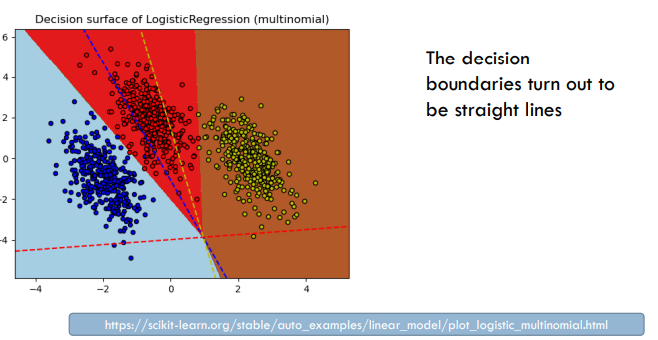
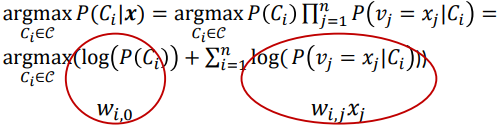
|  |  |  |
| --- | --- | --- |
| Logistic Regression – Decision | Logistic Regression – Learning | |
| Multinomial Logistic Regression |  | |
| MLR: Connections going into a node | MLR: Connections going out of a node | |
| Matrix Form | | |
| Training Multinomial Logistic Regression | |  |



# 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 for two features
  + A surface for each class
  + The intersections of the surfaces project to straight lines in the xy-plane: **decision boundaries**

# Naïve Bayes vs. Logistic Regression

* Both are probability-based and make hard decision by choosing argmaxCi **∈ C** P(*Ci* | *x*)
* For Naïve Bayes:
  + 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 and without any regularization
* Independence assumption:
  + When it assumption holds, NB will do as well as LogReg
  + When it does not hold, NB may put too much weight on some features 🡪 LogReg will not to this: if we add features that depend on other features, LogReg will put less weight on them
* **NB is a generative classifier**; it has a model of how the data are generated
* **LogReg is a discriminative classifier**, it only considers conditional probability

# Generative vs. Discriminative Classifiers

|  |  |
| --- | --- |
| **Generative** | **Discriminative** |
| 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 and weights | * The model is determined by the classes and the differences between them * Consider *different features and weights* for dog when comparing to wolf than to cat |

# Generating positive movie reviews

* Steps
  + 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. ^P(w = the | pos) = 0.1 vs. ^P(w = pitt | pos) = 0.000364
  + 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 suppresses “choose length of document” and assumes it is independent of class

# Discriminative classifiers

* Consider the probability of the class given the observation directly
* E.g. a discriminative classifier may focus on the features
  + *Terrible* and *terrific* for negative vs. positive film review
  + *Director* and *author* for positive film review vs. positive book review
* The discriminative classifier
  + May be more efficient but gives less explanation
  + May eventually focus on the wrong features

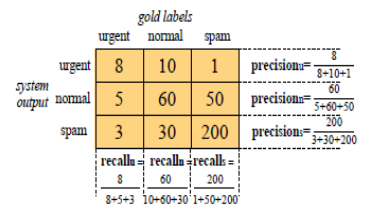
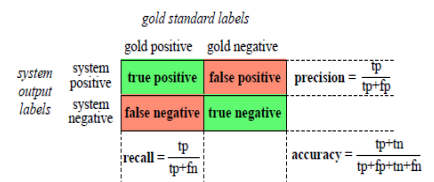
# Evaluation Measure: Accuracy

* What does the 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
* BUT:
  + For some tasks, the classes aren’t equally important (worse to loose an important email than to receive another spam mail)
  + For some tasks the different classes have different sizes

# Information Retrieval (IR)

* Goal: Find all the documents on a particular topic out of 100000 documents
* For these tasks, focus on
  + The relevant documents
  + The documents returned by the system
* Forget the
  + Irrelevant documents which are not returned

# IR Evaluation and Confusion Matrix



# Evaluation Measures

|  |  |
| --- | --- |
| * Accuracy: * Precision: | * Recall: * F-Score: *F1 =*   F-score combines P and R,  F1 is called “harmonic mean”  For some 0 < α < 1 |
| * Microaverage   Add up all respective tp, fp, fn from the different classes  Calculate P, R, F-score with these summed-up numbers | * Macroaverage   Take P, R and F-score of all classes, add them up and divide them by the number of classes (simply the average of those values) |

# 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)
* Goal: Ascribe probabilities to word sequences P(w1, w2, w3, …, wn)
* Related: the probability of the next word P(wn | w1, w2, w3, …, wn-1)
* A model which does either is called **Language Model (LM)**

# Chain Rule

* The two definitions above are related by the chain rule for probability
* P(w1, w2, w3, …, wn) = P(w1) x P(w2 |w1) x P(w3 | w2) x … x P(wn | w1, w2, …, wn-1) = 
* P(“its water is so transparent”) = P(its) x P(water | its) x P(is | its water) x P(so | its water is) x P(transparent | its water is so)
* BUT: This **does not work for long sequences**

# Markov Assumption

* A word depends only on the immediately preceding word
* P (w1, w2, w3, …, wn) = P(w1) x P(w2 | w1) x P(w3 | w2) x … x P(wn | wn-1) =
* P (“its water is so transparent”) = P(its) x P(water | its) x P(is | water) x P(so | is) x P(transparent | so)
* **Bigram Model**

# Estimating Bigram Probabilities

* The probabilities can be estimated by counting
* This yields maximum likelihood probabilities (= maximum probable on the training data)



# General n-gram Models

* A word depends only on the k-many immediately preceding words
* P(w1, w2, w3, …, wn) = 
* **Unigram** **model** – no preceding words
* **Bigram model** – one preceding word
* **Trigram model** – two preceding words
* **K-gram model** – k-1 preceding words
* Can be **trained similarly to a bigram model**
* Have to **be 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 ^P (wi | wi-1)
* k-gram
  + select word wi according to how probable it is given the k-1 preceding words

# Unknown Words

* There might be words that are 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

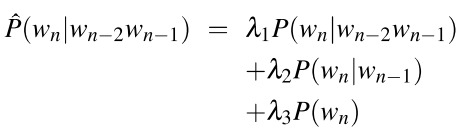
where |*V*| is the size of the vocabulary *V*

# Smoothing n-grams

|  |  |
| --- | --- |
| **Backoff** | **Interpolation** |
| * If you have good evidence, use the trigram model * If not, use the bigram model or even the unigram model | * Combine the models |

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

# Interpolation

* Simple interpolation:
* The λs can be tuned on a held-out corpus
* A more elaborate model will condition the λs on the context (brings in elements of backoff)

# Evaluation of n-gram Models

* Extrinsic evaluation:
  + To compare to LMs, see how well they are doing in an application, e.g. translation or speech recognition
* Intrinsic evaluation:
  + Use a held-out corpus and measure P(w1, w2, w3, …, wn) 1/n
  + The n-root compensates for different lengths
  + For a k-gram model:
  + It is normal to use the inverse of this, called the perplexity: 

# 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 contexts are restricted to a limited number of preceding words
* Advice: Use logarithms when working with n-grams