Natural Language Processing Specialization Formula Sheet

Fady Morris Ebeid
(2020)

Chapter 1 Classification and Vector Spaces

1 Logistic Regression

corpus: a language resource consisting of a large and structured set of texts.

1.1 Notation

V: Vocabulary size, the number of unique words in the entire set of sentences.
$\boldsymbol{\theta}$: Parameter vector, $\boldsymbol{\theta} = [\theta_0, \theta_1, \dots, \theta_n]$
m: Number of examples (sentences)
$P(\text{class})$: Probability that a sentence is in a given class.
$\text{class} \in \{\text{pos}, \text{neg}\}$.
$\text{freq}(w_i, \text{class})$: Frequency of a word w_i in a specific class.

1.2 Preprocessing

1. Eliminate handles and URLs.
2. Tokenize the string $\mathbf{w} = [w_1, w_2, \dots, w_n]$.
3. Remove stop words(and, is, are, at, has, for, a, . . .) and punctuation (, . : ! " ').
4. Stemming: Convert every word to its stem.(use Porter Stemmer [Por80]).
5. Convert words to lowercase.

1.3 Feature Extraction with Frequencies

$\boldsymbol{X}^{(m)}$: Features vector of a sentence m.It is a row vector.

$$\boldsymbol{X}^{(m)} = \left[\underbrace{1}_{\text{bias}}, \sum_{w} \text{freq}(w, \text{pos}), \sum_{w} \text{freq}(w, \text{neg})\right]$$

Then all the examples m can be represented as the matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix} \tag{1.1}$$

1.4 Logistic Regression: Regression and Sigmoid

The *logits* $z^{(i)}$ for an example i can be calculated as:

$$z^{(i)} = \boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \tag{1.2}$$

The hypothesis function h (sigmoid function σ):

$$h\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right) = h(z^{(i)}) = \sigma\left(z^{(i)}\right) = \frac{1}{1 + e^{-z^{(i)}}} \tag{1.3}$$

Note: All the h values are between 0 and 1.

1.5 Cost Function

The loss function for a single training example is:

$$\mathcal{L}(\boldsymbol{\theta}) = -\left[y^{(i)} \log\left(h(z^{(i)})\right) + \left(1 - y^{(i)}\right) \log\left(1 - h(z^{(i)})\right)\right]$$

The cost function used for logistic regression is the average of the log loss across all training examples:

$$\mathcal{J}(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(h(z^{(i)})\right) + \left(1 - y^{(i)}\right) \log\left(1 - h(z^{(i)})\right)\right] \tag{1.4}$$

Where:

- m is the number of training examples.
- $y^{(i)}$: is the actual label of the i^{th} training example.
- $h(z^{(i)})$ is the model prediction for the i^{th} training example.

1.6 Gradient Descent

The gradient of the cost function $\mathcal{J}$ with respect to one of the weights θ_j is

$$\nabla_{\theta_j} \mathcal{J}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(h(z^{(i)}) - y^{(i)}\right) x_j \tag{1.5}$$

To update the weight θ_j using gradient descent:

$$\theta_j := \theta_j - \alpha \nabla_{\theta_j} \mathcal{J}(\boldsymbol{\theta}) \tag{1.6}$$

Where α is the *learning rate*, a value to control how big a single update will be.

1.7 Vectorized Implementation

Putting all the examples in a matrix $\boldsymbol{X}$ (Equation 1.1), then the previous equations become:

$$\mathbf{z} \overset{(1.2)}{=} \boldsymbol{X\theta}$$

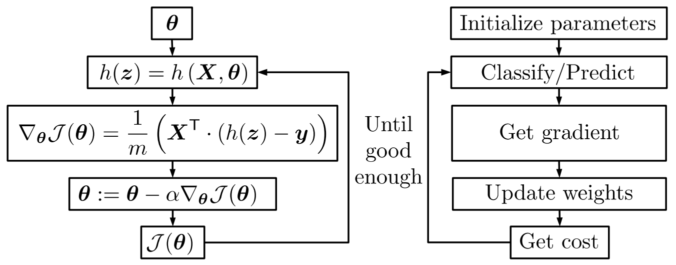
$$h(\boldsymbol{X}, \boldsymbol{\theta}) \overset{(1.3)}{=} h(\mathbf{z}) = \sigma(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}}$$

$$\mathcal{J}(\boldsymbol{\theta}) \overset{(1.4)}{=} -\frac{1}{m}\left[\mathbf{y}^{\mathsf{T}} \cdot \log(h(\mathbf{z})) + (1 - \mathbf{y})^{\mathsf{T}} \cdot \log(1 - h(\mathbf{z}))\right]$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \overset{(1.5)}{=} \frac{1}{m}\left(\boldsymbol{X}^{\mathsf{T}} \cdot (h(\mathbf{z}) - \mathbf{y})\right)$$

$$\boldsymbol{\theta} :\overset{(1.6)}{=} \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$$

Figure 1.1: Training Logistic Regression


$\boldsymbol{\theta}$
$h(\boldsymbol{z}) = h(\boldsymbol{X}, \boldsymbol{\theta})$
$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \frac{1}{m}\left(\boldsymbol{X}^{\mathsf{T}} \cdot (h(\boldsymbol{z}) - \boldsymbol{y})\right)$
$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$
$\mathcal{J}(\boldsymbol{\theta})$
Until good enough
Initialize parameters
Classify/Predict
Get gradient
Update weights
Get cost

1.8 Testing Logistic Regression

$m_{(\text{val})}$: Total number of examples (sentences) in validation set.
$y_i^{(\text{val})}$: Ground truth label for an example $i \in \{1, \dots, m_{(\text{val})}\}$ in the validation set. 1 for positive sentiment, 0 for negative sentiment.
$\hat{y}_i^{(\text{val})}$: Predicted label (sentiment) for the i^{th} example in the validation set.

1. Perform testing on unseen validation data $\boldsymbol{X}^{(\text{val})}, \mathbf{y}^{(val)}$ using trained weights $\boldsymbol{\theta}$.
2. Calculate $h(\boldsymbol{X}^{(\text{val})}, \boldsymbol{\theta}) = h(\mathbf{z})$
3. Predict $\hat{y}_i^{(\text{val})}$ for each example as follows

$$\hat{y}_i^{(\text{val})} = \begin{cases} 1, & \text{If } h(\mathbf{z})_i \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

4. Calculate the *accuracy score* for all examples in the validation set:

$$\begin{aligned} \text{accuracy} &= \frac{1}{m_{(\text{val})}} \sum_{i=1}^{m_{(\text{val})}} \left(\hat{y}_i^{(\text{val})} == y_i^{(\text{val})}\right) \\ &= 1 - \underbrace{\frac{1}{m_{(\text{val})}} \sum_{i=1}^{m_{(\text{val})}} \left|\hat{y}_i^{(\text{val})} - y_i^{(\text{val})}\right|}_{\text{error}} \end{aligned}$$

2 Naïve Bayes

2.1 Conditional Probability and Bayes Rule

Conditional Probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.7}$$

2.2 Naïve Bayes Assumptions

- Independence of events $P(A \cap B) = P(A)P(B)$. It assumes that the words in a piece of text are independent of one another, which is not true in reality, but it works well.
- Relative frequency in corpus: It relies on the distribution of the training data sets. A good data set will contain the same proportion of positive and negative tweets as a random sample would. However, most of available annotated corpora are artificially balanced. In reality positive sentences occur more frequently than negative.

2.3 Notation

class $\in \{\text{pos}, \text{neg}\}$.
w: A unique word in the vocabulary.
ratio(w_i): Ratio of the probability that the word w_i being positive to being negative.
N_{class}: The total number of words in a class.
N: total number of words in the corpus.

2.4 Naïve Bayes Introduction

$$N_{\text{class}} = \sum_{i=1}^{V} \text{freq}(w_i, \text{class}) \tag{1.8}$$

$$P(\text{class}) = \frac{N_{\text{class}}}{N} \tag{1.9}$$

$$N = N_{\text{pos}} + N_{\text{neg}}$$

$$P(\text{neg}) = 1 - P(\text{pos})$$

$$\begin{aligned} P(w|\text{class}) &= \frac{\text{freq}(w, \text{class})}{N_{\text{class}}} \\ &\approx \frac{\text{freq}(w, \text{class}) + 1}{N_{\text{class}} + V} \qquad \text{(Laplacian smoothing)} \end{aligned} \tag{1.10}$$

$$\sum_{i=1}^{V} P(w_i|\text{class}) = 1$$

The Naive Bayes inference condition rule for binary classification (of a sentence):

$$\prod_{i=1}^{n} \frac{P(w_i|\text{pos})}{P(w_i|\text{neg})}$$

Where n: number of words in a sentence.

Likelihood

$$\begin{aligned} \text{ratio}(w) &= \frac{P(w|\text{pos})}{P(w|\text{neg})} \\ &\overset{(1.10)}{\approx} \frac{P(w|\text{pos}) + 1}{P(w|\text{neg}) + 1} \qquad \text{(Laplacian smoothing)} \end{aligned} \tag{1.11}$$

$$\text{ratio}(w) = \begin{cases} 0 : 1 & \text{Negative sentiment.} \\ 1 & \text{Neutral Sentiment.} \\ 1 : \infty & \text{Positive sentiment.} \end{cases}$$

$$P(\text{class}|w_i) \overset{(1.7)}{=} \frac{P(\text{class})P(w_i|\text{class})}{P(w_i)} \tag{1.12}$$

$$\frac{P(\text{pos}|w_i)}{P(\text{neg}|w_i)} \overset{(1.12)}{=} \frac{P(\text{pos})P(w_i|\text{pos})}{P(\text{neg})P(w_i|\text{neg})} \tag{1.13}$$

$$\frac{P(\text{pos}|\text{sentence})}{P(\text{neg}|\text{sentence})} \overset{(1.13)}{=} \frac{P(\text{pos})}{P(\text{neg})} \prod_{i=1}^{n} \frac{P(\text{pos})P(w_i|\text{pos})}{P(\text{neg})P(w_i|\text{neg})} \tag{1.14}$$

$$= \frac{P(\text{pos})}{P(\text{neg})} \prod_{i=1}^{n} \text{ratio}(w_i)$$

$$\overset{(1.11)}{\approx} \underbrace{\frac{P(\text{pos})}{P(\text{neg})}}_{\substack{\text{prior} \\ \text{ratio}}} \underbrace{\prod_{i=1}^{n} \frac{P(w_i|\text{pos}) + 1}{P(w_i|\text{neg}) + 1}}_{\text{likelihood}} \tag{1.15}$$

Where n: number of words in a sentence.

Log Likelihood Score

Carrying repeated multiplications in 1.15 can result in numerical underflow. This problem is solved by taking log of both sides of the equation to calculate the *log likelihood score* of a sentence using the following equation:

$$\log \frac{P(\text{pos}|\text{sentence})}{P(\text{neg}|\text{sentence})} \overset{(1.15)}{=} \log \left[\frac{P(\text{pos})}{P(\text{neg})} \prod_{i=1}^{n} \text{ratio}(w_i) \right]$$

$$= \log \frac{P(\text{pos})}{P(\text{neg})} + \sum_{i=1}^{n} \log(\text{ratio}(w_i))$$

$$= \underbrace{\log \frac{P(\text{pos})}{P(\text{neg})}}_{\text{logprior}} + \underbrace{\sum_{i=1}^{n} \log \frac{P(w_i|\text{pos}) + 1}{P(w_i|\text{neg}) + 1}}_{\text{log likelihood}}$$

$$= \log \frac{P(\text{pos})}{P(\text{neg})} + \underbrace{\sum_{i=1}^{n} \lambda(w_i)}_{\substack{\text{log} \\ \text{likelihood}}} \tag{1.16}$$

Where

$$\begin{aligned} \lambda(w_i) &= \log(\text{ratio}(w_i)) \\ &\overset{(1.11)}{=} \log \frac{P(w_i|\text{pos}) + 1}{P(w_i|\text{neg}) + 1} \end{aligned} \tag{1.17}$$

$$\lambda(w_i) \begin{cases} < 0 & \text{Negative word.} \\ = 0 & \text{Neutral word.} \\ > 0 & \text{Positive word.} \end{cases}$$

If *log likelihood score* is > 0, the sentence is positive. If it is < 0, the sentence is negative.

2.5 Training Naïve Bayes

1. Collect and annotate corpus.
   Preprocess text:
   - Lowercase.
   - Remove punctuation, URLs, names.
   - Remove stop words.
   - Stemming [Por80].
   - Tokenize sentences $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]$
2. Word count.
   (a) Compute freq$(w, class)$ for every word in the vocabulary.
   (b) Compute N_{class} [equation 1.8]
3. Compute conditional probabilities $P(w|\text{pos}), P(w|\text{neg})$ [equation 1.10]
4. Calculate the *lambda score* ($\lambda(w)$) for each word [equation 1.17]
5. Get the *logprior*:

$$\log \frac{P(\text{pos})}{P(\text{neg})} \overset{(1.9)}{=} \log \frac{N_{\text{pos}}}{N_{\text{neg}}}$$

   If you are working with a balanced dataset ($N_{\text{pos}} = N_{\text{neg}}$), then logprior $= 0$

2.6 Testing Naïve Bayes

$m_{(\text{val})}$: Total number of examples (sentences) in validation set.
$y_i^{(\text{val})}$: Ground truth label for an example $i \in \{1, \ldots, m_{(\text{val})}\}$ in the validation set. 1 for positive sentiment, 0 for negative sentiment.
$\hat{y}_i^{(\text{val})}$: Predicted label (sentiment) for the i^{th} example in the validation set.

1. Perform testing on unseen validation data $\boldsymbol{X}^{(\text{val})}, \mathbf{y}^{(\text{val})}$
2. first, calculate *log likelihood score* for each sentence in the examples [equation 1.16]
3. Predict $\hat{y}_i^{(\text{val})}$ for each example as follows

$$\hat{y}_i^{(\text{val})} = \begin{cases} 1, & \text{If } \textit{log likelihood score} > 0 \\ 0, & \text{otherwise} \end{cases}$$

4. Calculate the *accuracy score* for all examples in the validation set:

$$\begin{aligned} \text{accuracy} &= \frac{1}{m_{(\text{val})}} \sum_{i=1}^{m_{(\text{val})}} \left(\hat{y}_i^{(\text{val})} == y_i^{(\text{val})} \right) \\ &= 1 - \underbrace{\frac{1}{m_{(\text{val})}} \sum_{i=1}^{m_{(\text{val})}} \left| \hat{y}_i^{(\text{val})} - y_i^{(\text{val})} \right|}_{\text{error}} \end{aligned}$$

   For a word not in the corpus, it is treated as neutral ($\lambda(w) = 0$)

3 Vector Space Models

- Represent words and documents as *vectors*.
- Representation that *captures* relative *meaning*.

3.1 Word by Word and Word by Doc.

Word by Word Design (W/W)

Counts the *co-occurrence* of two different words, which is the *number of times* they occur together within a certain distance k.
With *word by word* design you get a representation matrix with $n \times n$ entries, where n equals to vocabulary size V.

Word by Document Design (W/D)

Counts the *Number of times a word* occurs within a certain category.
Represented by a matrix with $n \times c$ entries, where c is the number of categories.

3.2 Euclidean Distance

The *euclidean distance* between two n-dimensional vectors:

$$\begin{aligned} d(\vec{v},\vec{w}) &= d(\vec{w},\vec{v}) \\ &= \|\vec{v}-\vec{w}\| \\ &= \sqrt{(v_1-w_1)^2+(v_2-w_2)^2+\ldots+(v_n-w_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(v_i-w_i)^2} \end{aligned}$$

Where

- n is the number of elements in the vector.
- The more similar the words, the more likely the Euclidean distance will be close to 0.

3.3 Cosine Similarity

The main advantage of this metric over the *euclidean distance* is that it isn't biased by the size difference between the representations.
Vector norm:

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Dot product:

$$\vec{v}\cdot\vec{w} = \sum_{i=1}^{n} v_i \cdot w_i$$

Cosine similarity:

$$\cos(\theta) = \frac{\vec{v}\cdot\vec{w}}{\|\vec{v}\|\|\vec{w}\|}$$

Cosine similarity gives values between -1 and 1.

$$\cos(\theta) = \begin{cases} 1 & \text{Parallel and in the same direction.} \\ 0 & \text{Orthogonal(perpendicular).} \\ -1 & \text{Point exactly in opposite directions.} \end{cases}$$

- Numbers in the range $[0,1]$ indicate a *similarity score.*
- Numbers in the range $[-1,0]$ indicate a *dissimilarity score.*

3.4 Manipulating Words in Vector Spaces

[Mik+13]

3.5 Visualization and PCA

PCA is used to visualize the embeddings on a k-dimensional subspace of the original n-dimensional subspace of the word embeddings.
Eigenvector: Uncorrelated features for your data.
Eigenvalue: The amount of information retained by each feature.
Perform PCA on a data matrix $\boldsymbol{X} = [\mathbf{x}_1|\mathbf{x}_2|\ldots|\mathbf{x}_n]^\mathsf{T} \in \mathbb{R}^{m\times n}$, where m is the number of examples, n is the dimension (length) of a word embedding.
Steps of PCA:

1. Mean normalize data and obtain the normalized data matrix $\bar{\boldsymbol{X}}$

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i, \qquad \boldsymbol{\sigma} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i^2 - \boldsymbol{\mu}^2}$$

$$\bar{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$x_i = \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$$

$$\bar{\boldsymbol{X}} = [\bar{\mathbf{x}}_1|\bar{\mathbf{x}}_2|\ldots|\bar{\mathbf{x}}_n]^\mathsf{T}$$

2. Get the $n\times n$ *covariance matrix* $\boldsymbol{\Sigma}$

$$\boldsymbol{\Sigma} = \frac{1}{m}\bar{\boldsymbol{X}}^\mathsf{T}\bar{\boldsymbol{X}}$$

3. Perform a *singular value decomposition* to get the *eigenvectors* $\boldsymbol{U}\in\mathbb{R}^{n\times n}$ and *eigenvalues* diagonal matrix $\boldsymbol{S}\in\mathbb{R}^{n\times n}$.

$$\boldsymbol{U},\boldsymbol{S} = \mathrm{SVD}(\boldsymbol{\Sigma})$$

4. Project data onto the k-dimensional principal subspace: Multiply your normalized data by the first k *eigenvectors* associated with the k largest *eigenvalues* to compute the projection $\boldsymbol{X'}\in\mathbb{R}^{m\times k}$.

$$\boldsymbol{B} = (\boldsymbol{U}_{ij})_{\substack{1\le i\le n\\ 1\le j\le k}}$$

$$\boldsymbol{X'} = \bar{\boldsymbol{X}}\boldsymbol{B}$$

The precentage of *retained variance* can be calculated from

$$\frac{\sum_{i=0}^{1} S_{ii}}{\sum_{j=0}^{d} S_{jj}}$$

4 Machine Translation and Document Search

4.1 Machine Translation

Transforming Word Vectors

Assume that we have a subset of a *source language* dataset of word embeddings $\boldsymbol{X} = [\mathbf{x}_1|\mathbf{x}_2|\ldots|\mathbf{x}_m]^\mathsf{T}$ and a translation subset of *destination language* dataset $\boldsymbol{Y} = [\mathbf{y}_1|\mathbf{y}_2|\ldots|\mathbf{y}_m]^\mathsf{T}$ We want to find a transformation matrix $\boldsymbol{R}$ such that:

$$\boldsymbol{XR}\approx\boldsymbol{Y}$$

Cost function:

$$\mathcal{J} = \frac{1}{m}\|\boldsymbol{XR}-\boldsymbol{Y}\|_F^2$$

where:

- m is the number of examples.
- $\|\boldsymbol{A}\|_F$ is the *Frobenius norm,*

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

- The reason for taking the square is that it's easier to compute the gradient of the squared Frobenius.

The gradient of the cost function with respect to the *transformation matrix*:

$$\begin{aligned} \frac{\partial\mathcal{J}}{\partial\boldsymbol{R}} &= \frac{\partial}{\partial\boldsymbol{R}}\frac{1}{m}\|\boldsymbol{XR}-\boldsymbol{Y}\|_F^2 \\ &= \frac{2}{m}(\boldsymbol{XR}-\boldsymbol{Y})^\mathsf{T}\boldsymbol{X} \\ &= \frac{2}{m}\boldsymbol{X}^\mathsf{T}(\boldsymbol{XR}-\boldsymbol{Y}) \end{aligned}$$

Then we use *gradient descent* to optimize the transformation matrix:

$$\boldsymbol{R} := \boldsymbol{R} - \alpha\frac{\partial\mathcal{J}}{\partial\boldsymbol{R}}$$

The predictions can be obtained using the trained $\boldsymbol{R}$ matrix:

$$\hat{\boldsymbol{Y}} = \boldsymbol{XR}$$

The translation of a word i can be found using k-nearest neighbor of $\hat{\mathbf{y}}_i$ from $\boldsymbol{Y}$ with $k=1$.

4.2 Document Search

Document Representation

1. **Bag-of-words (BOW) document models**
   Text documents are sequences of words. The ordering of words makes a difference.
2. **Document embeddings**
   A document can be represented as a *document vector* by summing up the word embeddings of every word in the document. If we don't know the embedding of a word, we can ignore that word.

Locality Sensitive Hashing

A more efficient version of k-nearest neighbors can be impelmented using locality sensitive hashing. Instead of searching the vector space we can only search in a subspace for the nearest neighboring vectors.
Assume we have a plane(hyperplane) π that divides the vector space that has a normal vector $\mathbf{p}$, then for any point with a position vector $\mathbf{v}$:

$$\mathbf{p}\cdot\mathbf{v}\begin{cases}>0, & \text{the point is above the plane.}\\ =0, & \text{the point is on the plane.}\\ <0, & \text{the point is below the plane.}\end{cases}$$

Multiplanes Hash Functions

- *Multiplanes hash functions* are based on the idea of numbering every single region that is formed by the intersection of n planes.
- We can divide the vector space into 2^n parts(*hash buckets*).

The hash value for a position of a vector $\mathbf{v}$ with respect to a plane $\mathbf{p}_i$ is:

$$h_i=\begin{cases}1, & \text{If } \operatorname{sign}(\mathbf{p}_i\cdot\mathbf{v})\geq 0.\\ 0, & \text{If } \operatorname{sign}(\mathbf{p}_i\cdot\mathbf{v})< 0.\end{cases}$$

Where $i=\{1,\ldots,n\}$
The combined hash bucket number for a vector (for all planes):

$$\text{hash}=\sum_{i=1}^{n}2^{i-1}\times h_i$$

Chapter 2 Probabilistic Models

1 Autocorrect and Minimum Edit Distance

1.1 Autocorrect

Reference: [Nor07]
How it works

1. Identify a misspelled word.
   Words not in the dictionary are misspelled words.
2. Find strings n edit distance away.
   Edit: an operation performed on a string to change it.
   Examples (for a string with n letters):

Operation	Description	Output Count
Insert	Add a letter	26(n+1)
Delete	Remove a letter	n
Replace	Change 1 letter to another	25n
Switch	Swap 2 adjacent letters	n-1

3. Filter candidates.
   Given a *vocabulary*, filter the *edit* list for *candidate words* found in the vocabulary.
4. Calculate word probabilities.

$$P(w)=\frac{\text{count}(w)}{M}$$

Where:

- $P(w)$: Probability of a word.
- $\text{count}(w)$: Number of times the word apprears.
- M: Total number of words in the *corpus*.

Then select the word with the highest probability as your *autocorrect* replacement.

Algorithm 1: Autocorrect

```
def autocorrect(word, n):
    Data:
    probs: a dictionary that maps each word to its probability in the corpus.
    $probs[w \to P(w)](x) = \begin{cases} P(w)=\frac{count(w)}{M} & \text{for } x=w \\ 0 & \text{otherwise}\end{cases}$
    vocab: a set containing all the vocabulary.
    Result: n-best: a set of tuples with the most probable n corrected words and their probabilities.
    suggestions = $\phi$
    $n$-best= $\phi$
    if word $\in$ vocab:
        suggestions = suggestions $\cup$ {word}
    else:
        one-edit-set = one-edit-distance(word) $\cap$ vocab
        if one-edit-set $\neq \phi$:
            suggestions = suggestions $\cup$ one-edit-set
        else:
            two-edit-set = two-edit-distance(word) $\cap$ vocab
            if two-edit-set $\neq \phi$:
                suggestions = suggestions $\cup$ two-edit-set
            else:
                suggestions = suggestions $\cup$ {word}
    best-words[$w \to probs(w)$]($x$) = {$x = w | w \in suggestions$}
    $n$-best = {The set of top $n$ words from best-words sorted by probabilities}
```

1.2 Minimum Edit Distance

Reference: [Jur12]

Minimum edit distance is the sum of costs of edits needed to transform one string into the other.It evaluates the similarity between two strings.

It is used in spelling correction, document similarity, machine translation, DNA sequencing and more.

Edits (operations) are:

Operation	Description	Cost
Insert	Add a letter	1
Delete	Remove a letter	1
Replace	Change 1 letter to another	2

Minimum Edit Distance Algorithm

Minimum edit distance can be calculated using *dynamic programming.* It breaks a problem down into subproblems which can be combined to form the final solution.To do this efficiently, we will use a table (see Figure 2.1) to maintain the previously computed substrings and use those to calculate larger substrings.

Initialization:

$$D[0,0]=0 \tag{2.1}$$
$$D[i,0]=D[i-1,0]+del\_cost(source[i]) \tag{2.2}$$
$$D[0,j]=D[0,j-1]+ins\_cost(target[j]) \tag{2.3}$$

Per cell operations:

$$D[i,j]=\min\begin{cases}D[i-1,j] & +\,del\_cost\\ D[i,j-1] & +\,ins\_cost\\ D[i-1,j-1] & +\begin{cases}rep\_cost; & \text{if } source[i]\neq target[j]\\ 0; & \text{if } source[i]=target[j]\end{cases}\end{cases} \tag{2.4}$$

$$\text{Minimum edit distance}=D[m,n] \tag{2.5}$$

	j	0	1	$\cdots$	n
i		#	T_0	$\cdots$	T_{n-1}
0	#	$D[0,0]$	$D[0,1]$	$\cdots$	$D[0,n]$
1	S_0	$D[1,0]$	$D[1,1]$	$\cdots$	$D[1,n]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$D[2,n]$
m	S_{m-1}	$D[m,0]$	$D[m,1]$	$\cdots$	$D[m,n]$

Figure 2.1: Minimum Edit Distance Table

Algorithm 2: Minimun Edit Distance

```
1  def min-edit-distance(source, target):
       Data:
       source: a string corresponding to the string you are
       starting with.
       target: a string corresponding to the string you want
       to end with.
       Result:
       D: a matrix of size $(m + 1 \times n + 1)$ containing
       minimum edit distances (see Figure 2.1)
       med: the minimum edit distance required to convert
       the source string to the target
2      $D[0,0] \overset{(2.1)}{=} 0$
3      for $i \in \{1, 2, \dots, m\}$:
4          $D[i,0] \overset{(2.2)}{=} [i-1,0] + del\_cost$
5          for $j \in \{1, 2, \dots, n\}$:
6              $D[0,j] \overset{(2.3)}{=} [0,j-1] + ins\_cost$
7              if $source[i-1] = target[j-1]$:
8                  r_cost = 0
9              else:
10                 r_cost = 2
11             $D[i,j] \overset{(2.4)}{=} \min \begin{cases} D[i-1,j] & + del\_cost \\ D[i,j-1] & + ins\_cost \\ D[i-1,j-1] & + r\_cost \end{cases}$
12     $med \overset{(2.5)}{=} D[m,n]$
```

Example:

$$\begin{aligned} &source \longrightarrow target \\ &\text{“play”} \longrightarrow \text{“stay”} \end{aligned}$$

$$\begin{aligned} D[i,j] &= source[:i] \longrightarrow target[:j] \\ D[2,3] &= \text{“pl”} \longrightarrow \text{“sta”} \\ D[0,0] &= \# \longrightarrow \# \\ D[m,n] &= source \longrightarrow target \end{aligned}$$

where #: empty string.

	j	0	1	2	3	4
i		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	l	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4

Figure 2.2: Minimum Edit Distance of “play” $\longrightarrow$ “stay”

2 Part of Speech Tagging and Hidden Markov Models

Reference: [JM19, Chapter 8]

2.1 Part of Speech Tagging

Part of speech (POS) tagging is the process of assigning tags that represent categories of *parts of speech* to words of a corpus.

Applications of POS tagging:

- Identifying named entities.

  Eiffel tower is located in *Paris*.

- Co-reference resolution.

  The Eiffel tower is located in Paris, *it is* 324 meters high.

- Speech recognition.

lexical term	tag	example
noun	NN	something, nothing
verb	VB	learn, study
determiner	DT	the,a
w-adverb	WRB	why, where
...	...	...

No.	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Table 2.1: Part-of-Speech Tags
Source: [San90] and [LJP03]

2.2 Markov Chains

Sates:

$$\mathcal{S} = \{s_1, s_2, \dots, s_N\}$$

Markov property: The probability of the next event only depends on the current event.

Initial Probability Vector

$$\boldsymbol{\pi} = [\pi_1, \pi_2, \dots, \pi_N]$$

Example:

$\boldsymbol{\pi} =$

	NN	VB	O
π (initial)	0.4	0.1	0.5

The Transition Matrix

The *transition matrix* has a dimension $(N \times N)$.

$$\boldsymbol{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \dots & a_{N,N} \end{pmatrix} = \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{pmatrix} \tag{2.6}$$

For all the outgoing transition probabilities:

$$\sum_{j=1}^{N} a_{ij} = 1$$

Example:

$\boldsymbol{A} =$

	NN	VB	O
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

2.3 Hidden Markov Models

Hidden states: parts of speech. States that are hidden and not directly observable from the text data.

Emission probabilities: The probability of a visible observation when we are in a particular state. Emission probabilities describe the transition probabilities between the hidden states $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$ (parts of speech) of hidden Markov model to the *observables* or *emissions* (words of corpus) $\mathcal{O} = \{o_1, o_2, \dots, o_V\}$.

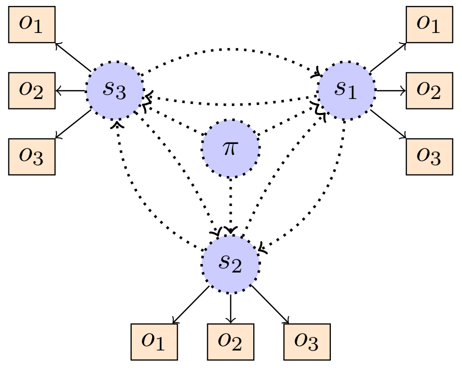


Figure 2.3: Hidden Markov Model

Emission Matrix

$$\boldsymbol{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1V} \\ b_{21} & b_{22} & \dots & b_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NV} \end{pmatrix} = \begin{pmatrix} P(o_1|s_1) & P(o_2|s_1) & \dots & P(o_V|s_1) \\ P(o_1|s_2) & P(o_2|s_2) & \dots & P(o_V|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(o_1|s_N) & P(o_2|s_N) & \dots & P(o_V|s_N) \end{pmatrix} \tag{2.7}$$

Emission matrix $\boldsymbol{B}$ has a dimension $(N \times V)$

Where N is the number hidden states (parts of speech tags), V is the number of observables (words in corpus).

$$\sum_{j=1}^{M} b_{ij} = 1$$

Example:

$\boldsymbol{B} =$

	going	to	eat	...
NN (noun)	0.5	0.1	0.02	
VB (verb)	0.3	0.1	0.5	
O (other)	0.3	0.5	0.68	

A word can have different parts of speech assigned depending on context in which they appear:

- "He lay on his $\overbrace{\text{back}}^{\textbf{NN}}$"
- "I will be $\overbrace{\text{back}}^{\textbf{RB}}$"

2.4 Calculating Transition Probabilities

1. Count the occurrences of tag pairs (the number of times each tag ($t_i \in S$) at time step i happened next to another tag ($t_{i-1} \in S$) at time step $i-1$).

$$C(t_i, t_{i-1})$$

2. Calculate probabilities: divide the counts by row sum to normalize the counts: The probability of a tag at position i given the tag at position $i-1$ becomes:

$$P(t_i|t_{i-1}) \overset{(2.6)}{=} a_{\text{rindex}(t_{i-1}),\text{cindex}(t_i)} = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)} \tag{2.8}$$

Where

- N is the total number of tags.

Smoothing

To avoid division by zero and zero probabilities apply smoothing to the equation (2.8)

$$P(t_i|t_{i-1}) \overset{(2.8)}{=} \frac{C(t_{i-1}, t_i) + \varepsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N \cdot \varepsilon} = \frac{C(t_{i-1}, t_i) + \varepsilon}{C(t_{i-1}) + N \cdot \varepsilon}$$

Where

- $C(t_{i-1})$ is the count of the previous POS tag occurrence in the corpus.
- ε is a smoothing parameter.

2.5 Calculating Emission Probabilities

Calculate the number of time a (tag, word) pair showed in the training set:

$$C(t_i, w_i)$$

Compute the probability of a word given its tag:

$$P(w_i|t_i) \overset{(2.7)}{=} b_{\text{rindex}(t_i),\text{cindex}(w_i)} = \frac{C(t_i, w_i) + \varepsilon}{\sum_{j=1}^{V} C(t_i, w_j) + V \cdot \varepsilon} = \frac{C(t_i, w_i) + \varepsilon}{C(t_i) + V \cdot \varepsilon}$$

Where

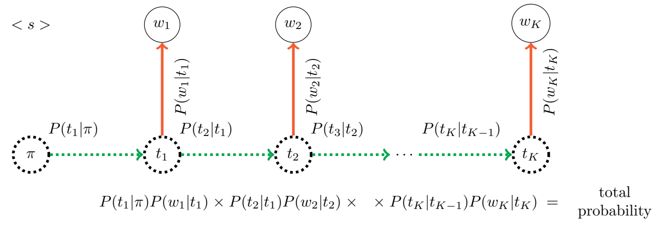
- w is a word (observable) in the corpus.
- $C(t_i)$ is the number of times a tag has occured in the corpus.
- V is the number of words in the vocabulary.

2.6 The Viterbi Algorithm

The Viterbi algorithm computes the most likely sequence of parts of speech tags for a given sentence (sequence of observations) $\mathbf{w} = [w_1, w_2, \dots, w_K]$

The *joint probability* (combined probability) of the observing a word is calculated by multiplying the transition probability with the emission probability.

The *total probability* is calculated by multiplying all joint probabilities of steps of the sequence.



Auxiliary Matrices

Given your transition and emission probabilities, you first populates and then use the auxiliary matrices $\boldsymbol{C}$ and $\boldsymbol{D}$.
The matrix $\boldsymbol{C} \in \mathbb{R}^{N \times K}$ holds the intermediate optimal probabilities.
The matrix $\boldsymbol{D} \in \mathbb{R}^{N \times K}$ holds the indices of the visited states (or *best paths*, the different states you're traversing when finding the most likely sequence of parts of speech tags for the given sequence of words).

$$\boldsymbol{C} = \begin{array}{|c|c|c|c|c|}\hline & w_1 & w_2 & \dots & w_K \\ \hline t_1 & & & & \\ \vdots & & & & \\ t_N & & & & \\ \hline \end{array}$$

$$\boldsymbol{D} = \begin{array}{|c|c|c|c|c|}\hline & w_1 & w_2 & \dots & w_K \\ \hline t_1 & & & & \\ \vdots & & & & \\ t_N & & & & \\ \hline \end{array}$$

Steps

1. **Initialization:**

   The first column of each of the matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ is populated.

$$\begin{aligned} c_{i,1} &= P(s_i|\pi)P(w_1|s_i) \\ &= \pi_i \cdot b_{i,\text{cindex}(w_1)} \end{aligned} \tag{2.9}$$

   For matrix $\boldsymbol{D}$, in the first column, set all entries to zero, as there are no proceeding parts of speech tags we have traversed.

$$d_{i,1} = 0$$

   Vectorized:

$$\mathbf{c}_1 = \boldsymbol{\pi} \odot \mathbf{b}_{\text{cindex}(w_1)} \tag{2.10}$$

$$\mathbf{d}_1 = \vec{0}$$

2. **Forward pass:**

$$c_{i,j} = \max_k c_{k,j-1} \cdot a_{k,i} \cdot b_{i,\text{cindex}(w_j)} \tag{2.11}$$

$$d_{i,j} = \arg\max_k c_{k,j-1} \cdot a_{k,i} \cdot b_{i,\text{cindex}(w_j)} \tag{2.12}$$

   where

   (a) $a_{k,i}$ is the transition probability from the parts of speech tag t_k to the current tag t_i.

   (b) $c_{k,j-1}$ represents the probability for the preceding path you've traversed.

   Vectorized:

$$\mathbf{c}_j = \max\left(\mathbf{c}_{j-1} \odot \mathbf{a}'_i\right) \cdot b_{i,\text{cindex}(w_j)} \tag{2.13}$$

$$\mathbf{d}_j = \arg\max\left(\mathbf{c}_{j-1} \odot \mathbf{a}'_i\right) \cdot b_{i,\text{cindex}(w_j)} \tag{2.14}$$

   where $\boldsymbol{A}^\mathsf{T} = [\mathbf{a}'_1|\mathbf{a}'_2|\dots|\mathbf{a}'_N]$

3. **Backward pass:**

   The probability at $c_{i,K}$ is the probability of the most likely sequence of hidden states, generating the given sequence of words.

   Get the index of $c_{i,K}$

$$z_K = \underset{i}{\text{argmax}}\, c_{i,K}$$

   Use this index z_K to traverse backwards through the matrix $\boldsymbol{D}$, to reconstruct the sequence of parts of speech tags.

Implementation note:
For numerical stability use log probabilities:

$$\log(c_{i,1}) \overset{(2.9)}{=} \log(\pi_i) + \log(b_{i,\text{cindex}(w_1)})$$

$$\log(c_{i,j}) \overset{(2.11)}{=} \max_k \log(c_{k,j-1}) + \log(a_{k,i}) + \log(b_{i,\text{cindex}(w_j)})$$

$$d_{i,j} \overset{(2.12)}{=} \arg\max_k \log(c_{k,j-1}) + \log(a_{k,i}) + \log(b_{i,\text{cindex}(w_j)})$$

Vectorized:

$$\log(\mathbf{c}_1) \overset{(2.10)}{=} \log(\boldsymbol{\pi}) + \log(\mathbf{b}_{\text{cindex}(w_1)})$$

$$\log(\mathbf{c}_j) \overset{(2.13)}{=} \max\left[\log(\mathbf{c}_{j-1}) + \log(\mathbf{a}'_i)\right] + \log(b_{i,\text{cindex}(w_j)})$$

$$\mathbf{d}_j \overset{(2.14)}{=} \arg\max\left[\log(\mathbf{c}_{j-1}) + \log(\mathbf{a}'_i) + \log(b_{i,\text{cindex}(w_j)})\right]$$

Algorithm 3: Viterbi Algorithm

Data:
- $\mathcal{O} = \{o_1, o_2, \dots, o_V\}$, the *observation space*
- $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$, the *state space* , where s_n is a tag.
- $\boldsymbol{\pi} = [\pi_1, \pi_2, \dots, \pi_N]$.An array of *initial probabilities*, where $\pi_i = P(x_1 = s_i)$
- $\mathbf{w} = [w_1, w_2, \dots, w_K]$, A sequence of observations, $w_i \in \mathcal{O}$
- $\boldsymbol{A} \in \mathbb{R}^{N \times N}$. Transition matrix
- $\boldsymbol{B} \in \mathbb{R}^{N \times V}$. Emission matrix, where $B_{ij} = P(o_j|s_i)$

Result:
$\mathbf{t} = [t_1, t_2, \dots, t_K]$, the most likely hidden state sequence of parts of speech tags.

```
def VITERBI($\mathcal{O}, \mathcal{S}, \boldsymbol{\pi}, \mathbf{y}, \boldsymbol{A}, \boldsymbol{B}$):
    /* Initialization */
    for each state $i = 1, 2, \dots, N$:
        $C_{i,1} \leftarrow \log(\pi_i) + \log(B_{i,\text{cindex}(w_1)})$
        $D_{i,1} \leftarrow 0$
    /* Forward pass */
    for each observation $j = 2, 3, \dots, K$:
        for each state $i = 1, 2, \dots, N$:
            $C_{ij} \leftarrow \max_k C_{k,j-1} + \log(A_{k,i}) + \log(B_{i,\text{cindex}(w_j)})$
            $D_{ij} \leftarrow \arg\max_k C_{k,j-1} + \log(A_{k,i}) + \log(B_{i,\text{cindex}(w_j)})$
    /* Backward pass */
    $z_K \leftarrow \arg\max_i C_{i,K}$
    $t_K \leftarrow s_{z_K}$
    for $j = K, K-1, \dots, 2$:
        $z_{j-1} \leftarrow D_{z_j,\, j}$
        $t_{j-1} \leftarrow s_{z_{j-1}}$
    return t
```

DOI: 10.5281/zenodo.3987961

References

[JM19] Daniel Jurafsky and James H Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2019. URL: https://web.stanford.edu/%20jurafsky/slp3/.

[Jur12] Dan Jurafsky. *Minimum Edit Distance*. Stanford University, Jan. 2012. URL: https://web.stanford.edu/class/cs124/lec/med.pdf (visited on 08/07/2020).

[LJP03] Mark Lieberman, Yoon-Kyoung Joh, and Marjorie Pak. *Alphabetical list of part-of-speech tags used in the Penn Treebank Project*. 2003. URL: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (visited on 08/08/2020).

[Mik+13] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., Oct. 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (visited on 07/22/2020).

[Nor07] Peter Norvig. *How to Write a Spelling Corrector*. Feb. 2007. URL: https://norvig.com/spell-correct.html (visited on 08/04/2020).

[Por80] Martin F. Porter. “An algorithm for suffix stripping”. In: *Program* 14.3 (1980), pp. 130–137. DOI: 10.1108/eb046814. URL: https://tartarus.org/martin/PorterStemmer/.

[San90] Beatrice Santorini. “Part-of-speech tagging guidelines for the penn treebank project (3rd revision)”. In: *Technical Reports (CIS)* (1990), p. 570. URL: https://catalog.ldc.upenn.edu/docs/LDC99T42/tagguid1.pdf.