IN4080 – 2019 FALL
NATURAL LANGUAGE PROCESSING

Jan Tore Lønning

Logistic regression

Lecture 7, 3 Oct

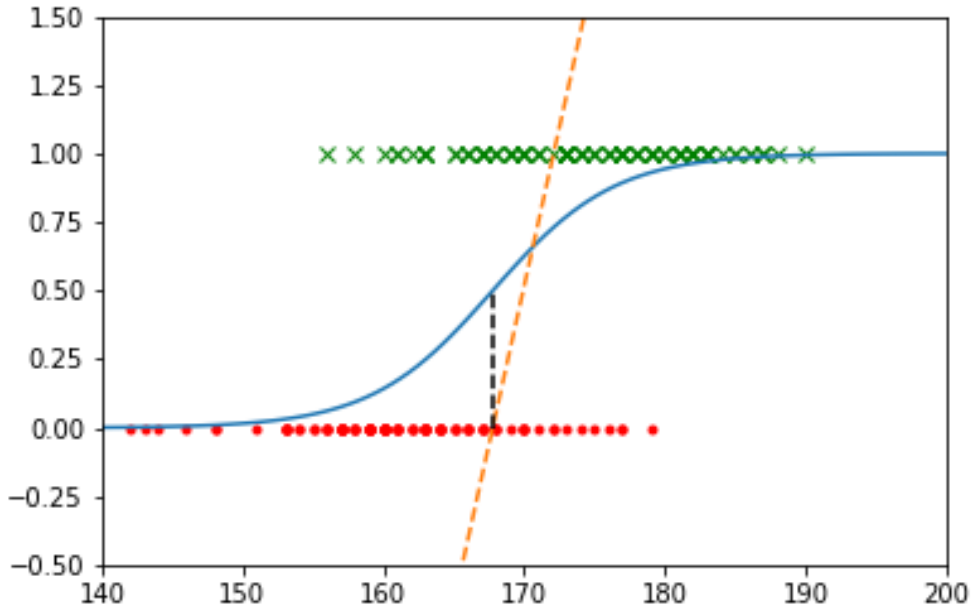
Last week

- Generative and discriminative classifiers
- Representing categorical features
- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Application to tagging
- Comparing to NB

Today

- Logistic regression, short recap
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Comparing Naïve Bayes and Logistic Regression
- Application to tagging

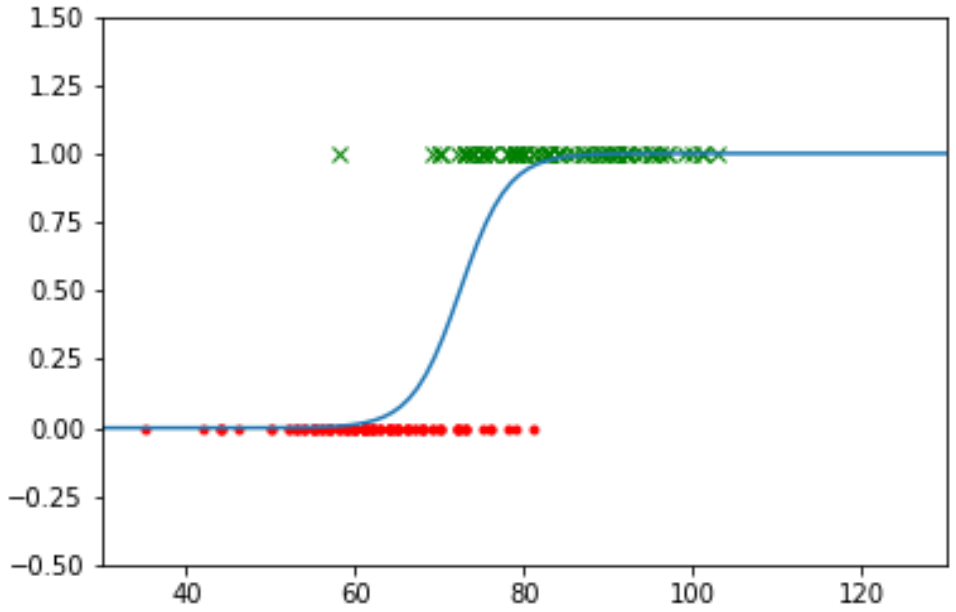
Logistic regression: repeat



- Instead of using the straight line
  - $y = ax + b$ and check $y > 0$?
- Apply the logistic function and check whether
  - $\sigma(y) = \frac{1}{1+e^{-(ax+b)}} > 0.5$
  - (here $ax + b = 0.23x - 38.7$)

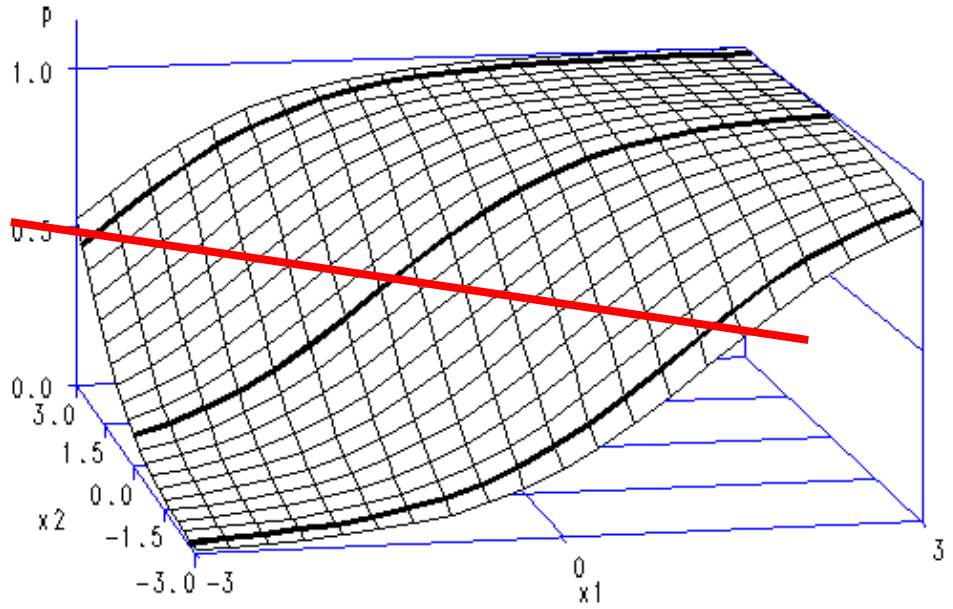
The figure shows the correct sigmoid curve for the height data.
Accuracy = 0.81 (on training data)

Another example



- This is the weights data
- Better separation, acc = 92.5
- The sigmoid on the figure is the one found by log.reg.
- Observe the different geometry of this curve compared to the heights curve

With two features

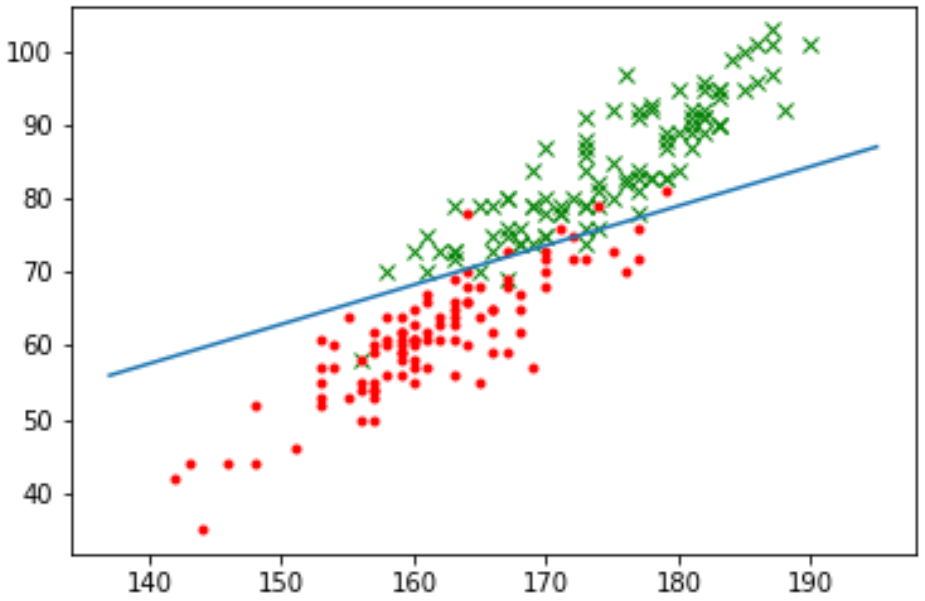


From IDRE, UCLA

- Two features: x_1, x_2
- Apply weights: w_0, w_1, w_2
- Let $y = w_0 + w_1 x_1 + w_2 x_2$
- Apply the logistic function, σ, and check whether
  - $\sigma(y) = \frac{1}{1+e^{-y}} > 0.5$

Geometrically:
Folding a plane along a sigmoid
The decision border is the intersection of this surface and the plane 0.5: a straight line

Example with two features

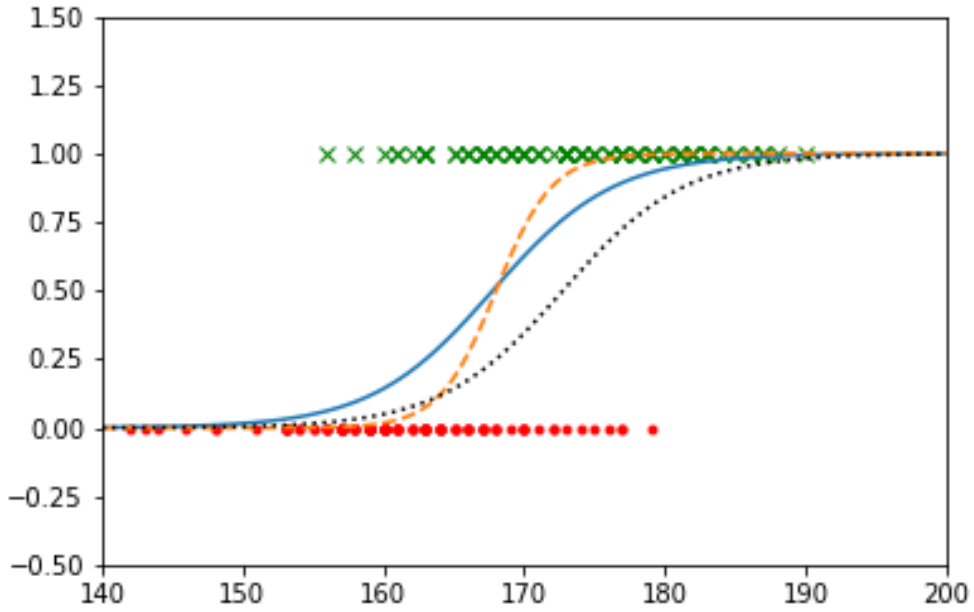


- Example:
  - Heights and weights
  - Acc.: = 0.95
- Blue line = deciscion border
  - Points above it gets a value > 0.5

Today

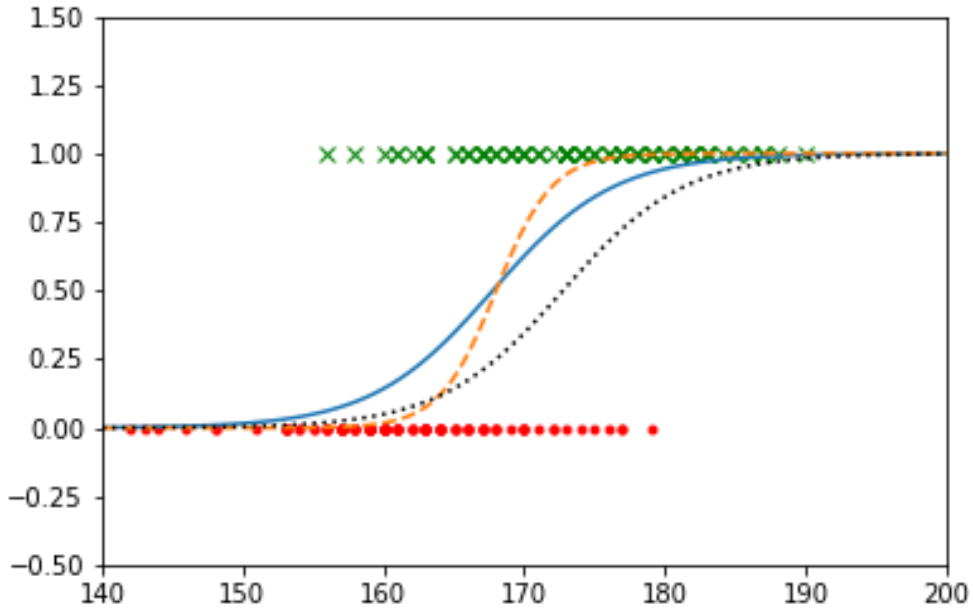
- Logistic regression, short recap
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Comparing Naïve Bayes and Logistic Regression
- Application to tagging

How to find the best curve?



- What are the best choices of a and b in $\frac{1}{1+e^{-(ax+b)}}$?
- Geometrically a and b determine the
  - Midpoint
  - Steepness
- of the curve

Learning in the logistic regression model



- A training instance consists of
  - a feature vector $\vec{x}$
  - a label (class), y, which is 1 or 0.
- With a set of weights, $\vec{w}$, the classifier will assign
  - $\hat{y} = P(C = 1|\vec{x}) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$ to this training instance $\vec{x}$
  - where $P(C = 0|\vec{x}) = 1 - \hat{y}$
- Goal: find $\vec{w}$ that maximize $P(C = y|\vec{x})$ or all training inst.s

Loss function

- In machine learning we have to determine an objective for the training.
- We can do that in terms of a loss function.
- The goal of the training is to minimize the loss function.
- Example: linear regression
  - Loss: Mean Square Error
- We can choose between various loss functions.
- The choice is partly determined by the learner.
- For logistic regression we choose (simplified) cross-entropy loss

Cross-entropy loss

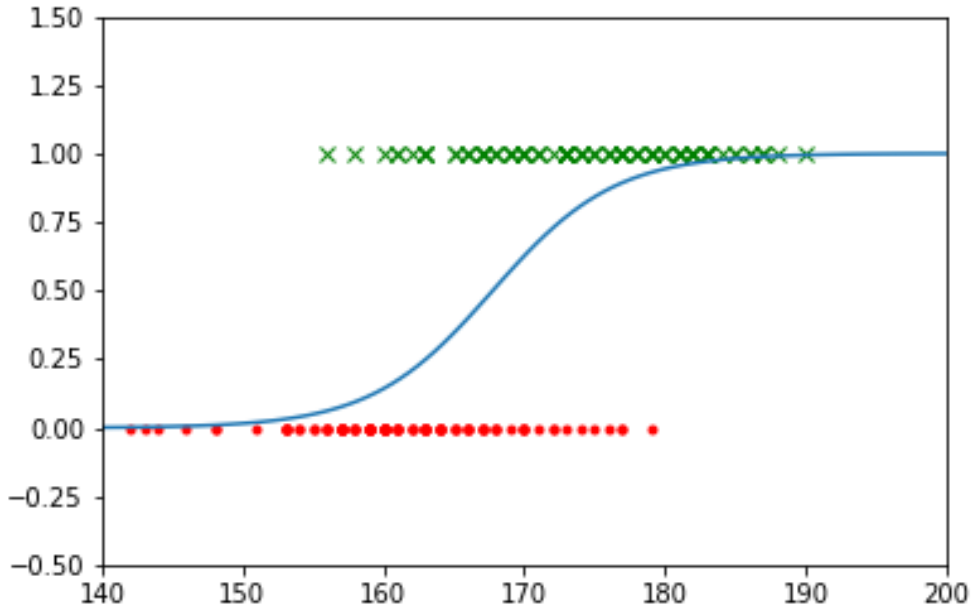
- The underlying idea is that we want to maximize the joint probability of all the predictions we make
  - $\prod_{i=1}^{m} P\left(y^{(i)} \mid \vec{x}^{(i)}\right)$, over all the training data i = 1, 2, …m
- This is the same as maximizing
  - $\log \prod_{i=1}^{m} P\left(y^{(i)} \middle| \vec{x}^{(i)}\right) = \sum_{i=1}^{m} \log P\left(y^{(i)} \middle| \vec{x}^{(i)}\right)$
- This is the same as minimizing
  - $L_{CE}(\vec{w}) = -\log \prod_{i=1}^{m} P(y^{(i)} | \vec{x}^{(i)}) = \sum_{i=1}^{m} -\log P(y^{(i)} | \vec{x}^{(i)})$
  - Which is an instance of what is called the cross-entropy loss

More on cross-entropy loss

- Observe that
  - When $y = 1$, $P(y \mid \vec{x}) = \hat{y} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
  - When $y = 0$, $P(y \mid \vec{x}) = 1 - \hat{y}$
- Since
  - $\hat{y}^y = \hat{y}$ when y=1
  - $\hat{y}^y = 1$ when y=0
  - $(1 - \hat{y})^{(1-y)} = 1$ when y=1
  - $(1 - \hat{y})^{(1-y)} = (1 - \hat{y})$ when y=0
- $P(y \mid \vec{x}) = \hat{y}^y (1 - \hat{y})^{(1-y)}$, always

- The loss function is calculated from the whole training set.
- It is obvious what we want to optimize if we only consider one item.
- The effect of the loss function is how the errors are balanced against each other.

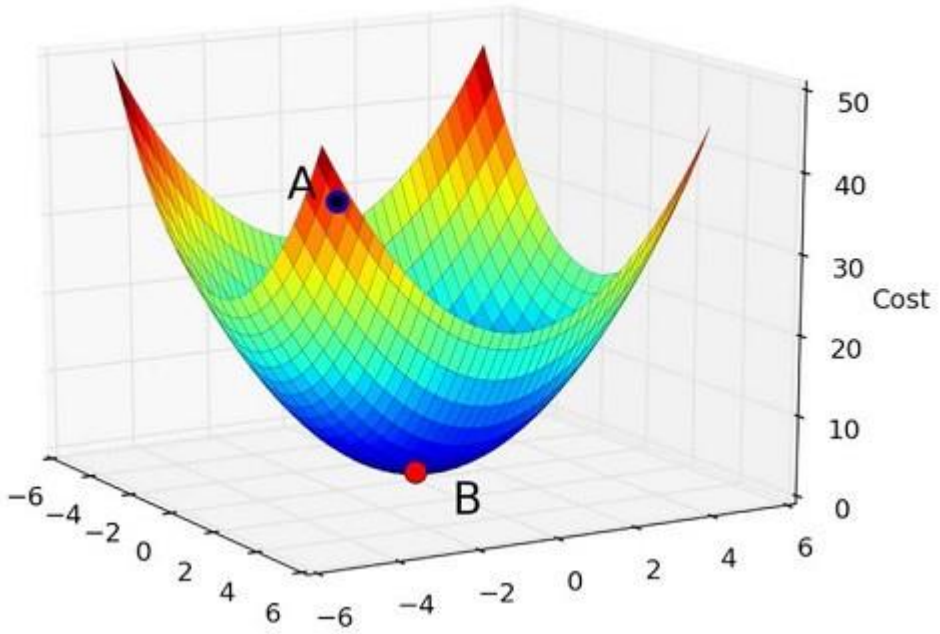
How the mistakes get punished



$\hat{y}$	$-log P(\hat{y} = y\|\vec{x}) = -log\hat{y}$
1/2	- log2
1/4	- 2 log2
1/8	- 3 log2
1/16	- 4 log2

Gradient descent

- We cannot find the best weights by counting in the training data
- Nor can we find them by some formula (like linear regression)
- We have successively to try new weights.
- Good news:
  - The loss function is convex: you are not stuck in local minima
  - We know which way to go
- We skip the details of sec. 5.4



Variations of gradient descent

Batch training:

- Calculate the loss for the whole training set
- Make one move in the correct direction
- Repeat (an epoch)

- Can be slow

Stochastic gradient descent:

- Pick one item
- Calculate the loss for this item
- Move in the direction of the gradient for this item

- Each move does not have to be in the direction of the gradient for the whole set.
- But the overall effect my be good
- Can be faster

Variations of gradient descent

Mini-batch training:

- Pick a subset of the training set of a certain size
- Calculate the loss for this training set
- Make one move in the direction of this gradient
- Repeat (an epoch)

- A good compromise between the two extremes
- (The other two are subcases of this)

Solvers/optimizers

- There are various different solvers and optimizers for gradient descent (which you may meet later).
- Observe that you may specify between solvers in scikit-learn.

Regularization

- LogReg is prone to overfitting to the training data, hence regularization

$$\hat{w} = \arg\max_{w} \sum_{i=1}^{m} \log P(c^i \mid \vec{f}^{\,i}) - \alpha R(w)$$

- The regularization punishes large weights
- Most common is L2-regularization $R(W) = \sum_0^n w_i^2$
- Alternative: L1-regularization $R(W) = \sum_0^n |w_i|$

scikit-learn – LogisticRegression

- `LogisticRegression(penalty='l2', …, C=1.0, …)`
- Uses L2-regularization as default
- By adjusting C you may get better results
- The optimal C varies from task to task
- Whether L1 or L2 may depend on the learner

Today

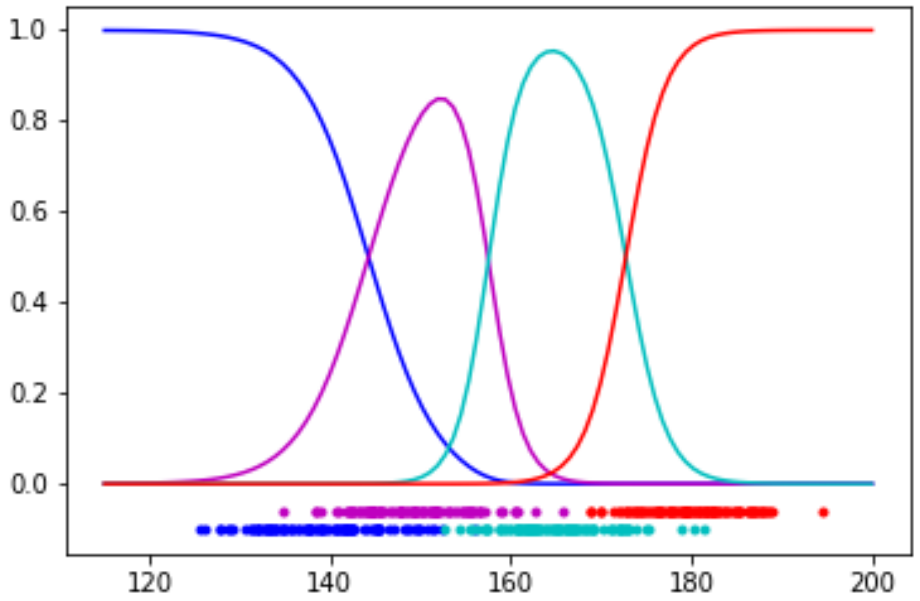
- Logistic regression, short recap
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Comparing Naïve Bayes and Logistic Regression
- Application to tagging

Multinomial Logistic Regression

- Also called maximum entropy (maxent) classifier, or softmax regression
- With one class we considered $P(C|\vec{x}) = \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
- and implicitly $P(nonC|\vec{x}) = 1 - \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{\vec{w}\cdot\vec{x}}}$
- We now consider a linear expression $\vec{w}_i$, for each class $C_i, i = 1, \ldots, k$
- The probability for each class is then given by the softmax function

$$P(C_j|\vec{x}) = \frac{e^{\vec{w_j}\cdot\vec{x}}}{\sum_{i=1}^{k} e^{\vec{w_i}\cdot\vec{x}}}$$

Example: softmax



- 4 different classes corresponding to the dots below the 0-line
- For each of them a corresponding softmax curve
- This expresses the probability of the observation belonging to this class
- For classification of new observation
- Choose the class with the largest probability.

Training Multinomial Logistic Regression

- This is done similarly to the binary task
- We skip the details

Features in Multinomial LR

- Multinomial LR constructs $P(C_j|\vec{x}) = \frac{e^{\overrightarrow{w_j}\cdot\vec{x}}}{\sum_{i=1}^{k} e^{\overrightarrow{w_i}\cdot\vec{x}}}$ for each class.
- This corresponds to one linear expression $\vec{w}_i$, for each $C_i, i = 1, \ldots, k$
- Alternatively, think of this
  - different features for each class:
    - notation $f_j(C, x)$ feature j for the class C and observation x
  - and one set of weights for the features and classes:
- In scikit-learn we write features as before and LogisticRegression constructs the match with labels during training

Today

- Logistic regression, short recap
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Comparing Naïve Bayes and Logistic Regression
- Application to tagging

Logistic regression and Naive Bayes

- Both are probability-based
- In the two-class case they consider whether $P(C|\vec{x}) > P(notC|\vec{x})$
- equivalently whether $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} > 0$

Logistic reg. and Naive Bayes are log-linear

- whether $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} > 0$
- For NB: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} =$

$$\log P(c_1 \mid \vec{f}) - \log P(c_2 \mid \vec{f}) = \log P(c_1) + \sum_{j=1}^{n} \log P(f_j \mid c_1) - \left(\log P(c_2) + \sum_{j=1}^{n} \log P(f_j \mid c2) \right) > 0$$

  - one particular linear expression,
- For LR: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
  - the linear expression that fits the training data best

Comparing NB and LogReg

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

Naive Bayes is an instance of log-linear

- LR: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
- NB: $\log P(c_1 \mid \vec{f}) - \log P(c_2 \mid \vec{f}) = \log P(c_1) + \sum_{j=1}^{n} \log P(f_j \mid c_1) - \left(\log P(c_2) + \sum_{j=1}^{n} \log P(f_j \mid c2) \right) > 0$
- Where:
  - $w_0 = P(c_1) - P(c_2)$
  - $w_i = P(f_j \mid c_1) - P(f_j \mid c_2)$

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
  - (without any smoothing)
- When the independence assumptions of NB 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

Ablation studies

- One way to see which features are important for LogReg
- Start with a classifier which uses many features
- Remove one feature f1, retrain and see whether it has an effect
- Remove another feature f2, instead of f1 or in addition to f1, and study the effect
- Beware of the possibility:
  - Removing f1 only has little effect
  - Removing f2 only has little effect
  - Removing both f1 and f2 might have a large effect
  - Why is this so?

Today

- Logistic regression, short recap
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Comparing Naïve Bayes and Logistic Regression
- Application to tagging

The HMM

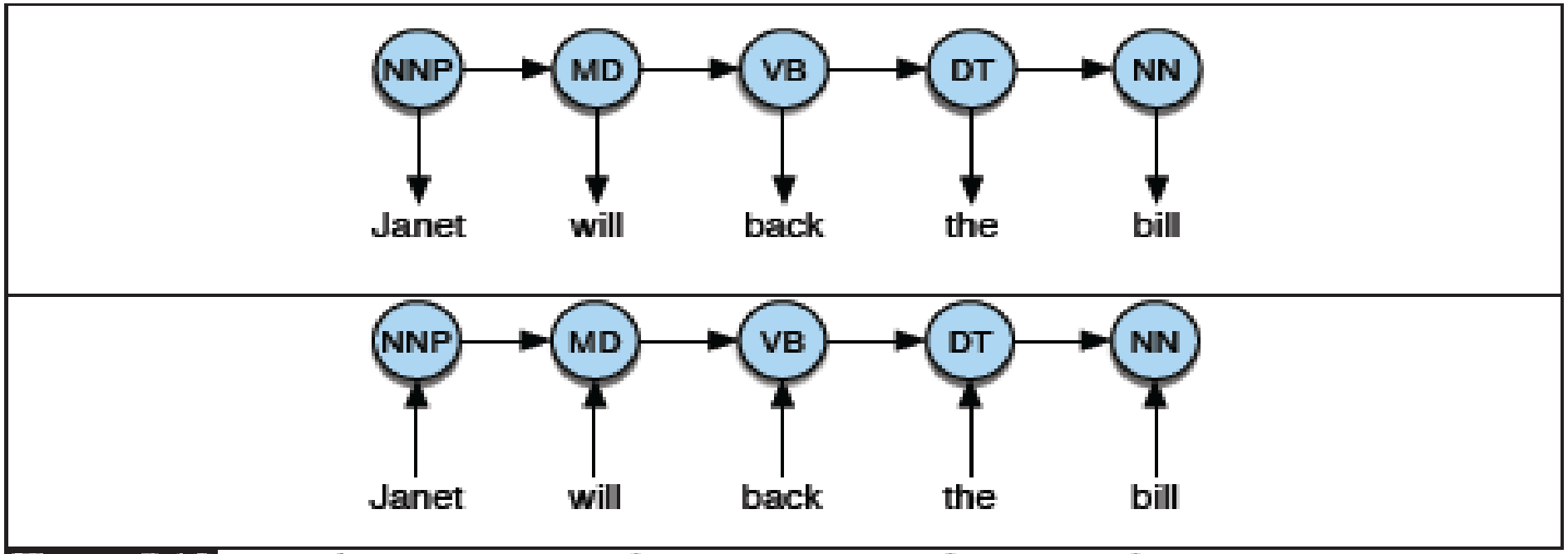


Figure 8.12 A schematic view of the HMM (top) and MEMM (bottom) representation of the probability computation for the correct sequence of tags for the *back* sentence. The HMM computes the likelihood of the observation given the hidden state, while the MEMM computes the posterior of each state, conditioned on the previous state and current observation.

Discriminative tagger

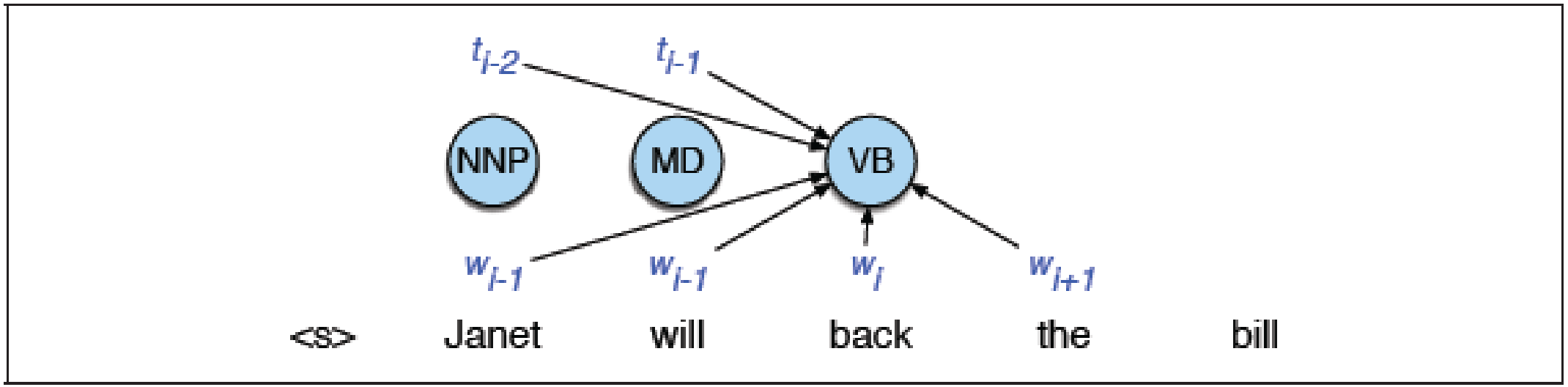


Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

Discriminative tagging 2

- $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P\left(t_i | t_{i-k}^{i-1} w_{i-m}^{i+m}\right)$
- A tag may use as features preceding tags and words in a neighbohood

t_i = VB and w_{i-2} = Janet
t_i = VB and w_{i-1} = will
t_i = VB and w_i = back
t_i = VB and w_{i+1} = the
t_i = VB and w_{i+2} = bill
t_i = VB and t_{i-1} = MD
t_i = VB and t_{i-1} = MD and t_{i-2} = NNP
t_i = VB and w_i = back and w_{i+1} = the

Examples

w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i contains a number
w_i contains an upper-case letter
w_i contains a hyphen
w_i is all upper case
w_i's word shape
w_i's short word shape
w_i is upper case and has a digit and a dash (like *CFC-12*)
w_i is upper case and followed within 3 words by Co., Inc., etc.

How can we use MultiLogReg in a tagger?

- Simplest alternative: Greedy sequence decoding:
  - Choose the best tag for the first word in the sentence
  - Then choose the best tag for the second word in the sentence, given the choice for the first word,
  - And so on, tagging one word at a time until we have finished the sentence.

Shortcomings

- Shortcomings of greedy decoding
  - Early decisions
  - Consider only one tag at a time
- Compare to HMM which considers whole tag sequences and choose the most probable sequence.

Maximum Entropy Markov Models (MEMM)

- Alternative: use the Viterbi algorithm with the LogReg decision:
- If the tagger uses the preceding tag as feature:
  - Use a trellis with a cell for each tag and word in the sentence
  - Fill the cell with best tag sequence leading to this tag and probability of being there
  - Like the bigram HMM-tagger
- If the tagger uses two preceding tags:
  - The trellis must have cells for each word and pairs of tags
  - Like the trigram HMM
- Etc.

However

- The greedy sequence decoding does surprisingly well
- And equally surprising: using preceding tags as features does not improve the tagger that much compared to not including them.
- See mandatory assignment 2B

- Beam search:
  - At each stage in the trellis keep the best hypotheses
    - But reject the hypotheses with a small probability for succeeding later on
- Also possible to produce the *n*-best hypotheses, e.g., the 5 best, from the trellis

More refinements

- J&M considers some finer details that may be a problem for the MEMM-tagger, we will not go into the details
- Conditional Random Fields (CRFs) is a generalization compared to MEMM:
  - Makes it possible to optimize training for whole tag sequences
  - Slow in training
  - Considered the best tool for sequence labelling until a few years ago
- Currently, neural networks ("deep learning") are considered the best tool:
  - Take the IN5550 in the spring