Ensemble learning

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

---

Admin

- Class last Tuesday?
- Assignment grading
- Assignment 9
- Midterm 2
- Final project
  - 11/27 (Wed) submit project proposal
Ensemble learning

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

---

**Testing**

example to label

\[ \text{model 1} \rightarrow \text{prediction 1} \]
\[ \text{model 2} \rightarrow \text{prediction 2} \]
\[ \vdots \]
\[ \text{model m} \rightarrow \text{prediction m} \]

How do we decide on the final prediction?

---

**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?**

---

**Testing**

- take majority vote
- if they output probabilities, take a weighted vote

**How does having multiple classifiers help?**

---

**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>
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>.6*.6*.6=0.216</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>I</td>
<td>.6*.6*.4=0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>C</td>
<td>.6*.4*.6=0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>I</td>
<td>.6*.4*.4=0.096</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>C</td>
<td>.4*.6*.6=0.144</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>I</td>
<td>.4*.6*.4=0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>C</td>
<td>.4*.4*.6=0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>.4*.4*.4=0.064</td>
</tr>
</tbody>
</table>

0.096+ 0.096+ 0.096+ 0.064 = 35% error!

Benefits of ensemble learning

3 classifiers in general, for $r = \text{probability of mistake for individual classifier}$:

$$p(\text{error}) = 3r^2(1-r) + r^3$$

Binomial distribution

<table>
<thead>
<tr>
<th>$r$</th>
<th>$p(\text{error})$</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>0.0073</td>
</tr>
</tbody>
</table>

Benefits of ensemble learning

5 classifiers in general, for $r = \text{probability of mistake for individual classifier}$:

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

<table>
<thead>
<tr>
<th>$r$</th>
<th>$p(\text{error})$</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>0.0073</td>
</tr>
</tbody>
</table>

Benefits of ensemble learning

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

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

(Cumulative probability distribution for the binomial distribution)
Given enough classifiers...

\[
p(\text{error}) = \sum_{i=0}^{m} \binom{m}{i} r^i (1-r)^{m-i}
\]
\[r = 0.4\]

What is 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

Pros:
- Lots of existing classifiers already
- Can work well for some problems

Cons/concerns:
- Often, classifiers are not independent, that is, they make the same mistakes!
  - e.g. many of these classifiers are linear models
  - Voting won’t help us if they’re making the same mistakes

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 → learning alg → model 1
... ... ...
Training Data m → learning alg → model m

data generating distribution

Ideal situation

Training data 1  Training data 2  ...

Use training data as a proxy for the data generating distribution
sampling with replacements

"Training" data 1

Training data

pick a random example from the real training data

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

add it to the new "training" data
**Sampling with replacements**

- "Training" data 1
- Pick another random example
- Training data
- Keep going until you've created a new "training" data set

**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
Won't these all be basically the same?

For a data set of size n, what is the probability that a given example will \textbf{NOT} be select in a "new" training set sampled from the original?

What is the probability it isn't chosen the first time?

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

What is the probability it isn't chosen \textbf{any} of the n times?

\[ (1 - \frac{1}{n})^n \]

Each draw is independent and has the same probability.
When does bagging work

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
  - Prone to overfitting

Often has 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>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

“training” data 2

<table>
<thead>
<tr>
<th>Data</th>
<th>Label</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

“training” data 3

<table>
<thead>
<tr>
<th>Data</th>
<th>Label</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.05</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>1</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

“Strong” learner

Given
- a reasonable amount of training data
- a target error rate $\varepsilon$
- 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

<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>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Which of our algorithms can handle weights?

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:

Boosting

Weights: [ ] [ ] [ ] [ ]
Examples: E1 E2 E3 E4 E5

classified correct classified incorrect

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

How should we change the example weights?

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

Weights

Examples
E1  E2  E3  E4  E5

Learn another weak classifier:

Boosting

Weights

Examples
E1  E2  E3  E4  E5

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

Boosting

Weights

Examples
E1  E2  E3  E4  E5

Learn another weak classifier:

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: $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$
- $\text{classifier}_k(x_i)$: $+1/-1$ prediction of classifier $k$ example $i$

AdaBoost: train

For $k = 1$ to iterations:
- $\text{classifier}_k = \text{learn a weak classifier based on weights}$
- calculate weighted error for this classifier
  
  $\epsilon_k = \sum_{i=1}^{n} w_i \cdot [\text{label} \neq \text{classifier}_k(x_i)]$

- calculate "score" for this classifier:
  
  $\alpha_k = \frac{1}{2} \log \left( \frac{1-\epsilon_k}{\epsilon_k} \right)$

- change the example weights
  
  $w_{i} = \frac{1}{Z} w_i \exp(-\alpha_k \cdot \text{label} \neq \text{classifier}_k(x_i))$

What does this say?

Weighted error for this classifier is:

$$\epsilon_k = \sum_{i=1}^{n} w_i \cdot [\text{label} \neq \text{classifier}_k(x_i)]$$

What is the range of possible values? 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:

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

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

weighted sum of the errors/mistakes

“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: classify

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

What does this do?

- ranges from $-\infty$ to $-\infty$
- for most reasonable values: ranges from -1 to 1
- errors of 50% = 0
**AdaBoost: classify**

classify(x) = \text{sign}\left( \sum_{k=1}^{\infty} \alpha_k * \text{classifier}_k(x) \right)

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

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

Remember, we want to enforce:

\[ w_i \geq 0 \]
\[ \sum_{i=1}^{\infty} w_i = 1 \]

\( Z \) 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

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

Remember, we want to enforce:

\[ w_i \geq 0 \]
\[ \sum_{i=1}^{\infty} w_i = 1 \]

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

\[ Z = \sum_{i=1}^{\infty} w_i \exp(-\alpha_k * \text{label}_i * \text{classifier}_k(x_i)) \]
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 this do?

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

update the example weights

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

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

---

AdaBoost justification

update the example weights

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

Does this look like anything we’ve seen before?

---

AdaBoost justification

update the example weights

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

Exponential loss!
$$l(y, y') = \exp(-yy')$$

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

---

Other boosting variants

Adaboost = $e^{-y(u^*)}$

Logitboost
Brownboost
0-1 loss
Mistakes
Correct

---
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 => p(error) = 0.5 it is at chance

Boosting example

This one seems to be the best
This is a ‘weak classifier’: It performs slightly better than chance.

Boosting example

How should we reweight examples?

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

reds on this side get less weight
blues on this side get more weight

reds on this side get more weight
blues on this side get less weight
How should we reweight examples?

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

The strong (non-linear) classifier is built as the combination of all the weak (linear) classifiers.
AdaBoost: train

for k = 1 to iterations:
- classifier_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?

AdaBoost: train

for k = 1 to iterations:
- classifier_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
- Anything that can train on weighted examples
- For most applications, must be fast!
  Why?

Boosted decision stumps

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?
Boosted decision stumps

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 boosted decision stumps?

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!

Adaboost application example:
face detection

- Linear classifier!
  - Each stump defines the weight for that dimension
  - If you learn multiple stumps for that dimension then it’s the weighted average
Adaboost application example: face detection

To give you some context of importance:

Rapid object detection using a Boosted Cascade of Simple Features

Paul Viola
Viola@eecsnet.com
Mitsubishi Electric Research Labs
201 Broadway, 8th Fl.
Cambridge, MA 02139

Michael Jones
mjones@eecsnet.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

Rapid object detection using a boosted cascade of simple features

This work 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

g(x) = \sum(\text{WhiteArea}) - \sum(\text{BlackArea})

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

Based on 24x24 grid:
160,000 features to choose from
“weak” learners

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

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

Example output

Solving other “Face” Tasks

Facial Feature Localization
Profile Detection
Demographic Analysis

“weak” classifiers learned
Bagging vs Boosting

Popular Ensemble Methods: An Empirical Study

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

Richard Maclin
Computer Science Department
University of Minnesota
Minneapolis, MN 55455 USA

Change in error rate over standard classifier

Ada-Boosting
Arcing
Bagging
White bar represents 1 standard deviation