IN4080 – 2022 FALL

NATURAL LANGUAGE PROCESSING

Jan Tore Lønning

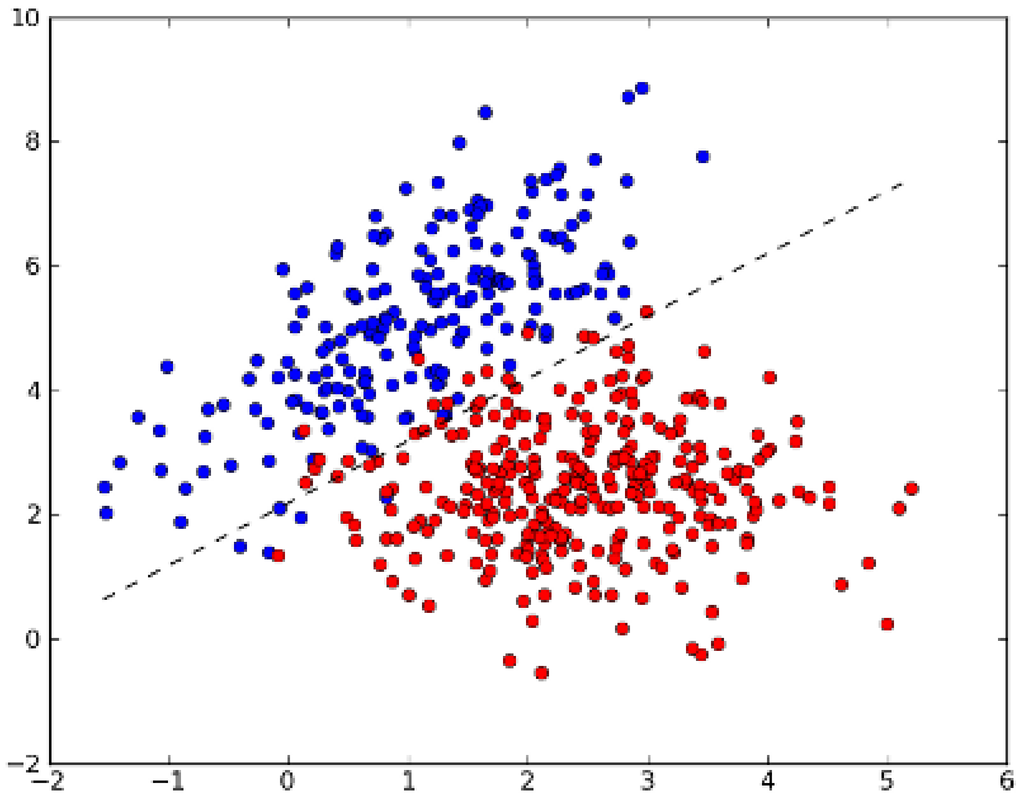
Lecture 5, 22 Sept

Today

- Multinomial Logistic Regression
- Representing categorical features
- Naïve Bayes vs. Logistic Regression
- Evaluation
- Language models

Repeat: Logistic Regression - Decision

- Two classes: C and $\bar{C}$
- An observation: $\boldsymbol{x} = (x_1, \dots, x_n)$
- Model weights: $\mathbf{w} = (w_0, \dots, w_n)$
- Assign class C to $\boldsymbol{x}$ iff
  - $z = \sum_{i=0}^{n} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x} > 0$
  - $e^z > 1$
  - $\hat{y} = P(C|\boldsymbol{x}) = \sigma(z) = \frac{1}{1+e^{-z}} > 0.5$

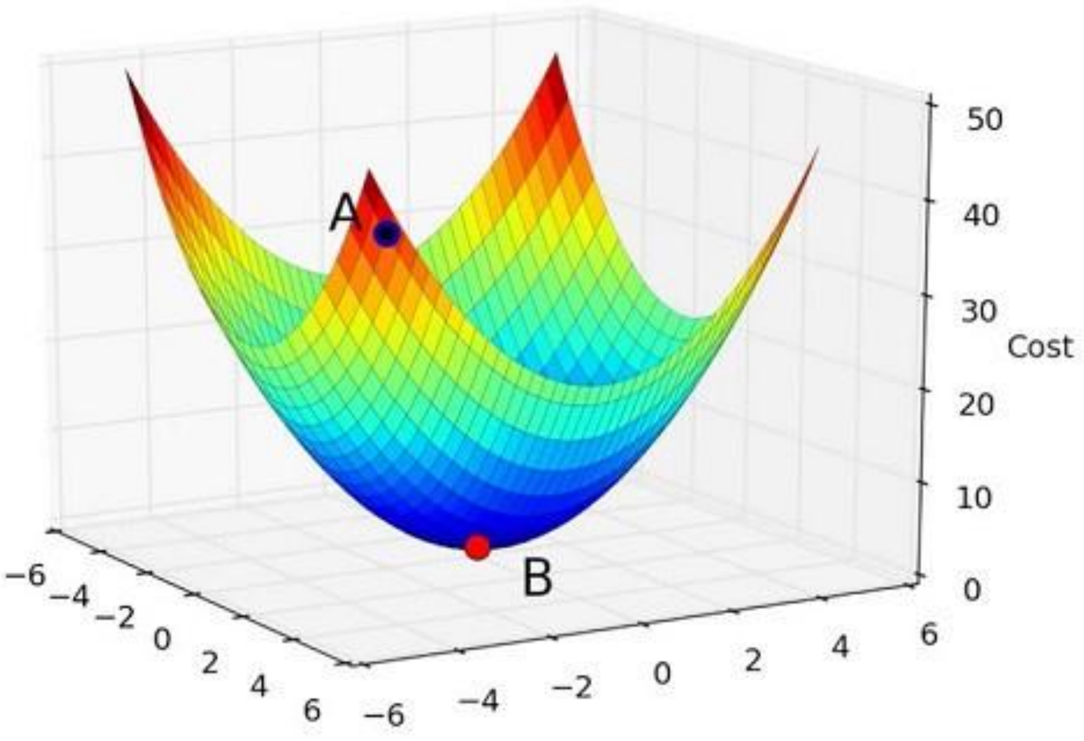


n features

Logistic Regression: Learning

- Objective: reduce the loss
- Cross-entropy loss:
  - (= max. joint probability)
  - $L_{CE}(\vec{w}) = \sum_{j=1}^{m} - \log P(y^{(j)} | \vec{x}^{(j)})$
- Gradient descent:
  - $w_i \leftarrow (w_i - \eta \frac{\partial}{\partial w_i} L_{CE}(\hat{y}, y))$
- For one observation $\boldsymbol{x}^{(j)}$:
  - $w_i \leftarrow (w_i - \eta (\hat{y}^{(j)} - y^{(j)}) x_i^{(j)})$

m observations, observation j, feature i

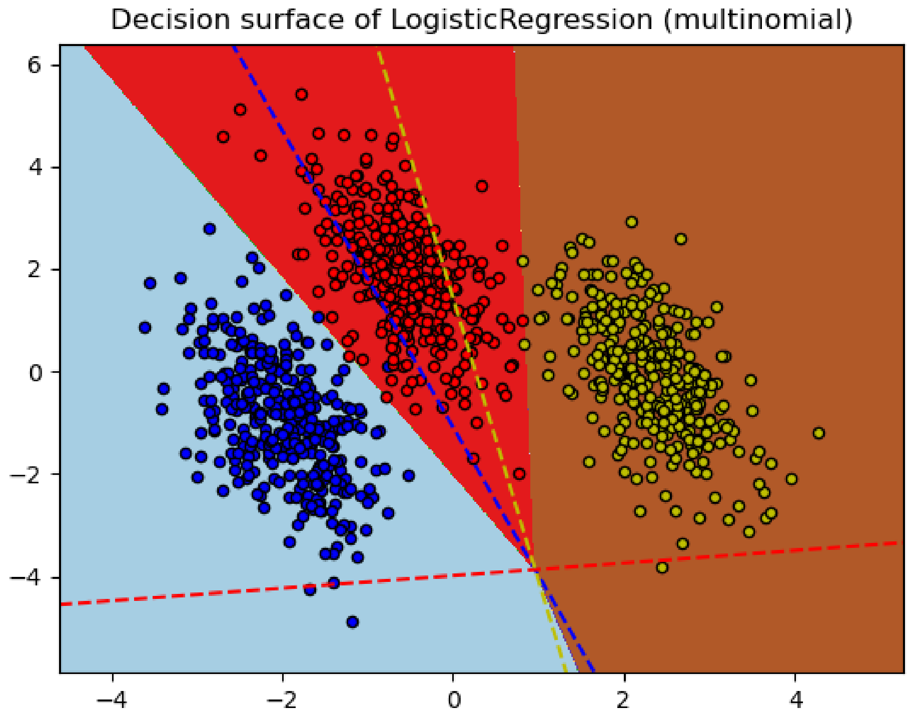


weights

Multinomial Logistic Regression

- A type of multi-class classifier:
  - A finite set of classes $C_i, i = 1, \dots, k$
  - An observation $\boldsymbol{x} = (x_1, \dots, x_n)$ is assigned to exactly one of the classes
- A model consists of weights for each class:
  - $\boldsymbol{w_i} = (w_{i,0}, \dots, w_{i,n})$
- Consider a linear expression for each class
  - $z_i = \boldsymbol{w}_i \cdot \boldsymbol{x} = \sum_{j=0}^{n} w_{i,j} x_j$
- Choose the class C_i with the largest z_i

n features, k classes, class i, feature j



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html

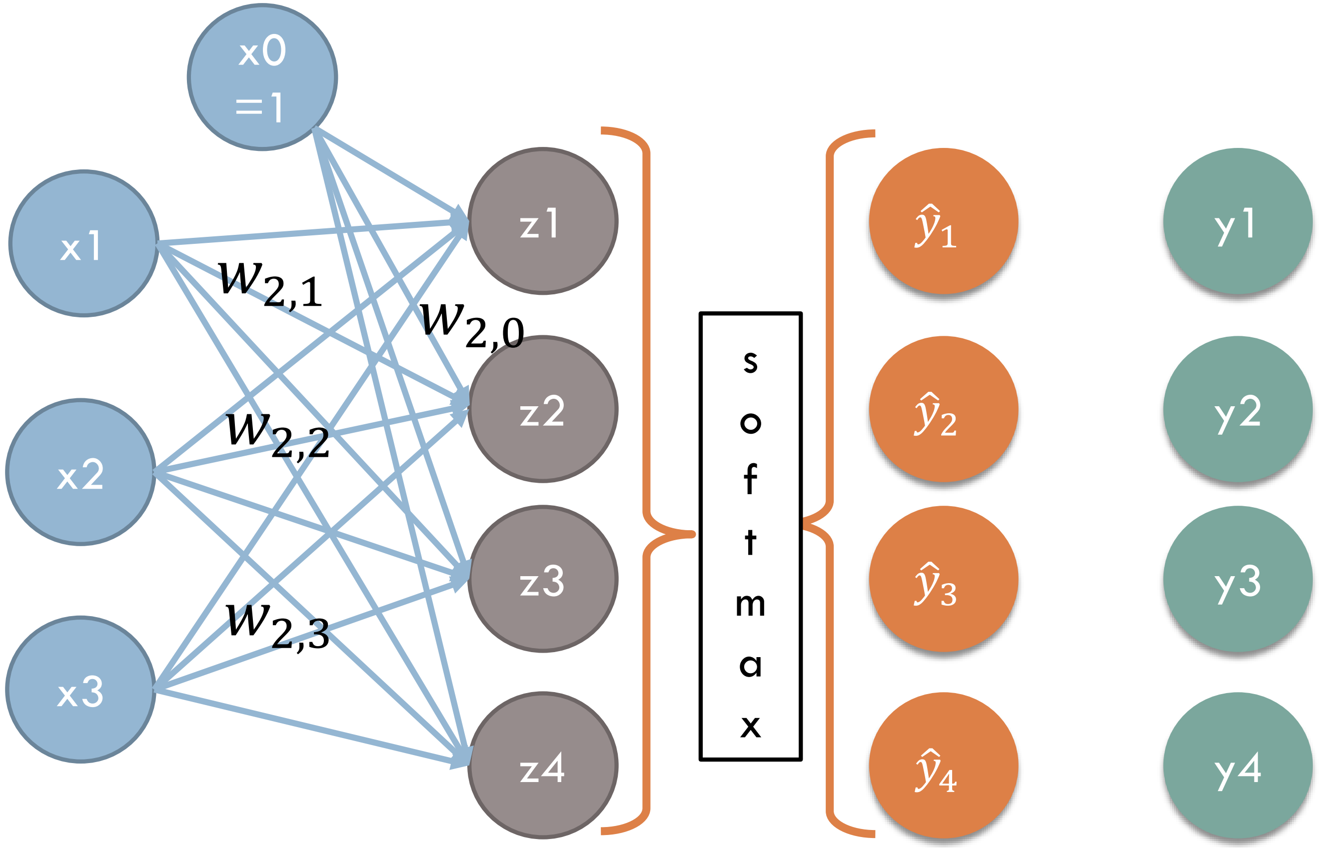
Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

Multinomial Logistic Regression

- $z_i = \boldsymbol{w_i} \cdot \boldsymbol{x} = \sum_{j=0}^{n} w_{i,j} x_j$
- The probability of class C_i:
  $\hat{y}_i = P(C_i|\boldsymbol{x}) = (softmax(z_1, \dots, z_k))_i$

$$= \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$

- Choose the class C_i with
  - the largest z_i
  - the largest $\hat{y}_i = P(C_i|\boldsymbol{x})$



n features, k classes

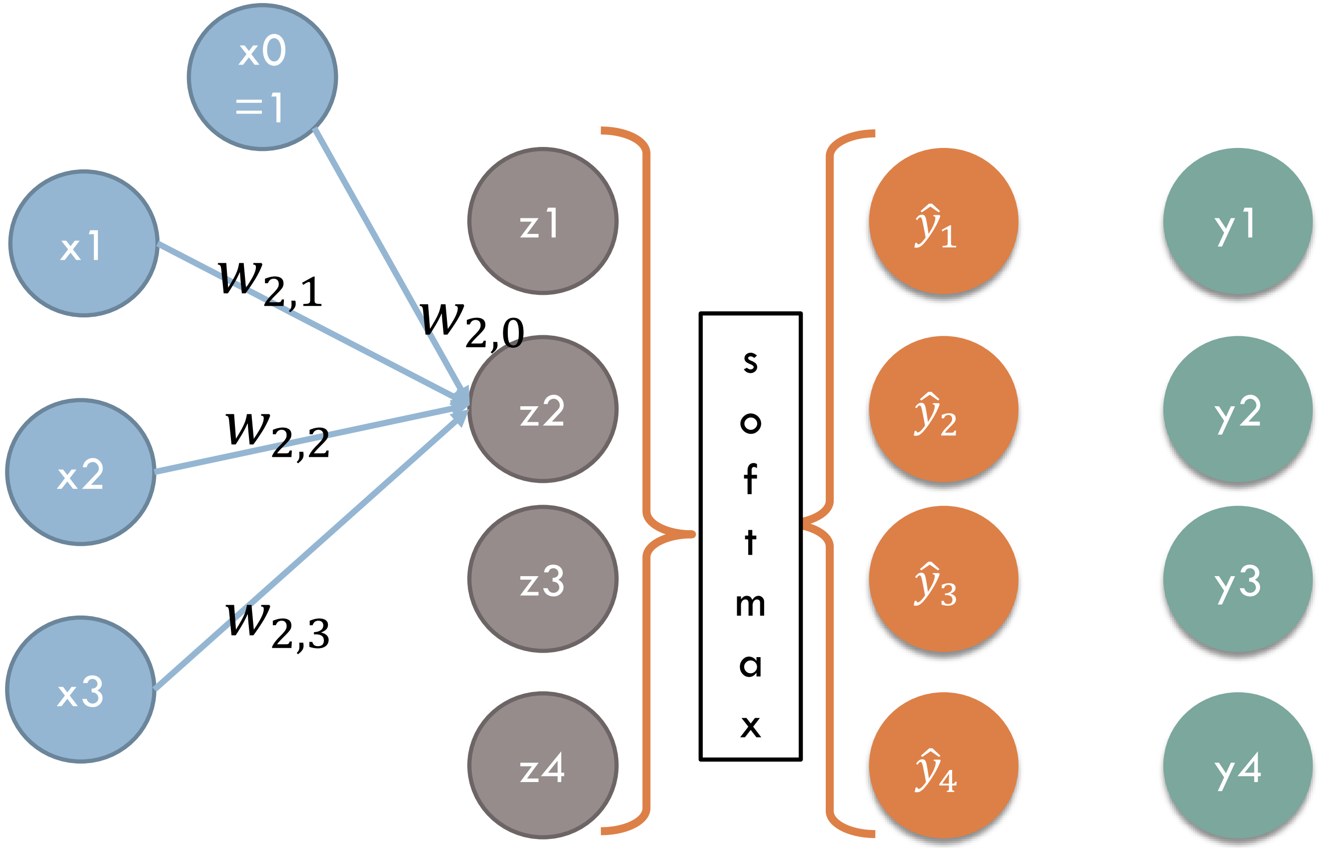
Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

Connections going into a node

- $z_i = \boldsymbol{w_i} \cdot \boldsymbol{x} = \sum_{j=0}^{n} w_{i,j} x_j$
- The probability of class C_i: $\hat{y}_i = P(C_i|\boldsymbol{x}) = (softmax(z_1, \ldots, z_k))_i$

$$= \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$

- Choose the class C_i with
  - the largest z_i
  - the largest $\hat{y}_i = P(C_i|\boldsymbol{x})$



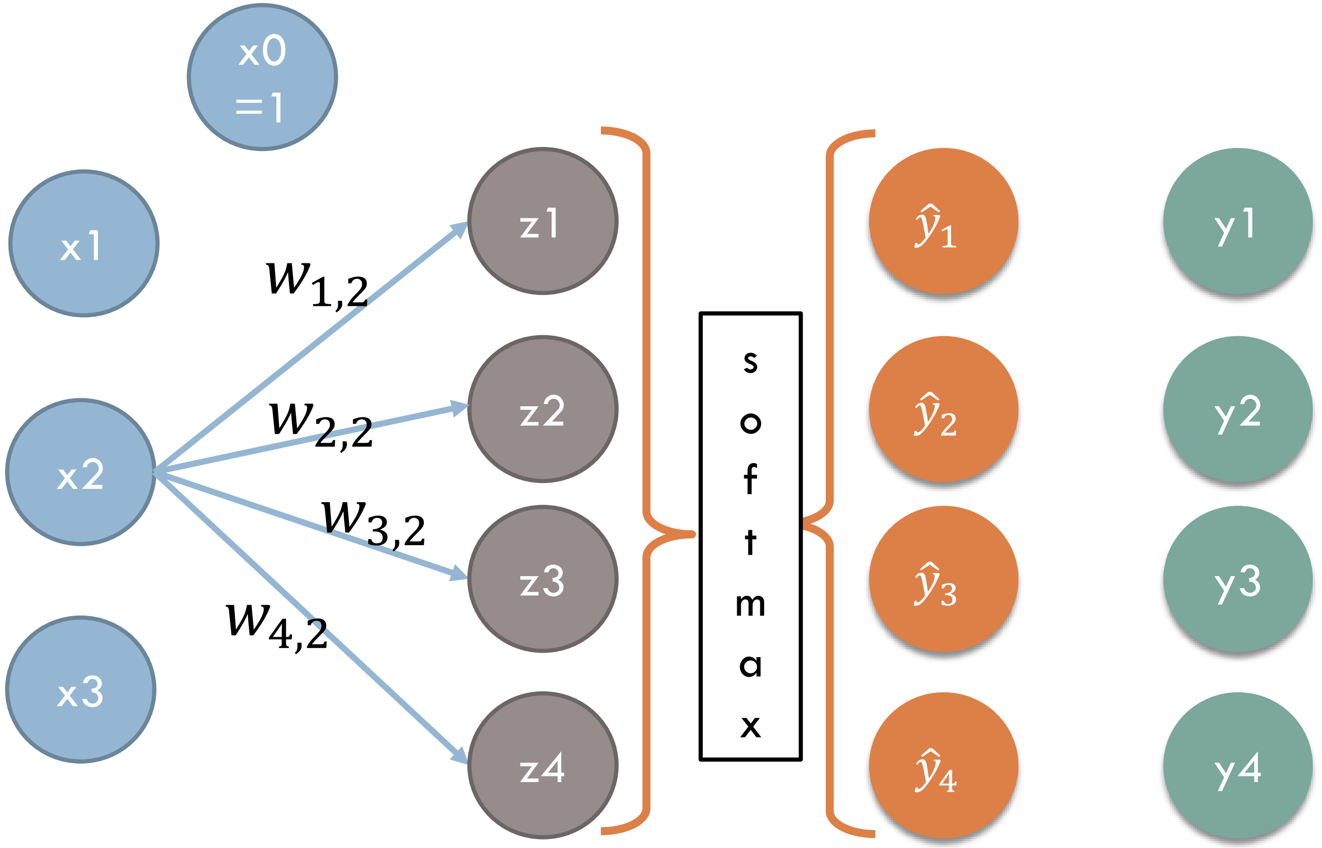
n features, k classes

Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

Connections going out of a node

- $z_i = \boldsymbol{w_i} \cdot \boldsymbol{x} = \sum_{j=0}^{n} w_{i,j} x_j$
- The probability of class C_i:
  $\hat{y}_i = P(C_i|\boldsymbol{x}) =$
  $(softmax(z_1, \ldots, z_k))_i$

  $$= \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$
- Choose the class C_i with
  - the largest z_i
  - the largest $\hat{y}_i = P(C_i|\boldsymbol{x})$



n features, k classes

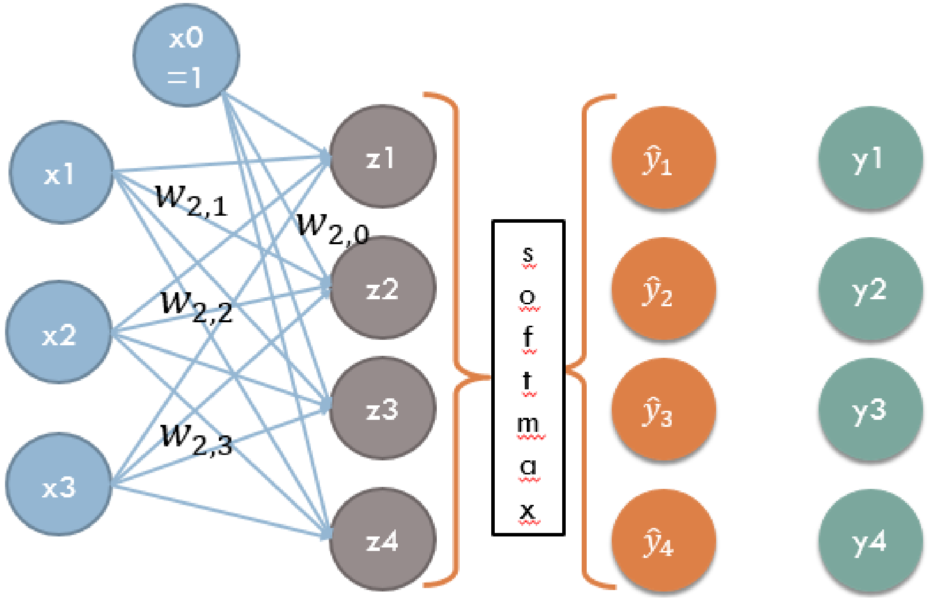
Beware: Jurafsky and Martin uses $w_{i,j}$ where Marsland, IN3050, uses $w_{j,i}$

Matrix form

$$W\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ \boxed{b_2} \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} z_1 \\ \boxed{z_2} \\ \vdots \\ z_m \end{bmatrix} = \mathbf{z}$$

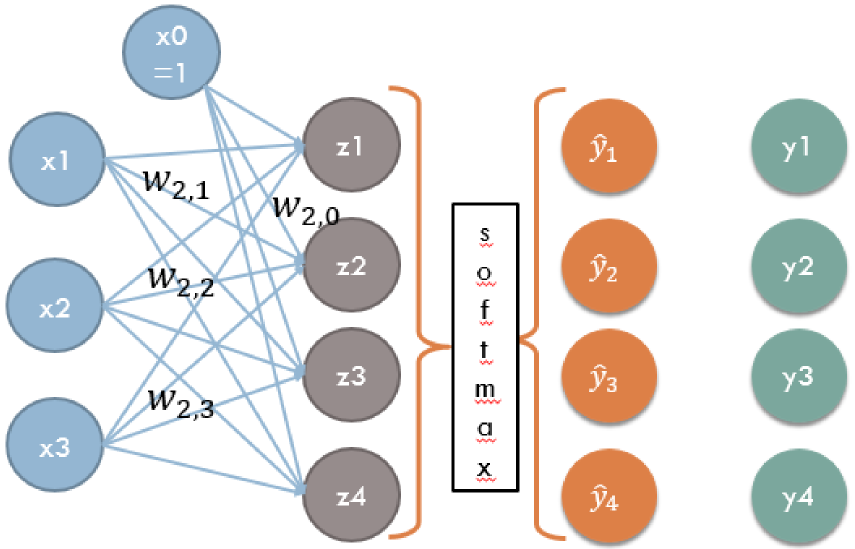
Oops: n features, m classes

- For those of you who know matrices:
  - The connections between the layers: a matrix
  - Running it through the connections: matrix multiplication



Training Multinomial Logistic Regression

- One observation
  - Target of form $\mathbf{y} = (0, \ldots, 0,1,0, \ldots, 0)$
    - say $y_c = 1$ and $y_j = 0$ for $j \neq c$
  - Compare the predicted $\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, .. \hat{y}_k)$
  - to the target labels using cross-entropy loss
    - $L_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{j=1}^{k} y_j \log \hat{y}_j$



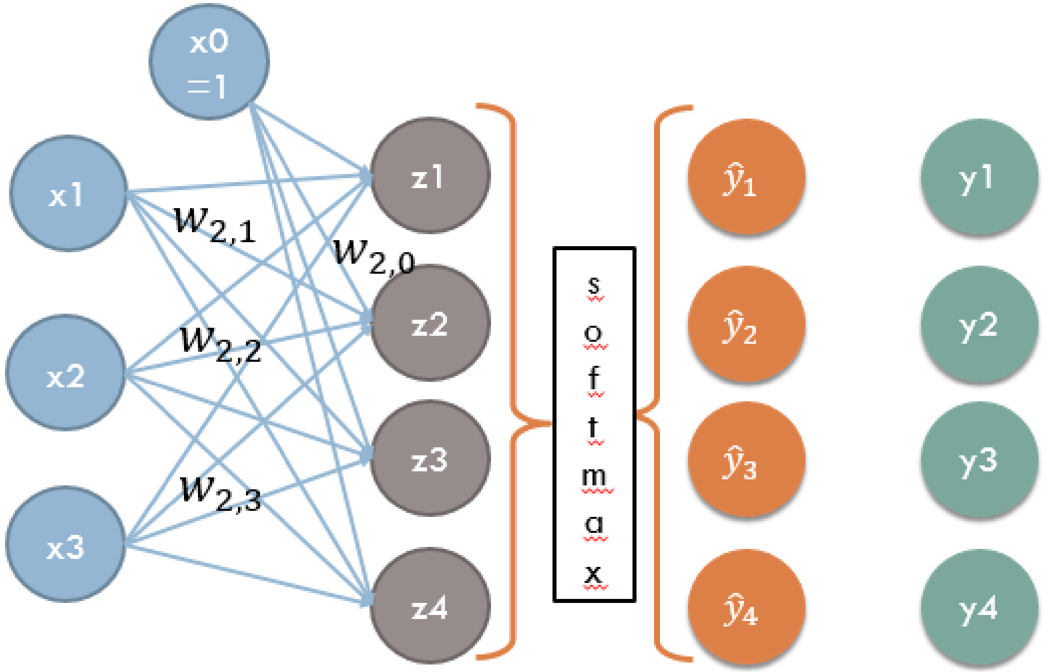
- A batch $Y = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$:
  - $L_{CE}(Y, \hat{Y}) = \sum_{j=1}^{m} L_{CE}(\hat{y}^{(j)}, y^{(j)})$

m observations, n features, k classes

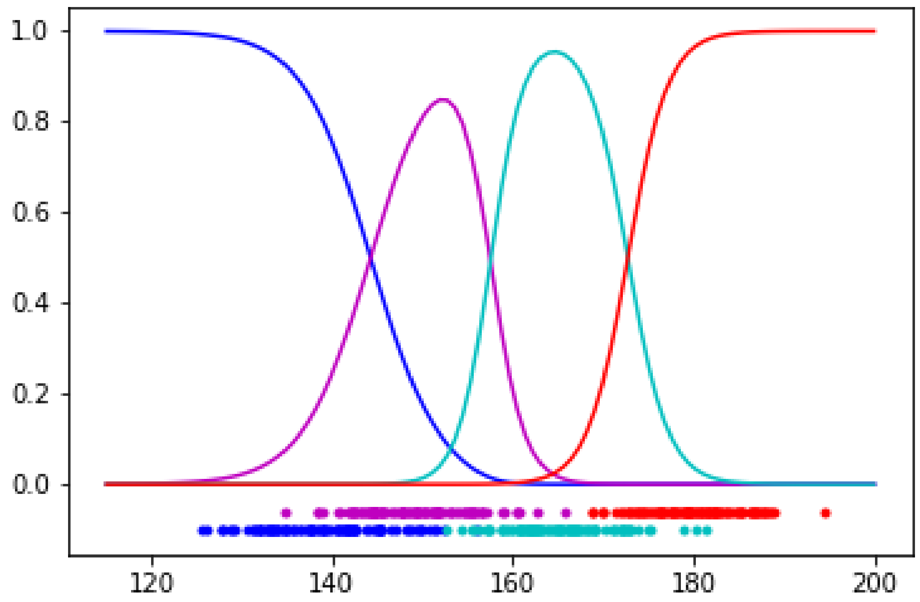
Training Multinomial Logistic Regression

- Gradient descent:
  - partial derivatives
  - + some algebra
  - yield update rule:
    - $w_{i,j} = w_{i,j} - \eta(\hat{y}_i - y_i)x_j$
  - which means
    - $w_{c,j} = w_{c,j} + \eta(1 - \hat{y}_c)x_j$
    - $w_{i,j} = w_{i,j} - \eta(\hat{y}_i)x_j$, for $j \neq c$
  - c.f. J&M (5.47)

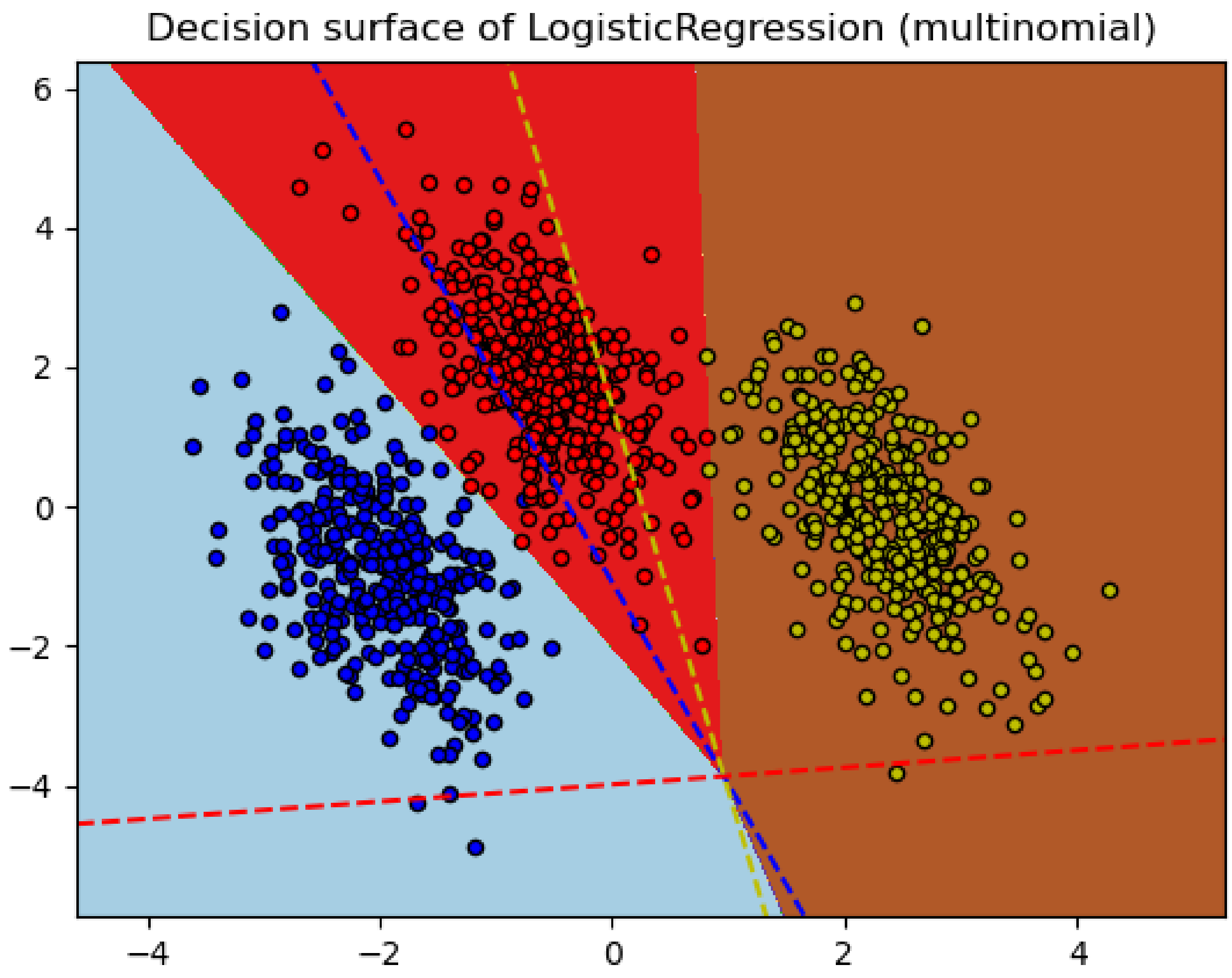
n features, k classes



Example: softmax



- 4 different classes corresponding to the dots below the 0-line
- For each of them:
  - a corresponding softmax curve
  - = the probability of the observation belonging to this class
- Similarly with two features
  - A surface for each class
  - The intersections of the surfaces project to straight lines in the xy-plane
    - = decision boundaries



The decision boundaries turn out to be straight lines

https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html

Categorical features

Categories as numbers

- In the naive Bayes model we could handle categorical values directly, e.g., characters:
  - What is the probability that c_n = ‘z’
- But many classifier can only handle numerical data
- How can we represent categorical data by numerical data?
- (In general, it is not a good idea to just assign a single number to each category: ~~‘a’→1, ‘b’→2, ‘c’→ 3, …)~~

Data representation

Assume the following example

	4 different featues				Classes
feature	f1	f2	f3	f4	
type	cat	cat	Bool (num)	num	
Value set	a, b, c	x, y	True, False	0, 1, 2, 3, …	Class1, class2

Dictionary representation in NLTK

```
[({'f1': 'a', 'f2': 'y', 'f3': True, 'f4': 5}, 'class_1'),
 ({'f1': 'b', 'f2': 'y', 'f3': False, 'f4': 2}, 'class_2'),
 ({'f1': 'c', 'f2': 'x', 'f3': False, 'f4': 4}, 'class_1')]
```

3 training instances

4 features

class

One-hot encoding

feature 1				feature 2	
a	b	c		x	y
(1,0,0)	(0,1,0)	(0,0,1)		(1,0)	(0,1)

- Represent categorical variables as vectors/arrays of numerical variables

Representation in scikit: "one hot" encoding

NLTK

```
[({'f1': 'a', 'f2': 'y', 'f3': True, 'f4': 5}, 'class_1'),
 ({'f1': 'b', 'f2': 'y', 'f3': False, 'f4': 2}, 'class_2'),
 ({'f1': 'c', 'f2': 'x', 'f3': False, 'f4': 4}, 'class_1')]
```

3 training instances

4 features

class

scikit

```
X_train:
array([[ 1.,  0.,  0.,  0.,  1.,  1.,  5.],
       [ 0.,  1.,  0.,  0.,  1.,  0.,  2.],
       [ 0.,  0.,  1.,  1.,  0.,  0.,  4.]])
```

7 features

3 training instances

```
train_target: ['class_1', 'class_2', 'class_1'], or
train_target: [1, 2, 1]
```

3 corresponding classes

One-hot encoding		
a	b	c
[1, 0, 0]	[0, 1,0]	[0, 0, 1]

Converting a dictionary

- We can construct the data to scikit directly
- Scikit has methods for converting Python-dictionaries/NLTK-format to arrays

```
» train_data = [inst[0] for inst in train]
» train_target = [inst[1] for inst in train]
» v = DictVectorizer()
» X_train=v.fit_transform(train_data)

» X_test=v.transform(test_data)
```

1. Constructs (=fit) repr. format
2. Transform

Transform
Use same v as for train

Multinomial NB in scikit

- We can construct the data to scikit directly
- Scikit has methods for converting text to bag of words arrays

```
» train_data=["en rose er en rose",
              "anta en rose er en fiol"]
» v = CountVectorizer()
» X_train=v.fit_transform(train_data)

» print(X_train.toarray())
  [[0 2 1 0 2]
  [1 2 1 1 1]]
```

- Positions corresponds to [anta, en, er, fiol, rose]

Sparse vectors

- One hot encoding uses space
- 26 English characters:
  - Each is represented as a vector with 25 ‘0’-s and a single ‘1’
- Bernoulli NB text. classifier with 2000 most frequent words
  - Each word represented by a vector with 1999 ‘0’-s and a single ‘1’.
- scikit-learn uses internally a dictionary-like representation for these vectors, called ”sparse vectors”

Naïve Bayes vs. Logistic Regression

Naïve Bayes vs. Logistic Regression

- Both are probability-based and make a hard decision by choosing
  - $\underset{C_i \in \mathcal{C}}{\operatorname{argmax}} P(C_i|\boldsymbol{x})$
- For Naïve Bayes:
  - $\underset{C_i \in \mathcal{C}}{\operatorname{argmax}} P(C_i|\boldsymbol{x}) = \underset{C_i \in \mathcal{C}}{\operatorname{argmax}} P(C_i) \prod_{j=1}^{n} P\left(v_j = x_j|C_i\right) =$
    $\underset{C_i \in \mathcal{C}}{\operatorname{argmax}} (\log\left(P(C_i)\right) + \sum_{i=1}^{n} \log(P\left(v_j = x_j|C_i\right)))$

    $w_{i,0}$ $\qquad$ $w_{i,j}x_j$
  - a linear expression for each class like the Log.Reg

Comparing NB and LogReg

- NB is an instance of LogReg,
  - i.e. one possible choice of weights
- LogReg will do at least as well as NB on the training data
  - with respect to the cross-entropy loss
  - (without any regularization)
- When the independence assumptions holds, NB will do as well as LogReg
- When the independence assumptions does not hold, NB may put too much weight on some features
- LogReg will not do this: If we add features that depend on other features, LogReg will put less weight on them

Comparing NB and LogReg

- NB is a generative classifier:
  - It has a model of how the data are generated
  - $P(C)P\left(\vec{f}\middle|C\right) = P(\vec{f}, C)$
- LogReg is a discriminative classifier
  - It only considers the conditional probability $P\left(C|\vec{f}\right)$

Comparing cats and dogs

Generative

- Comparing cats and dogs:
  - a cat model/distribution
  - a dog model
- If we also want to compare dogs and wolfs
  - we use the same dog model:
    - features
    - weights

Discriminative

- The model is determined by the classes and the differences between them
- Consider other features and weights for dog when comparing to wolf than to cat.

Generating positive movie reviews

- First choose the length of the review, say *n=1000* words
- Then choose the first word
  - according to the probability distribution P(w | 'pos') e.g.
    - $\hat{P}(w = the|pos) = 0.1$
    - $\hat{P}(w = pitt|pos) = \frac{31}{798\,742}$
- Then choose word 2, etc. up to word 1000

- Observation:
  - Whether we compare to negative film reviews or positive book reviews, we will use the same features

- Footnote:
  - The multinomial text model tacitly suppress "choose length of document", and assumes it is independent of class

Discriminative classifiers

- A discriminative classifier considers the probability of the class given the observation directly.
- E.g. a discriminative text classifier may focus on the features:
  - *terrible* and *terrific* for pos. vs. neg film review
  - *director* and *author* for pos. film vs. pos. book review
- The discriminative classifier
  - may be more efficient
  - but gives less explanation
  - and may eventually focus on wrong features

Evaluation

Evaluation measure: Accuracy

- What does accuracy 0.81 tell us?
- Given a test set of 500 documents:
  - The classifier will classify 405 correctly
  - And 95 incorrectly
- A good measure given:
  - The 2 classes are equally important
  - The 2 classes are roughly equally sized
  - Example:
    - Woman/man
    - Movie reviews: pos/neg

But

- For some tasks, the classes aren't equally important
  - Worse to loose an important mail than to receive yet another spam mail

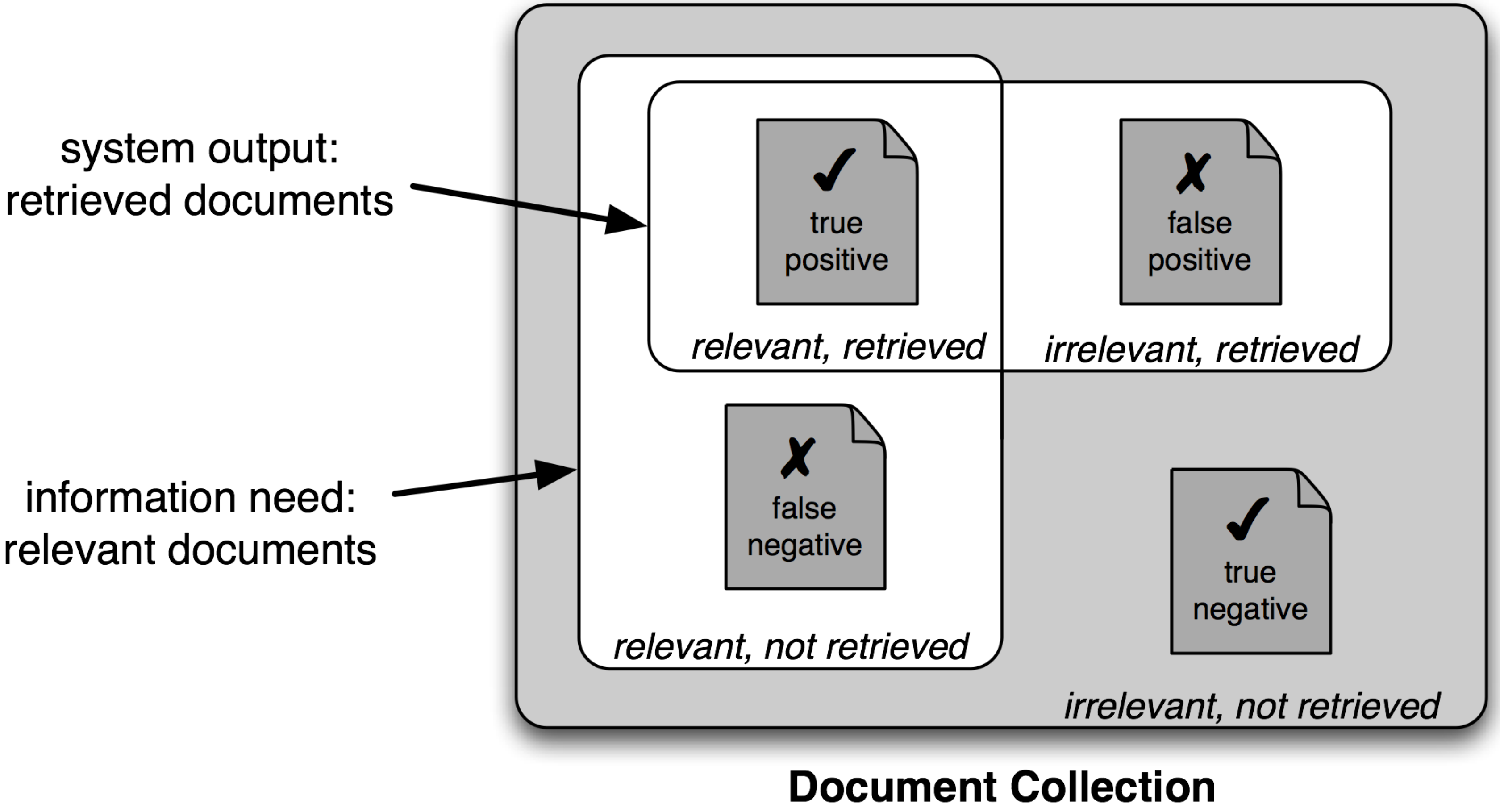
- For some tasks the different classes have different sizes.

Information retrieval (IR)

- Traditional IR, e.g. a library
  - Goal: Find all the documents on a particular topic out of 100 000 documents,
    - Say there are 5
  - The system delivers 10 documents: all irrelevant
    - What is the accuracy?

- For these tasks, focus on
  - The relevant documents
  - The documents returned by the system
- Forget the
  - Irrelevant documents which are not returned

IR - evaluation



Confusion matrix

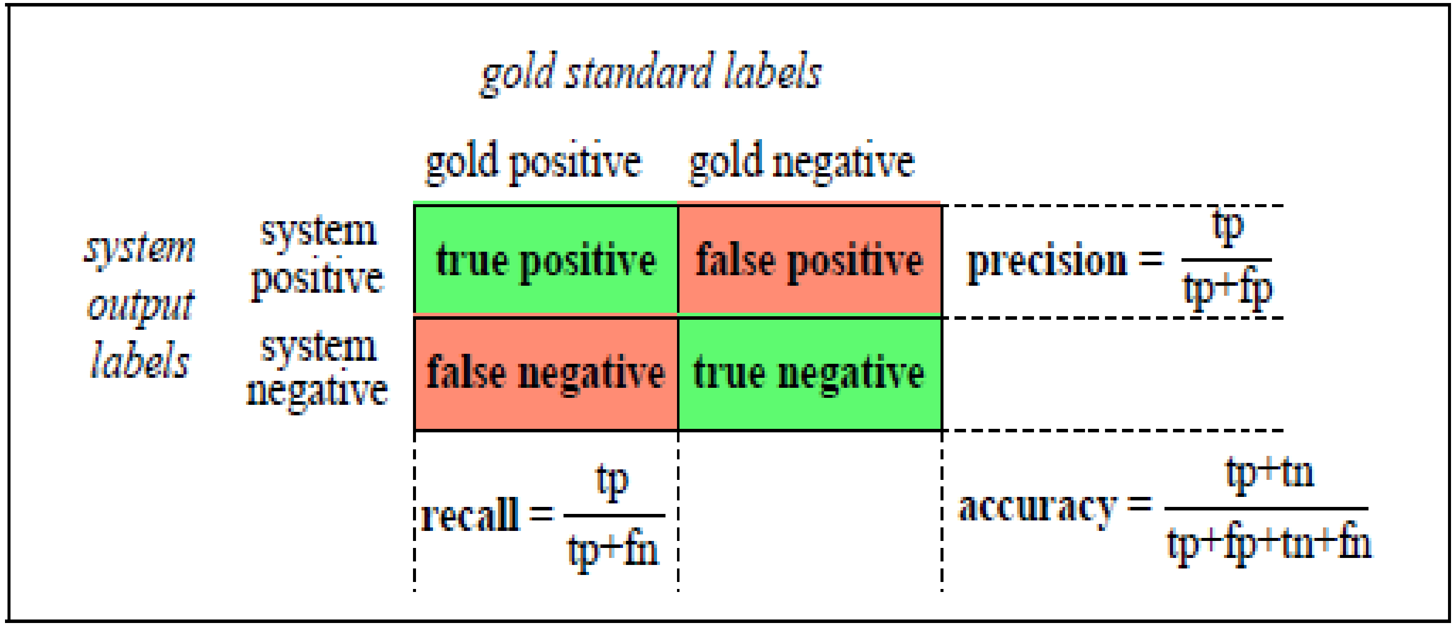


Figure 6.4 Contingency table

- Beware what the rows and columns are:
  - NLTKs ConfusionMatrix swaps them compared to this table

Evaluation measures

		Is in C	
		Yes	NO
Classifier	Yes	tp	fp
	No	fn	tn

- Accuracy: (tp+tn)/N
- Precision:tp/(tp+fp)
- Recall: tp/(tp+fn)

- F-score combines P and R
- $F_1 = \frac{2PR}{P+R} \left(= \frac{1}{\frac{\frac{1}{R}+\frac{1}{P}}{2}} \right)$
- F_1 called ''harmonic mean''
- General form
  - $F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$
  - for some $0 < \alpha < 1$

Confusion matrix

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Figure 6.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

- Precision, recall and f-score can be calculated for each class against the rest

Language Models

Probabilistic Language Models

- Goal: Ascribe probabilities to word sequences.
- Motivation:
  - Translation:
    - P(she is a tall woman) > P(she is a high woman)
    - P(she has a high position) > P(she has a tall position)
  - Spelling correction:
    - P(She met the prefect.) > P(She met the perfect.)
    - P(She met the prefect match.) < P(She met the perfect match.)
  - Speech recognition:
    - P(I saw a van) > P(eyes awe of an)

Probabilistic Language Models

- Goal: Ascribe probabilities to word sequences.
  - $P(w_1, w_2, w_3, \dots, w_n)$
- Related: the probability of the next word
  - $P(w_n \mid w_1, w_2, w_3, \dots, w_{n-1})$
- A model which does either is called a Language Model, LM
  - Comment: The term is somewhat misleading
    - (Probably origin from speech recognition where it is combined with an acoustic model)

Chain rule

- The two definitions are related by the chain rule for probability:
- $P(w_1, w_2, w_3, \ldots, w_n) =$
- $P(w_1) \times P(w_2 | w_1) \times P(w_3 | w_1, w_2) \times \cdots \times P(w_n | w_1, w_2, \ldots, w_{n-1}) =$
- $\prod_i^n P(w_i | w_1, w_2, \ldots, w_{i-1}) = \prod_i^n P\left(w_i | w_1^{i-1}\right)$

- *P("its water is so transparent") =*
  P(its) × P(water|its) × P(is|its water)
  × P(so|its water is) × P(transparent|its water is so)

- But this does not work for long sequences
  - (we may not even have seen before)

Markov assumption

- A word depends only on the immediate preceding word
- $P(w_1, w_2, w_3, \dots, w_n) \approx$
- $P(w_1) \times P(w_2|\, w_1) \times P(w_3|w_2\,) \times\cdots\times P(w_n|\, w_{n-1}) =$
- $\prod_i^n P(w_i|\, w_{i-1})$

- P("its water is so transparent") ≈

  P(its) × P(water | its) × P(is | water) × P(so | is) × P(transparent | so)

- This is called a bigram model

Estimating bigram probabilities

- The probabilities can be estimated by counting
- This yields maximum likelihood probabilities
  - (=maximum probable on the training data)
- $\hat{P}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)}{count(w_{i-1})}$

Example from J&M

$$\hat{P}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

General ngram models

- A word depends only on the k many immediately preceding words
- $P(w_1, w_2, w_3, \dots, w_n) \approx$
- $\prod_i^n P(w_i | \, w_{i-k}, w_{i+1-k}, \dots, w_{i-1}) = \prod_i^n P\left(w_i | \, w_{i-k}^{i-1}\right)$

- This is called a
  - unigram model – no preceding words
  - trigram model – two preceding words
  - *k*-gram model – *k*-1 preceding words

- We can train them similarly to the bigram model.
- Have to be more careful with the smoothing for larger *k*-s.

Generating with n-grams

- Goal: Generate a sequence of words
- Unigram:
  - Choose the first word according to how probable it is
  - Choose the second word according to how probable it is, etc.
  - = the generative model for multinomial NB text classification
- Bigram
  - Select word *k* according to $\hat{P}(w_i|w_{i-1})$
- *k*-gram
  - Select word w_i according to how probable it is given the $k-1$ preceding words $P\left(w_i|\, w_{i-k}^{i-1}\right)$

Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Unknown words

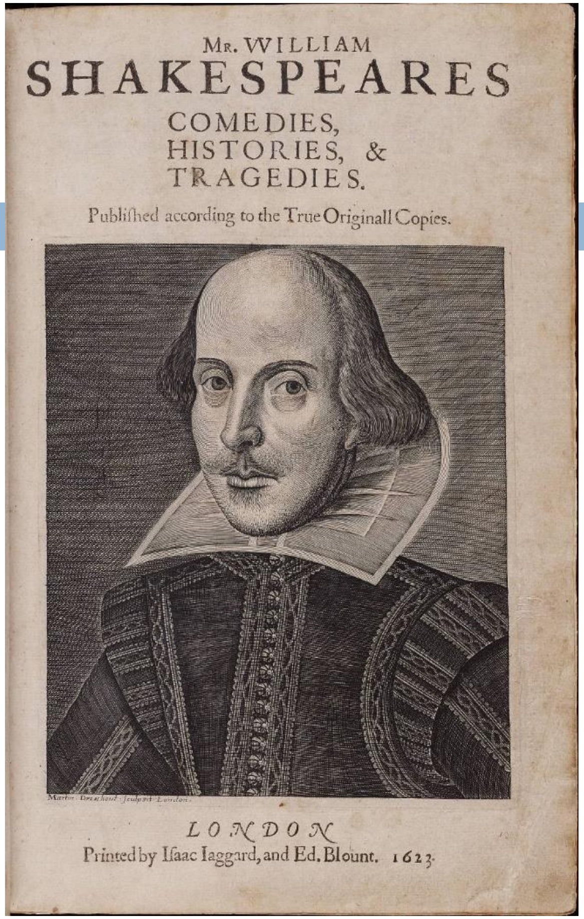
- There might be words that is never observed during training.
- Use a special symbol for unseen words during application, e.g. UNK
- Set aside a probability for seeing a new word
  - This may be estimated from a held-out corpus
- Adjust
  - the probabilities for the other words in a unigram model accordingly
  - the conditional probabilities of the *k-gram model*

Smoothing, Laplace, Lidstone

- Since we might not have seen all possibilities in training data, we might use Lidstone or, more generally, Laplace smoothing
- $\hat{P}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)+k}{count(w_{i-1})+k\,|V|}$
  - where $|V|$ is the size of the vocabulary V.

But:

- Shakespeare produced
  - N = 884,647 word tokens
  - V = 29,066 word types
- Bigrams:
  - Possibilities:
    - $V^2 = 844{,}000{,}000$
  - Shakespeare,
    - bigram tokens: 884,647
    - bigram types: 300,000



- Add-k smoothing is not appropriate

Smoothing n-grams

Backoff

- If you have good evidence, use the trigram model,
- If not, use the bigram model,
- or even the unigram model

Interpolation

- Combine the models

Use either of this. According to J&M interpolation works better

Interpolation

- Simple interpolation:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

- The λ-s can be tuned on a held out corpus
- A more elaborate model will condition the λ-s on the context
  - (Brings in elements of backoff)

Evaluation of n-gram models

- Extrinsic evaluation:
  - To compare two LMs, see how well they are doing in an application, e.g. translation, speech recognition
- Intrinsic evaluation:
  - Use a held out-corpus and measure $P(w_1, w_2, w_3, \ldots, w_n)^{\frac{1}{n}}$
    - The n-root compensate for different lengths
  - $\prod_i^n P\left(w_i \middle|\, w_{i-k}^{i-1}\right)^{\frac{1}{n}}$ for a k-gram model
  - It is normal to use the inverse of this, called the perplexity
  - $PP(w_1^n) = \dfrac{1}{P(w_1,w_2,w_3,\ldots,w_n)^{\frac{1}{n}}} = P(w_1, w_2, w_3, \ldots, w_n)^{-\frac{1}{n}}$

Properties of LMs

- The best smoothing is achieved with Kneser-Ney smoothing
- Short-comings of all n-gram models
  - The smoothing is not optimal
  - The context are restricted to a limited number of preceding words.

A practical advice: Use logarithms when working with n-grams