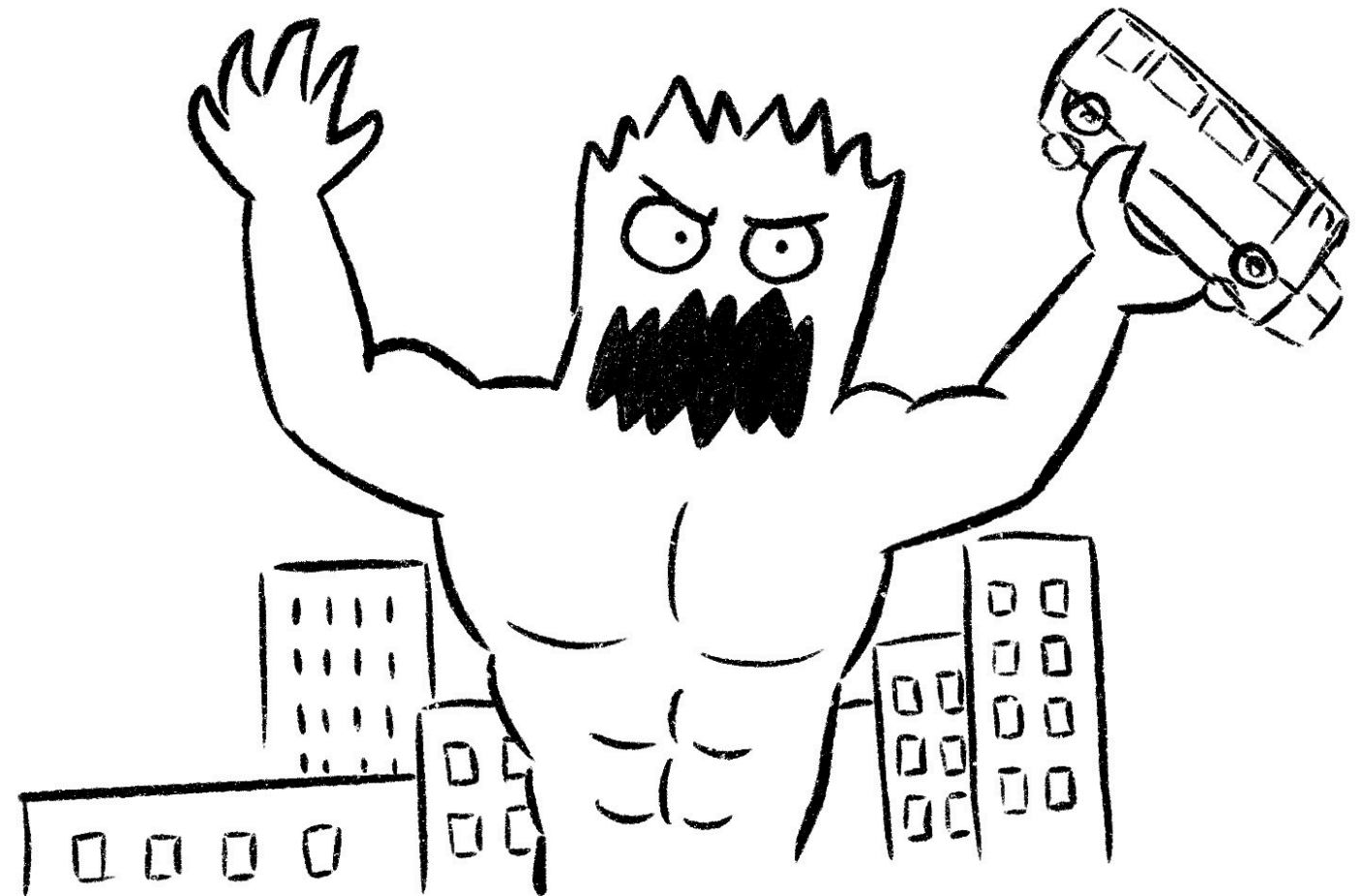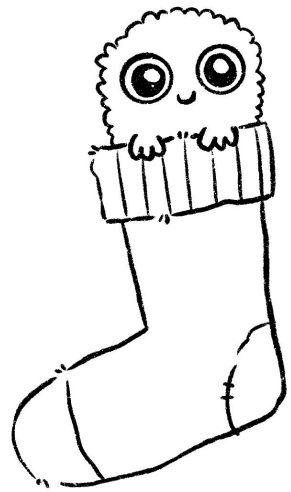# Feature Engineering

Reza Rawassizadeh

what we think about a dataset

what is the reality of a dataset

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- Feature Generation

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Outline

- **Feature Engineering Concepts**

- Feature Selection Methods

- Feature Generation

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Feature Engineering Concepts

- **Feature** is a representation of a raw data in a machine understandable format. For instance, we can convert {null, low, medium, high, all} to a numerical features as follows: {0, 0.25, 0.5, 0.75, 1}. Assuming the dataset as table, each attribute or feature presents one column in a table of data, which can contribute to the execution of machine learning algorithm [Ozdemir '18].

- **Feature engineering** is the process of identifying, filtering and/or reconstructing the most relevant data (from the original raw data) that we intend to use.

Feature Engineering Concepts

- **Feature selection** is the process of selecting a set of attributes which are useful for the machine learning algorithm and removing the attributes which are not important for the algorithm.

According to Chanrashekar and Satin [Chanrashekar '14] feature selection is the process of selecting inputs that reduce noise and effect of invariant variables.

- **Feature generation** is the process of making new features, which are useful for the machine learning algorithm, out of the existing raw data or existing features.

For example, assume we have *X, Y, Z* and an arithmetic mean of three attributes, generates a new feature, which might help the algorithm better discriminating the data, rather each of these attributes alone.

Feature Selection Example

- How to select the best chickens for breeding

- We have following information in our chicken table:
  'Chicken id', 'name', 'weight', 'height', 'daily mood score', 'eye shape', 'singing voice score', 'peak size', 'feet size', 'last vaccine received date', 'number of vaccine received', 'flu-shot received', 'number of eggs hatching per week', 'feather color'.

- Not all examples are that simple, sometimes we do not have enough domain knowledge to identify useful features, and we should automate this process.

Outline

- Feature Engineering Concepts

- **Feature Selection Methods**

- Feature Generation

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Feature Selection Methods

- Filtering

- Wrapper

- Embedded

Filtering Methods

- There are three known feature selection methods, **filtering**, **wrapper** and **embedded** methods.

- **Filtering method** process features and <u>removes the ones which are not relevant to the machine learning algorithm goal</u>, e.g. prediction.

- While we are dealing with numerical data, **correlation analysis** (Pearson, Kendall and Spearman) is one of the most convenient filtering methods to remove the redundant feature.

- Another filtering method is to use **covariance**. e.g. positive covariance -> two features tend to change together.

Wrapper Methods

- These methods are using a blackbox algorithm (we do not know what is happening inside the algorithm) and the algorithm decides about which features to use. Ideally, the algorithm should **combine *n* number of features in *n* different methods** to identify the best features, but this is not efficient.

- To address the efficiency challenge, **heuristic algorithms** or **search optimization** algorithms can be used to improve the search process.

- A good example of heuristic methods are deep learning optimization algorithms that use gradient descent, e.g., Adam, RMSprop and Adagrad.
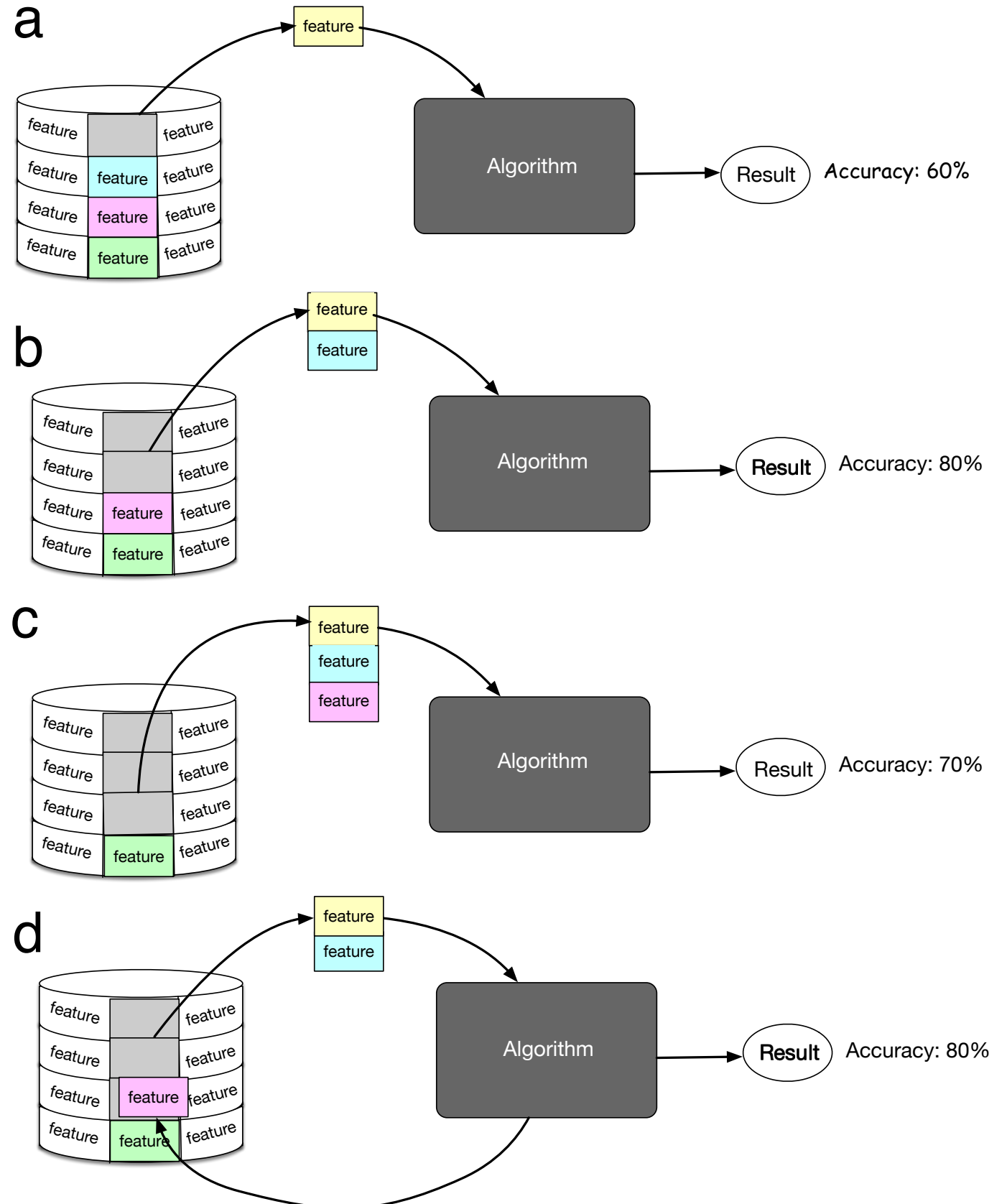
Wrapper Methods

- There are two categories of wrapper methods:

  - Sequential search methods

  - Heuristic search methods

Wrapper Methods
Sequential Search Method

Sequential Feature Selection

(SFS)

Wrapper Methods
Sequential Search Method

Sequential Backward Selection (SBS)

It starts with the largest set of features, then removes features one by one and checks the accuracy (or other objective function), until the desired accuracy has been achieved.

It operates in contrast to SFS, because first, it feeds all features into the algorithm, then it reduces them one by one to study changes in the objective function.

SBS
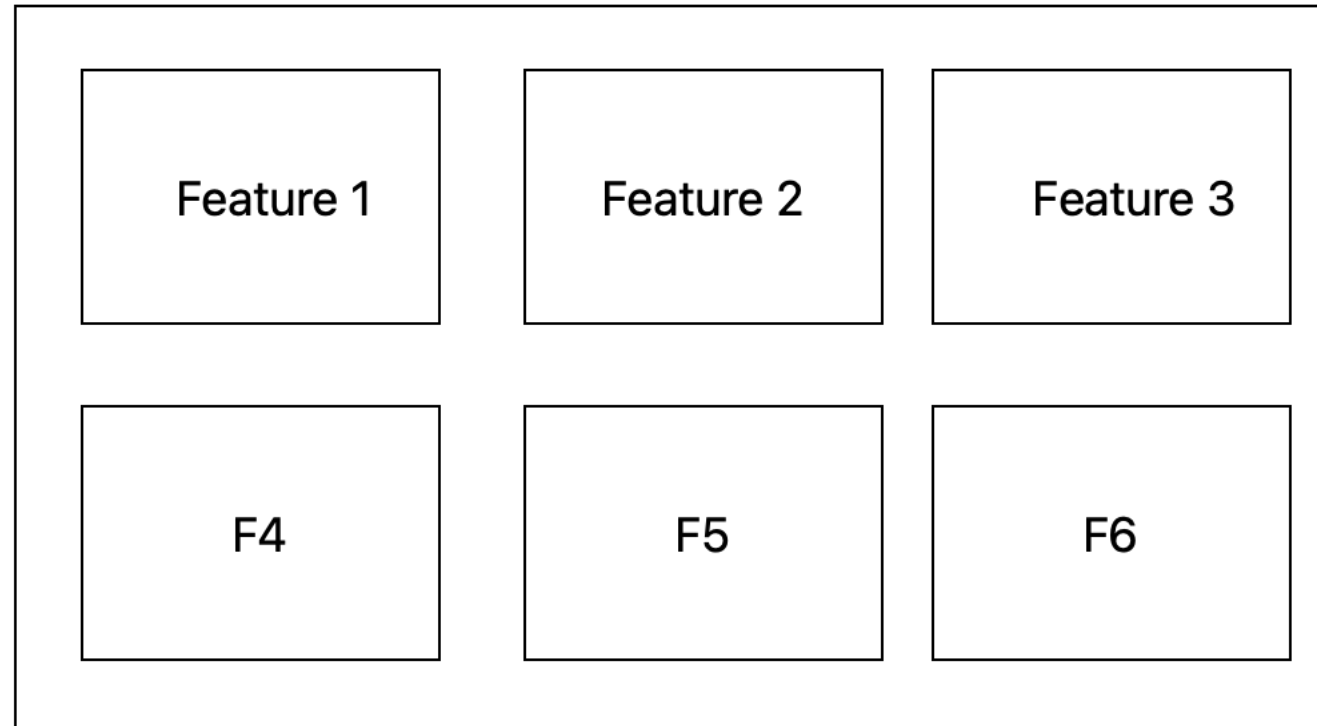
1- F1, F2, F3, F4, F5, F6 --> Algorithm => accuracy : 70%

2- F1, F2,, F4, F5, F6 --> Algorithm => accuracy : 75%

3- F1, F4, F5, F6 --> Algorithm => accuracy : 90%

4- F1, F4, F5 --> Algorithm => accuracy : 80%

5- F1, F3, F4, F5, F6 --> Algorithm => accuracy : 95%

Wrapper Methods
Heuristic Methods (Genetic Algorithms)

- Heuristics are techniques that *"try to solve a problem by finding an approximation instead of a best answer"*.

- **Genetic Algorithms** are well known methods to solve a problem in heuristic fashion.

- They are based on the Darwin's theory of *"Survival of the Fittest"*. Genetic algorithms are used to find 'optimal' or 'near optimal' solution for a problem which is hard to solve (these problems are called NP-Hard problems).

- Genetic algorithms try to search for a solution by somehow a random search, but they perform better than purely random search or brute force search.

Genetic Algorithm

We have a pool or **population** of solutions (or answers). Each solution is called **chromosome** and each chromosome has a set of variables or parameters which is called **gene**. The value of each gene is called **allele**. Think of a solution as a set of parameters that their combination yield a result and this result will be represented as a **fitness score**.

Entities which compose the population in genetic algorithms

gene = variable
allele = value
chromosome = set of variables

F(x,y,z)

Genetic Algorithm

Population: All possible solutions.
Chromosome: A single solution.
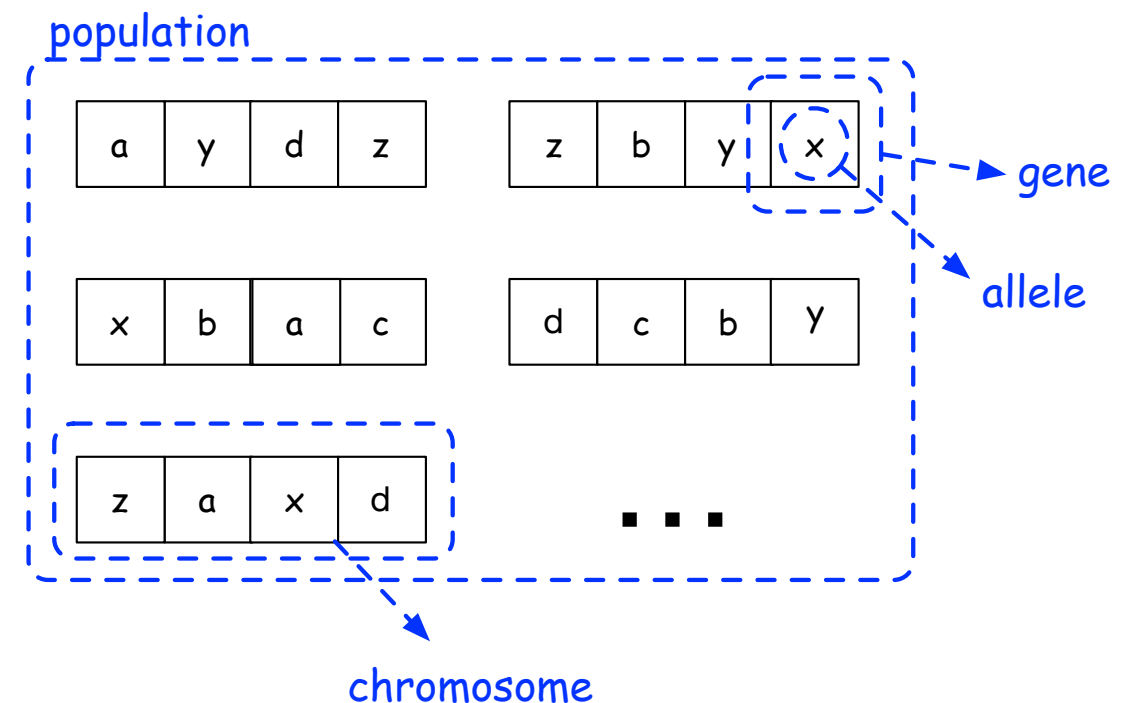**Gene:** Variables or parameters, which characterize the chromosome.
Allele: The value inside a gene. In other word, the value of the variable (gene).
Fitness score & Fitness function: A cost of a solution (chromosome), is acquired by a function called fitness function and this cost is presented in a value that is called fitness score.
Selection: Is a process in which it tries to find chromosomes with a high fitness scores for mating. The **mating process** here means they pass their genes together and make new children.
Crossover: Is a process in which two chromosomes (parents) combine together and pass over their genes to create new children (offspring).
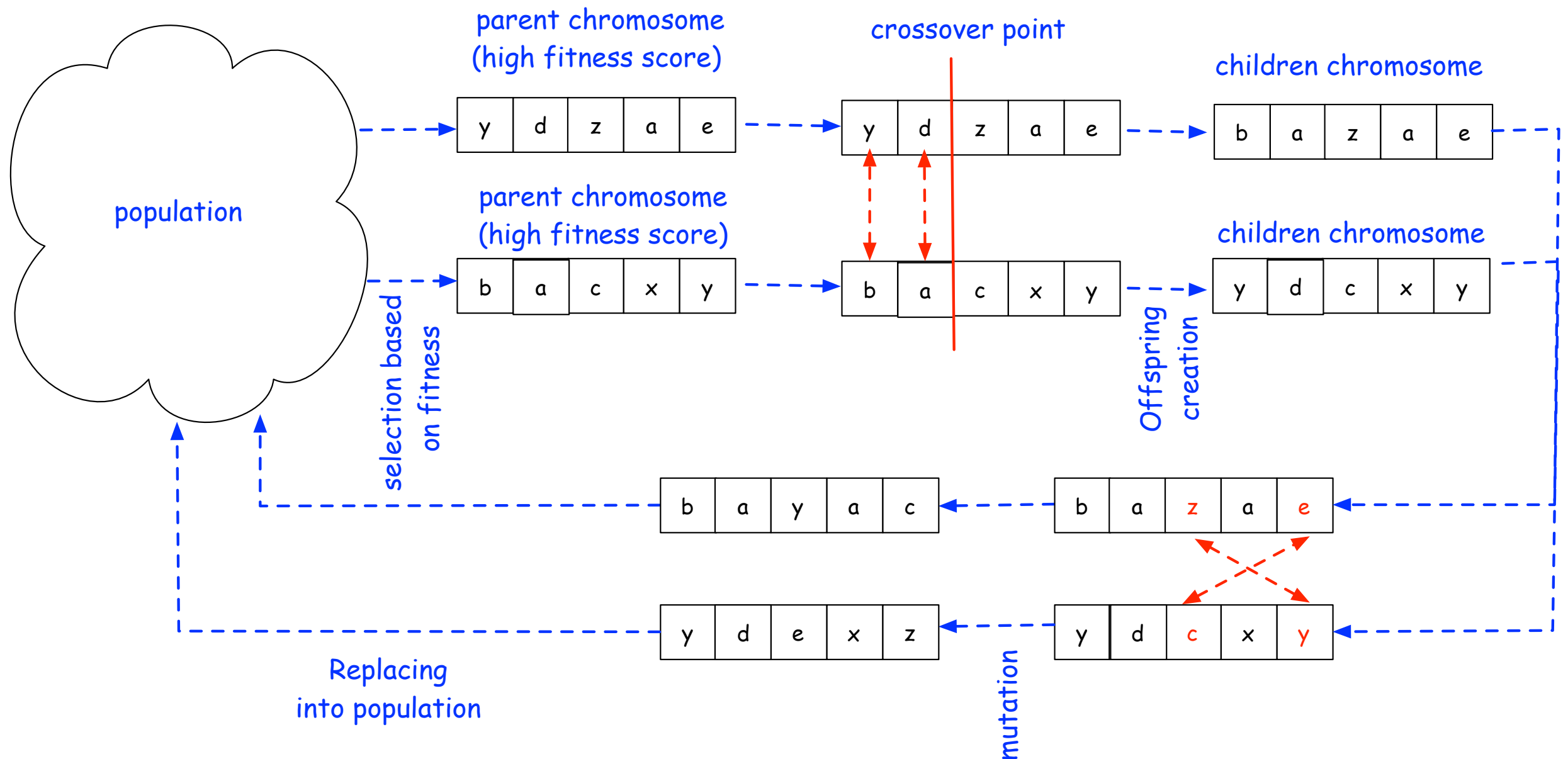
population

| a | y | d | z |

| z | b | y | x |

→ gene

→ allele

| x | b | a | c |

| d | c | b | y |

| z | a | x | d |

. . .

chromosome

Genetic Algorithm

Two chromosome with high fitness score will be selected and based on the given crossover point two children chromosome will be created. Then **mutation will be done to increase their diversity**, and newly generated chromosomes will be replaced low fitness score chromosomes in the population.
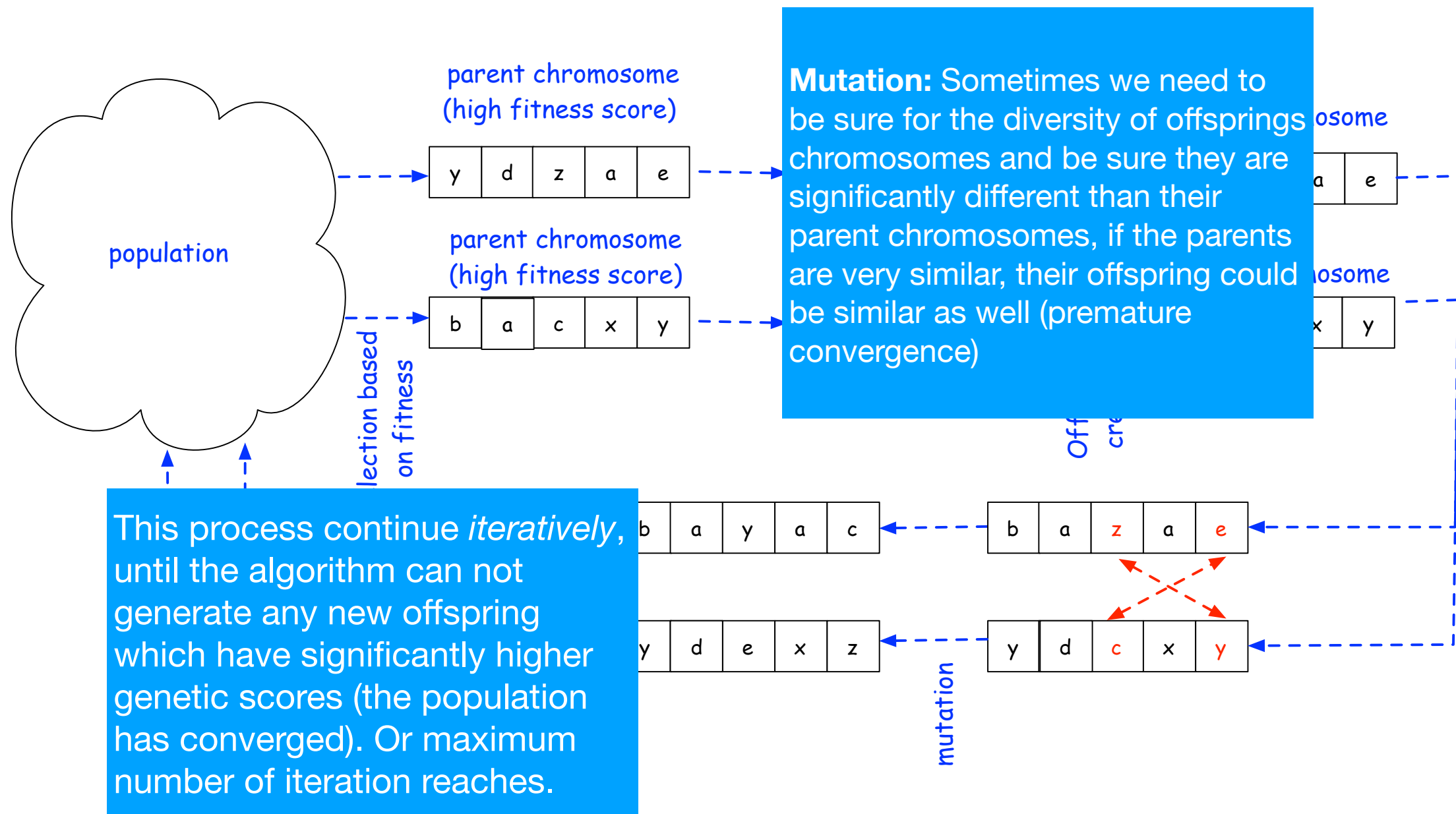
Genetic Algorithm

population

parent chromosome
(high fitness score)

| y | d | z | a | e |

parent chromosome
(high fitness score)

| b | a | c | x | y |

selection based on fitness

Mutation: Sometimes we need to be sure for the diversity of offsprings chromosomes and be sure they are significantly different than their parent chromosomes, if the parents are very similar, their offspring could be similar as well (premature convergence)

osome

| a | e |

osome

| x | y |

This process continue *iteratively*, until the algorithm can not generate any new offspring which have significantly higher genetic scores (the population has converged). Or maximum number of iteration reaches.

| b | a | y | a | c | ← | b | a | z | a | e |

| y | d | e | x | z | ← | y | d | c | x | y |

mutation

Two chromosome with high fitness score will be selected and based on the given crossover point two children chromosome will be created. Then mutation will be done to increase their diversity, and newly generated chromosomes will be replaced low fitness score chromosomes in the population.

Genetic Algorithm Example

- https://twitter.com/i/status/1182033107668496384

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- **Feature Generation**

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Feature Generation

- Feature generation is the process of combining, mixing, merging, … two or more raw data objects and create a new data object from them.

- Why do we do that? Because of reducing the computational cost, while maintaining or even improving the accuracy of the data.

- E.g. Fitness tracker and three axis data, X, Y, Z …

- A very simple feature generation method multiply all features together, **interaction features** or **polynomial feature creation**. For instance, if we have following features: x_1, x_2, x_3, x_4

Their interaction feature could be as follows:

$$x_1^2, x_1 . x_2, x_1 . x_3, x_1 . x_4, x_2^2, x_2 . x_3, x_2 . x_4, x_3^2, x_3 . x_4, x_4^2 .$$

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- Feature Generation

- **Feature Engineering for Numerical Data**

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Feature Engineering for Numerical Data

- **Sanity check** checks whether the data is in proper form. For example, it checks whether there is a negative value, missing values, etc.

- **Normalizing** is the process of converting data objects from different format into a comparable and unit-independent format. The normalization could also include identifying the dataset's maximum and minimum boundaries, binning or quantizing the data as well.

- **Quantizing** the data or **binning** the data means substituting the original values of the data with a discrete and more generalized value. For example, we can quantize the "time of the day" into four bins as follows:
    * Original data: {00:00, 01:00 , 02:00, … 23:59 } => Quantized data: {mid night, morning, afternoon, evening, night}
    * Quantization in the context of neural network is bringing the data from float-32 to smaller range such as int-8.

- **Discretizing** the data is converting continuous data into a discrete unit of data. For example, 11.22 (float) —> 11 (integer), 13.3462 (float)—> 13 (int)

Scaling Numerical Data

- While working with ML algorithm we always have the challenge to deal with computer resources, and to mitigate this challenge one solution is to scale data, e.g. getting rid of large floating numbers.

(1) **Min-Max scaling** is used to compress (squeeze) or stretch feature values. Assume we have x, which is vector of variable its Min-Max scaling factorial be shown as $\tilde{x}$.
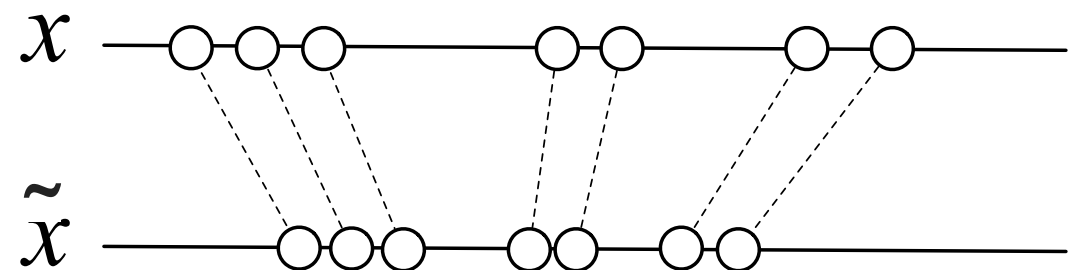
$$\tilde{x}_i = \frac{x_i - min(x)}{max(x) - min(x)}$$
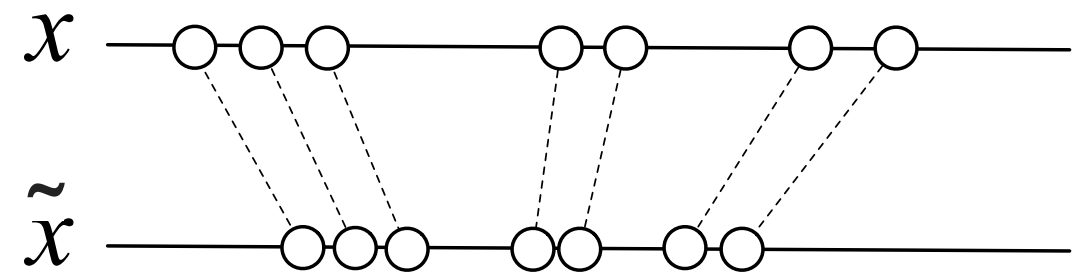
Scaling Numerical Data

(1) **Min-Max scaling** is used to compress (squeeze) or stretch feature values. Assume we have x, which is vector of variable its Min-Max scaling factorial be shown as $\tilde{x}$.

$$\tilde{x} = \frac{x - min(x)}{max(x) - min(x)}$$

(2) **Standardization** or **variance scaling** or **z-score**

$$\tilde{x} = \frac{x - \bar{x}^{\text{ mean}}}{\sigma_{\text{ sd}}}$$

(3) **L²-norm** or **Euclidean norm**

$$||x||_2 = \sqrt{x_1^2 + x_2^2 + \ldots x_n^2}$$

L¹-norm

$$||x||_1 = x_1 + x_2 + \ldots x_n$$

$$\tilde{x} = \frac{x}{||x||_2}$$

Scaling Numerical Data in R

```
#---- Scaling in R -----

# Load sample dataset
data(iris)
# View sample matrix
a = iris$Sepal.Width
# Before min-max normalization
a
#Normalize a with min-max function
scale(a, center = TRUE, scale = TRUE)

# normalizing the content of a matrix with R default scaling package.
#
x <- matrix(1:10, ncol = 2)
x
scale(x)

#------- min-max scaling ——
minmax <- function(x) {
  return((x- min(x)) /(max(x)-min(x)))
}

samplevec = c(-23,2,34,12,11,20,0)
minmax(samplevec)

#--------norms------
#Lp=1 norm
lpnorm = norm(x, type = c("O")) # one norm, (maximum absolute column sum)
lpnorm


lpnorm = norm(x, type = c("2")) #  "spectral" or 2-norm (largest singular value)
lpnorm
```

Transformation

Original   Result
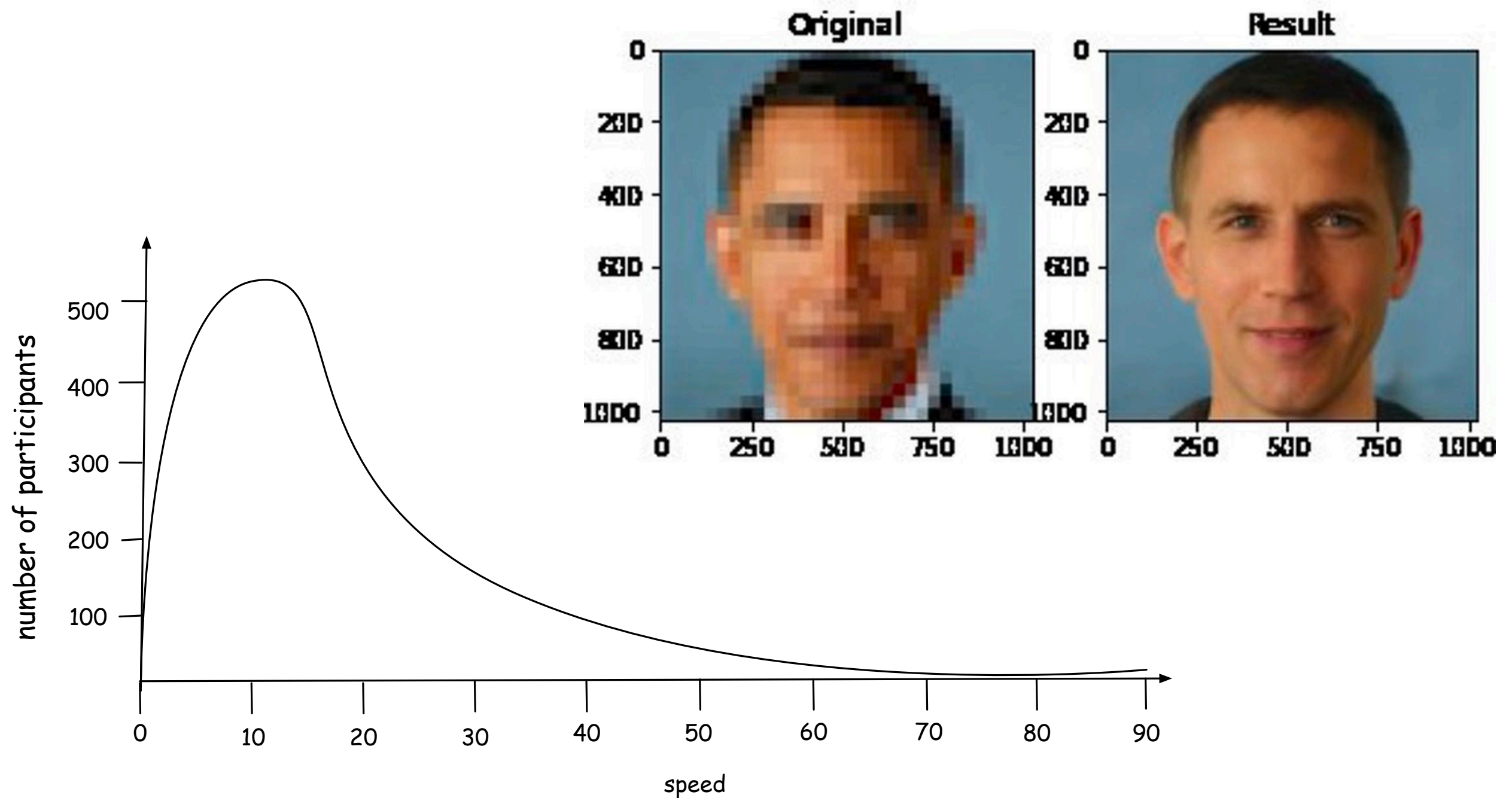
Transformation

(1) **Logarithm Transformation**
It is used to reduce the emphasis on high density data and increase the emphasize on low density data.

(2) **Box-Cox Transformation** It trying to convert the shape of non-normal distribution into a normal distribution.

$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda} & if \quad \lambda \neq 0 \\ log(y) & if \quad \lambda = 0 \end{cases}$$

The value of λ varies between -5 and +5.

Review Counts Histogram

Log Transformed Counts Histogram

Box-Cox Transformed Counts Histogram
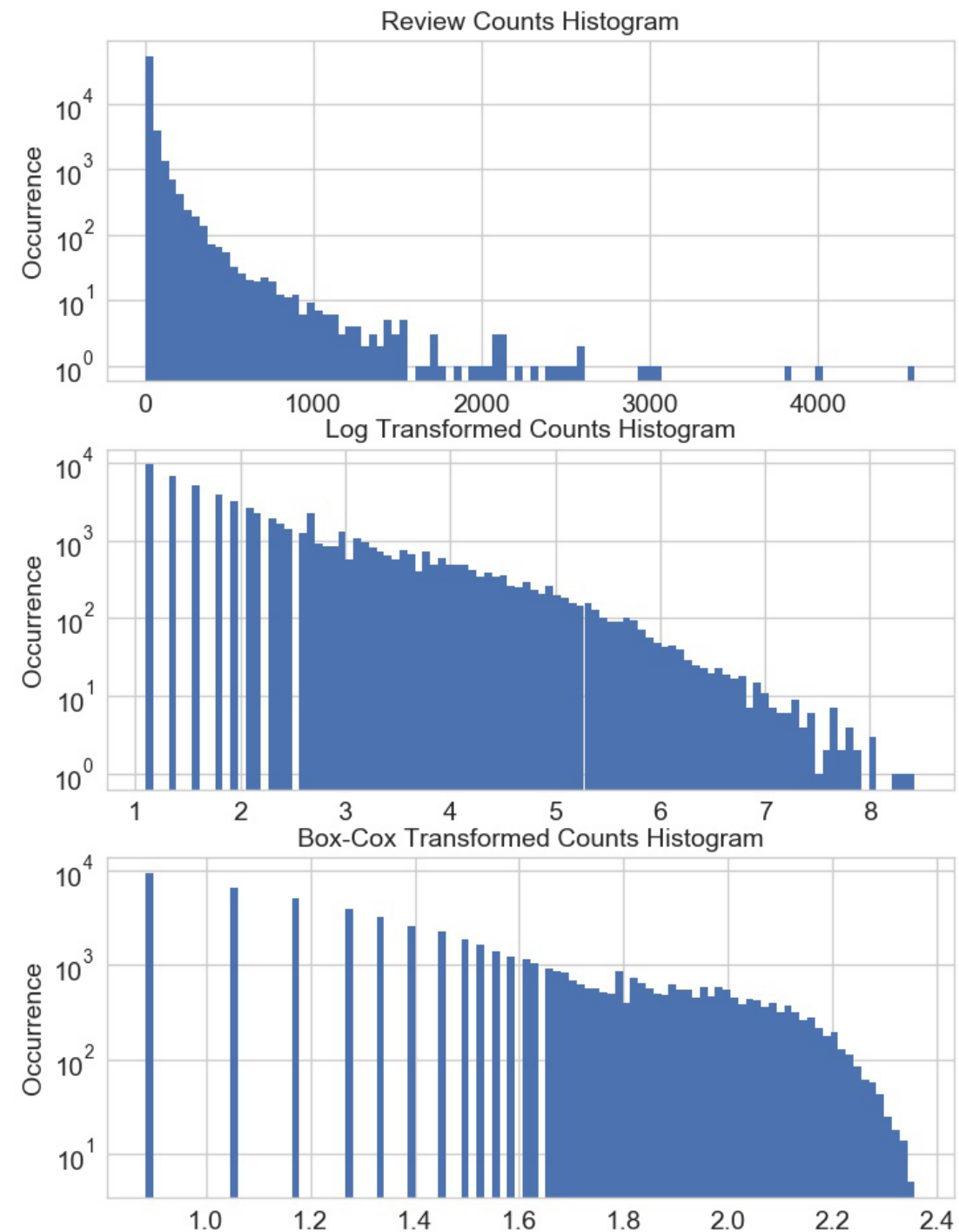
Different transformation of the data in Yelp Reviews.

Yelp is similar to 大众点评 Dàzhòng Diǎnpíng

Some Notes About Transformation

*Both **Box-cox** and **log** transform operate when the data is **positive**. If you are dealing with a dataset that includes negative values as well, you can add a fixed number (constant) to all numbers (the number could be equal to the positive value of the lowest number) and to all data points and make all your data objects positive.

*The logarithmic transform and box-cox transform are both members of a group called **power-transform**. These transformations operate based on "stabilizing the variance" of the dataset. In other words, power transformation changes the distribution so that variance is *no longer dependent on the mean.*

*Feature selection can increase the bias as well. Usually, in deep learning algorithms, there will be no feature selection, and we give all features to the model. Then, the algorithm sorts out the useful features itself. However, we do not advocate generalizing a method that works for deep learning algorithms to all other algorithms. Especially for battery-powered devices with energy limitations, having to check too many features is a major challenge.

*Some literature suggests using transformation to change the shape of data into the normal distribution and thus add more flexibility to statistical analysis.

Transforming Numerical Data in R

```r
#---------- Logarithm Transformation
a <- c(190,220,210,200,150,4,5,6,2,3,1,1)
plot(a, pch =21, cex=0.9, col="blue", bg="blue")

a2 <- log10(a)
a2
plot(a2, pch =21, cex=0.9, col="blue", bg="blue")

#---------- box-cox transformation
library("car")

b1 = bcPower(1:10, 2) # lambda = 1 means the original dataset
plot(b1,  pch =21, cex=0.9, col="blue", bg="blue")

b2 <- bcPower(a,3) # experiment with different lambda
plot(b2,  pch =21, cex=0.9, col="blue", bg="blue")
```

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- Feature Generation

- Feature Engineering for Numerical Data

- **Feature Engineering for Categorical Data**

- Feature Engineering for Textual Data

Feature Engineering for Categorical Data

(1) One-hot Encoding: It converts any possible values of a dataset into a bit (0 or 1).

Ultra fat:	1, 0, 0, 0
fat:	0, 1, 0, 0
fit:	0, 0, 1, 0
thin:	0, 0, 0, 1

(2) Dummy Coding: It is very similar to One-hot Encoding with one difference, but instead of using k bits to present the k numbers of data points, it uses $k-1$ bits to present data.

Ultra fat:	1, 0, 0
fat:	0, 1, 0
fit:	0, 0, 1
thin:	0, 0, 0

(3) Effect Coding: It is similar to dummy coding, but the reference category (all 0 bits) are presented with -1. This is very similar to Dummy coding, but it makes results of <u>linear regression</u> more easy to interpret.

Ultra fat:	1, 0, 0
fat:	0, 1, 0
fit:	0, 0, 1
thin:	-1, -1, -1

(4) Feature Hashing: It maps each feature into a hash value.

"the cat on the roof"	00a9
"the cat in the bin"	01xb
"the chicken who eats the cat"	82n4
"the chicken who knew how to deal with cats"	z12d

Assuming n is the number of hash keys its computational complexity is $O(\frac{1}{\sqrt{n}})$

Feature Engineering for Categorical Data

(5) Bin Counting: It converts the target data points into a <u>set of probabilities</u>. Therefore, instead of dealing with many different features, the algorithm deals with a set of probabilities. These probabilities were extracted by analyzing historical data. In other words, the attribute or feature will be converted into a likelihood.

machine ID	input	 	#chicken nugget	#cat	#elephant	#burgers
1	3200		3000	200	0	0
2	5631		5000	-	31	-
3	2000		1500	200	-	-
4	1500		1000	100	200	100

Performance of four machines which receive chickens as input and should deliver nuggets as output. Other outputs are assumed to be incorrect.

machine ID	Correct output	Incorrect output
1	3000÷3200=0.93	200÷3200=0.06
2	5000÷5631=0.89	31÷5631=0.005
3	1500÷2000=0.75	200÷2000=0.1
4	1000÷1500=0.66	(100+200+100)÷1500=0.267

Performance of four machines based on the likelihood of correct or incorrect output.

One-hot Encoding in Python

```python
#------- One hot coding ---------

from numpy import array
from numpy import argmax
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

import pandas as pd

# define example
sampledata = ['chicken', 'chicken', 'cat', 'chicken', 'dog', 'dog', 'horse',
'chicken', 'dog', 'horse']


### integer mapping using LabelEncoder
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(sampledata)
print(integer_encoded)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)

### One hot encoding
onehot_encoder = OneHotEncoder(sparse=False)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

print(onehot_encoded)
```

Dummy Coding in Python

```python
import pandas as pd
df=pd.DataFrame(data=sampledata,
columns=['dummy'])
dummies = pd.get_dummies(df['dummy'])

print("dummies:")
print(dummies)
```

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- Feature Generation

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- Feature Engineering for Textual Data

Outline

- Feature Engineering Concepts

- Feature Selection Methods

- Feature Generation

- Feature Engineering for Numerical Data

- Feature Engineering for Categorical Data

- **Feature Engineering for Textual Data**

Feature Engineering for Textual Data

- A large segment of data community is dedicated to working with textual data and extracting knowledge from textual corpus, such as getting data from online social media, online news media, HTML pages, books and articles, recorded voices of human conversations, transcript of movies, etc.

- There are several types of tasks done with analyzing textual data, including **classification**, **clustering, sentiment analysis, entity recognition,** and **text generation**.

- Example: classifying emails based on their emotional tone in customer support.

- "I am not rich, but I am a true lover of your product. I wish I have enough money to buy all of the existing colors. Every night I dream about your product, which has one in the kitchen, one in the office, and one in each room.

How to feed the Text into the Algorithm

- Bag-of-words

- nGrams

- Part-of-Speech tagging

- WordEmbedding

Bag-of-words

- **Bag-of-words:** This method treats the tokenized document (e.g. a sentence or a web page, or a paragraph or…) as a list of words. It does not care about their position in the sentence. For example, bag-of-words method does not distinguish between following two lists:

  - there, book, luxury, hotel, eat, break fast, overview, mountains, island.
  - overview, mountains, island, there, book, luxury, hotel, eat, break fast.

- Then it transforms the token into a list of words and their frequency. For example, 'I am not loser, I am amazing, I am very good, I will change the world', will be fed into a bag-of-words algorithm and following is the algorithm output:

  I = 4, am = 3, will = 1, loser = 1 …

Subword Tokenization

- By looking at space between words in a text, we can tokenize a text easily into words, but some languages, such as Chinese, Japanese, and Korean do not have space between their words, and therefore, we need another approach to performing tokenization.

- Subword tokenization mitigates this challenge. For example, the word 'education' will be broken down into 'edu', 'cat', 'ion'.

- Subword tokenization can also handle out-of-vocabulary (OOV) words in languages that use spaces. By breaking down words into smaller units, models can better handle rare or unknown words by understanding their sub-components.

- Subword tokenization can be used even on the character level, and despite its extreme resource utilization, it works with acceptable accuracy for NLP tasks with neural networks. However, character level tokenization is too inefficient from a resource usage perspective.

- A well-known approach for subword tokenization is **Byte Pair Encoding (BPE)**

nGram

- Bag-of-words is very useful method, but it doesn't care about the sequence of words at all. For example, 'I am not losing...', is a sentence that is saying something positive about the subject, but if we treat it as a bag-of-words it does consider 'losing' as a single term and 'not' as another term. Most probably, the algorithm interprets it as a negative sentence, which is incorrect.

- N Gram is similar to bag of words, but it separates words based on their adjacent words. For example, a 2 Gram (bi-Gram) of '*I am not loosing.*' will be '*I am*', '*am not*' and '*not losing*', respectively its 3 Gram (tri-Gram) will be '*I am not*' and '*am not losing*'.

Part of Speech

The part of speech tagging (PoS) will annotate this sentence as follows.

I am very happy chicken and I love myself.

pronoun
preposition
adverb
adjective
noun
conjunction
pronoun
verb
pronoun
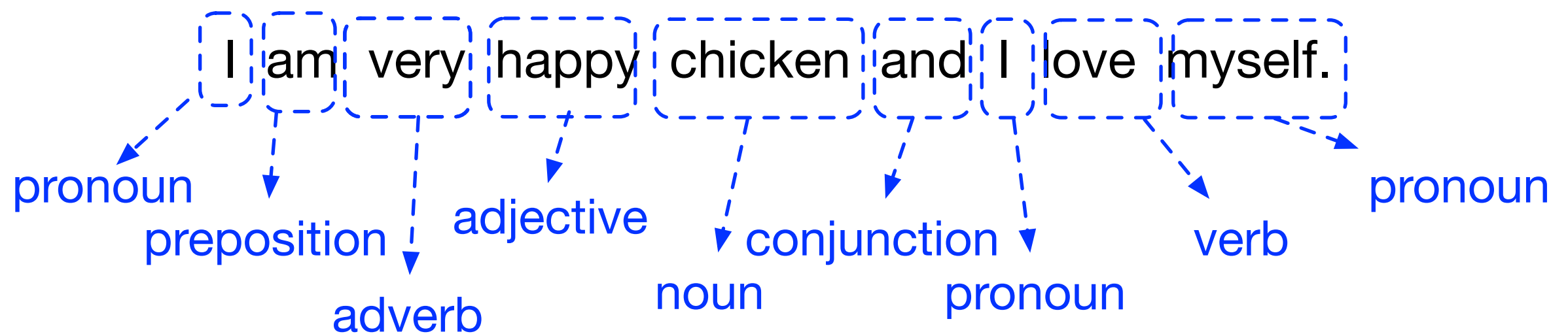
There the algorithm will output a list of two tuples, the word in the sentence and its role. such as `<I, pronoun>`, `<am, preposition>`, …
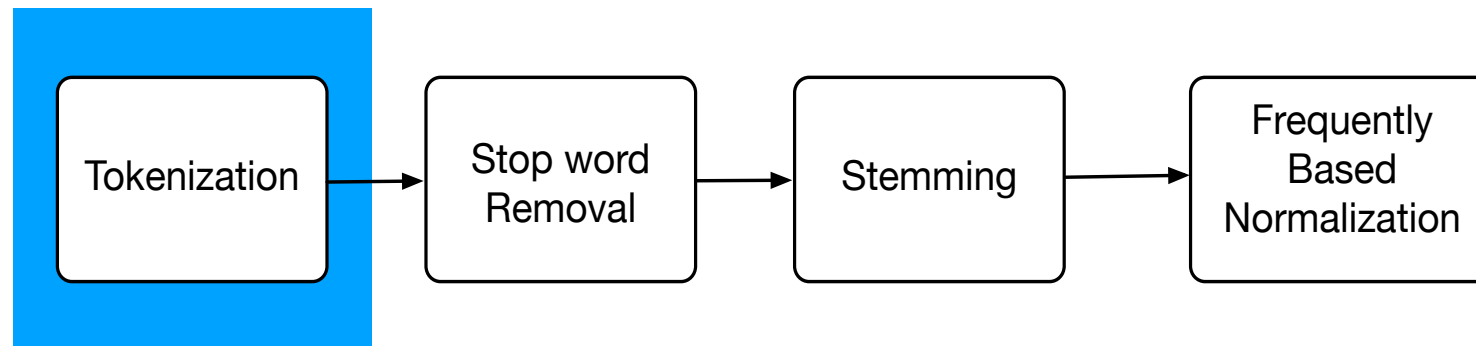
- *I am not rich, but I am a true lover of your product. I wish I had enough money to buy all of the existing colors. Every night I dream about your product, which has one in the kitchen, one in the office, and one in each room.*

Tokenized version:

1. *I am not rich, but I am a true lover of your product.*

2. *I wish I had enough money to buy all of the existing colors.*

3. *Every night I dream about your product, which has one in the kitchen, one in the office, and one in each room.*
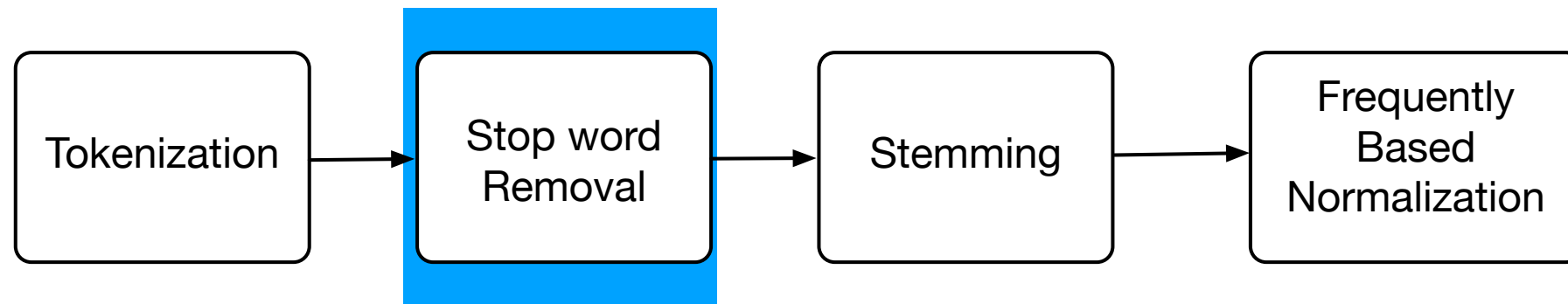
1. I am not rich, but I am a true lover of your product.

2. I wish I had enough money to buy all of the existing colors.

3. Every night I dream about your product, which has one in the kitchen, one in the office, and one in each room.

Stop word removal (”articles“, ”prepositions“, ”conjunctions“ and sometimes “pronouns“ are considered stop words):

1. not, rich, but, true, lover, product.

2. wish, had, enough, money, buy, all, existing, colors.

3. Every, night, dream, about, product, have, one, kitchen, office, each, room.

```
┌──────────────┐      ┌──────────────┐      ┌──────────────┐      ┌──────────────┐
│              │      │              │      │              │      │  Frequently  │
│ Tokenization │ ───▶ │  Stop word   │ ───▶ │   Stemming   │ ───▶ │    Based     │
│              │      │   Removal    │      │              │      │ Normalization│
└──────────────┘      └──────────────┘      └──────────────┘      └──────────────┘
```
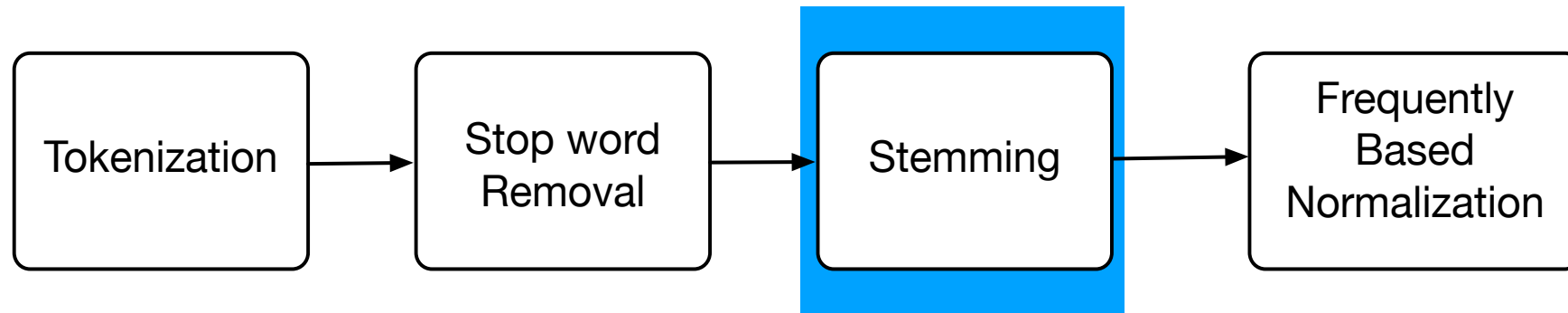
Stemming is converting words into their original root

- had —> have

- humans —> human

- swimming —> swim

After stemming the document/term matrix could be constructed as well. But a normalization step is highly recommended.

```
┌──────────────┐     ┌──────────────┐     ┌──────────────┐     ┌──────────────┐
│              │     │   Stop word  │     │              │     │  Frequently  │
│ Tokenization │ ──▶ │   Removal    │ ──▶ │   Stemming   │ ──▶ │    Based     │
│              │     │              │     │              │     │ Normalization│
└──────────────┘     └──────────────┘     └──────────────┘     └──────────────┘
```
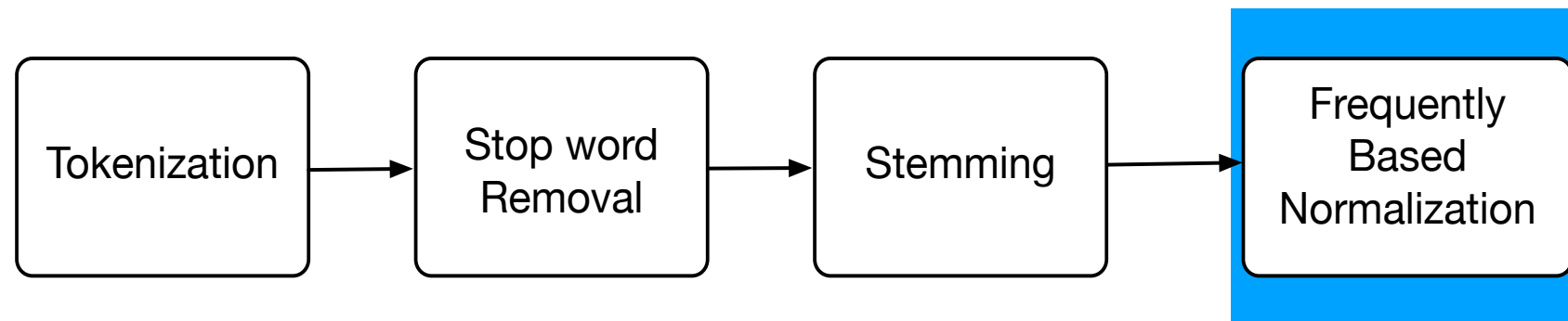
A widely used text normalization is Term Frequency/ Inverse Document Frequency (TF/IDF) which is a **vector space based-representations** normalization. TF/IDF evaluates how important the word is in the document. TF measures the **frequency of a term (word) in a document**, which means the number of target terms, divided by all terms in that document.

$$TF = \frac{\#\ term\ appearance\ in\ the\ document}{\#\ total\ terms\ in\ the\ document}$$

Nevertheless, the size of document could be large or small and this affect the importance of a term. IDF measures **how rare the term is in the target document.** It will be written as follows:

$$IDF = log_e(\frac{\#\ total\ documents}{\#documents\ which\ includes\ the\ term})$$

Therefore, TF/IDF will be written as a product of TF and IDF, i.e. $TF \times IDF$

Text Feature Engineering in R

```
┌─────────────┐     ┌─────────────┐     ┌─────────────┐     ┌─────────────┐
│             │     │             │     │             │     │ Frequently  │
│ Tokenization│ --> │  Stop word  │ --> │  Stemming   │ --> │    Based    │
│             │     │   Removal   │     │             │     │Normalization│
└─────────────┘     └─────────────┘     └─────────────┘     └─────────────┘
```
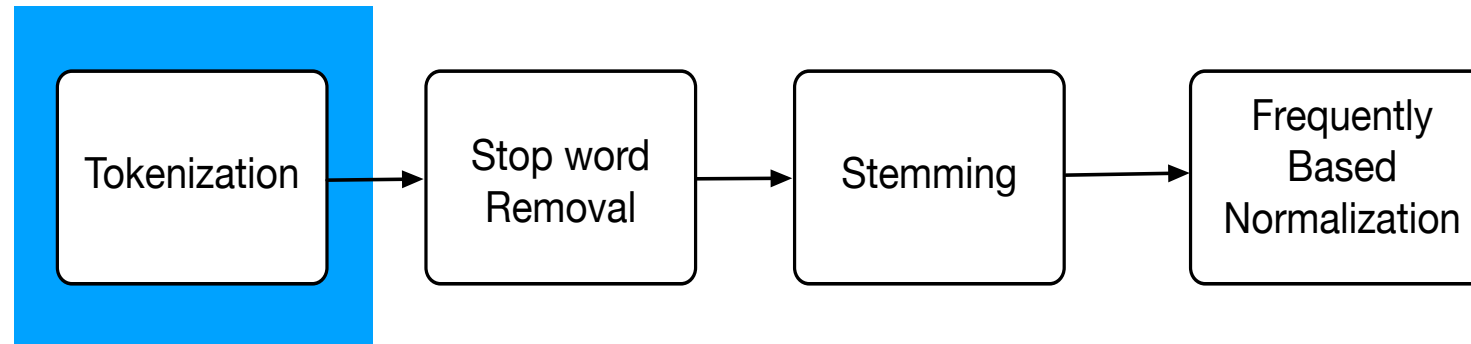
```r
install.packages('tokenizers')

library('tokenizers')

rawtxt <- "Hello Mr. Smith, how are you doing today? The weather is great, and city is
awesome. The sky is pinkish-blue. Seeing the sky is amazing. You shouldn't eat
cardboard"

tktxt <- tokenize_sentences(rawtxt, lowercase= FALSE, strip_punct= FALSE, simplify=
FALSE)
tktxt
class(tktxt)

tktxt2 <- tokenize_lines(rawtxt, simplify = FALSE)
tktxt2

tktxt3 <- tokenize_regex(rawtxt, pattern = "\\s+", simplify = FALSE) # \\s+ means space
tktxt3
```

Text Feature Engineering in R

```
┌──────────────┐   ┌──────────────┐   ┌──────────────┐   ┌──────────────┐
│              │   │  Stop word   │   │              │   │  Frequently  │
│ Tokenization │──▶│   Removal    │──▶│   Stemming   │──▶│    Based     │
│              │   │              │   │              │   │ Normalization│
└──────────────┘   └──────────────┘   └──────────────┘   └──────────────┘
```
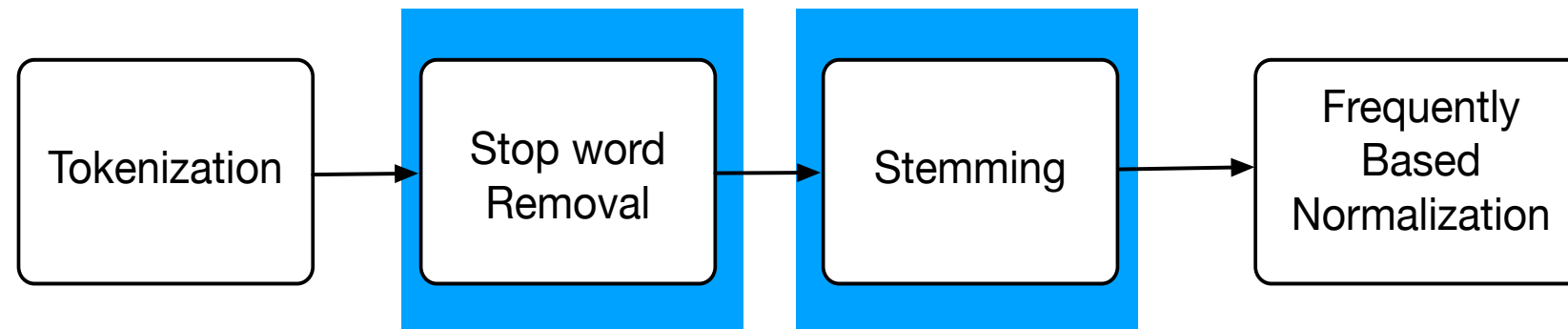
```
install.packages('tm')
library('tm')

rawtxt <- "Hello Mr. Smith, how are you doing today? The weather is great,
and city is awesome. The sky is pinkish-blue. Seeing the sky is amazing. You
shouldn't eat cardboard"


corp1 <- Corpus(VectorSource(rawtxt))
inspect(corp1)

# remove stop words
nostoptxt <- tm_map(corp1, removeWords, stopwords("english"))
inspect(nostoptxt)

# Stemming
stemmed <- tm_map(nostoptxt, stemDocument)
inspect(stemmed)
```
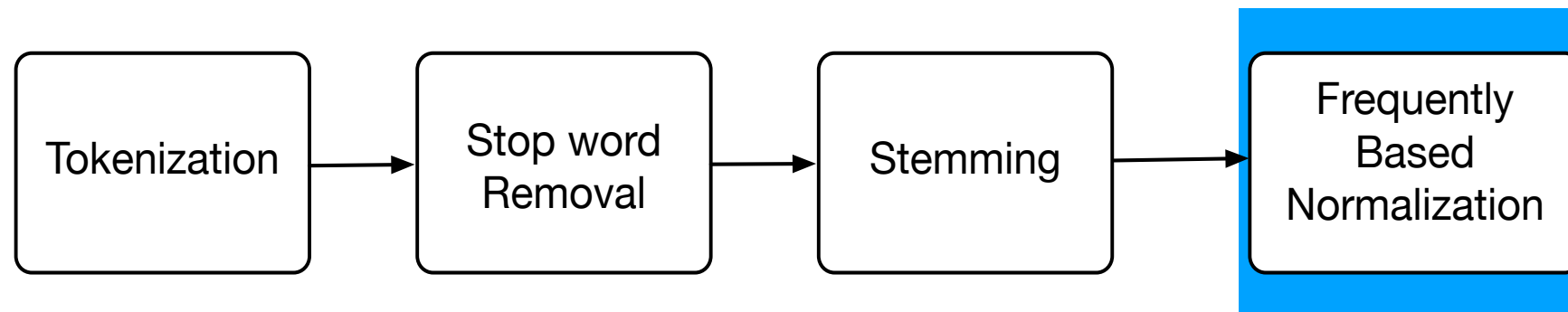
Text Feature Engineering in R

```
┌──────────────┐     ┌──────────────┐     ┌──────────────┐     ┌──────────────┐
│              │     │              │     │              │     │  Frequently  │
│ Tokenization │────▶│  Stop word   │────▶│   Stemming   │────▶│    Based     │
│              │     │   Removal    │     │              │     │ Normalization│
└──────────────┘     └──────────────┘     └──────────────┘     └──────────────┘
```

Step 1: Create a Document/Term Matrix from existing documents

```
txt <- c("banana is a fruit, which is yellow",
         "banana is the tastiest fruit in the world",
         "banana tastes good",
         "banana looks like a yellow cucumber... ha ha ha yellow",
"there are pink banana and blue banana as well as yellow")

corp2 <- Corpus(VectorSource(txt))
#--- first lets create a document term matrix
tdmatrix <- DocumentTermMatrix(corp2)
inspect(tdmatrix)
```

Text Feature Engineering in R

```
┌──────────────┐     ┌──────────────┐     ┌──────────────┐     ┌──────────────┐
│              │     │              │     │              │     │  Frequently  │
│ Tokenization │ ──> │  Stop word   │ ──> │   Stemming   │ ──> │    Based     │
│              │     │   Removal    │     │              │     │ Normalization│
└──────────────┘     └──────────────┘     └──────────────┘     └──────────────┘
```
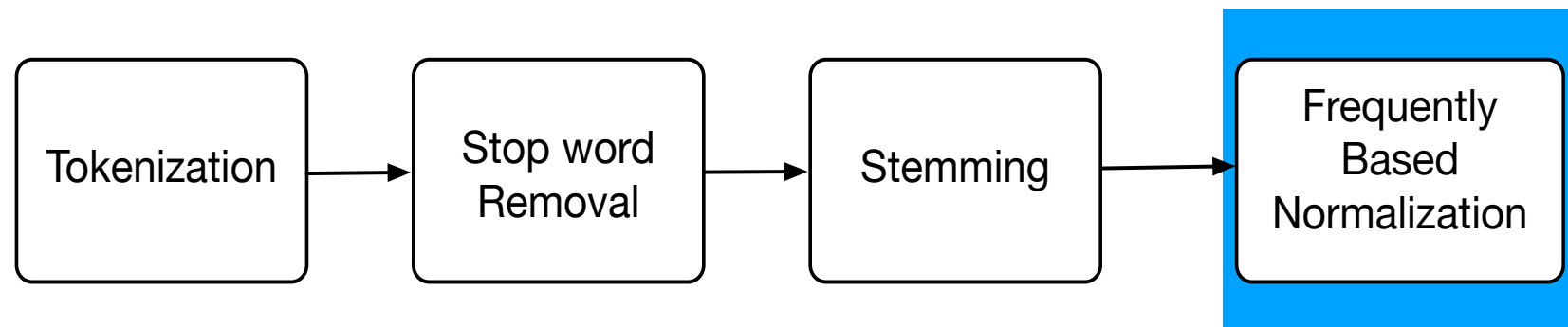
Step 2: Create a TF/IDF for that particular matrix

```r
# calcualte tf
tf <- as.matrix(tdmatrix)
# calcualte idf
idf <- log( ncol(tf) / ( 1 + rowSums(tf != 0) ) )
idf
# transform idf into a diagonal matrix to be able to multiply it by TF
idf <- diag(idf)
idf

tf_idf <- crossprod(tf, idf)
colnames(tf_idf) <- rownames(tf)
tf_idf
```

Text Feature Engineering in R

Terms	1	2	3	4	5
banana	1.386294	1.049822	1.609438	1.049822	1.5970154
fruit	1.386294	1.049822	0.000000	0.000000	0.0000000
which	1.386294	0.000000	0.000000	0.000000	0.0000000
yellow	1.386294	0.000000	0.000000	2.099644	0.7985077
most	0.000000	1.049822	0.000000	0.000000	0.0000000
tastiest	0.000000	1.049822	0.000000	0.000000	0.0000000
the	0.000000	2.099644	0.000000	0.000000	0.0000000
world	0.000000	1.049822	0.000000	0.000000	0.0000000
good	0.000000	0.000000	1.609438	0.000000	0.0000000
tastes	0.000000	0.000000	1.609438	0.000000	0.0000000
...	0.000000	0.000000	0.000000	1.049822	0.0000000
cucumber	0.000000	0.000000	0.000000	1.049822	0.0000000
like	0.000000	0.000000	0.000000	1.049822	0.0000000
looks	0.000000	0.000000	0.000000	1.049822	0.0000000
and	0.000000	0.000000	0.000000	0.000000	0.7985077
are	0.000000	0.000000	0.000000	0.000000	0.7985077
blue	0.000000	0.000000	0.000000	0.000000	0.7985077
pink	0.000000	0.000000	0.000000	0.000000	0.7985077
there	0.000000	0.000000	0.000000	0.000000	0.7985077
well	0.000000	0.000000	0.000000	0.000000	0.7985077

Step 2

```
# ca
tf <
# ca
idf
idf
# tr
idf
idf

tf_i
coln
tf_i
```

Word Embedding

- Previous representations we have explained can not recognize synonyms, without having a dictionary that 'Good' and 'Nice' belong to the same category of words.

- Example:

  - *Mr. Smith is a nice colleague, but he always wears a tie at work, nowadays who is wearing a tie? His boss also wears a tie.*

- Word embedding, approaches try to _identify the role of the word based on its context in the document_. Referring to a word based on its context is a very useful feature, which is not possible in other approaches.

Word Embedding

- Word embedding algorithms can also establishes an accurate association between words, e.g. "chicken" is associated to "bird" or "boy" is associated to "man".

- By converting a text into numerical vector we could perform numerical comparison between words as well. For example, we can compare chicken ID2345 with another chicken in the text as well.

- We can perform addition or subtraction on vectors using Euclidean, cosine, or other distances. For example, 'king - man + woman = queen', 'Syria - Damascus = Iraq - Baghdad', or 'Syria - Damascus + Baghdad = Iraq'.
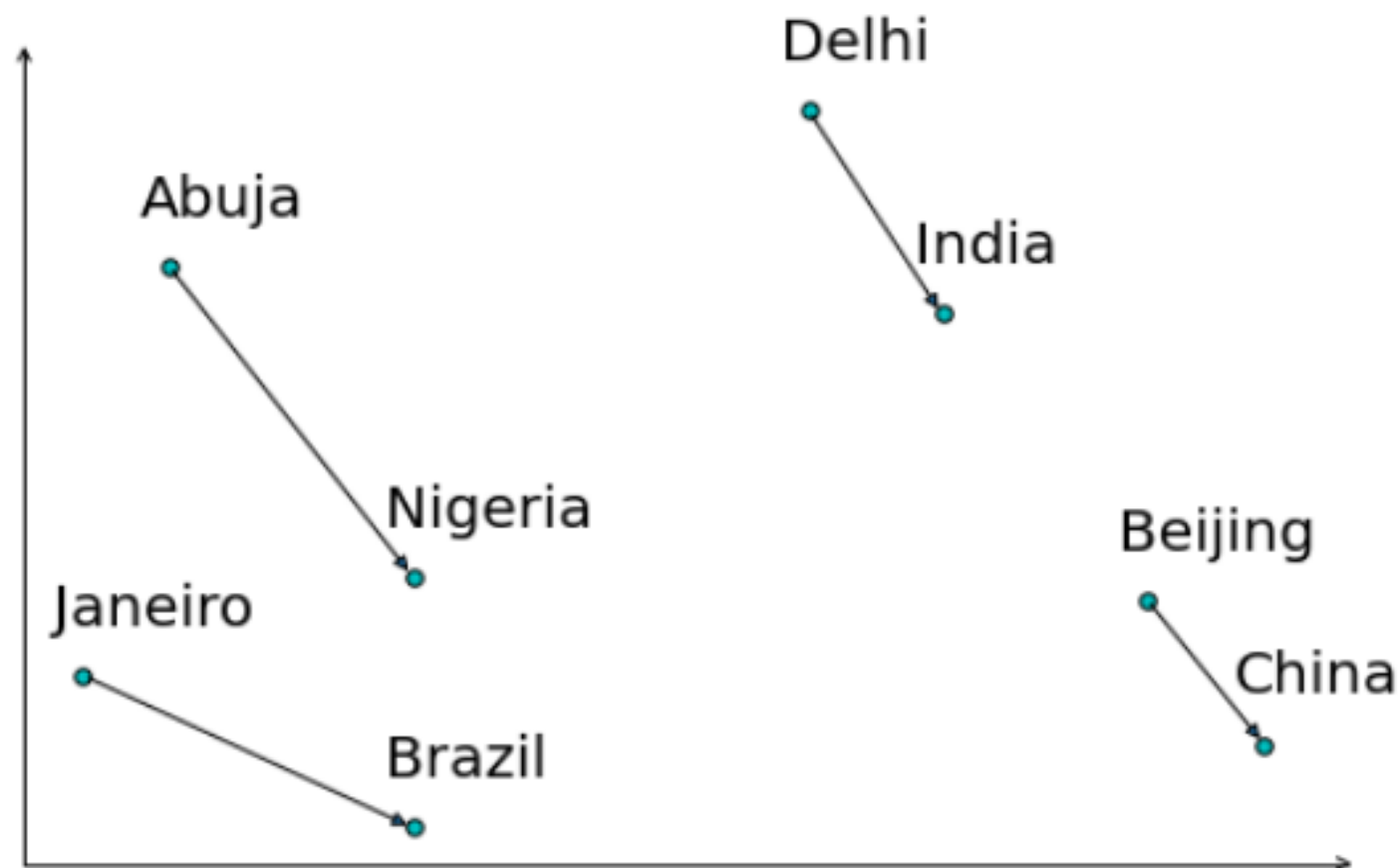
Word as Vector

chicken = [0.9, 0.8, 0.01, 0.02, 0.01]

bird = [0.79, 0.91, 0.02, 0.0, 0.01]

boy = [0.0, 0.01,0.0, 0.01,0.2]

Man = [0.01, 0.05, 0.01, 0.02, 0.4]

Word Embedding visualization in 2D space

Language Model

- Word embedding algorithm works by building a *language model*.

- Training a language model refers to the process of giving a *large text to an algorithm and the algorithms builds a model based on the given text*.

- For example, we can give a book, Wikipedia or any other large corpus of textual data into an algorithm to train the language model.

Building a Language Model for Autocomplete

- A very simple approach to constructing a language model is by moving a sliding window over the input text.

- The algorithm starts by using a sliding window and moves it along the text. The content of each window is the training instances used to construct the model. The next word will be the label of the instance. For example, assuming the window size is three sentences: *'The wound is the place where the light enters you.'* will be converted into a language model as:

Instance: 'the wound is', label: 'the'

Instance: 'is the place', label: 'where'
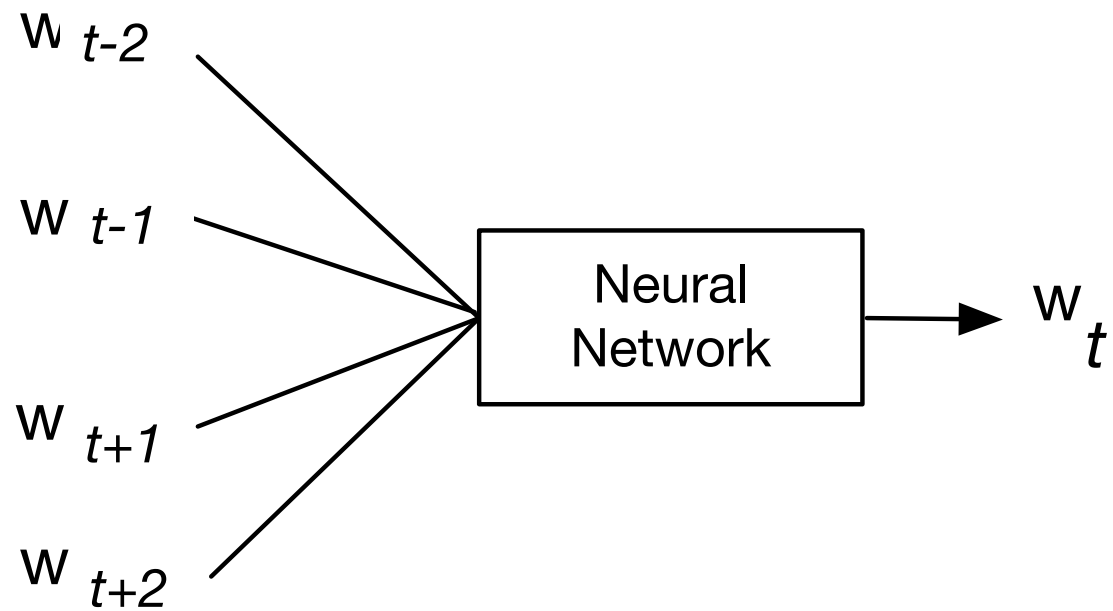
…

Back to Word Embedding

- The basic idea of word embedding is to <u>identify the context</u> (c) by a vector of input words, i.e. $\{w_1, w_2, \ldots w_t\}$, which can be formalized as $p(c \mid w_t)$, and this probability will be estimated by a neural network.

- In other words, the objective of this neural network is to <u>predict a center word based on the given input words that describe the context (Continuous BoW)</u>. Or <u>predict context words by given center words (Skip Gram).</u>

- For example, consider this sentence: 'The white cow is heavy'. Using the 'cow' as the input center word the algorithm can predict the neighbor words. Or using the neighbor words as context words the word embedding algorithm can predict the word 'cow'.

  - Using a <u>context to predict words</u> to that context, i.e. **Continuous Bag of Words (CBOW)**

  - Using the <u>word to predict the context</u> **(Skip-Gram)**.
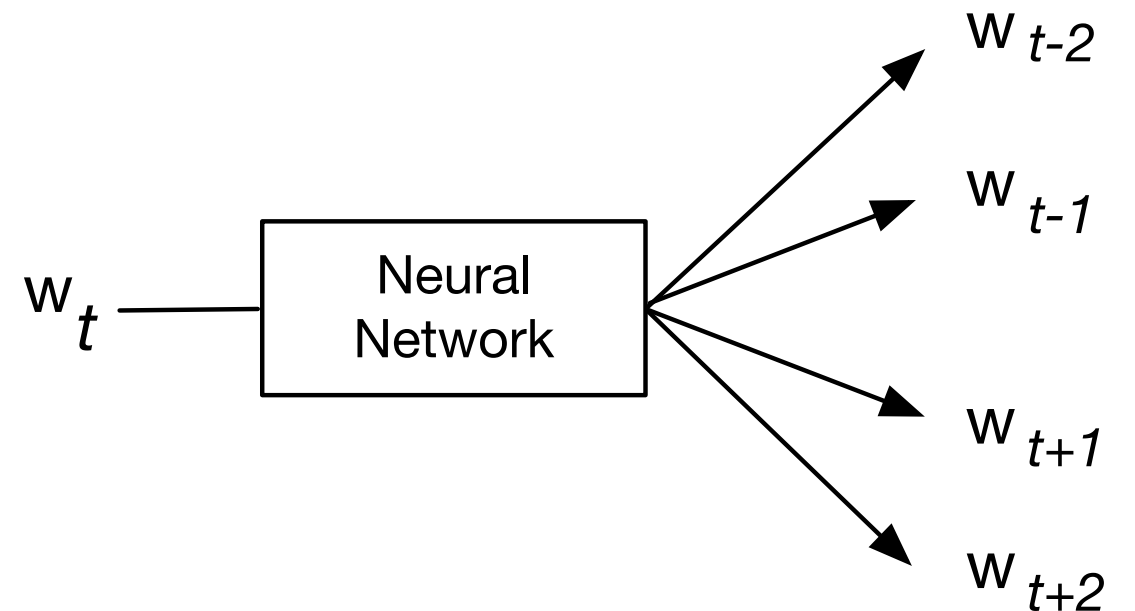
CBOW & Skip-Gram

- Assume we give the input (context) 'Today is ____ and the weather is also very good', CBOW can predict the missing word is 'amazing'.

  - The missing word is selected from a set of words (e.g. {amazing, probability =0.8}, {snowy, probability = 0.03}, …) and each of them has a probability, the word "amazing" has the highest probability and thus it is selected by CBOW to be returned as the predicted word.

- Skip-Gram is reverse to CBOW. It finds the context words for the given target word.

  - For example, giving the context word "cow" the skip-gram can find the surrounding words such as 'heavy', 'white', etc. Because, there are sentences in the text corpus such as 'The big white cow is heavy to move'. Skip-gram uses those sentences to identify surrounding words for 'cow'.

Word Embedding

CBOW

w_{t-2}

w_{t-1}

w_{t+1}

w_{t+2}

Neural Network

w_t

Skipgram

w_t

Neural Network

w_{t-2}

w_{t-1}

w_{t+1}

w_{t+2}

Popular Word Embedding Methods

- Word2Vec

- GLoVe

- FastText

Word2Vec

- Word2Vec uses a neural network used to **convert a given text corpus into a vector of words**. The vector which represents the word is called *Neural Word Embeddings*.

- Word2Vec can perform both CBOW and SkipGram tasks.

- The input layer converts each word into a one-hot encoded vector, where only one element (corresponding to the index of the word in the vocabulary) is 1, and all other elements are 0. The projection layer, also known as the embedding layer, indeed maps the one-hot encoded input vectors into dense vectors of lower dimensionality. These dense vectors represent the word embeddings and are typically much smaller in dimensionality compared to the one-hot encoded vectors. In the original Word2Vec architectures (e.g., Continuous Bag of Words (CBOW) and Skip-gram), there's typically no hidden layer.

Word2Vec Example

```python
# Python example to generate word vectors using Word2Vec
from nltk.tokenize import sent_tokenize, word_tokenize

import nltk
nltk.download('punkt')

import warnings
warnings.filterwarnings(action = 'ignore')

import gensim
from gensim.models import Word2Vec

# source to download https://archive.org/stream/OneHundredYearsOfSolitude_201710/One_Hundred_Years_of_Solitude_djvu.txt
#  Reads 'ggmarques.txt' file
sample = open('/Users/reza/EVERYTHING/TEACHING/CS 688_WebAnalyticsMining/toGithub/Feature Engineering/ggmarques.txt',
"r")
s = sample.read()

# Replaces escape character with space
f = s.replace("\n", " ")

data = []

# iterate through each sentence in the file
for i in sent_tokenize(f):
    temp = []

    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())

    data.append(temp)


#------ Create CBOW model
# Create CBOW model
model1 = gensim.models.Word2Vec(data) # , min_count = 1, size = 100, window = 5
print("Cosine similarity between 'banana' and 'orange' - CBOW : ",
        model1.wv.distance('banana', 'orange'))

print("Cosine similarity between 'human' and 'house' - CBOW : ",
        model1.wv.distance('human', 'house'))

#------ Create Skip Gram model
```

```
pip install gensim
pip install nltk
```

GloVe

- The abbreviation of GloVe is for Global Vector for Word Representation, and it is introduced in 2014 [Pennington '14] one year after word2Vec, to address a Word2Vec co-occurrence limitation.

- Word2Vec goes through *each word in the entire corpus* and *predicts the surrounding words once for every word*.

- In particular, Word2Vec's skip-gram ignores whether some context words appear more often than others. A frequent co-occurrence of words creates more instances for the training set, but those instances do not have any additional information; they only increase the size of the training dataset.

- GloVe resolves this challenge by **scanning the entire corpus once** and **counting the co-occurrences once for all words**. The result will be a matrix that involves all co-occurrences, i.e., a window-based *global co-occurrence matrix*. It is a dense representation that presents the overall (global) statistics of word co-occurrences. GloVe uses this matrix to learn word embeddings by considering the global statistical properties of word co-occurrences, not just the *local context* (Word2Vec focuses on local context only).

GloVe

- For example, consider two following sentences and their simplified symmetric *co-occurrence matrix*, which we only check one neighbor words (window size=1). Based on the existence of a window word it adds one point into the matrix. Since, 'I' is neighbor to 'like' two times, their value in the matrix will be 2.

- `I like NLP`

- `I like Word-Embedding methods.`

	I	like	NLP	Word-Embedding	methods
I	0	2	0	0	0
like	2	0	1	1	0
NLP	0	1	0	0	0
Word-Embedding	0	1	0	0	1
methods	0	0	0	1	0

By looking at this matrix you can see 'NLP' and 'Word-Embedding' which are two similar words have the same vector representation.

GloVe

- For example, consider two following sentences and their simplified symmetric *co-occurrence matrix*, which we only check one neighbor words (window size=1). Based on the existence of a window word it adds one point into the matrix. Since, 'I' is neighbor to 'like' two times, their value in the matrix will be 2.

- I like NLP

- I like Wor

Summary:

Skipgram of word2Vec models tries to capture co-occurrence one window at a time, but *Glove* it tries to capture the overall counts how often the co-occurrence appears.

	NLP	Word-Embedding	methods
	0	0	0
	1	1	0
NLP	0	1	0	0	0
Word-Embedding	0	1	0	0	1
methods	0	0	0	1	0

By looking at this matrix you can see 'NLP' and 'Word-Embedding' which are two similar words have the same vector representation.

GloVe Example

```
##  Files needed can be download at https://nlp.stanford.edu/projects/glove/
##  Reference: https://web.stanford.edu/class/cs224n/materials/
Gensim%20word%20vector%20visualization.html

import numpy as np
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec

# Import file and train model
# Files can be download at https://nlp.stanford.edu/projects/glove/
glove_file = 'XXX/glove.6B/glove.6B.50d.txt'  # Pre-trained word vectors
word2vec_glove_file = 'XXX/glove.6B/glove.6B.100d.word2vec.txt'
glove2word2vec(glove_file, word2vec_glove_file)

model = KeyedVectors.load_word2vec_format(word2vec_glove_file)

print (model['china'])
print (model['beijing'])

print (model['india'])
print (model['mumbai'])
```

FastText

- Both Word2Vec and GloVe have the problem of generalization to unknown words.

- Skip-gram of Word2Vec uses uni-gram (one complete word) and takes every single word as a single unit, but <u>FastText uses n-gram</u> and considers *n* number of characters together as a single unit.

- In simple words, while using FastText, we have a *vector for a single word* instead of having *a vector for a set of adjacent words.*

- For example, assuming a window size of three, in the training set the word 'aquarium' can have one vector in word2Vec, but in FastText we will have 'aqu', 'qua', 'uar', 'ari', and 'ium'. Now if the test set we encounter the word 'aqua' for the first time, FastText can see that it is similar to 'aquarium', because the vector values of 'aqu' and 'qua' same.

FastText Example

```python
from gensim.models import FastText
from gensim.test.utils import common_texts  # some example
sentences

print(common_texts[0])
print(len(common_texts))

model = FastText(vector_size=4, window=3, min_count=1) #
instantiate

model.build_vocab(corpus_iterable=common_texts)
model.train(corpus_iterable=common_texts,
total_examples=len(common_texts), epochs=10)  # train

print('----- queen ----')
print(model.wv['queen'])
print('-----king ----')
print(model.wv['king'])
```

From Word Embedding to Context Understanding

- I went to the bank […]

From Word Embedding to Context Understanding

- I went to the bank of river, the view is very nice.
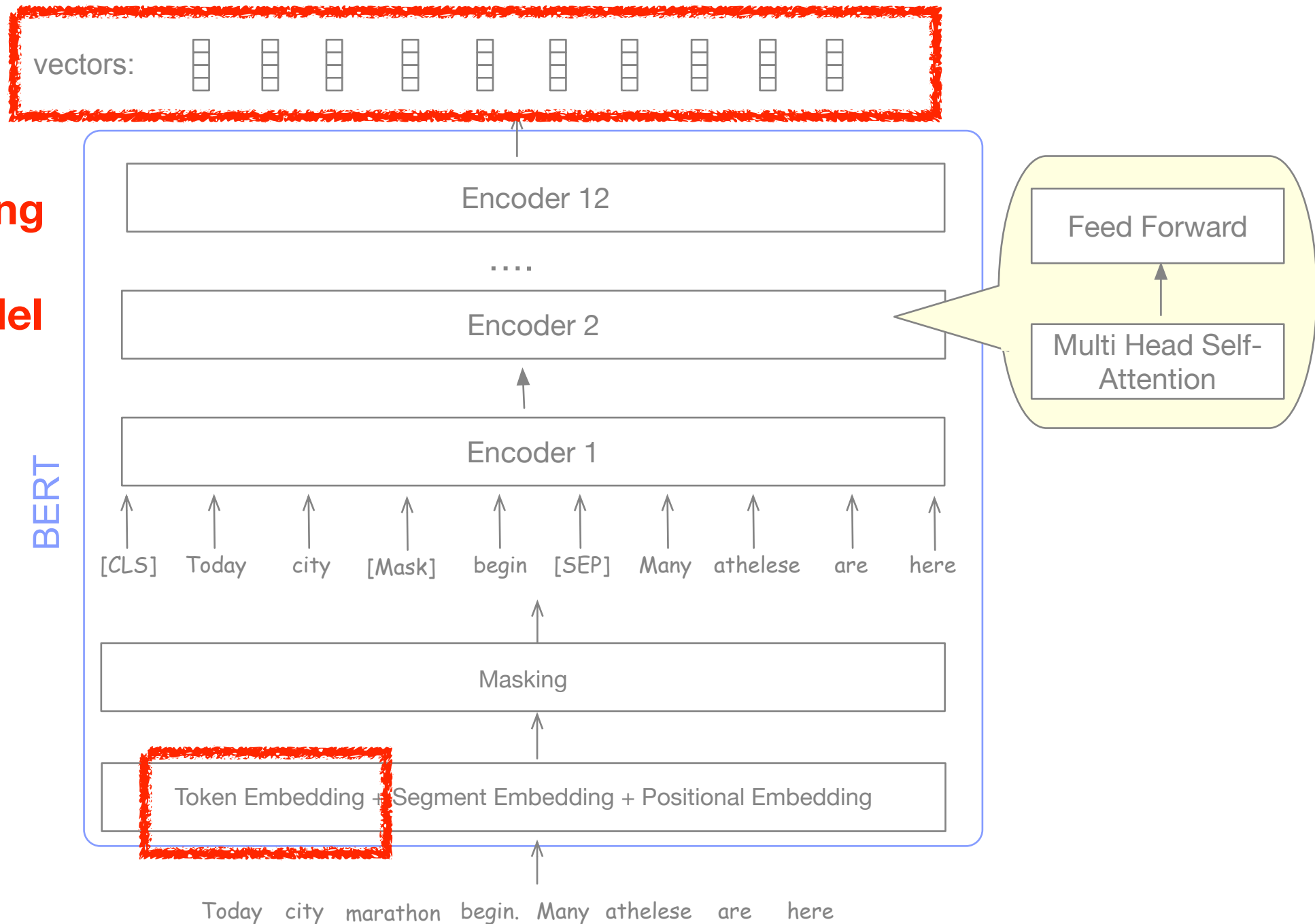
Advances After Word Embeddings

- Seq2Seq Neural Networks (2014)

- Seq2Seq with Attention (2014, 2015)

- Transformer architecture (2017)

- BERT (2018)

- After BERT a competition to make a bigger and thus more accurate model (e.g. GPT-3, LLama, Bloom, ….)

- In November 2022, ChatGPT was released based on the architecture explained in InstructGPT.

Transformer Architecture

Decoder:

Output probability

Softmax

Fully Connected Layer

Normalization

$\oplus$ ← Add

Feed Forward Layer

Normalization

$\oplus$ ← Add

Multi-Head Attention

K ↑ V ↑ Q

Normalization

$\oplus$ ← Add

Masked Multi-Head Attention

n times

Add

Positional Encoding $\oplus$

Word Embedding

Output Sequence

Accurate word embedding

Encoder:

Normalization

$\oplus$

Add

Feed Forward Layer

n times

Normalization

$\oplus$

Multi-Head Attention

Q ↑ K ↑ V ↑

Add

Positional Encoding $\oplus$

Word Embedding

Input Sequence

word embedding

BERT Architecture

vectors:

Accurate word embedding = Language Model

BERT

Encoder 12

....

Encoder 2

Encoder 1

[CLS]   Today   city   [Mask]   begin   [SEP]   Many   athelese   are   here

Masking

Token Embedding + Segment Embedding + Positional Embedding

Today   city   marathon   begin.   Many   athelese   are   here

Feed Forward

Multi Head Self-Attention

ChatGPT Architecture

(i) Supervised Fine Tuning

Text Prompt Dataset

Sample Text Prompts

Human Label (Specify desired output)

Pre-trained LM

Neural Network (SFT)

Fine-tuned LM (SFT model)

(ii) Reward Training

Freezed SFT Model

Generated outputs

Neural Network (RM)

Reward Model

Human Ranks Generated outputs

Use SFT weights to initialize PPO

(iii) Optimize the Policy using PPO

Sample Text Prompt

PPO

Generated output

Freezed Reward Model

Reward

References and Further Readings
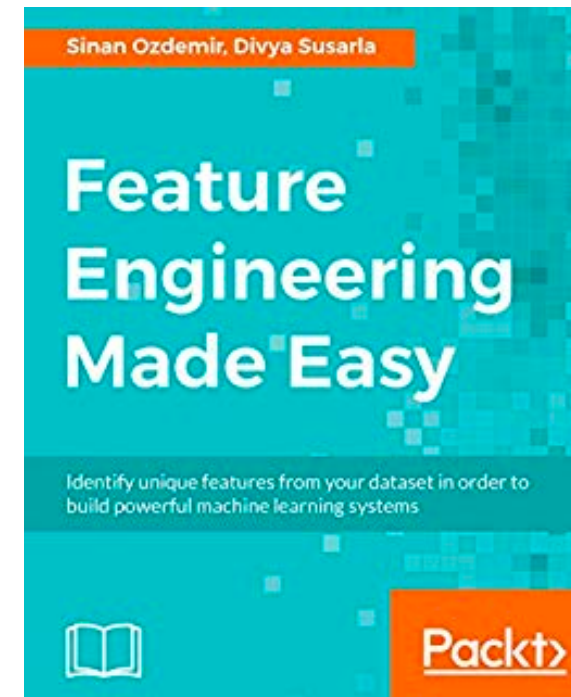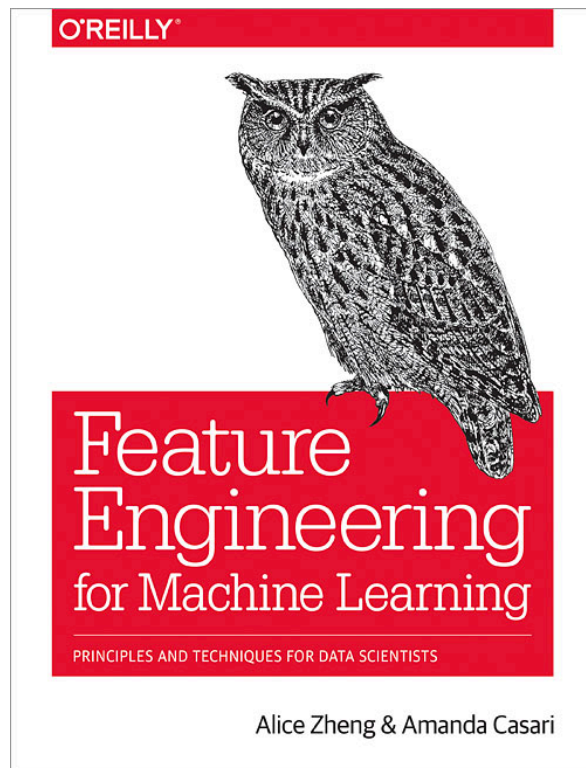
- A very brief and helpful tutorial has been proposed for Genetic Algorithm here. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

- A good tutorial about Word2Vec existed here: https://skymind.ai/wiki/word2vec