Admin

Assignment grading

Assignment 9

Midterm 2

Final project
  - No formal class Tuesday: figure out project ideas
  - 11/23 (Wed) submit project proposal

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**
  - learning alg → model 1
  - learning alg → model 2
  - ... → model m

- **Testing**
  - example to label
  - model 1 → prediction 1
  - model 2 → prediction 2
  - ... → 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)

- How does having multiple classifiers help us?

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)

**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>0.6\times0.6\times0.6=0.216</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>I</td>
<td>0.6\times0.6\times0.4=0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>C</td>
<td>0.6\times0.4\times0.6=0.144</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>I</td>
<td>0.6\times0.4\times0.4=0.096</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>C</td>
<td>0.4\times0.6\times0.6=0.144</td>
</tr>
<tr>
<td>I</td>
<td>C</td>
<td>I</td>
<td>0.4\times0.6\times0.4=0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>C</td>
<td>0.4\times0.4\times0.6=0.096</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>0.4\times0.4\times0.4=0.064</td>
</tr>
</tbody>
</table>

**Benefits of ensemble learning**

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

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

**Benefits of ensemble learning**

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

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

<table>
<thead>
<tr>
<th>$r$</th>
<th>$p(\text{error})$</th>
<th>$p(\text{error})$</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>
Benefits of ensemble learning

Given enough classifiers...

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

Obtaining independent classifiers

Idea 1: different learning methods

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
data generating distribution

Ideal situation

bagging

sampling with replacements

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

"Training" data 1

pick a random example from the real training data

Training data

sampling with replacements

"Training" data 1

add it to the new "training" data

Training data

sampling with replacements

"Training" data 1

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

Training data

sampling with replacements

"Training" data 1

pick another random example

Training data
sampling with replacements

"Training” data 1

pick another random example

Training data

sampling with replacements

"Training” data 1

keep going until you’ve created a new “training” data set

Training data

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

Training Data 1

Won’t these all be basically the same?

Training Data m
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?

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

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

What is the probability it isn’t chosen any of the \( n \) times?

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

Each draw is independent and has the same probability

Converges very quickly to \( 1/e \approx 63\% \)
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 a third of the original noisy examples

For some classifiers that have trouble with noisy classifiers, this can help

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

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

<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>0.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

<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>0.1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.5</td>
<td></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: E1 E2 E3 E4 E5

Examples: E1 E2 E3 E4 E5

Learn a weak classifier:

---

**Boosting**

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

**Learn another weak classifier:**
Boosting

**Weights:**

- weak 1
- weak 2
- weak 3
- weak 4
- weak 5

**Examples:**

- E1
- E2
- E3
- E4
- E5

Examples:

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

Classifying

- prediction 1
- prediction 2

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 enforce: $w_i \geq 0$
- $\sum w_i = 1$
- $\text{classifier}_k(x_i)$: +1/-1 prediction of classifier $k$ for example $i$
AdaBoost: train
for $k = 1$ to iterations:
- classifier$_k$ = learn a weak classifier based on weights
- calculate weighted error for this classifier
  \[ \varepsilon_k = \sum_{i=1}^{n} w_i \cdot [\text{label} \neq \text{classifier}_k(x_i)] \]
- calculate “score” for this classifier:
  \[ \alpha_k = \frac{1}{2} \log \left( \frac{1 - \varepsilon_k}{\varepsilon_k} \right) \]
- change the example weights
  \[ w_i = \frac{1}{Z} w_i \exp \left( -\alpha_k \cdot \text{label} \cdot \text{classifier}_k(x_i) \right) \]

What does this say?

What is the range of possible values?

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

weighted sum of the errors/mistakes
AdaBoost: train

classifierᵢ = learn a weak classifier based on weights

"score" or weight for this classifier is:

$$αᵢ = \frac{1}{2} \log \left( \frac{1 - eᵢ}{eᵢ} \right)$$

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

- ranges from +∞ to −∞
- for most reasonable values: ranges from −1 to 1
- errors of 50% = 0

AdaBoost: classify

classify(x) = sign \left( \sum_{i=1}^{T} αᵢ * classifierᵢ(x) \right)

What does this do?

The weighted vote of the learned classifiers weighted by α (remember α generally varies from −1 to −1 training error)

What happens if a classifier has error >50%
AdaBoost: classify

\[ \text{classify}(x) = \text{sign} \left( \sum_{i=1}^{T} \alpha_i \cdot \text{classifier}_i(x) \right) \]

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

We actually vote the opposite!

---

AdaBoost: train, updating the weights

update the example weights

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

Remember, we want to enforce:

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

\( Z \) 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

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

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

\[ Z = \sum_{i=1}^{T} w_i \exp(-\alpha_i \cdot \text{label} \cdot \text{classifier}_i(x_i)) \]

What does this do?
AdaBoost: train

update the example weights

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

<table>
<thead>
<tr>
<th>correct</th>
<th>positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>incorrect</td>
<td>negative</td>
</tr>
</tbody>
</table>

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

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 back (>50% error) \( \alpha \) is negative classifier is so bad, consider opposite prediction of classifier
AdaBoost: train

update the example weights

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

- correct: positive
- incorrect: negative
- correct: small value
- incorrect: large value

If the classifier was good (<50% error) \( \alpha \) is positive
If the classifier was back (>50% error) \( \alpha \) is negative

---

AdaBoost justification

update the example weights

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

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

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

---

Other boosting variants

- Logitboost
- Brownboost
- 0-1 loss

Adaboost = \( e^{-\gamma(y'y)} \)

- Loss
- Correct
- Mistakes
11/17/16

Start with equal weighted data set

Boosting example

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

weak learner = line

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

This one seems to be the best

This is a 'weak classifier': it performs slightly better than chance.

How should we reweight examples?

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

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

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:
- \( \text{classifier}_k = \text{learn a weak classifier based on weights} \)
- 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:
- \( \text{classifier}_k = \text{learn a weak classifier based on weights} \)
- 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?
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?

Boosted decision stumps

Linear classifier!
- Each stump defines the weight for that dimension
- If you learn multiple stumps for that dimension then it's the weighted average

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
Adaboost application example: face detection

To give you some context of importance:

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

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

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

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 Massachusetts
Amherst, MA 01003 USA

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


Boosting Neural Networks

Change in error rate over standard classifier

Ada-Boosting
Arcing
Bagging

White bar represents 1 standard deviation

Boosting Decision Trees