Machine Learning: Yoshinari Fujinuma

University of Colorado Boulder

LECTURE 7

Slides adapted from Chenhao Tan, Noah Smith, Chris Ketelsen

Logistics

- HW1 due on Friday, ask questions

Learning objectives

How do you represent the data?

- Understand why features matter
- Understand feature engineering techniques

Outline

Features matter

Feature engineering techniques

Outline

Learning Objective: More Best Practices

You already know:

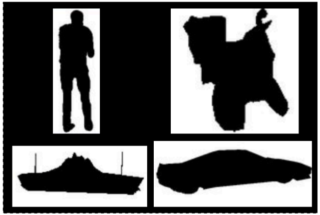
- Separating training and test data
- Hyperparameter tuning on development data

Today:

- Adding, scaling, pruning features

Features Matter

Shape representation



Features Matter

Original

Features Matter

Bag of words

a, algorithms, and, applications, arthur, artificial, building, by, can, coined, computational, computer, computing, construction, data, decisions, designing, detection, difficult, driven, email, employed, evolved, example, explicit, explores, filtering, following, from, good, in, include, infeasible, inputs, instructions, intelligence, intruders, is, learn, learning, machine, make, making, model, name, network, of, on, or, overcome, pattern, performance, predictions, program, programming, range, recognition, sample, samuel, static, strictly, study, such, tasks, that, the, theory, through, vision, was, where, with

Features Matter

Original

The name machine learning was coined in 1959 by Arthur Samuel.[1] Evolved from the study of pattern recognition and computational learning theory in artificial intelligence,[3] machine learning explores the study and construction of algorithms that can learn from and make predictions on data[4] - such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions,[5]:2 through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision.

Outline

Features matter

Feature engineering techniques

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!

- 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!

- 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!

- K-nearest neighbors?
- Perceptron?

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

Irrelevant Features

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

- K-nearest neighbors?
- Perceptron?

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

Irrelevant Features

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

- K-nearest neighbors? ☹
- Perceptron?

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

Irrelevant Features

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

- K-nearest neighbors? ☹
- Perceptron?

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

Irrelevant Features

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

- 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.

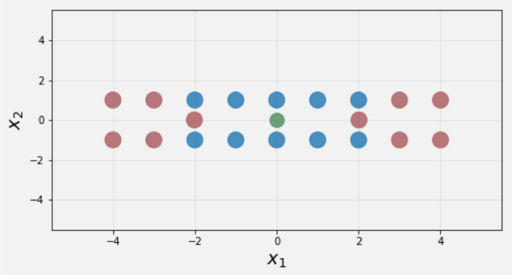
Example: $\phi(x) = [\![$the word *the* occurs in document $x]\!]$

Generalization: if a feature has variance (in D) **lower** than some threshold value, remove it.

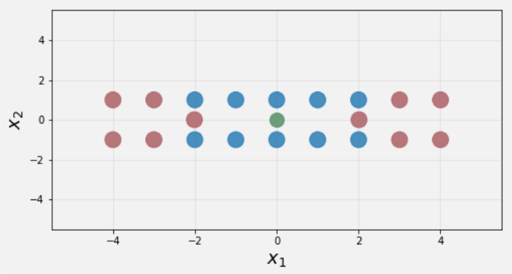
$$\text{sample\_mean}(\phi; D) = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n) \qquad \text{(call it "}\bar{\phi}\text{")}$$

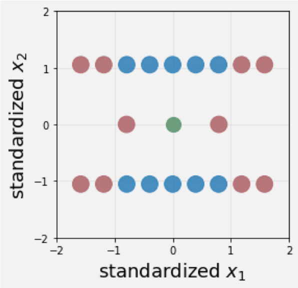
$$\text{sample\_variance}(\phi; D) = \frac{1}{N-1} \sum_{n=1}^{N} \left(\phi(x_n) - \bar{\phi}\right)^2 \qquad \text{(call it "Var}(\phi)\text{")}$$

Technique: Feature Normalization



Technique: Feature Normalization





Technique: Feature Normalization

Standardization

$$\phi(x) \rightarrow \frac{\phi(x) - \bar{\phi}}{\sqrt{\mathrm{Var}(\phi)}}$$

Absolute scaling

$$\phi(x) \rightarrow \frac{\phi(x)}{\max_n |\phi(x_n)|}$$

Technique: Feature Normalization

Standardization

$$\phi(x) \rightarrow \frac{\phi(x) - \bar{\phi}}{\sqrt{\text{Var}(\phi)}}$$

Absolute scaling

$$\phi(x) \rightarrow \frac{\phi(x)}{\max_n |\phi(x_n)|}$$

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

Technique: Example Normalization

We have been talking about normalizing columns.

We can also normalize **rows**. e.g., l_2 normalization

$$\boldsymbol{x} = \frac{\boldsymbol{x}}{||\boldsymbol{x}||_2} = \frac{\boldsymbol{x}}{\sqrt{\sum_j \boldsymbol{x}[j]^2}}$$

e.g., if $\boldsymbol{x} = (1, 2, 3)$, then

$$\boldsymbol{x} = (\frac{1}{\sqrt{1+4+9}}, \frac{2}{\sqrt{1+4+9}}, \frac{3}{\sqrt{1+4+9}})$$

Techniques: Creating New Features

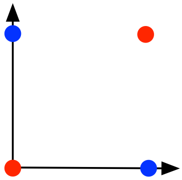
1. Consider two binary features, ϕ_j and $\phi_{j'}$. A new *conjunction* feature can be defined by:

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

Techniques: Creating New Features

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

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

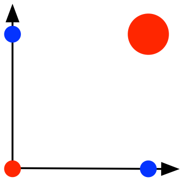


The classic “xor” problem: these points are *not* linearly separable.

Techniques: Creating New Features

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

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

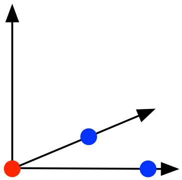


Define $\boldsymbol{x}[3] = \boldsymbol{x}[1] \wedge \boldsymbol{x}[2]$.

Techniques: Creating New Features

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

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

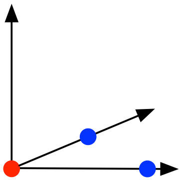


Rotating the view.

Techniques: Creating New Features

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

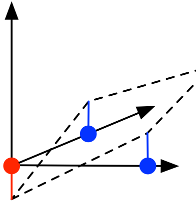
$$\phi_{j \wedge j'}(x) = \phi_j(x) \wedge \phi_{j'}(x)$$



Techniques: Creating New Features

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

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



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

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.

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.
3. Transformations on features can be useful. E.g., log-scaling,

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

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

Techniques: Creating New Features

Remember that adding features does not always bring benefits.

Could be irrelevant, redundant, or features that make linearly separable datasets not linearly separable.

Feature engineering

Features are really important for machine learning.

Feature engineering

Features are really important for machine learning.

- Pruning
- Normalization
- Creating new features

Feature engineering

Features are really important for machine learning.

- Pruning
- Normalization
- Creating new features

In practice, feature engineering requires a deep understanding of the problem.