CS 446 Machine Learning Fall 2016 Aug 25, 2016

Introduction to Machine Learning

Professor: Dan Roth *Scribe: Ben Zhou, C. Cervantes, C. Cheng*

1 Supervised Learning

1.1 The Badges Game

The key learning protocol in this class is **supervised learning**: given labeled data, learn a model that can predict labels on unseen data.

In the badges game, we could say that a function is correct if it is consistent with all the given examples. However, this justification enables us to say that the list of badges is itself a function; it gets all the names correct, but it has no predictive power.

The solution *the second letter of the first name is a vowel* may be correct and concise, but given the data (list of names), there is no notion of vowels distinct from other characters. Representation – and necessary external knowledge – is crucial to developing a good model.

1.2 Introduction

We consider systems that apply function f to input items x and return an output $y = f(x)$

Given an **instance space** X and a **label space** Y, there exists some target function $y = f(x)$ where $y \in Y$, so that for any $x \in X$, this function outputs the correct y in the label space. A supervised learner deals with a system where $f(x)$ is learned from a given set of (x, y) pairs.

We want this learning when we don't know what f is. Our goal is to find a function $g(x)$ that is close (preferably identical) to $f(x)$.

1.3 Training

Given a set of labeled training data $D^{train} = (x_1, y_1), (x_2, y_2)...(x_n, y_n)$, we want to use learning algorithms to find a learned model $g(x)$.

Other learning protocols may include

Giving examples X without labels Y

Giving labeled examples (x, y) one at a time and adjusting the model after each

1.4 Testing

We typically reserve some labeled data for testing – D^{test} – and compare the output of the algorithm with the true label (the example is correct if $g(x_1) == y_1$). If we have some notion of a distance – measuring how far $g(x)$ is from y – then we can measure error: how incorrect is $g(x)$.

1.5 Definitions

Supervised Learning Given examples $(x, f(x))$ of some unknown function f, we want to find a good approximation of f
Feature Extraction The process of mapping a domain element into a representation x
Target Function The domain of $f(x)$: binary classification ($f(x) \in \{-1, +1\}$); multiclass classification ($f(x) \in \{1, 2, ..., k\}$); real valued number ($f(x) \in \mathbb{R}$)

1.6 Examples

1. Disease diagnosis: x is the properties of patient; f is disease
2. Part-of-Speech tagging: x is an English sentence; f is the POS tag of a word
3. Face recognition: x is a bitmap picture of person's face; f is the name (a property of) the person
4. Automatic Steering: x is bitmap picture of road surface in front of car; f is degrees to turn the steering wheel

2 Key issues in Machine Learning

Modeling

How do we formulate application problems as machine learning problems?

How do we represent the data?

Learning protocols (where are the data and labels coming from?)

Representation

What functions should we learn (hypothesis spaces)?

How do we map raw input to an instance space? What kind of features are we using?

What are rigorous ways to find these? Are there general approaches?

Algorithms

What are good algorithms?

How do we define success?

Generalization vs. over-fitting

The computational problem

Using supervised learning

What is our instance space?

What is our label space?

What is our hypothesis space?

What learning algorithm do we use?

What is our loss function/evaluation metric?

3 Instance Space

Designing an appropriate instance space X is crucial for how well we can predict y. When we apply machine learning to a task, we first need to define the instance space X.

Instances $x \in X$ are defined by features:

- Boolean features (e.g. Does this email contain the word 'money'?)
- Numerical features (e.g. How often does 'money' occur in this email?)

In the badges game, possible features include:

- Gender/age/country of the person
- Length of their first or last name
- Does the name contain letter x?
- How many vowels does their name contain?
- Is the n^{th} letter a vowel?

3.1 Feature Encoding

Assume X is an N-dimensional vector space (eg. $\mathbb{R}^N$), where each x is a feature vector. We can then think of $x = [x_1, x_2...x_n]$ as a point in X.

We can encode a name in the badges game by encoding its characters, where each group of features represents a character. In each group we want $26 \times 2 + 1$ positions in order to encode the alphabetic character and its case, as in
Abe $\rightarrow$ [10000...010000...00001]

The choice of features is crucial to how well a task can be learned, but while there are general principles, features are application specific.

4 Label Space

The label space Y determines what kind of supervised learning task we are dealing with. In this class we focus on binary classification, and make the case that most other classification tasks can be reduced to binary classification.

Other label spaces include

Binary Classification $y \in \{-1, 1\}$

Multiclass Classification $y \in \{1, 2, ..., k\}$

Regression $y \in \mathbb{R}$

Ranking Labels are ordinal and we learn and ordering over input ($f(x_1) > f(x_2)$)

5 Hypothesis Space

In order to learn a model $g(x)$, we must choose which kind of function we expect $g(x)$ to be.

Consider input with four binary features ($\mathbf{x} = [x_1x_2x_3x_4]; x \in \{0, 1\}$), and an unknown function $f(x)$ that returns y. On four features, we have 16 possible instances. In the binary classification task, there are $2^{16} = 65536$ possible functions to describe our data ($|Y|^{|X|}$). Given seven examples, we now have 2^9 possible functions. Without restrictions on the set of possible functions $g(x)$, learning is not possible.

We must put restrictions on the **hypothesis space** – H – such that $H \subseteq |Y|^{|X|}$. Our hypothesis space could be the set of simple conjunctions ($x_1 \wedge x_2$; $x_1 \wedge x_2 \wedge x_3$), or the set of m-of-n rules (m out of the n features are 1, etc.). Many other restrictions are also possible.

6 Views of Learning

Learning is the removal of the remaining uncertainty
Suppose we knew that the unknown function was an m-of-n Boolean function, we could use the training data to infer which function it is.

Learning requires a good, small hypothesis space
We could start with a very small class an enlarge it until it contains a hypothesis that fits the data, but we could be wrong. Our prior knowledge or guess of hypothesis space may not fit the data.

We want general strategies for machine learning. We could either think about flexible hypothesis spaces (decision tress, neural networks, nested collections, etc.) or we could develop some special representation languages for restricted classes of functions (limit the expressivity of the target models and get flexibility by augmenting the feature space).

In either case we're going to develop algorithms for finding a hypothesis in our space and try to guarantee that it generalizes well.

7 Context-Sensitive Spelling Example

I don't know {whether, weather} to laugh or cry.

We want to choose which word is correct. Thus, our label space Y is binary (whether/weather) and we're looking for a function (f) that maps sentences (s) to this label.

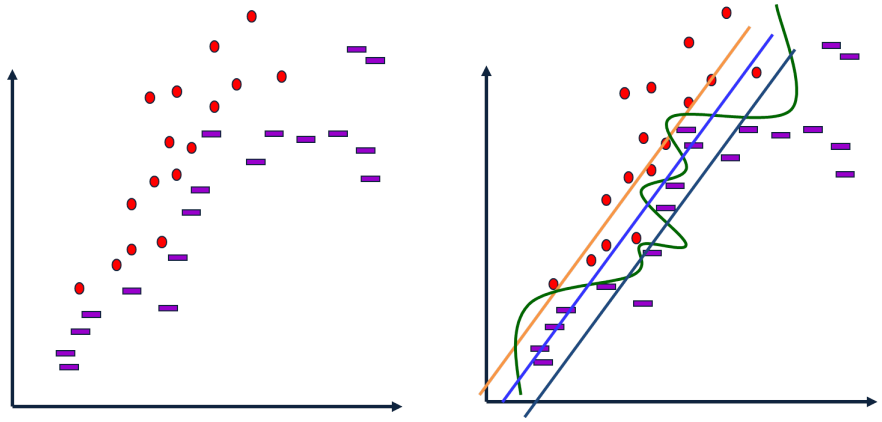
$f(s) \rightarrow \{\text{whether/weather}\}$

We must define the domain. For each word, we define a Boolean feature x_w where $x_w = 1$ *iff* w is in the sentence. This maps a sentence to a point in $\{0,1\}^{50000}$, where for this example we assume there are 50,000 words in English. We can thus encode every sentence as a bit vector of dimensionality 50,000. Given the label, some points are *whether* and some are *weather*. See Figure 2a.

We want to learn a function that best separates the data. Some examples are given in Figure 2b. The green example separates the red points from the purple points perfectly, while the blue does not. However, intuitively the blue is much simpler and gets most of the examples correct.

The key issue between the green and blue lines is memorizing vs. learning, or accuracy vs. simplicity.

One possibility is to change the learning problem from finding a function that best separates the data to finding a **linear function** that best separates the



(a) Examples shown in 2d space

(b) Examples shown with possible separating functions

data[1]. If x is our data representation, the line that best separates the data is w, such that we can make predictions according to $y = sgn(w^T x)$.

7.1 Expressivity

If we assume that $f(x)$ is a linear function, as in

$f(x) = sgn\{w^T x - \theta\} = sgn\{\sum_{i=1}^{n} w_i x_i - \theta\}$

we can express many functions in this way

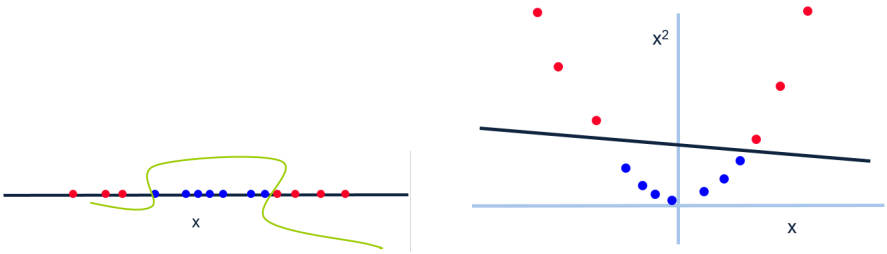
- Conjunctions (e.g. $y = x_1 \wedge x_3 \wedge x_5 = sgn\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}$)
- At least m of n: (e.g. $y =$ at least 2 of $\{x_1, x_3, x_5\} = sgn\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 2\}$)

Many functions are not linear, however, like exclusive-or (xor)

$(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
$x_i \in \{0, 1\}, f(x_1, x_2 ... x_n) = 1 \quad iff \quad \sum x_i$ is even

The function is not linearly separable. However, data that is not linearly separable at first may be separable with another representation. As shown in the figures below, we could represent each data point x as x^2, increasing the dimensionality without adding information, which then makes our data linearly separable.

[1] A linear function that is linear in the feature space; that is, a function that is a linear combination of x



(a) Data not linearly separable in 1D (b) Data separable in $\langle x, x^2 \rangle$ space

7.2 Modeling

Assume the following (DNF) function

$$x_1x_2x_4 \vee x_2x_4x_5 \vee x_1x_3x_7,$$

as shown in Figure 2 below.

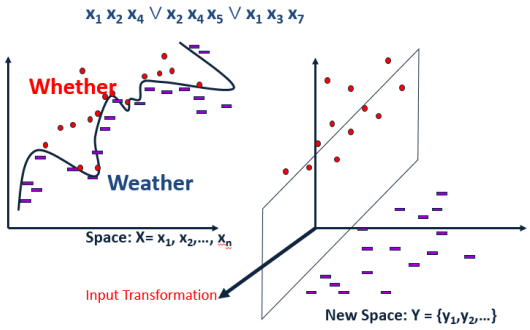


Figure 3: Directions of weight vectors

While this function is not linear over these x_is, it can be made linear by inventing a new set of new features.

Suppose the original feature space is $X = x_1, x_2, ..., x_n$. A new feature space can be invented by taking conjunction of every subset of three x_is from X, such that

$$Y = \{y_1, y_2, ...\} = \{x_ix_jx_k, ...\} \quad \forall i, j, k.$$

Then, this feature space would have $\binom{n}{3} = O(n^3)$ features. In this new space, a function that looked very complicated can be written as

$$y_3 \vee y_4 \vee y_7,$$

and looks like a disjunction in a larger space. The new discriminator is functionally simpler, and is linearly separable.

Can we always do this?

In a finite-dimensional space, we can always do this. It may be very expensive, due to the significant increase in the feature space dimensionality, but it is always possible.

8 Learning

8.1 Local Search

One typical way to learn is by choosing an initial model and correcting it. This local search approach can be described as below.

1. Start with some linear threshold function
2. Determine how well it separates the data
3. Correct the function based on errors
4. Repeat until convergence

8.2 General Framework for Learning

In learning, the goal is to predict an unobserved value $y \in Y$ based on an observed input vector $\mathbf{x} \in X$. Put another way, we want to estimate a functional relationship $y \sim f(\mathbf{x})$ based on a set of examples $\{(\mathbf{x}, y)_i\}_{i=1,\ldots,n}$. We want $f(\mathbf{x})$ to minimize some kind of risk.

For example, we want the function to minimize the expected number of mistakes that it makes. Let $E_{X,Y}$ denote the expected number of mistakes with respect to the true distribution[2]. We then want to minimize the *loss* given by

$$L(f()) = E_{X,Y}(f(\mathbf{x}) \neq Y)$$

However, computing the expectation in this loss function is both computationally and information theoretically difficult. Thus, we minimize the empirical classification error: that is, how many errors we make in the training data.

$$L'(f()) = \frac{1}{n} \sum_i f(\mathbf{x}_i) \neq y_i$$

This minimization problem is typically NP hard. As a result, we want to instead minimize a convex upper bound on the true number of mistakes. There are many such loss functions that we can define.

[2]If the true distribution is known ($P(y = 1|\mathbf{x})$ and $P(y = 0|\mathbf{x})$), no learning takes place; one can simply take the max at test time

Squared Loss	$L(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$
Input Dependent Loss	$L(f(\mathbf{x}), y) = 0$ *if* $f(\mathbf{x}) = y$; *else* $L(f(\mathbf{x}), y) = c(\mathbf{x})$
0-1 Loss	$L(f(\mathbf{x}), y) = \frac{1}{2}(1 - sgn(yf(\mathbf{x})))$
Hinge Loss	$L(f(\mathbf{x}), y) = max(0, 1 - yf(\mathbf{x}))$

8.3 Linear Threshold Units (LTU)

We want to learn a linear separator (or LTU) given by

$$f(\mathbf{x}) = \text{sgn}(\mathbf{x}^T \cdot \mathbf{w} - \theta) = \text{sgn}(\sum_{i=1}^{n} w_i x_i - \theta),$$

where

- $\mathbf{x}^T = (x_1, x_2, ..., x_n) \in \{0, 1\}^n$ is the feature encoding of a data point
- $\mathbf{w}^T = (w_1, w_2, ..., w_n) \in \mathbb{R}^n$ is the target function
- θ determines the shift of the linear separator with respect to the origin

One useful trick to simplify the notation is to add one dimension to our space to allow our hyperplanes to go through the origin.

Let $\mathbf{x}' = (\mathbf{x}, -1)$ and $\mathbf{w}' = (\mathbf{w}, \theta)$, then

$$\text{sgn}(\mathbf{x}^T \cdot \mathbf{w} - \theta) = \text{sgn}((\mathbf{w}')^T \cdot \mathbf{x}').$$

Our goal is to find a $\mathbf{w}$ that best separates the data set. We thus need to minimize the expected risk function $J(\mathbf{w}) = E_{X,Y} Q(\mathbf{x}, y, \mathbf{w})$.

To find this minimum, we can use a batch gradient descent algorithm.

8.4 Gradient Descent

Gradient descent can be used to determined the weight vector that minimizes $J(\mathbf{w}) = \text{Err}(\mathbf{w})$. At each step, the weight vector is modified in the direction that produces the steepest descent along the error surface, as shown in Figure 4.

Here, the hypothesis space is the collection of LTUs. To find the best, we can use Least Mean Squares (LMS) as the loss function, which is given by

$$Q(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \frac{1}{2}(\mathbf{w}^T \mathbf{x} - \mathbf{y})^2.$$

Let $\mathbf{w}^{(j)}$ be the weight vector at time j, the prediction on the dth example of $\mathbf{x}$ is $o_d = \mathbf{w}^{(j)} \cdot \mathbf{x}$

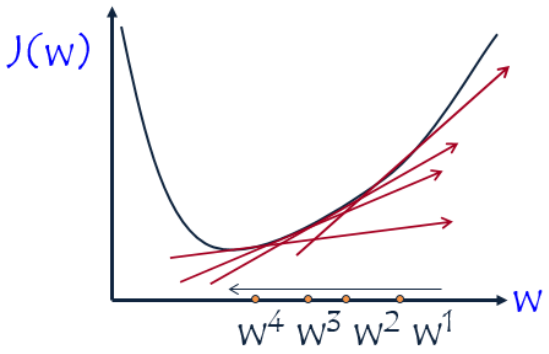


Figure 4: Directions of weight vectors

Let t_d be the target value for this example, the error the current hypothesis makes on the data set is

$$J(\mathbf{w}) = \text{Err}(\mathbf{w}^{(j)}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2.$$

To find the best direction in the weight space $\mathbf{w}$ we compute the gradient of E with respect to each of the components of

$$\bigtriangledown E(\mathbf{w}) = [\frac{\partial E}{\partial \mathbf{w}_1}, \frac{\partial E}{\partial \mathbf{w}_2}, ..., \frac{\partial E}{\partial \mathbf{w}_n}],$$

this vector specifies the direction that produces the steepest increase in E. We want to modify in the direction of $\mathbf{w}$ in the direction of $-\bigtriangledown E$, such that

$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w},$$

where

$$\Delta\mathbf{w} = -R \bigtriangledown E.$$

Here R is the fixed learning rate, which specifies how large each step is.

Since $\text{Err}(\mathbf{w}^{(j)}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$, we can compute the derivative as

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{w}_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \mathbf{w}_d \cdot \mathbf{x}_d) \\
&= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}$$

Hence, we get the weight update rule as

$$\Delta w_i = R\sum_{d\in D}(t_d - o_d)(x_{id}).$$

The gradient descent algorithm for training linear units can be concluded as follows

- Start with an initial random weight vector
- For each example x_d with target value t_d
  - Evaluate the LTU $o_d = \mathbf{w} \cdot \mathbf{x}_d$
- Update $\mathbf{w}$ by adding Δw_i to each component
- Continue until E below some threshold

This algorithm always converges to a local minimum of $J(\mathbf{w})$, for small enough steps. Here (LMS for linear regression), the surface contains only a single global minimum, so the algorithm converges to a weight vector with minimum error, regardless of whether the examples are linearly separable.

The surface may have local minimum if the loss function is different.

8.5 Stochastic Gradient Descent

If we drop the averaging operation and choose a sample $(\mathbf{x}, y)$ at random to update $\mathbf{w}^t$, the weight update rule becomes

$$\Delta w_i = R(t_d - o_d)x_{id},$$

This is stochastic gradient descent (SGD), which is an example of an *on-line* algorithm, as it updates $\mathbf{w}$ using a single example at a time, rather than needing the entire training set.

In general SGD does not converge to a global minimum, but by decreasing the learning rate R, convergence can be guaranteed.

On-line algorithms are advantageous in many respects, but most notably because one need not touch the entire data set on each update.

8.6 Model Considerations

There are two important considerations when trying to model a learning problem.

Sample complexity: For a given learning problem, we are interested in the number of examples the must be seen to determine if a learned model will generalize well.

Computational Complexity: For a given loss function, we are interested in the relation between convergence and loss, and under what conditions can a given loss be computed efficiently.