CSC 439/539
Statistical Natural Language Processing
Lecture 2: Crash Course in Machine Learning for Text Categorization

Mihai Surdeanu

Fall 2017

Take-away

- Understand the task of text categorization
- A few fundamental machine learning (ML) algorithms
  - K Nearest Neighbors (kNN)
  - Perceptron
  - Logistic regression
  - Feed-forward neural networks

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

Text categorization example


```
<REUTERS NEWID="11">
<DATE>26-FEB-1987 15:18:59.34</DATE>
<TOPICS><D>earn</D></TOPICS>
<TEXT>
<TITLE>COBANCO INC &lt;CBCO> YEAR NET</TITLE>
<DATELINE>    SANTA CRUZ, Calif., Feb 26 - </DATELINE>
<BODY>Shr 34 cts vs 1.19 dlrs
    Net 807,000 vs 2,858,000
    Assets 510.2 mln vs 479.7 mln
    Deposits 472.3 mln vs 440.3 mln
    Loans 299.2 mln vs 327.2 mln
    Note: 4th qtr not available. Year includes 1985
extraordinary gain from tax carry forward of 132,000 dlrs,
or five cts per shr.
 Reuter
</BODY></TEXT>
</REUTERS>
```

What would you do with such an algorithm?

Text categorization example

- **ham** Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
- **ham** Ok lar... Joking wif u oni...
- **spam** Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
- **ham** U dun say so early hor... U c already then say...
- **ham** Nah I don't think he goes to usf, he lives around here though
- **spam** FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv

Overview

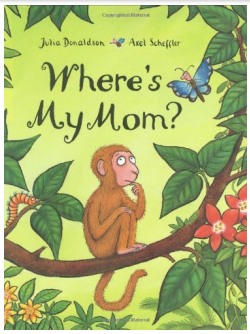
- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

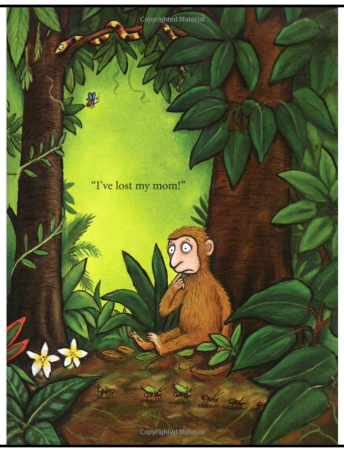
Motivation

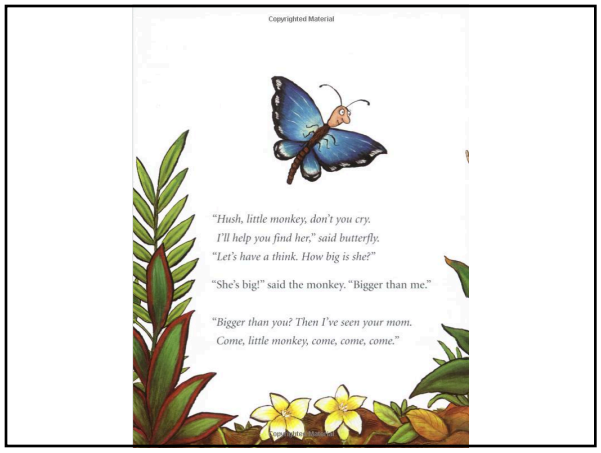
- "I don't crush the corolla of wonders of the world"
  - Lucian Blaga, 1919
- But I think that for ML, we really should destroy it
  - To help us become better users of ML software, and, hopefully, designers of new ones
  - To squash impostor syndrome. We all suffer from it. See this humorous description: http://www.isi.edu/natural-language/people/bayes-with-tears.pdf

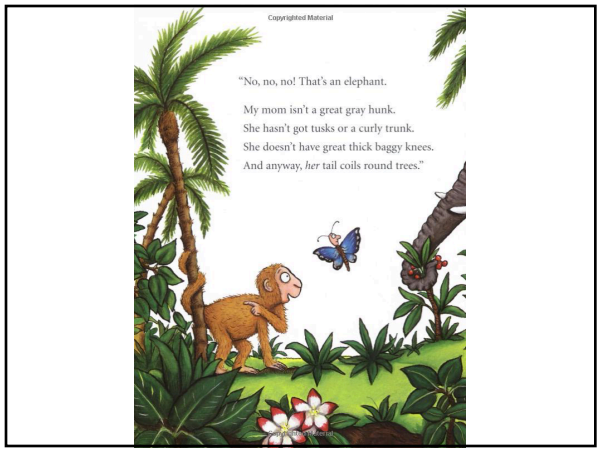
MACHINE LEARNING IS NOT HARD

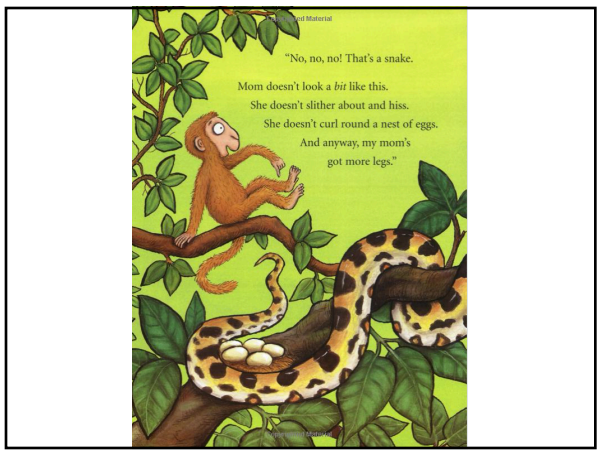
Let's read a book






Copyrighted Material
"Hush, little monkey, don't you cry.
I'll help you find her," said butterfly.
"Let's have a think. How big is she?"
"She's big!" said the monkey. "Bigger than me."
"Bigger than you? Then I've seen your mom.
Come, little monkey, come, come, come."


Copyrighted Material
"No, no, no! That's an elephant.
My mom isn't a great gray hunk.
She hasn't got tusks or a curly trunk.
She doesn't have great thick baggy knees.
And anyway, her tail coils round trees."


"No, no, no! That's a snake.
Mom doesn't look a bit like this.
She doesn't slither about and hiss.
She doesn't curl round a nest of eggs.
And anyway, my mom's
got more legs."

Don't worry. Eventually, the baby monkey finds her mom.

What have we learned from this book?

- Objects are described by their properties, or *features*
  - Vector representation, vector length, normalization
- A classification algorithm has at its core a *similarity function*, which measures how similar properties of two objects are
  - Vector similarity
- A good algorithm aggregates its decisions over multiple properties
- Algorithms may *overfit*, if not designed well, or not trained on enough data
  - Overfitting: when an algorithm performs well in training, but poorly on unseen data (testing)

Vector length + cosine similarity

- Vector length:

$$||\vec{x}||_2 = \sqrt{\sum_i x_i^2}$$

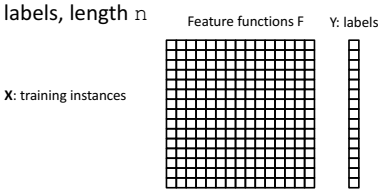
- Cosine similarity:

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Let's build some vectors

Representing NLP problems numerically

- **X**: instance-feature matrix, size $n \times m$ (n rows, m columns)
  - n training instances X_i
  - m feature functions f_j
  - $\mathbf{X}_{i,j} = f_j(X_i)$, i.e., the value of feature function f_j for X_i
- Y: vector of labels, length n



Slide by Erwin Chan

Definitions: instances, features, and labels

- Instances
  - $\mathbf{X} = \{ X_1, X_2, \ldots, X_n \}$: set of all instances in the data
  - X_i: an individual instance
- Features
  - $F = \{ f_1, f_2, \ldots, f_m \}$: a set of *feature functions*
  - $X_i = [f_1(X_i), f_2(X_i), \ldots, f_m(X_i)]$ is a vector of *features*, i.e., the result of applying each feature function in F to the instance X_i
- Labels
  - Set of possible labels: $Y = \{ y_1, y_2, \ldots, y_k \}$
    - Binary classification: $|Y| = 2$
    - Multiclass classification: $|Y| > 2$
  - Classification: for an instance X_i, predict a label $y_i \in Y$
  - Regression: labels are continuous numeric values

Adapted from slide by Erwin Chan

(Powerpoint) Notations

- Scalars: lower case characters
- Vectors of scalars: upper case characters
- Matrices, or vectors of vector: bold upper case characters

Have we understood?

- Features
- Feature vectors
- Vector normalization
- Cosine similarity, dot product

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

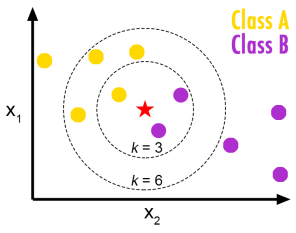
kNN

- Perhaps the simplest categorization algorithm
- Rationale: to classify a new object find the object in the training set that is most similar (1NN)
  - Not very robust: one document could be mislabeled or atypical
- Generalization: assign each object to the majority class of its *k* nearest neighbors in the training set (kNN)

Probabilistic kNN

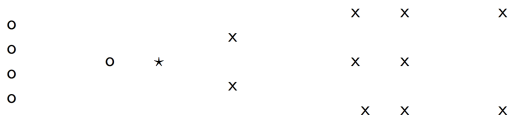
- Probabilistic version of kNN: P(c|d) = fraction of *k* neighbors of *d* that are labeled with class *c*
- kNN classification rule for probabilistic kNN: Assign *d* to class *c* with highest P(c|d)

Exercise



How is star classified for k = 3; k = 6?

Exercise



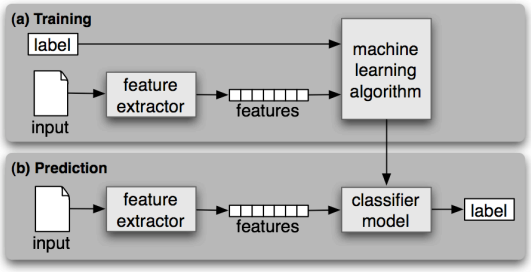
How is star classified by:

(i) 1-NN (ii) 3-NN (iii) 9-NN (iv) 15-NN

What kNN needs

- Representation of objects as vectors
- A similarity function between two vectors
  - Cosine similarity
  - Euclidian distance

Architecture of ML algorithms

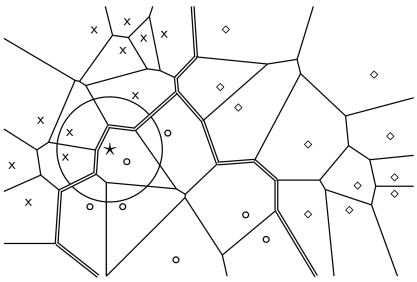


kNN algorithm

TRAIN-KNN($\mathbb{C}, \mathbb{D}$)
1 $\mathbb{D}' \leftarrow$ PREPROCESS($\mathbb{D}$)
2 $k \leftarrow$ SELECT-K($\mathbb{C}, \mathbb{D}'$)//*tuning*
3 **return** $\mathbb{D}', k$

APPLY-KNN($\mathbb{D}', k, d$)
1 $S_k \leftarrow$ COMPUTENEARESTNEIGHBORS($\mathbb{D}', k, d$)
2 **for each** $c_j \in \mathbb{C}(\mathbb{D}')$
3 **do** $p_j \leftarrow |S_k \cap c_j|/k$
4 **return** $\arg\max_j p_j$

kNN is based on Voronoi tesselation



kNN is a non-linear classifier! That is, separators are not "lines".

What is the fundamental problem with kNN?

- The overhead of prediction is high
  - We potentially have to inspect all the objects seen in training to discover the closest neighbors for a new object
- We must condense what we learn from training into something more compact, which could be applied quickly at prediction time

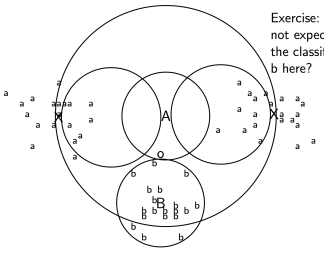
Averaging the vectors per class does not work too well

- Rocchio algorithm:
  - Compute a centroid for each class
    - The centroid is the average of all documents in the class.
  - Assign each test document to the class of its closest centroid.
- What is the runtime of this algorithm at prediction time?

Exercise

- Let's build a Rocchio classifier

Rocchio cannot handle multimodal classes



Exercise: Why is Rocchio not expected to do well for the classification task a vs. b here?

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

Rationale

- Error driven learning: update what is learned only when the classifier makes a mistake in training
  - Ignore objects that are classified correctly
  - Focus on the "hard" ones

Perceptron decision

- Learns a single vector W (for binary classification)
- Decision:
  - Decide "yes" iff f(X) = W . X – Theta > 0
  - Decide "no" otherwise

Bias term

Dot product

Algorithm (1/2)

```
1  comment: Categorization Decision
2  funct decision(x⃗, w⃗, θ) ≡
3     if w⃗ · x⃗ > θ then
4                          return yes
5                    else
6                          return no
7     fi.
```

Algorithm (2/2)

```
9  comment: Initialization
10 w⃗ = 0
11 θ = 0
12 comment: Perceptron Learning Algorithm
13 while not converged yet do
14        for all elements x⃗_j in the training set  do
15           d = decision(x⃗_j, w⃗, θ)
16           if class(x⃗_j) = d then
17                                continue
18           elsif class(x⃗_j) = yes and d = no then
19                                                θ = θ − 1
20                                                w⃗ = w⃗ + x⃗_j
21           elsif class(x⃗_j) = no and d = yes then
22                                                θ = θ + 1
23                                                w⃗ = w⃗ − x⃗_j
24           fi
25        end
26 end
```

Exercise

- Let's build a Perceptron

Visualization

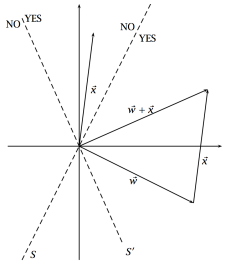


Figure 16.8 One error-correcting step of the perceptron learning algorithm. Data vector $\vec{x}$ is misclassified by the current weight vector $\vec{w}$ since it lies on the "no"-side of the decision boundary S. The correction step adds $\vec{x}$ to $\vec{w}$ and (in this case) corrects the decision since $\vec{x}$ now lies on the "yes"-side of S', the decision boundary of the new weight vector $\vec{w} + \vec{x}$.

Real-world example

Word w^i	Weight
vs	11
mln	6
cts	24
;	2
&	12
000	−4
loss	19
'	−2
"	7
3	−7
profit	31
dlrs	1
1	3
pct	−4
is	−8
s	−12
that	−1
net	8
lt	11
at	−6
θ	37

Table 16.10 Perceptron for the "earnings" category. The weight vector $\vec{w}$ and θ of a perceptron learned by the perceptron learning algorithm for the category "earnings" in Reuters.

Simplifying the bias term

- In many implementations, the bias term is folded into the vectors W and X:

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \\ \theta \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \\ -1 \end{pmatrix}$$

- The decision then becomes simply $W \cdot X > 0$
- And the updates work only with W (let's check!)

Simplifying the bias term

```
9  comment: Initialization
10 w⃗ = 0
11 θ = 0
12 comment: Perceptron Learning Algorithm
13 while not converged yet do
14     for all elements x⃗_j in the training set do
15         d = decision(x⃗_j, w⃗, θ)
16         if class(x⃗_j) = d then
17                                 continue
18         elsif class(x⃗_j) = yes and d = no then
19                                                 θ ← θ − 1
20                                                 w⃗ = w⃗ + x⃗_j
21         elsif class(x⃗_j) = no and d = yes then
22                                                 θ ← θ + 1
23                                                 w⃗ = w⃗ − x⃗_j
24         fi
25     end
26 end
```

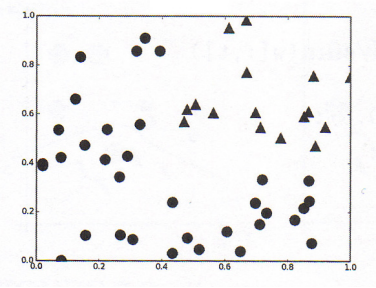
Ensemble models and average Perceptron

- In most situations, voting between multiple classifiers improves performance
  - It is also a cheap way to build a non-linear classifier from a bunch of linear ones

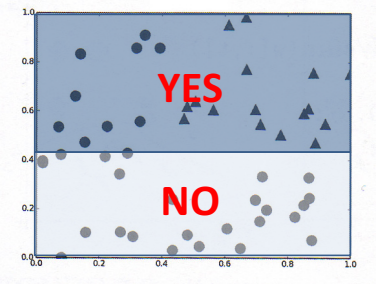
Voting

- Term typically used when the individual classifiers are not directly related
  - Different ML models; different features; etc.
  - Worth $1M: see the Netflix prize
- Types of voting
  - Unweighted voting: pick majority choice
  - Weighted voting: weigh votes by, say, accuracy over a controlled set (development) or confidence of each prediction

Example: Apply voting to this data

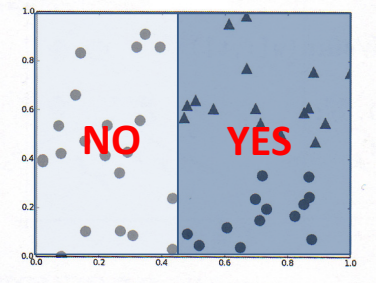


Classifier A



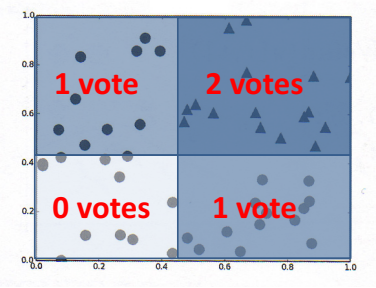
Slide by Erwin Chan

Classifier B



Slide by Erwin Chan

Correct decision boundary through majority vote



Slide by Erwin Chan

Prediction overhead

- But prediction overhead is high again, linear in the number of individual classifiers
- Averaging is a cheap way of reaping most benefits of voting

Averaging

```
comment: Initialization
w⃗ = 0
θ = 0
comment: Perceptron Learning Algorithm
while not converged yet do
    for all elements x⃗_j in the training set do
        d = decision(x⃗_j, w⃗, θ)
        if class(x⃗_j) = d then
                continue
        elsif class(x⃗_j) = yes and d = no then
            θ = θ
            w⃗ = w⃗ + x⃗_j
        elsif class(x⃗_j) = no and d = yes then
            θ = θ + 1
            w⃗ = w⃗ − x⃗_j
        fi
    end
end
```

avgW += W

avgW += W

avgW /= number-of-updates

Averaging

- Decision:
  - Decide “yes” iff f(X) = avgW · X > 0
  - Decide “no” otherwise

What are the fundamental problems with the Perceptron?

- The basic or averaged Perceptrons only model linearly-separable data
- No ”smooth” updates
  - The vector W is updated equally for big or small mistakes…
  - Hard to generalize, because the model “jumps around” good solutions

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

Binary classification notations

Let's now talk about the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** problem in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then $x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the **negative class**, and 1 the **positive class**, and they are sometimes also denoted by the symbols "-" and "+." Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** for the training example.

All slides on logistic regression come from Andrew Ng's CS229 course at Stanford

Some notation changes

- Just as before, the bias term (Theta in the Manning textbook) is folded in the X and W vectors
- But here we will use Theta to denote the weights learned by the model (known as W in the Manning textbook)

Looking ahead: A better update rule

$$\theta := \theta + \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$

This is what LR does and it is much better than the Perceptron update. Why?

(h is the decision function)

Let's see how we can derive this update rule

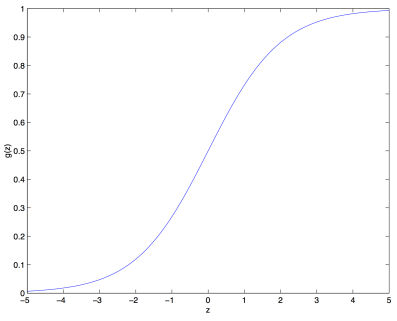
Improvement 1: Decision function

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

g = sigmoid function, or logistic function

Decision function



Decision function

Let us assume that

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Why does this make sense?

Note that this can be written more compactly as

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Decision function's derivative (for later)

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} \left(e^{-z}\right) \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

Use the differentiation rules to derive this yourself:
https://en.wikipedia.org/wiki/Differentiation_rules

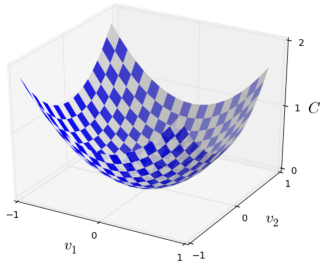
Improvement 2: Cost function and gradient descent/ascend

- Given a training set, how do we learn the parameters Theta to make h(x) close to y?
- To formalize this, we will use a *cost function*, J, which measures how close h(x) is to y
- Given such a function, we will use the gradient descent algorithm, which starts with some initial Theta, and repeatedly performs the update:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

  - Takes a step in the direction of steepest decrease of J.

Gradient descent



Cost function for LR = Log likelihood

J = l(Theta) = log L(Theta) = log p(Y|X; Theta)

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X; \theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}
$$

Note

- In the case of LR, we want to maximize J, so we will perform gradient ascent:

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta)$$

Computing the partial derivatives

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{\partial}{\partial \theta_j}g(\theta^T x) \\
&= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)g(\theta^T x)(1-g(\theta^T x))\frac{\partial}{\partial \theta_j}\theta^T x \\
&= \left(y(1-g(\theta^T x)) - (1-y)g(\theta^T x)\right)x_j \\
&= \left(y - h_\theta(x)\right)x_j
\end{aligned}$$

We are applying the chain rule twice here! See:
https://en.wikipedia.org/wiki/Differentiation_rules

Stochastic gradient ascent

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}$$

- "Stochastic" because we perform updates after every training example
- "Batch"
  - Perform 1 update for 1 pass over the entire training dataset
  - Almost never used in practice (why?)

The update rule

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}$$

This is much better than the Perceptron update. Why?

Exercise: write the whole LR training algorithm

Limitations

- The decision function is linear (in log space)
- Feature design is hard for most real-world NLP problems…

Overview

- Text categorization
- Introduction to categorization algorithms
- K Nearest Neighbors (kNN)
- Perceptron
- Logistic regression
- Feed-forward neural networks

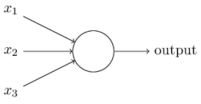
What did you find hard about ML so far?

- We need to design custom decision functions (e.g., dot product, sigmoid), and custom cost functions
- But then we have to recompute all the derivatives. Derivatives are a pain…
- Feature design is hard for real-world NLP applications

Advantages of neural networks

- Custom, non-linear decision and cost functions
- Auto-differentiation to compute derivatives automatically
- Automatic generation of feature vectors for all data points (next lecture)

Perceptron architecture



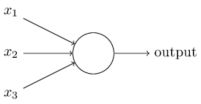
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

or:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

All slides on logistic regression come from http://neuralnetworksanddeeplearning.com/

LR architecture

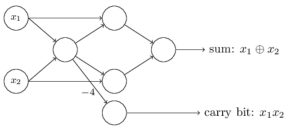


$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

where *z* is the dot product of W and X (plus bias):

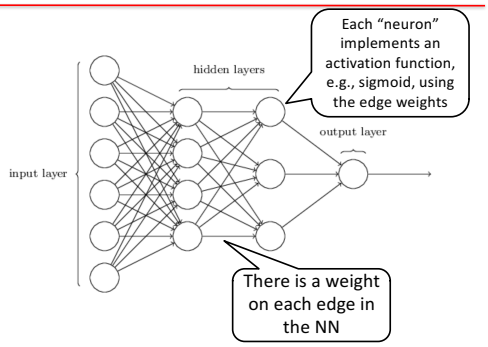
$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

Many other architectures are possible…

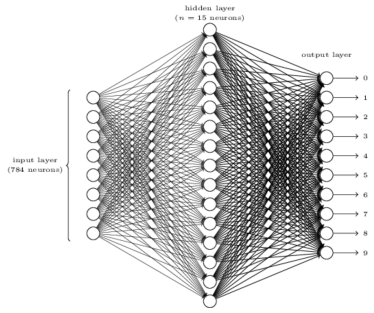


See http://neuralnetworksanddeeplearning.com/chap1.html for this discussion.

Feed forward neural networks



NN for recognizing digits



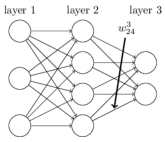
What needs to be trained

- The network (i.e., number of layers, nodes, and edges), and activation functions are static. You define them.
- During training, stochastic gradient descent (SGD) adjust the edge weights such that a cost function (J or C), e.g., negative log likelihood, is minimized.

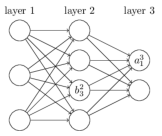
What needs to be trained

- Very similar to LR (in fact, LR is a one-layer NN…), but
- Computing the derivatives for the hidden layers (which don't exist in LR) is tricky
- We will use the backpropagation algorithm for this purpose
  - Let's do this for the feed-forward NN
  - Required for 539 only

Notations



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

$$z^l \equiv w^l a^{l-1} + b^l$$

Goal of backpropagation

The goal of backpropagation is to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network.

Cost function in this example

$$C = \frac{1}{2n}\sum_x \|y(x) - a^L(x)\|^2$$

Or, for a single training example x:

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a^L_j)^2$$

- C = quadratic cost, mean squared error, MSE
- But many other cost functions are possible!
- (Incomplete) rule of thumb:
  - If binary classification: MSE
  - If multiclass classification: cross entropy

Notes: a few useful formulas

- Hadamard product: Element wise multiplication of two vectors:

$$\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1*3\\2*4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

- Error or layer l in the network:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

4 equations

- Which will compute the partial derivatives of C with respect to w and b, so we can perform gradient descent

Summary: the equations of backpropagation

$\delta^L = \nabla_a C \odot \sigma'(z^L)$ (BP1)

$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ (BP2)

$\frac{\partial C}{\partial b_j^l} = \delta_j^l$ (BP3)

$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ (BP4)

BP1

- Compute the error in the last layer (L):

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

- Or, in vector form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Proof of BP1

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

BP2

- Computes the error in layer l as a function of the error in layer l + 1 (hence "backpropagation"):

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Proof of BP2 (1/2)

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1},\end{aligned}$$

Proof of BP2 (2/2)

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

BP3 and BP4

- Rate of change of cost wrt any bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

- Rate of change of cost wrt any weight in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

At-home exercise: proofs of BP3 and BP4

- These could be great midterm questions (for grad students)…

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation + stochastic gradient descent

- SGD learns network weights (and bias) for any network, with any decision functions in the nodes, and any cost function
  - Assuming we can differentiate the decision and cost functions…
- SGD is not guaranteed to be optimal!
  - But it works reasonably well, and it is reasonably fast

Coming up

- Next lecture: Automatic feature generation through distributional similarity
- Following lecture: DyNet tutorial for feed-forward networks
- Rest of the course: structured learning, aka how to learn when the points to be categorized depend on each other?

Readings

- For kNN and Rocchio: see chapter 14 in Introduction to Information Retrieval (http://informationretrieval.org)
- For Perceptron: see chapter 16 in our textbook
- For logistic regression (LR): see Andrew Ng's notes for LR: Part II in the file cs229_notes1.pdf (in D2L)
- For neural networks: see chapters 1 and 2 in Michael Nielsen's book: http://neuralnetworksanddeeplearning.com

Take-away

- Understand the task of text categorization
- A few fundamental machine learning (ML) algorithms
  - K Nearest Neighbors (kNN)
  - Perceptron
  - Logistic regression
  - Feed-forward neural networks