IT 469 Human Language Technologies : 3- Text Classification

College of Computer and Information Sciences,

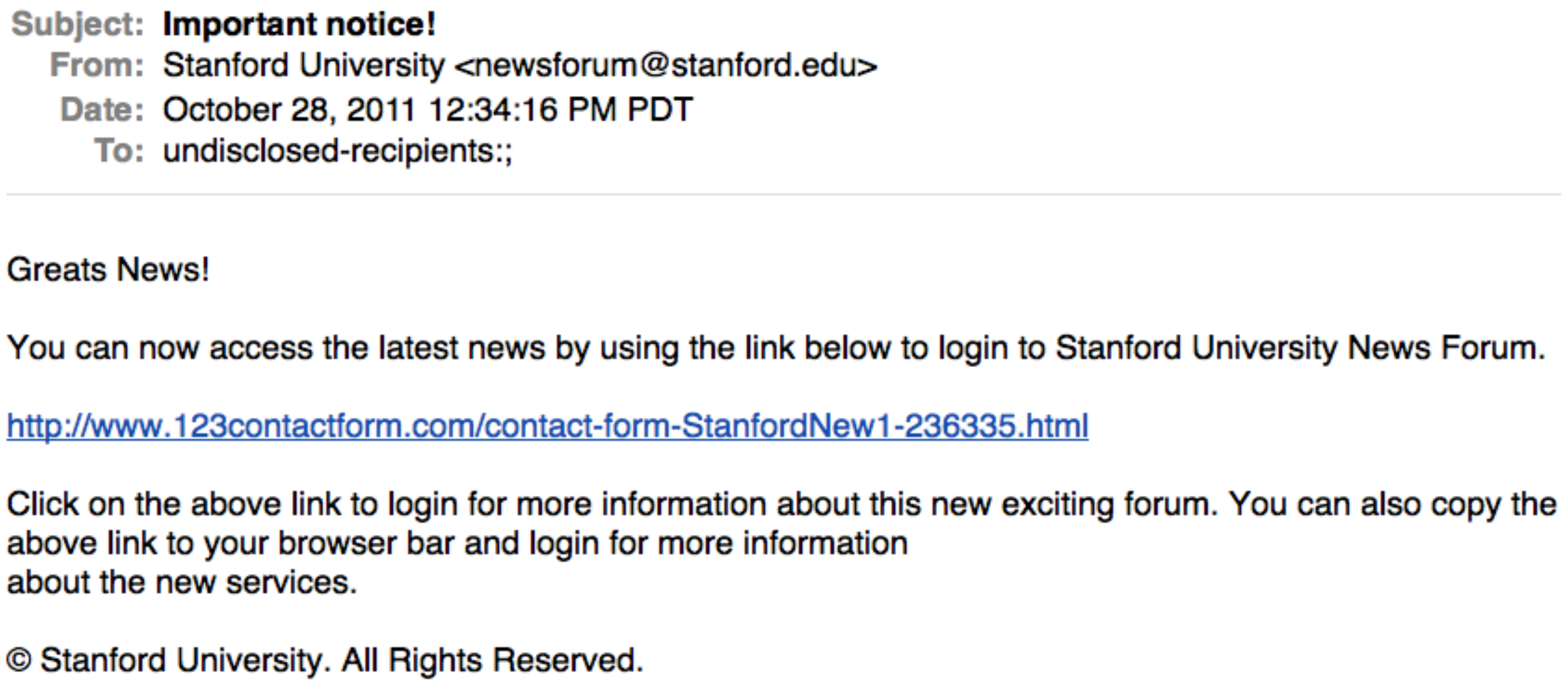
Information Technology Department

King Saud University

Outline

- Text classification (part 1)
  - Definition
  - Classification method (supervised learning)
  - Closer look at Naïve Bayes classifier
  - Evaluation metrics
  - Harm mitigation
- Generative and Discriminative Classifiers (part 2)
  - Logistic Regression
  - Learning components
  - Overfitting
  - Regularization

Is this spam?



Subject: **Important notice!**
From: Stanford University <newsforum@stanford.edu>
Date: October 28, 2011 12:34:16 PM PDT
To: undisclosed-recipients:;

Greats News!

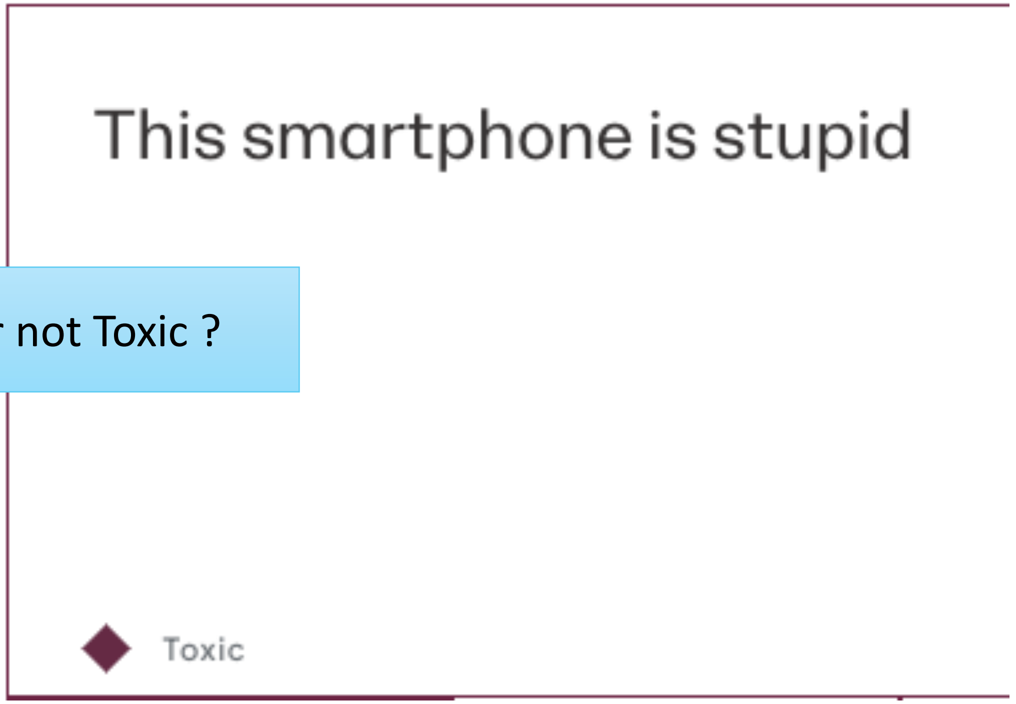
You can now access the latest news by using the link below to login to Stanford University News Forum.

http://www.123contactform.com/contact-form-StanfordNew1-236335.html

Click on the above link to login for more information about this new exciting forum. You can also copy the above link to your browser bar and login for more information
about the new services.

© Stanford University. All Rights Reserved.

Identify "toxic" comments

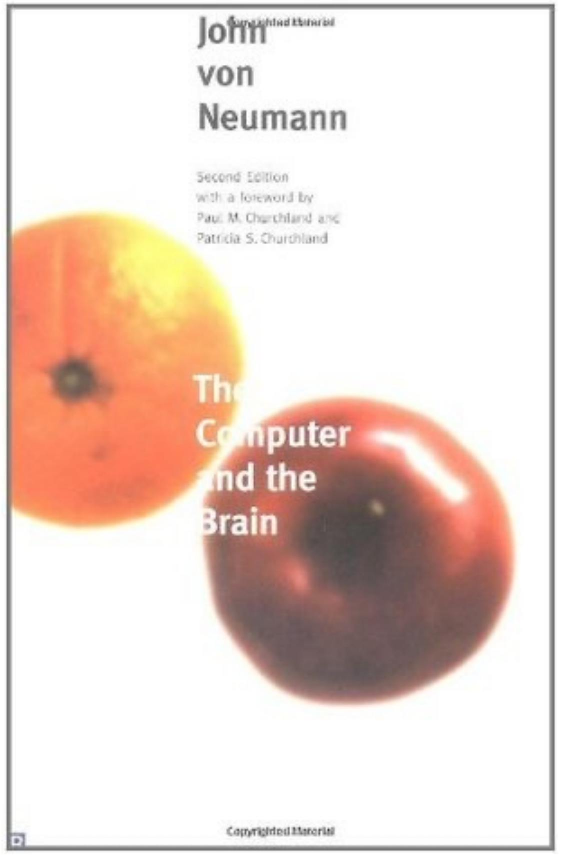

This smartphone is stupid

◆ Toxic

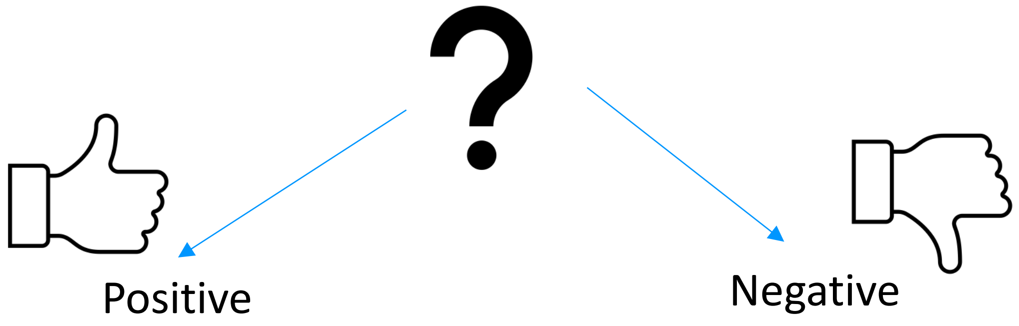
Toxic or not Toxic ?

https://www.perspectiveapi.com/

Book Reviews



I absoultely loved this book !



Text Classification: definition

- *Input*:
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

- *Output*: a predicted class $c \in C$

Classification Methods: Supervised Machine Learning

- Any kind of classifier
  - Naïve Bayes
  - Logistic regression
  - Neural networks
  - k-Nearest Neighbors
  - …

Naive Bayes Classifier

Naive Bayes Intuition

- Simple ("naive") classification method based on Bayes rule
- Relies on very simple representation of document
  - **Bag of words**

The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

fairy
always
love
to
it
it
whimsical
it
I
and
seen
are
friend
anyone
happy
dialogue
recommend
adventure
who
sweet
of
satirical
it
movie
it
I
but
to
romantic
I
several
yet
again
it
the
humor
the
seen
would
to
scenes
I
the manages
fun
the
times
I
and
and
about
while
whenever
have
conventions
with

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Bayes' Rule Applied to Documents and Classes

- For a document *d* and a class *c*

$$P(c \mid d) = \frac{P(d \mid c) P(c)}{P(d)}$$

Laplace (add-1) smoothing for Naïve Bayes

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1\right)}$$

$$= \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

E 4.14

Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class $= c_j$

$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

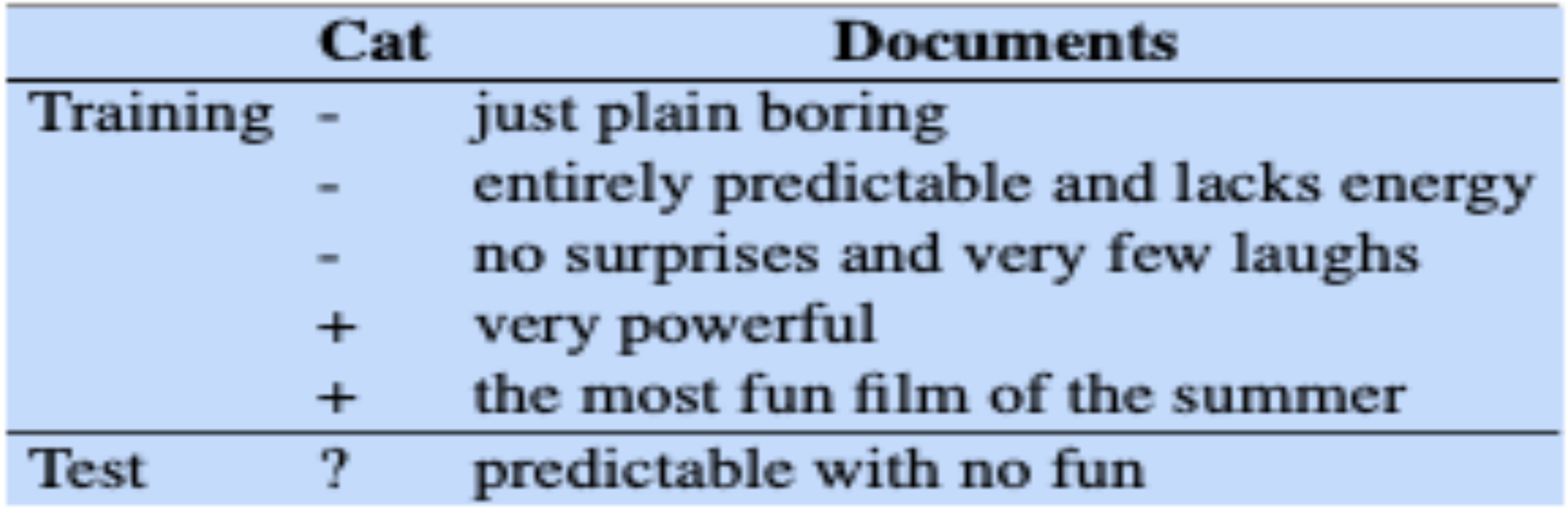
- Calculate $P(w_k \mid c_j)$ terms
  - $Text_j \leftarrow$ single doc containing all $docs_j$
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

$$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$$

Worked example

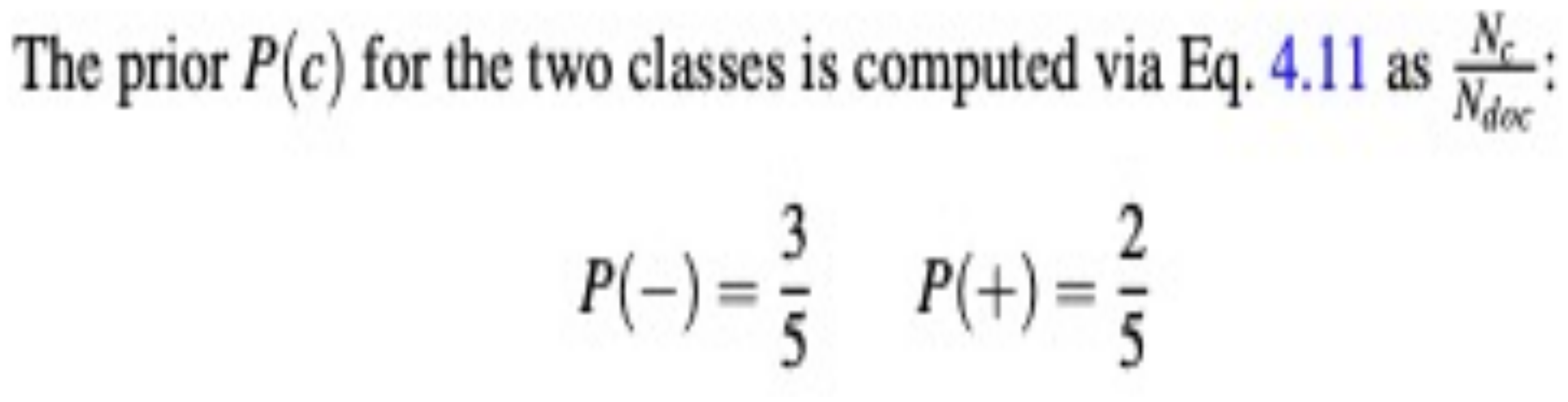
- Let’s walk through an example of training and testing naive Bayes with add-one smoothing. We’ll use a sentiment analysis domain with the two classes positive (+) and negative (-), and take the following miniature training and test documents simplified from actual movie reviews.



	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

Worked example Cont.

- The prior P(c) for the two classes is computed via Eq. 4.11 as



The prior $P(c)$ for the two classes is computed via Eq. 4.11 as $\frac{N_c}{N_{doc}}$:

$$P(-) = \frac{3}{5} \qquad P(+) = \frac{2}{5}$$

The word with doesn’t occur in the training set, so we drop it completely (we don’t use unknown word models for naive Bayes)

Worked example Cont.

- The likelihoods from the training set for the remaining three words “predictable”, “no”, and “fun”, are as follows, from Eq. 4.14

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \quad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \quad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \quad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

For the test sentence S = “predictable with no fun”,

- after removing the word ‘with’, the chosen class via Eq. 4.9:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

The model thus predicts the class negative for the test sentence

Unknown words

- What about unknown words
  - that appear in our test data
  - but not in our training data or vocabulary?
- We **ignore** them
  - Remove them from the test document!
  - Pretend they weren't there!
  - Don't include any probability for them at all!
- Why don't we build an unknown word model?
  - It doesn't help: knowing which class has more unknown words is not generally helpful!

Stop words

- Some systems ignore stop words
  - **Stop words:** very frequent words like *the* and *a*.
    - Sort the vocabulary by word frequency in training set
    - Call the top 10 or 50 words the **stopword list**.
    - Remove all stop words from both training and test sets
      - As if they were never there!
- But removing stop words doesn't usually help
  - So in practice most NB algorithms use **all** words and **don't** use stopword lists

Summary: Naive Bayes is Not So Naive

- Very Fast, low storage requirements
- Work well with very small amounts of training data
- Robust to Irrelevant Features

    Irrelevant Features cancel each other without affecting results
- Very good in domains with many equally important features

    Decision Trees suffer from *fragmentation* in such cases – especially if little data
- Optimal if the independence assumptions hold: If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
- A good dependable baseline for text classification
    - **But we will see other classifiers that give better accuracy**

Slide from Chris Manning

Evaluation Metrics
Precision, Recall, and F measure

Evaluation

- Let's consider just binary text classification tasks
- Imagine you're the CEO of Delicious Pie Company
- You want to know what people are saying about your pies
- So you build a "Delicious Pie" tweet detector
  - Positive class: tweets about Delicious Pie Co
  - Negative class: all other tweets

The 2-by-2 confusion matrix

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	**true positive**	**false positive**	$\textbf{precision} = \frac{tp}{tp+fp}$
	system negative	**false negative**	**true negative**	
		$\textbf{recall} = \frac{tp}{tp+fn}$		$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$

Evaluation: Accuracy

- Why don't we use **accuracy** as our metric?
- Imagine we saw 1 million tweets
    - 100 of them talked about Delicious Pie Co.
    - 999,900 talked about something else
- We could build a dumb classifier that just labels every tweet "not about pie"
    - It would get 99.99% accuracy!!! Wow!!!!
    - But useless! Doesn't return the comments we are looking for!
    - That's why we use **precision** and **recall** instead

Evaluation: Precision

- % of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels)

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluation: Recall

- % of items actually present in the input that were correctly identified by the system.

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Why Precision and recall

- Our dumb pie-classifier
    - Just label nothing as "about pie"

Accuracy=99.99%

but

Recall = 0

- (it doesn't get any of the 100 Pie tweets)

Precision and recall, unlike accuracy, emphasize true positives:

- finding the things that we are supposed to be looking for.

A combined measure: F

- F measure: a single number that combines P and R:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

- We almost always use balanced F_1 (i.e., $\beta = 1$)

$$\mathrm{F}_1 = \frac{2PR}{P + R}$$

How to combine P/R from 3 classes to get one metric

- Macro averaging:
  - compute the performance for each class, and then average over classes
- Micro averaging:
  - collect decisions for all classes into one confusion matrix
  - compute precision and recall from that table.

Macroaveraging and Microaveraging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Statistical Significance Testing

How do we know if one classifier is better than another?

- Given:
  - Classifier A and B
  - Metric M: M(A,x) is the performance of *A* on testset *x*
  - δ(x): the performance difference between A, B on x:
    - δ(x) = M(A,x) – M(B,x)

  - We want to know if δ(x)>0, meaning A is better than B
  - δ(x) is called the **effect size**
  - Suppose we look and see that δ(x) is positive. Are we done?
  - No! This might be just an accident of this one test set, or circumstance of the experiment. Instead:

Statistical Hypothesis Testing

- Consider two hypotheses:
  - Null hypothesis: A isn't better than B
  - A is better than B

$$H_0 : \delta(x) \leq 0$$
$$H_1 : \delta(x) > 0$$

- We want to rule out H_0
- We create a random variable X ranging over test sets
- And ask, how likely, if H_0 is true, is it that among these test sets we would see the δ(x) we did see?
  - Formalized as the p-value:

$$P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$$

Statistical Hypothesis Testing

$$P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$$

- In our example, this p-value is the probability that we would see δ(*x*) assuming H_0 (=*A* is not better than *B)*.
    - If H_0 is true but δ(*x*) is huge, that is surprising! Very low probability!
- A very small p-value means that the difference we observed is very unlikely under the null hypothesis, and we can reject the null hypothesis
- Very small: .05 or .01
- A result(e.g., "*A* is better than *B*") is **statistically significant** if the δ we saw has a probability that is below the threshold and we therefore reject this null hypothesis.

Statistical Hypothesis Testing

- How do we compute this probability?
- In NLP, we don't tend to use parametric tests (like t-tests)
- Instead, we use non-parametric tests based on sampling: artificially creating many versions of the setup.
- For example, suppose we had created zillions of testsets x'.
    - Now we measure the value of δ(x') on each test set
    - That gives us a distribution
    - Now set a threshold (say .01).
    - So if we see that in 99% of the test sets $\delta(x) > \delta(x')$
        - We conclude that our original test set delta was a real delta and not an artifact.

Statistical Hypothesis Testing

- Two common approaches:
  - approximate randomization
  - bootstrap test
- Paired tests:
  - Comparing two sets of observations in which each observation in one set can be paired with an observation in another.
  - For example, when looking at systems A and B **on the same test set**, we can compare the performance of system *A* and *B* on each same observation x_i

Avoiding Harms in Classification

Harms in text classifiers

- Kiritchenko and Mohammad (2018) found that most sentiment classifiers assign lower sentiment and more negative emotion to sentences with African American names in them.
  - This perpetuates negative stereotypes that associate African Americans with negative emotions
- Toxicity detection is the task of detecting hate speech, abuse, harassment, or other kinds of toxic language
- But some toxicity classifiers incorrectly flag as being toxic sentences that are non-toxic but simply mention identities like blind people, women, or gay people.
  - This could lead to censorship of discussion about these groups.

What causes these harms?

- Can be caused by:
  - Problems in the training data; machine learning systems are known to amplify the biases in their training data.
  - Problems in the human labels
  - Problems in the resources used (like lexicons)
  - Problems in model architecture (like what the model is trained to optimized)
- Mitigation of these harms is an open research area
- Meanwhile: **model cards**

Model Cards

(Mitchell et al., 2019)

- For each algorithm you release, document:
  - training algorithms and parameters
  - training data sources, motivation, and preprocessing
  - evaluation data sources, motivation, and preprocessing
  - intended use and users
  - model performance across different demographic or other groups and environmental situations

Part2:Generative and Discriminative Classifiers

2-Logistic Regression

Logistic Regression

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks

Generative and Discriminative Classifiers

- Naive Bayes is a **generative** classifier

- by contrast:

- Logistic regression is a **discriminative** classifier

Generative and Discriminative Classifiers

Suppose we're distinguishing cat from dog images

imagenet

imagenet

Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Now given a new image:

Run both models and see which one fits better

Discriminative Classifier

Just try to distinguish dogs from cats

Oh look, dogs have collars!
Let's ignore everything else

Finding the correct class c from a document d in Generative vs Discriminative Classifiers

- Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overbrace{P(d|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}$$

- Logistic Regression

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overset{\text{posterior}}{P(c|d)}$$

Components of a probabilistic machine learning classifier

Given *m* input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \dots, x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

The two phases of logistic regression

- **Training**: we learn weights *w* and *b* using **stochastic gradient descent** and **cross-entropy loss**.

- **Test**: Given a test example *x* we compute $p(y|x)$ using learned weights *w* and *b*, and return whichever label ($y = 1$ or $y = 0$) is higher probability

Logistic Regression for one observation x

- Input observation: vector $x = [x_1, x_2, \ldots, x_n]$
- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
- Output: a predicted class $\hat{y} \in \{0,1\}$

(multinomial logistic regression: $\hat{y} \in \{0, 1, 2, 3, 4\}$)

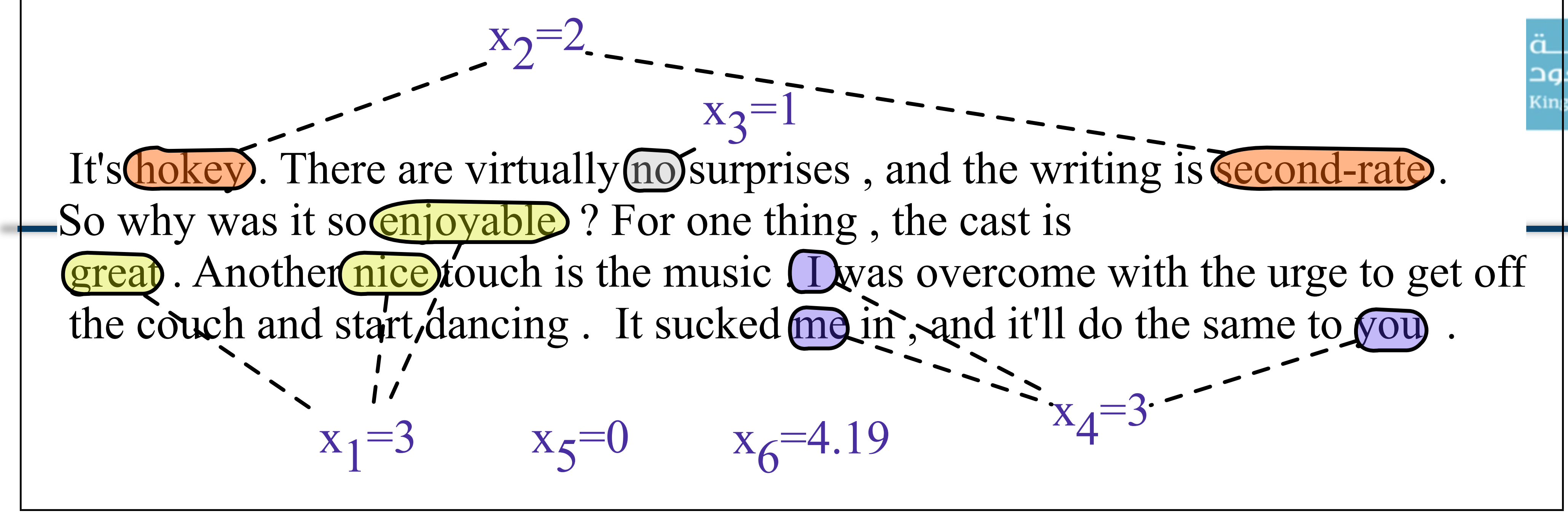
Making probabilities with sigmoids

$$\begin{aligned}
P(y=1) &= \sigma(w \cdot x + b) \\
&= \frac{1}{1+\exp\left(-(w \cdot x + b)\right)}
\end{aligned}$$

$$\begin{aligned}
P(y=0) &= 1 - \sigma(w \cdot x + b) \\
&= 1 - \frac{1}{1+\exp\left(-(w \cdot x + b)\right)} \\
&= \frac{\exp\left(-(w \cdot x + b)\right)}{1+\exp\left(-(w \cdot x + b)\right)}
\end{aligned}$$

Sentiment example: does y=1 or y=0?

- It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is
- great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .


x2=2
x3=1
It's hokey. There are virtually no surprises , and the writing is second-rate.
So why was it so enjoyable ? For one thing , the cast is
great . Another nice touch is the music . I was overcome with the urge to get off
the couch and start dancing . It sucked me in , and it'll do the same to you .
x1=3
x5=0
x6=4.19
x4=3

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Classifying sentiment for input x

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Suppose w = $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

b = 0.1

Classifying sentiment for input x

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70
\end{aligned}
$$

$$
\begin{aligned}
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{if } w \cdot x + b > 0$$

$$\text{if } w \cdot x + b \leq 0$$

Wait, where did the W's come from?

- Supervised classification:
- We know the correct label y (either 0 or 1) for each x.
- But what the system produces is an estimate, $\hat{y}$
- We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update w and b to minimize the loss.

Learning components

- A loss function:
  - **cross-entropy loss**
- An optimization algorithm:
  - **stochastic gradient descent**

Hyperparameters

- The learning rate η is a **hyperparameter**
  - too high: the learner will take big steps and overshoot
  - too low: the learner will take too long
- Hyperparameters:
- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

Overfitting

- A model that perfectly match the training data has a problem.
- It will also **overfit** to the data, modeling noise
  - A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.
- A good model should be able to **generalize**

Regularization

- A solution for overfitting
- Add a regularization term *R*(θ) to the loss function (for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

- Idea: choose an *R*(θ) that penalizes large weights
  - fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 Regularization (= ridge regression)

- The sum of the squares of the weights
- The name is because this is the (square of the) **L2 norm** $||\theta||_2$, = **Euclidean distance** of θ to the origin.

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

- L2 regularized objective function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 Regularization (= lasso regression)

- The sum of the (absolute value of the) weights
- Named after the **L1 norm** $||W||_1$, = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

- L1 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

References:

- Chapter 4 and Chapter 5 Daniel Jurafsky and James H. Martin. " Speech and Language Processing, 3rd edition ."2021