Machine Learning (CSE 446): Practical Issues

Noah Smith

University of Washington
nasmith@cs.washington.edu

October 18, 2017

scary words

Outline of CSE 446

We've already covered stuff in blue!

- Problem formulations: classification, regression
- Supervised techniques: decision trees, nearest neighbors, perceptron, linear models, probabilistic models, neural networks, kernel methods
- Unsupervised techniques: clustering, linear dimensionality reduction
- "Meta-techniques": ensembles, expectation-maximization
- Understanding ML: limits of learning, practical issues, bias & fairness
- Recurring themes: (stochastic) gradient descent, bullshit detection

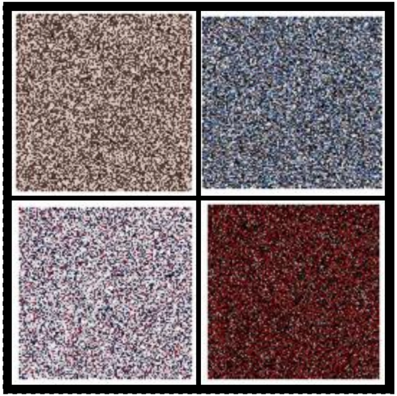
Today: (More) Best Practices

You already know:

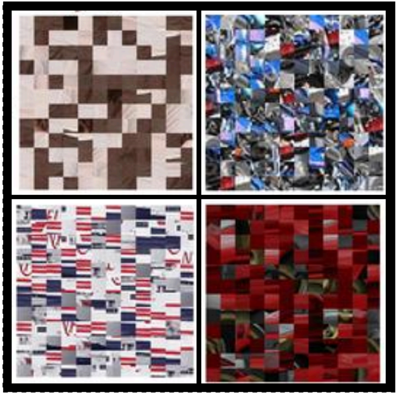
- Separating training and test data
- Hyperparameter tuning on development data

Understanding machine learning is partly about knowing algorithms and partly about the art of mapping abstract problems to learning tasks.

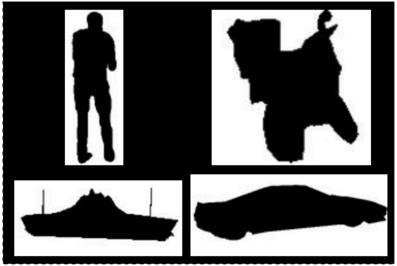
Features Matter



Features Matter



Features Matter



Features Matter

Irrelevant Features

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!
- K-nearest neighbors?

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!

- K-nearest neighbors? ☹

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!
- K-nearest neighbors? ☹
- Perceptron?

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!

- K-nearest neighbors? ☹
- Perceptron? ☺

Irrelevant Features

One irrelevant feature isn't a big deal; what we're worried about is when irrelevant features *outnumber* useful ones!

- Decision trees (not too deep)?
  Somewhat protected, but beware spurious correlations!

- K-nearest neighbors? ☹
- Perceptron? ☺

What about *redundant* features ϕ_j and $\phi_{j'}$ such that $\phi_j \approx \phi_{j'}$?

Technique: Feature Pruning

If a binary feature is present in too small or too large a fraction of D, remove it.

Technique: Feature Pruning

If a binary feature is present in too small or too large a fraction of D, remove it.

Example: $\phi(x) = [\![$the word *the* occurs in document $x]\!]$

Technique: Feature Pruning

If a binary feature is present in too small or too large a fraction of D, remove it.

Example: $\phi(x) = [\![$the word *the* occurs in document $x]\!]$

Generalization: if a feature has variance (in D) **lower** than some threshhold value, remove it.
Note: in lecture, I mistakenly said to remove high-variance features. Mea culpa.

$$\text{sample\_mean}(\phi; D) = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n) \qquad \text{(call it “}\bar{\phi}\text{”)}$$

$$\text{sample\_variance}(\phi; D) = \frac{1}{N-1} \sum_{n=1}^{N} \left(\phi(x_n) - \bar{\phi}\right)^2 \qquad \text{(call it “Var}(\phi)\text{”)}$$

Technique: Feature Normalization

Center a feature:

$$\phi(x) \to \phi(x) - \bar{\phi}$$

(This was a required step for principal components analysis!)

Scale a feature. Two choices:

$$\phi(x) \to \frac{\phi(x)}{\sqrt{\mathsf{Var}(\phi)}} \qquad \text{“variance scaling”}$$

$$\phi(x) \to \frac{\phi(x)}{\max_n |\phi(x_n)|} \qquad \text{“absolute scaling”}$$

Technique: Feature Normalization

Center a feature:

$$\phi(x) \to \phi(x) - \bar{\phi}$$

(This was a required step for principal components analysis!)

Scale a feature. Two choices:

$$\phi(x) \to \frac{\phi(x)}{\sqrt{\mathrm{Var}(\phi)}} \qquad \text{“variance scaling”}$$

$$\phi(x) \to \frac{\phi(x)}{\max_n |\phi(x_n)|} \qquad \text{“absolute scaling”}$$

Remember that you'll need to normalize test data before you test!

Techniques: Creating New Features

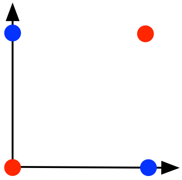
1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

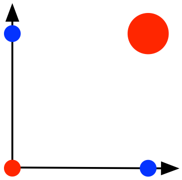


The classic "xor" problem: these points are *not* linearly separable.

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

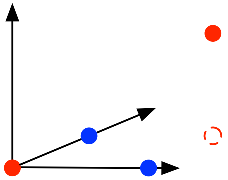


Define $\mathbf{x}[3] = \mathbf{x}[1] \wedge \mathbf{x}[2]$.

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

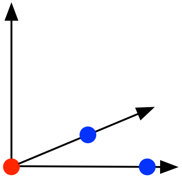


Rotating the view.

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

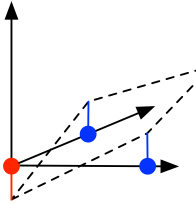
$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$



Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$



$$2 \cdot \mathbf{x}[1] + 2 \cdot \mathbf{x}[2] - 4 \cdot \mathbf{x}[3] - 1 = 0$$

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

Generalization: take the *product* of two features.

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

   Generalization: take the *product* of two features.

2. Even more generally, we can create conjunctions (or products) using as many features as we'd like.

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

   $$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

   Generalization: take the *product* of two features.
2. Even more generally, we can create conjunctions (or products) using as many features as we'd like.
   This is one view of what decision trees are doing!
   - Every leaf's path (from root) is a conjunction feature.
   - Why not build decision trees, extract the features and toss them into the perceptron?

Techniques: Creating New Features

1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$

   Generalization: take the *product* of two features.

2. Even more generally, we can create conjunctions (or products) using as many features as we'd like.
   This is one view of what decision trees are doing!
   - Every leaf's path (from root) is a conjunction feature.
   - Why not build decision trees, extract the features and toss them into the perceptron?

3. Transformations on features can be useful. For example,

$$\phi(x) \rightarrow \text{sign}(\phi(x)) \cdot \log\left(1 + |\phi(x)|\right)$$

   Example: $\phi(x)$ is the count of the word *cool* in document x.

Evaluation

Accuracy:

$$\begin{aligned} \mathrm{A}(f) &= \sum_{x} \mathcal{D}(x, f(x)) \\ &= \sum_{x,y} \mathcal{D}(x, y) \cdot \begin{cases} 1 & \text{if } f(x) = y \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{x,y} \mathcal{D}(x, y) \cdot [\![f(x) = y]\!] \end{aligned}$$

where $\mathcal{D}$ is the *true* distribution over data. Error is $1 - \mathrm{A}$; earlier we denoted error "$\epsilon(f)$."

This is *estimated* using a test dataset $\langle x_1, y_2 \rangle, \ldots, \langle x_{N'}, y_{N'} \rangle$:

$$\hat{\mathrm{A}}(f) = \frac{1}{N'} \sum_{i=1}^{N'} [\![f(x_i) = y_i]\!]$$

Issues with Test-Set Accuracy

Issues with Test-Set Accuracy

- Class imbalance: if $\mathcal{D}(*, \text{not spam}) = 0.99$, then you can get $\hat{\text{A}} \approx 0.99$ by always guessing "not spam."

Issues with Test-Set Accuracy

- Class imbalance: if $\mathcal{D}(*, \text{not spam}) = 0.99$, then you can get $\hat{\mathrm{A}} \approx 0.99$ by always guessing "not spam."
- Relative importance of classes or cost of error types.

Issues with Test-Set Accuracy

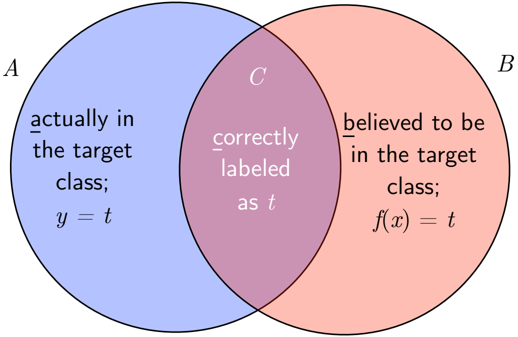
- Class imbalance: if $\mathcal{D}(*, \text{not spam}) = 0.99$, then you can get $\hat{\text{A}} \approx 0.99$ by always guessing "not spam."
- Relative importance of classes or cost of error types.
- Variance due to the test data.

Evaluation in the Two-Class Case

Suppose we have two classes, and one of them, t, is a "target."

- E.g., given a query, find relevant documents.

Precision and **recall** encode the goals of returning a "pure" set of targeted instances and capturing *all* of them.



$$\hat{\mathrm{P}}(f) = \frac{|C|}{|B|} = \frac{|A \cap B|}{|B|}$$

$$\hat{\mathrm{R}}(f) = \frac{|C|}{|A|} = \frac{|A \cap B|}{|A|}$$

$$\hat{F}_1(f) = 2 \cdot \frac{\hat{\mathrm{P}} \cdot \hat{\mathrm{R}}}{\hat{\mathrm{P}} + \hat{\mathrm{R}}}$$

Another View: Contingency Table

	$y = t$	$y \neq t$	
$f(x) = t$	C (true positives)	$B \setminus C$ (false positives)	B
$f(x) \neq t$	$A \setminus C$ (false negatives)	(true negatives)	
	A		