, Michael Jones
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Rapid Object Detection using a Boosted Cascade of Simple Features
... overlap. Each partition yields a simple feature detector. The ... and a Real-World Test. We tested our system on the MIT+CMU frontal face test set [5].
This set consists of 100 images with 607 labeled frontal faces. A ... Cited by 842 Related articles: All 128 versions. Cite. Save. More.

To give you some context of importance:
The anatomy of a large-scale hypertextual web search engine
S. Brin, L. Page -- Computer networks and ISDN systems, 1998 - ScienceDirect.com
This paper describes a machine learning approach for visual object detection which is capable of processing images extremely rapidly, and achieving high detection rates. This work is distinguished by three key contributions. The first is the introduction of a new image...
“weak” learners

4 Types of "Rectangle filters"
(Similar to Haar wavelets
Papageorgiou, et al.)

Based on 24x24 grid:
160,000 features to choose from

\[ g(x) = \frac{\text{sum(WhiteArea)}}{\text{sum(BlackArea)}} \]

94

“weak” learners

\[ F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \ldots \]

\[ f_i(x) = \begin{cases} 1 & \text{if } g_i(x) > \theta_i \\ -1 & \text{otherwise} \end{cases} \]

95

Example output

96

Solving other “Face” Tasks

Facial Feature Localization
Profile Detection
Demographic Analysis

97
“weak” classifiers learned

Bagging vs Boosting

Popular Ensemble Methods: An Empirical Study

David Opits  
Department of Computer Science  
University of Minnesota  
Minneapolis, MN 55455 USA

Richard Maelin  
Computer Science Department  
University of Minnesota  
Duluth, MN 55812 USA


Boosting Neural Networks

Boosting Decision Trees

Change in error rate over standard classifier

Ada-Boosting
Arcing
Bagging

White bar represents 1 standard deviation