Natural Language Processing Specialization Formula Sheet

Fady Morris Milad (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.
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:

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

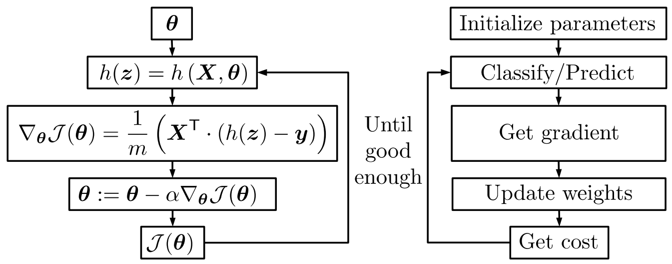
$$h\left(\boldsymbol{X}, \boldsymbol{\theta}\right) \overset{(1.3)}{=} h(\mathbf{z}) = \sigma\left(\mathbf{z}\right) = \frac{1}{1 + e^{-\mathbf{z}}}$$

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

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

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

Figure 1.1: Training Logistic Regression



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.
$\text{ratio}(w_i)$: 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

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

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

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

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

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

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

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:

$$\begin{aligned} \log \frac{P(\text{pos}|\text{sentence})}{P(\text{neg}|\text{sentence})} &\overset{(1.15)}{=} \log \left[\frac{P(\text{pos})}{P(\text{neg})} \prod_{i=1}^{n} \text{ratio}(w_i) \right] \\ &= \log \frac{P(\text{pos})}{P(\text{neg})} + \sum_{i=1}^{n} \log(\text{ratio}(w_i)) \\ &= \underbrace{\log \frac{P(\text{pos})}{P(\text{neg})}}_{\text{logprior}} + \underbrace{\sum_{i=1}^{n} \log \frac{P(w_i|\text{pos}) + 1}{P(w_i|\text{neg}) + 1}}_{\text{log likelihood}} \\ &= \log \frac{P(\text{pos})}{P(\text{neg})} + \underbrace{\sum_{i=1}^{n} \lambda(w_i)}_{\text{log likelihood}} \end{aligned} \tag{1.16}$$

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 $\boldsymbol{w} = [w_1, w_2, \dots, w_n]$
2. Word count.
   (a) Compute $\text{freq}(w, \mathit{class})$ 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 $i \in \{1, \dots, m_{(\text{val})}\}$ 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 $\boldsymbol{X}^{(\text{val})}, \mathbf{y}^{(\text{val})}$
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:

$$\boldsymbol{X R} \approx \boldsymbol{Y}$$

Cost function:

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

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*:

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

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:

$$\hat{\boldsymbol{Y}} = \boldsymbol{X R}$$

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):

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

References

[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).

[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/.

Fady Morris Milad (2020)