1.1 N-Grams

$$P(w_1 w_2 w_3 \cdots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \cdots P(w_n|w_{1:n-1})$$

$$= \prod_{k=1}^{n} P(w_1|w_{1:k-1})$$

$\Downarrow$ "Markov Assumption"

$$\approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \quad \text{(e.g. for bigram)}$$

MLE (Maximum Likelihood Estimation)

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad \cdots \text{(bigram)}$$

general

$$\approx\!\!> \quad \underbrace{P(w_n|w_{n-N+1.n-1}) = \frac{C(w_{n-1})}{C(w_{n-N+1:n-1})}}_{\text{"relative frequency"}} \quad \cdots \text{N-Gram}$$

Smoothing

↳ to avoid assigning zero probabilities to unseen events

1) Laplace Smoothing

: add 1 to every count $\Rightarrow$ $P(w_n|w_{n-N+1:n-1}) = \dfrac{C(w_{n-N+1:n})+1}{\sum_{k=1}^{n}(C(w_{n-N+1:n-1})+1)}$

$$= \underbrace{\frac{C(w_{n-N+1:n})+1}{C(w_{n-N+1:n-1})+V}}$$

where V is, the number of tokens

2) Add-k Smoothing

: add k to every count (Similar to Laplace Smoothing)

$\Rightarrow$ Sharp change in probabilities may occur because too much probability mass may move to zeros

3) Backoff

1.2 Naive Bayes Classifiers

❀ Generative vs Discriminative Classifiers

- Generative : Build a model of how a class could generate some input data (e.g. Naive Bayes)
- Discriminative : Learn useful features from the input to discriminate between classes (e.g. Logistic Regression)

The class of a document (d) : $\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$ (Bayes' rule)

$= \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$

W.L.O.G for a document of features $f_1, f_2, \cdots f_n$, $\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(f_1 f_2 \cdots f_n | c) P(c)$

❀ Assumptions

- bag-of-words assumption : position doesn't matter
- naive Bayes assumption : feature independence $P(f_1 f_2 \cdots f_n | c) = P(f_1|c) \cdots P(f_n|c)$

$$\Rightarrow \quad c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

Linear Function of input features

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

(from training data)

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \qquad \hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)}$$

❀ How to improve Naive Bayes Classifiers?

1) Delete Duplicates ~ binary Naive Bayes
   : whether a word occurs or not may be more important than its frequency
2) Negation
   e.g. didn't like this movie, but I
   ⇓
   didn't NOT_like NOT_this NOT_movie, but I
3) Use existing sentiment lexicons

1.3 Logistic Regression

✤ Generative vs Discriminative model

⇒ generative는 $\hat{c} = \text{argmax}\, P(d|c)P(c)$로, $P(d|c)$를 구하려 하지만,
discriminative는 $P(c|d)$를 바로 구하려 한다.

✤ When to use Logistic Regression : Correlation

⇒ Naive Bayes는 independence assumption 때문에 correlation이 강한 정보에 적합 ×
⇝ larger dataset에는 logistic regression이 더 나음.

✤ Sigmoid Classifier

$z = \left(\sum_{i=1}^{n} wx\right) + b$ 를 0~1 range로 mapping

$$\Rightarrow \sigma(z) = \frac{1}{1+\exp(-z)} \quad \approx \quad \begin{cases} p(y=1) = \sigma(w \cdot x + b) \\ p(y=0) = 1 - \sigma(w \cdot x + b) \end{cases}$$

✤ Softmax Regression (Multinomial Logistic Regression)

⇒ multiclass 일 때는 output y가 one-hot vector

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \quad \approx \quad p(y_k = 1|x) = \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)} \quad (K \text{ is the number of classes})$$

$$\Rightarrow \hat{y} = \text{softmax}(Wx + b)$$

✤ Learning in Logistic Regression

1) Loss Function (Cost Function)
: a metric for how close the current label ($\hat{y}$) is to the true gold label (y). (e.g. cross-entropy)

2) Gradient Descent
: optimization algorithm for updating the weights

1) Cross-Entropy Loss (binary classifier)

$L(\hat{y}, y)$ = How much $\hat{y}$ differs from y, where $\hat{y} = \sigma(w \cdot x + b)$

$$\approx p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y} \quad (\because \text{Bernoulli Distribution})$$

따라서, $\log p(y|x) = y \log \hat{y} + (1-y)\log(1-\hat{y})$ should be maximized

$$\Leftrightarrow L_{CE} = -[y \log \hat{y} + (1-y)\log(1-\hat{y})] \text{ should be } \textbf{Minimized}$$

$$\Rightarrow L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

2) Gradient Descent

↳ Goal : Minimize the loss (averaged over all examples)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)};\theta), y^{(i)})$$

* (Logistic Regression에서는 loss function이 convex라서 무조건 minimum이 보장되는 방면, multi-layer neural network에서는 local minima에 stuck될 수 있다.)

$$\Rightarrow W^{t+1} = W^{t} - \eta \frac{d}{dw} L(f(x;w), y)$$

(η는 learning rate)

$$\Downarrow$$

$$\nabla L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1}(f(x;\theta), y) \\ \frac{\partial}{\partial w_2}(f(x;\theta), y) \\ \vdots \\ \frac{\partial}{\partial b}(f(x;\theta), y) \end{bmatrix}$$

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = (\hat{y} - y)\, x_j$$

derivation

$$\frac{\partial L_{CE}}{\partial w_j} = \frac{\partial}{\partial w_j}\left(-y \log \sigma(wx+b) + (1-y)\log(1-\sigma(wx+b))\right)$$

$$= -\frac{y}{\sigma(wx+b)} \cdot \frac{\partial}{\partial w_j}\sigma(wx+b) + \frac{1-y}{1-\sigma(wx+b)} \cdot \frac{\partial}{\partial w_j}\sigma(wx+b)$$

$$= \frac{\partial}{\partial w_j}\sigma(wx+b)\left[\frac{\sigma(wx+b) - y}{\sigma(wx+b)(1-\sigma(wx+b))}\right]$$

$$= \cancel{\sigma(wx+b)}\,(1-\cancel{\sigma(wx+b)}) \cdot \underbrace{\frac{\partial}{\partial w_j}(wx+b)}_{= x_j} \cdot \left[\frac{\sigma(wx+b) - y}{\cancel{\sigma(wx+b)}\,(1-\cancel{\sigma(wx+b)})}\right]$$

$$= (\sigma(wx+b) - y)\, x_j$$

$$= (\hat{y} - y)\, x_j$$

* Mini-batch Training

- Stochastic Gradient Descent : 1 example at a time

$$\Updownarrow$$

- Mini-batch Training : train on a group of m examples
  ~ Cost function is the average loss over the batch

$$Cost(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

$$\Rightarrow \frac{\partial Cost(\hat{y}, y)}{\partial w} = \frac{1}{m}(\hat{y} - y)^T X = \frac{1}{m}(\sigma(wX+b) - y)^T X$$

* Regularization

→ to avoid overfitting, add a regularization term, $R(\theta)$

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

→ to penalize large weights

L2 Regularization	L1 Regularization
$R(\theta) = \lVert \theta \rVert_2^2 = \sum_{i=1}^{n} \theta_i^2$ (Euclidian distance)	$R(\theta) = \lVert \theta \rVert_1 = \sum_{i=1}^{n} \lvert \theta_i \rvert$ (Manhattan Distance)
Ridge Regression	Lasso Regression
• prefer weight vectors with many small weights	• prefer sparse solutions with larger weights, but many more zero weights
• Corresponds to assuming that weights are distributed in a Gaussian distribution with $\mu = 0$	• viewed as a Laplace prior on the weights

3) Multinomial Logistic Regression

$\rightsquigarrow$ output : vector $\hat{y}$ with K elements, where each element $\hat{y}_k = p(y_k = 1 | x)$

$$\Rightarrow L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

$$= -\log \hat{y}_c \quad \text{(where c is the correct class)}$$

$$= -\log \hat{p}(y_c = 1 | x)$$

$$= -\log \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)} \quad \text{(where K is the number of classes)}$$

$$\frac{\partial L_{CE}}{\partial w_{k,i}} = (\hat{y}_k - y_k)\, x_i = \left(\frac{(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)} - y_k \right) x_i$$

2.1. Vector Semantics (Idea)

Vector Semantics : learning representations of the meaning of words (**embeddings**)

⤳ Finding self-supervised ways to learn word representations is an important focus in NLP.

- the idea is to represent a word as a point in the multidimensional semantic space

⇒ by representing words as embeddings, classifiers can assign classes as long as it has "similar meanings"

↕

(naive Bayes / Logistic Regression 에서는 특정 단어가 나온 횟수에 의존한 반면, embedding 사용시 직접 나타나지 않아도, 비슷한 의미의 단어들도 고려)

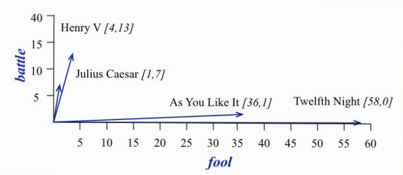
2.2. Cosine Similarity

✿ Document Vectors

<Term-Document Matrix>

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

⇓



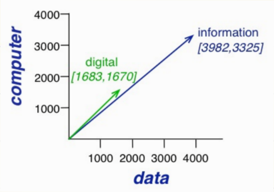
✿ Word Vectors

<Word-Word Matrix>

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

⤳ nearby words

⇓



$$\Rightarrow \text{cosine}(v,w) = \frac{v \cdot w}{|v||w|}$$

: 두 vector 사이 각도로 similarity 판별

2.3 TF-IDF Weighting

* Problem
  : word-word matrix에서 nearby에 많을수록 important,
  반면에 too frequent words may be unimportant

1) Term Frequency : frequency of a word(t) in the document(d)

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

"High term frequency → more important"

2) Inverse Document Frequency : number of documents the word occurs in

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$ (N is the number of documents)

"Low document frequency → more important"

$$\Rightarrow w_{t,d} = tf_{t,d} \times idf_t$$

2.4. PPMI Weighting

(PPMI : Positive Pairwise Mutual Information)

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

$P(w,c)$ - - - - → co-occur in the corpus

$P(w)P(c)$ - - - → prior-expected to appear by chance

↳ 각각 단어의 출현 비율로 예상한 것보다 실제로 두 단어가 함께 나타나는 횟수가 많으면 두 단어는 관련이 있다고 생각할 수 있다

⇒ useful to find words that are strongly associated

$$PPMI(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)$$

* Problem : Bias toward infrequent events (rare words → high PMI)

↓

Solution
- Power Count : $P(c)$ 대신 $P_\alpha(c)$ 사용 $\rightsquigarrow P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$

  $$\rightsquigarrow PPMI_\alpha(w,c) = \max\left(\log \frac{P(w,c)}{P(w)P_\alpha(c)}, 0\right)$$

  α는 주로 0.75
- Laplace Smoothing

⇒ Represent a document with a Centroid document vector

$$d = \frac{w_1 + w_2 + \cdots + w_k}{k}$$ (given k word vectors w_i $1 \le i \le k$)

2.5. Word2vec

(Word vector는 sparse 하니깐, short dense vector (= embedding)로 변환하자
⇒ avoid overfitting + better capture synonymy ~ dimensions 50~1000)

* Static Embedding : the method learns 1 fixed embedding for each vocabulary

* Intuition
"instead of counting how often each word occurs near, train a classifier on a binary prediction task"
: appear near, or not?
~ take the weights as the word embeddings
⇒ **Self-Supervision** : We can just use running text as training data (x labeling)

< SGNS : Skip Gram Negative Sampling >

↳ method for computing embeddings

1) Intuition

1) If a word (c) is a context word to ω, treat as positive (+)
2) Randomly sample other words in the lexicon to create negative (−) examples
3) Use logistic regression to train a classifier to distinguish (+/−)
4) Use the learned weights as embeddings

Similarity $(c, \omega) \approx c \cdot w$ (두단어의 word vector의 dot product 높을수록 high similarity)

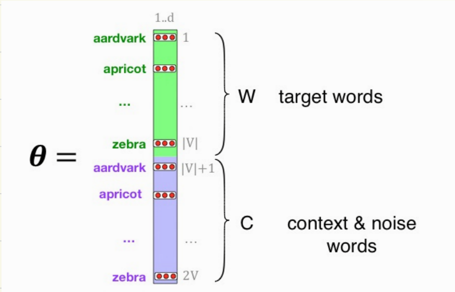
$\rightarrow \; P(+|\omega, c) = \sigma(c \cdot w) \; / \; P(-|\omega, c) = \sigma(-c \cdot w)$

2) Windowing

→ go over windows (multiple words)

$$\Rightarrow P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$ (← assume that all context words are independent)

$$\leftrightarrow \sum_{i=1}^{L} \log \sigma(c_i, w)$$ how similar the context window is to the word



$|V| \times |V|$ 에서 $2|V| \times d$로 줄음.

2) Learning Skip-Gram Embeddings

... lemon, a (tablespoon of apricot jam, a) pinch ...
c1 c2 w c3 c4

randomly sampled from the lexicon based on $P_\alpha(\omega)$
weighted unigram frequency

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if