$\binom{n}{k} = \frac{n!}{n!(n-k)!}$ number of ways to select k words out of n given words (“unordered samples without replacement”)

Estimating Naïve Bayes

- In relative frequency estimation, the parameters are set to empirical frequencies:

$$\hat{\phi}_{y,j} = \frac{\text{count}(y,j)}{\sum_{j'=1}^{V} \text{count}(y,j')} = \frac{\sum_{i:y^{(i)}=y} x_j^{(i)}}{\sum_{j'=1}^{V} \sum_{i:y^{(i)}=y} x_{j'}^{(i)}}$$

$$\hat{\mu}_y = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')}.$$

- This turns out to be identical to the maximum likelihood estimate:

$$\hat{\phi}, \hat{\mu} = \underset{\phi,\mu}{\text{argmax}} \prod_{i=1}^{N} \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}) = \underset{\phi,\mu}{\text{argmax}} \sum_{i=1}^{N} \log \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)})$$

Perception classification is **discriminative**
Naïve Bayes is **probabilistic**
Logistic Regression is both **discriminative and probabilistic** (it

Derivative of sigmoid(x) = sig(x)(1 – sig(x))
Derivative of tanh(x) = 1 – tanh(x)^2

A Probability Model for Text Classification

- First, assume each instance is independent of the others
  - $p(\boldsymbol{x}^{(1:N)}, y^{(1:N)}) = \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)}, y^{(i)})$
- Apply the chain rule of the probability
  - $p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y) \cdot p(y)$
- Define the parametric form of each probability
  - $p(y) = \text{Categorical}(\mu)$ $\quad p(\boldsymbol{x}|y) = \text{Multinomail}(\phi)$
  - The multinomial is a distribution over vectors of counts
  - The parameters μ and ϕ are vectors of probabilities

We estimate the parameters of Naïve Bayes through **maximum likelihood** (maximizing likelihood of dataset)

Learning Logistic Regression

- Maximization of the conditional log-likelihood

$$\log \mathrm{p}(\boldsymbol{y}^{(1:N)} \mid \boldsymbol{x}^{(1:N)}; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$
$$= \sum_{i=1}^{N} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')\right)$$

- Minimization of the negative log-likelihood/logistic loss

$$\ell_{\text{LOGREG}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y'))$$

Stochastic Gradient Descent (Online Optimization)

- Computing the gradient over all instances is expensive
- Stochastic gradient descent approximates the gradient by its value on a single data:

$$\sum_{i=1}^{N} \ell(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) \approx N \times \ell(\boldsymbol{\theta}; \boldsymbol{x}^{(j)}, y^{(j)})$$

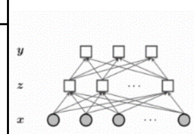
theoretically guaranteed!

- $(\boldsymbol{x}^{(i)}, y^{(i)})$ is sampled at random from the training set $(\boldsymbol{x}^{(j)}, y^{(j)}) \sim \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$

Summary of Linear Classification

	Pros	Cons
Naive Bayes	Simple, probabilistic, fast Closed-form solution	Not very accurate
Perceptron	Simple, accurate	Not probabilistic, may overfit
Logistic Regression	Error-driven learning, regularized	More difficult to implement

A Simple Feedforward Architecture



Next predict y from $\boldsymbol{z}$, again via logistic regression:

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z\to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z\to y)} \cdot \boldsymbol{z} + b_{j'})}$$

where each b_j is an offset. This is denoted:

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \text{SoftMax}(\Theta^{(z\to y)}\boldsymbol{z} + \boldsymbol{b})$$

Softmax + negative LL loss = cross-entropy loss in deep learning

Evaluation metrics:

1) Accuracy (problem = doesn’t take into account precision/recall for unbalanced datasets)
2) Recall = TP / (TP + FN) (fraction of positive instances which were correctly classified)
3) Precision = TP / (TP + FP) (fraction of positive predictions that were correct)
4) F1 score = 2*recall*precision / (recall + precision) (tradeoff between just recall or just precision)
5) Recall/precision imply binary classif.. In mcc, instances are positive for one class and negative for other classes
6) Two ways to combine performance across classes:
   a. Macro F-measure: compute F-score per class, and average across all classes (treats all classes equally)
   b. Micro F-measure: compute the total number of TP, FP, and FN across all classes, and compute a single F-score. This emphasizes performance on high-frequency classes

Markov processes: have sequence of N r.v.s, want a sequence probability model. There are |V|^n possible sequences

First order markov process: P(x1, x2, …, xn) = P(x1)P(xi | xi-1)

Second order markov process: P(x1, x2, …, xn) = P(x1)P(x2 | x1)P(xi | xi-1, xi-2)

How to evaluate a language model:

1) Extrinsic: build a new language model, use it for some task (speech recognition, machine translation)

Time-consuming and **Bad approximation** unless test data looks like training data (so, only useful in pilot experiments)

2) Intrinsic: measure how good we are at modeling language
   a. Perplexity (Cannot compare perplexities of language models trained on different corpora)

- Perplexity is the normalized inverse probability of S

$$p(S) = \prod_{i=1}^{sent} p(s_i) \qquad \log_2 p(S) = \sum_{i=1}^{sent} \log_2 p(s_i)$$

$$l = \frac{1}{M} \sum_{i=1}^{sent} \log_2 p(s_i) \qquad \text{perplexity} = 2^{-l}$$

$\text{perplexity} = 2^{-l}, l = \frac{1}{M}\sum_{i=1}^{sent} \log_2 p(s_i)$

- Sent is the number of sentences in the test data
- M is the number of words in the test corpus
- A better language model has higher p(S) and lower perplexity

Problem with add one smoothing: with a large vocabulary, it thinks we are extremely likely to see novel events, rather than words we’ve actually seen (a large dictionary makes novel events too probable)

Interpolation – mixture of unigram, bigram, trigram, etc. models

Backoff – use trigram if good evidence, otherwise bigram, otherwise unigram

Absolute Discounting Interpolation

- Instead of multiplying the higher-order by lambdas
- Save ourselves some time and just subtract some d!

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \underbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}_{\text{discounted bigram}} + \underbrace{\lambda(w_{i-1})}_{\text{Interpolation weight}} \underbrace{P(w)}_{\text{unigram}}$$

- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing

- The unigram is useful exactly when we haven’t seen this bigram
- Instead of $p(w)$: how likely is w
- $p_{continuation}(w)$: how likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing (for bigrams)

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we’ve discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left|\{w : c(w_{i-1}, w) > 0\}\right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

TFIDF (term-frequency inverse document frequency):

$tf_{t,d} = \log_{10}(\text{count}(t,d) + 1)$

$idf_i = \log_{10}\left(\frac{N}{df_i}\right)$ — N: Total # of docs in collection; df_i: # of docs that have word i

- Words like "the" or "good" have very low idf

$$w_{t,d} = tf_{t,d} \times idf_i$$

$$\text{PMI}(w,c) = \log_2 \frac{p(w,c)}{p(w)p(c)}$$

PMI (Pointwise Mutual Information):

Do word w and c co-occur more than if they were independent?

$$\text{PPMI}_\alpha(w,c) = max(\log_2 \frac{p(w,c)}{p(w)p_\alpha(c)}, 0) \qquad P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

PPMI (Positive Pointwise Mutual Information):

- **PMI is biased toward infrequent events**. Very rare words have very high PMI values. We want to give rare words slightly higher probabilities. **PPMI vectors are long and sparse.** Want to learn vectors that are short and dense (because easier to use, generalize better than storing counts, and in practice work better)

How to get short, dense vectors?

1) SVD (a special case of this is called LSA – Latent Semantic Analysis)
2) Brown clustering
3) Neural language modeling (skip-grams and CBOW – Continuous bag-of-words

- How do we measure the quality of a partition C?

$$\text{Quality}(C) = \sum_{i=1}^{n} \log e(w_i|C(w_i))q(C(w_i)|C(w_{i-1})) = \sum_{c=1}^{k}\sum_{c'=1}^{k} p(c,c') \log \frac{p(c,c')}{p(c)p(c')} + G$$

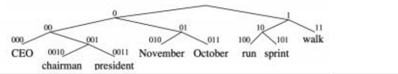
a constant

- Where $p(c,c') = \frac{n(c,c')}{\sum_{c,c'} n(c,c')}$ $\quad p(c) = \frac{n(c)}{\sum_c n(c)}$

Here, n(c) is the number of times class c occurs in the corpus, n(c, c') is the number of times c' is seen following c, under the function C

Brown Clusters as Vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman = 0010, "months" = 01, and verbs = 1



Brown Clustering

- $\mathcal{V}$ is a vocabulary
- $C: \mathcal{V} \to \{1, 2, \dots k\}$ is a partition of the vocabulary into k clusters
- $q(C(w_i)|C(w_{i-1}))$ is a probability of cluster w_i of to follow the cluster of w_{i-1}
- $e(w_i|C(w_i)) = \frac{count(w_i)}{\sum_{x \in C(w_i)} count(x)}$

$$p(w_1, w_2, \dots, w_T) = \prod_{i=1}^{n} e(w_i|C(w_i))q(C(w_i)|C(w_{i-1}))$$

Brown Clustering: A First Algorithm

- Start with |V| clusters: each word gets its own cluster
- The goal is to get k clusters
- We run |V|-k merge steps:
  - Pick 2 clusters and merge them
  - Each step picks the merge maximizing Quality(C)
- Cost ?
  - $O(|V| - k) \times O(|V|^2) \times O(|V|^2) = O(|V|^5)$

    # iters   # pairs   compute Quality(C)

A Second Algorithm

- m: a hyper-parameter, sort words by frequency
- Take the top m most frequent words, put each of them in its own cluster $c_1, c_2, c_3, \dots c_m$
- For $i = (m+1) \dots |V|$
  - Create a new cluster c_{m+1} *(we have $m+1$ clusters)*
  - Choose two clusters from $m+1$ clusters based on quality(C) and merge *(back to m clusters)*
- Carry out $m-1$ final merges *(full hierarchy)*
- Running time $O(|V|m^2 + n)$, n=#words in corpus

Skip-Gram

- Maximize the log likelihood of context word $w_{t-m}, w_{t-m+1}, \dots, w_{t-1}, w_{t+1}, w_{t+2}, \dots, w_{t+m}$ given word w_t

$$J(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} p(w_{t+j}|w_t;\theta)$$

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log p(w_{t+j}|w_t)$$

- m is usually 5~10

Skip-Gram

- How to model log $P(w_{t+j}|w_t)$?

$$p(w_{t+j}|w_t) = \frac{\exp(u_{w_{t+j}} \cdot v_{w_t})}{\sum_{w'} \exp(u_{w'} \cdot v_{w_t})}$$

- Softmax function Again!
- Every word has 2 vectors
  - v_w : when w is the center word
  - u_w: when w is the outside word (context word)

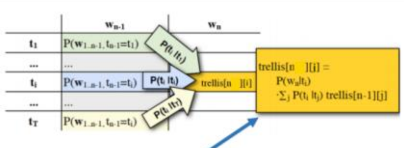
Hidden Markov Model (Formal)

- States T = $t_1, t_2 \dots t_N$;
- Observations O= $o_1, o_2 \dots o_M$;
  - Each observation is a symbol from a vocabulary V = $\{v_1, v_2, \dots v_V\}$
- Transition probabilities
  - Transition probability matrix A = $\{a_{ij}\}$
- Observation likelihoods
  - Output probability matrix B = $\{b_i(o_t)\}$
- Initial probability vector π
  - $\pi_i = P(t_1 = i) \;\; 1 \le i \le N$

Hidden Markov Model

- Two independent assumptions
  - Approximate p(**t**) by a bi(or N)-gram model
    - Markov assumption: $P(q_i|q_1 \dots q_{i-1}) = P(q_i|q_{i-1})$
  - Assume each word depends only on its POS tag
    - Output independence: $P(o_i|q_1 \dots q_i, o_1 \dots o_i \dots o_T) = P(o_i|q_i)$

Implementation using an array

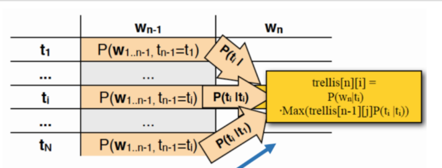


Induction:
$$\alpha_k(q) = P(w_k \mid t_k = q) \sum_{q'} \alpha_{k-1} P(t_k = q \mid t_{k-1} = q')$$

Three Basic Problems for HMMs

- **Likelihood** of the input: How likely the sentence "I love cat" occurs
  - Forward algorithm
- **Decoding** (tagging) the input: POS tags of "I love cat" occurs
  - Viterbi algorithm
- **Estimation** (learning): How to learn the model?
  - Find the best model parameters
    - **Case 1: supervised – tags are annotated (MLE)**
    - Case 2: unsupervised -- only unannotated text (Forward-backward algorithm)

Implementation using an array



Induction:
$$\delta_k(q) = P(w_k \mid t_k = q) \max_{q'} \delta_{k-1}(q') P(t_k = q \mid t_{k-1} = q')$$

← Forward Algorithm ; Viterbi Algorithm →

Context-free grammar - A CFG gives a formal way to define what meaningful constituents are and exactly how a constituent is formed out of other constituents (or words). It defines valid structure in a language.

Context-free Grammar

N	Finite set of non-terminal symbols	NP, VP, S
Σ	Finite alphabet of terminal symbols	the, dog, a
R	Set of production rules, each A → β, β ∈ (Σ, N)	S → NP VP, Noun → dog
S	Start symbol	

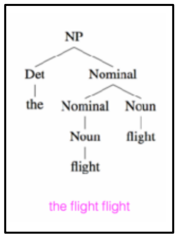
Context-free Grammar

- A context-free grammar defines how symbols in a language combine to form valid structures

NP	→	Det Nominal	non-terminals
NP	→	ProperNoun	
Nominal	→	Noun | Nominal Noun	
Det	→	a | the	lexicon/ terminals
Noun	→	flight	

Summary: Syntax Through The Use Of Context-free Grammar

- Groups of consecutive words act as a group or a **constituent**
- A **context-free grammar** consists of a set of **rules** or **productions**, expressed over a set of **non-terminal** symbols and a set of **terminal** symbols.
- A particular **context-free language** is the set of strings that can be derived from a particular **context-free grammar**
- Verbs can be **subcategorized** by the types of **complements** they expect. Simple subcategories are **transitive** and **intransitive**
- **Treebanks** of parsed sentences exist for many genres of English and for many languages.



PCFG

- Probabilistic context-free grammar: each production is also associated with a probability
- This lets us calculate the probability of a parse for a given sentence; for a given parse tree T for sentence S comprised of n rules from R (each $A \to \beta$):

$$P(T,S) = \prod_{i=1}^{n} P(\beta|A)$$

PCFG

N	Finite set of non-terminal symbols	NP, VP, S
Σ	Finite alphabet of terminal symbols	the, dog, a
R	Set of production rules, each A → β [p], p = P(β | A)	S → NP VP, Noun → dog
S	Start symbol	

CKY

- In CNF, each non-terminal generates two non-terminals

S → NP VP

[s [NP I] [VP shot an elephant in my pajamas]]

- The recursive step of our dynamic program needs to select a split point and rule
- If the parent spans tokens i-j, then there is a k where the children span i-k and k-j.

Estimating PCFGs

How do we calculate $P(A \to \beta)$?

Maximum likelihood estimates

CKY Algorithm

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
     for all {A | A → words[j] ∈ grammar}
           table[j−1, j] ← table[j−1, j] ∪ A
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
           for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
                 table[i,j] ← table[i,j] ∪ A
```

Figure 13.5 The CKY algorithm.

CKY PCFG

- We calculate the max probability parse using CKY by storing the probability of each phrase within each cell as we built it up.
- In particular, we fill the cell for span i-j and label A with the maximum over splits and rules of

$$table(i,j,A) = P(A \to BC) \times table(i,k,B) \times table(k,j,C)$$

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                   and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
        table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
           for all { A | A → BC ∈ grammar,
                    and table[i,k,B] > 0 and table[k,j,C] > 0 }
              if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                 table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                 back[i,j,A] ← {k,B,C}
  return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Estimating PCFGs

$$\sum_\beta P(\beta \mid A) = \frac{C(A \to \beta)}{\sum_\gamma C(A \to \gamma)}$$

(equivalently)

$$\sum_\beta P(\beta \mid A) = \frac{C(A \to \beta)}{C(A)}$$