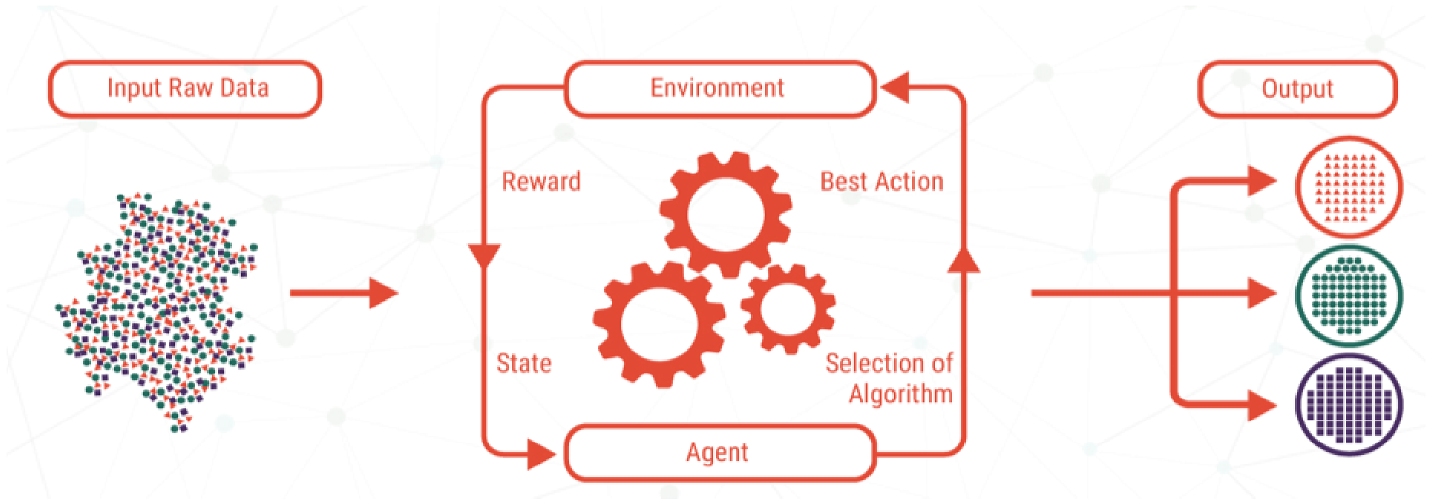
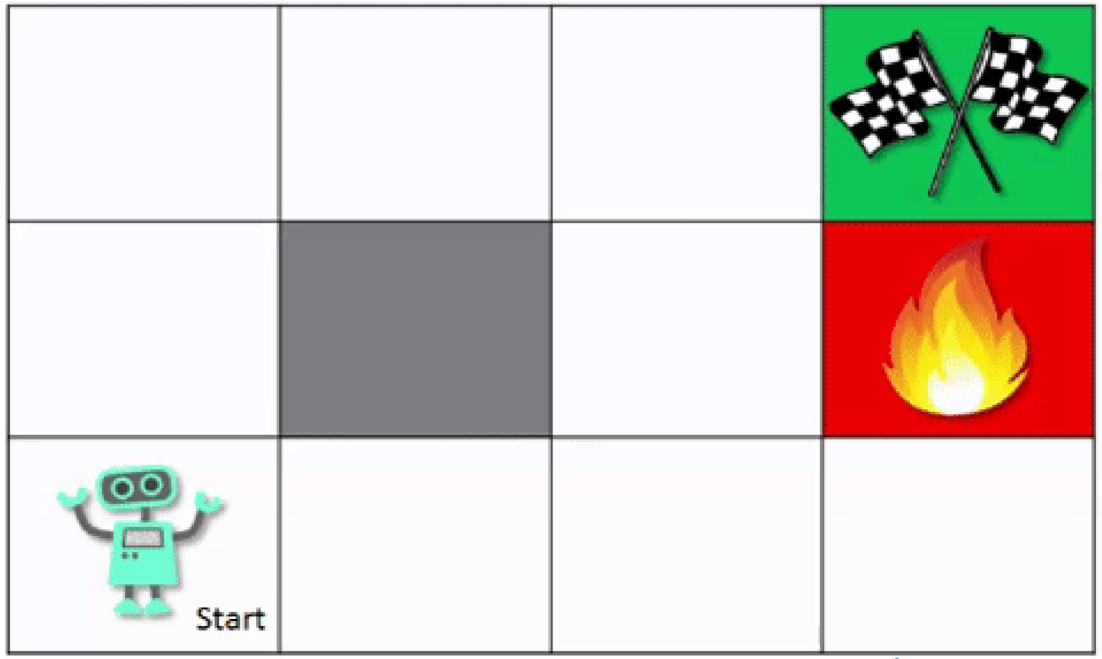
Reinforcement Learning

Implementation of Q-Learning

HOW REINFORCEMENT LEARNING WORKS?



Step 1: Formulate Problem
Step 2: Create Environment
Step 3: Define Reward
Step 4: Create Agent and Train Agent
Step 5: Validate Agent
Step 6: Deploy Policy



EXAMPLE: SELF-DRIVING CAB – REINFORCEMENT LEARNING

The Smartcab's job is to **pick up the passenger at one location and drop them off in another.** Smartcab should take care of:

- Drop off the passenger to the **right location.**
- Save passenger's time by **taking minimum time possible to drop off**
- Take care of passenger's **safety and traffic rules**

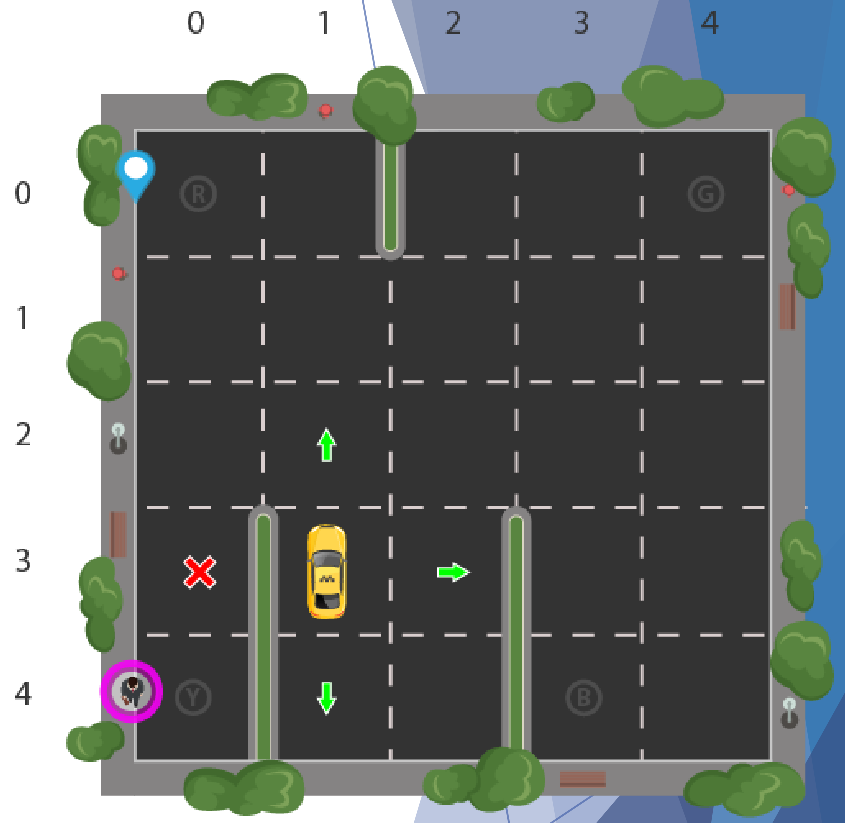
RL algorithm needs only two information a **state** and **actions** to perform.

All we need is a way to identify a state uniquely by **assigning a unique number to every possible state**, and RL learns to choose an action number from 0-5 where:

0 = south	1 = north	2 = east
3 = west	4 = pickup	5 = dropoff

Reinforcement Learning will learn a mapping of **states** to the optimal **action** to perform in that state by *exploration*, i.e. the **agent explores the environment and takes actions based off rewards defined in the environment.**

The optimal action for each state is the action that has the **highest cumulative long-term reward.**

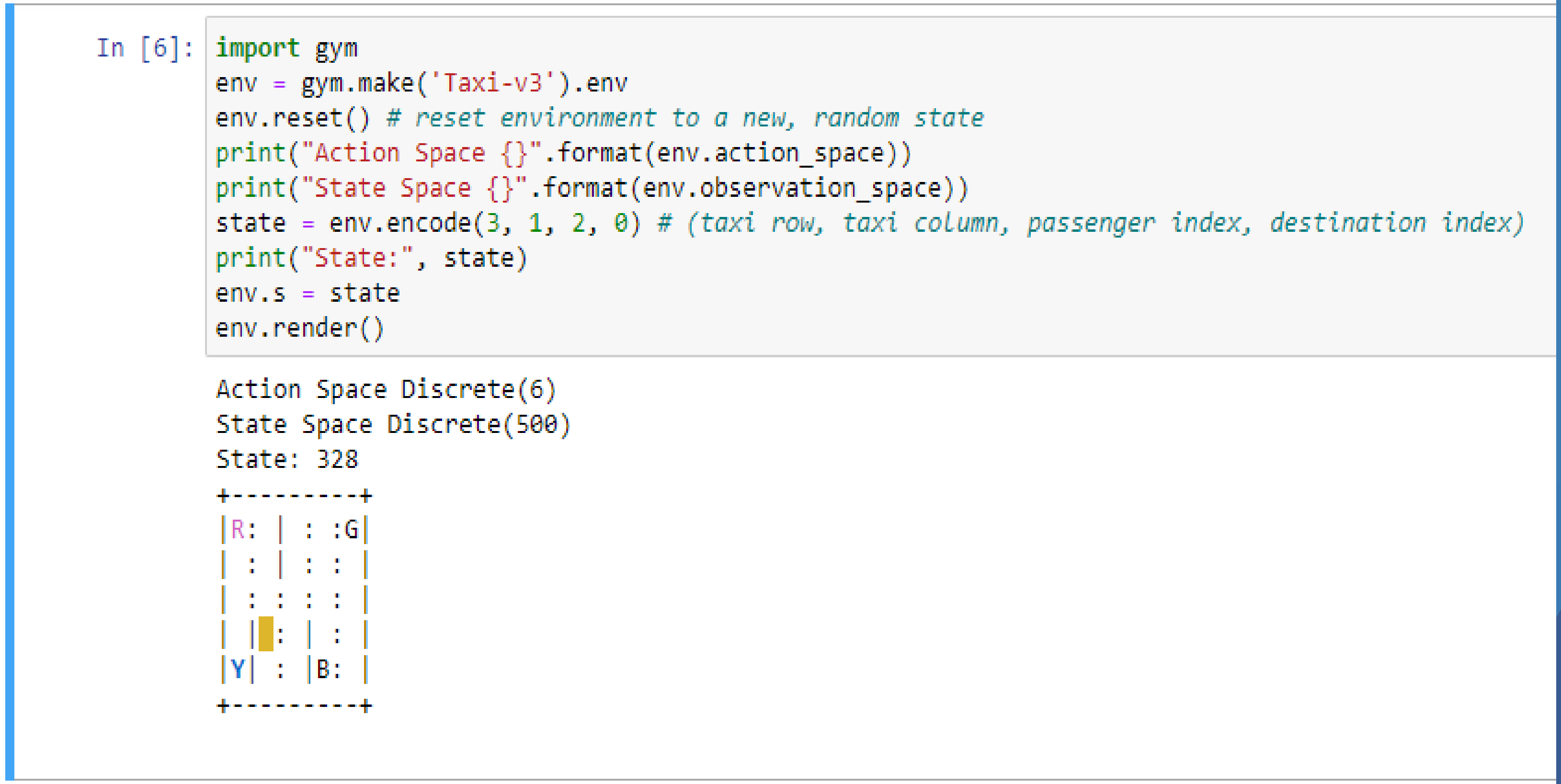


Actions: 6
States: 500
The 500 states correspond to a encoding of the taxi's location, the passenger's location, and the destination location.

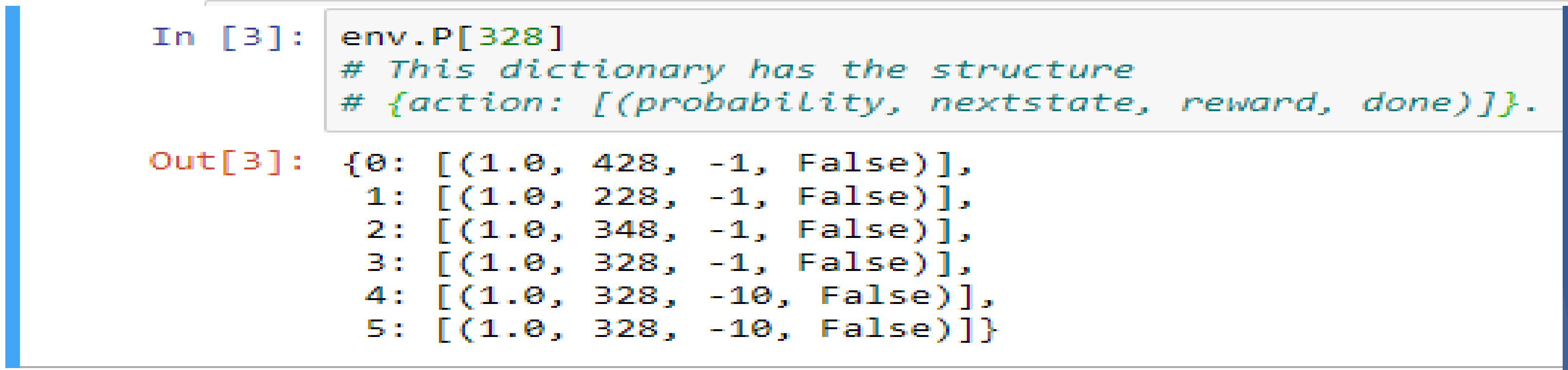
Q-LEARNING IN PYTHON

The code snippet simulate the environment

- The **filled square** represents the taxi, which is yellow without a passenger and green with a passenger.
- The **pipe ("|")** represents a wall which the taxi cannot cross.
- **R, G, Y, B** are the possible pickup and destination locations. The **blue letter** represents the current passenger pick-up location, and the **purple letter** is the current destination.

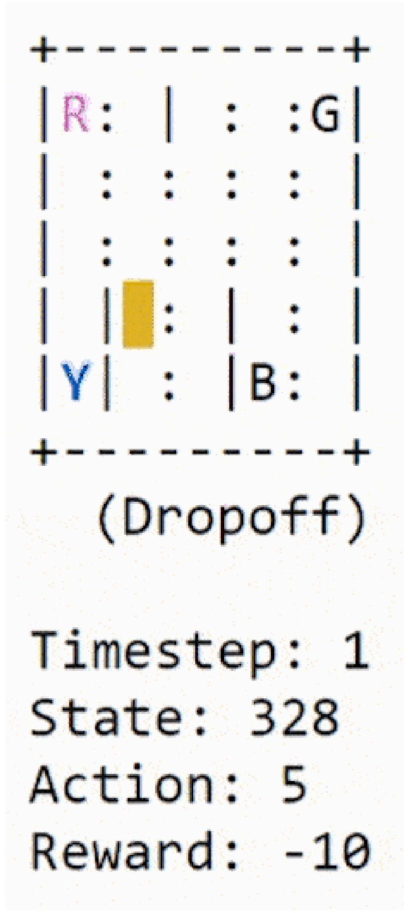


```
In [6]: import gym
        env = gym.make('Taxi-v3').env
        env.reset() # reset environment to a new, random state
        print("Action Space {}".format(env.action_space))
        print("State Space {}".format(env.observation_space))
        state = env.encode(3, 1, 2, 0) # (taxi row, taxi column, passenger index, destination index)
        print("State:", state)
        env.s = state
        env.render()

        Action Space Discrete(6)
        State Space Discrete(500)
        State: 328
        +---------+
        |R: | : :G|
        | : | : : |
        | : : : : |
        | | : | : |
        |Y| : |B: |
        +---------+
```



```
In [3]: env.P[328]
        # This dictionary has the structure
        # {action: [(probability, nextstate, reward, done)]}.

Out[3]: {0: [(1.0, 428, -1, False)],
         1: [(1.0, 228, -1, False)],
         2: [(1.0, 348, -1, False)],
         3: [(1.0, 328, -1, False)],
         4: [(1.0, 328, -10, False)],
         5: [(1.0, 328, -10, False)]}
```

What happens if we try to implement the same without Reinforcement Learning ?



```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Timestep: 1
State: 328
Action: 5
Reward: -10
```

Timesteps taken: 3280
Penalties incurred: 1055

- Our agent takes **thousands of timesteps** and makes **lots of wrong drop offs** to deliver just **one passenger to the right destination.**
- This is because the agent is not ***learning*** **from past experience.**
- Optimizing is difficult as the **agent has no memory** of which action was best for each state, which is exactly **what Reinforcement Learning will do for us.**

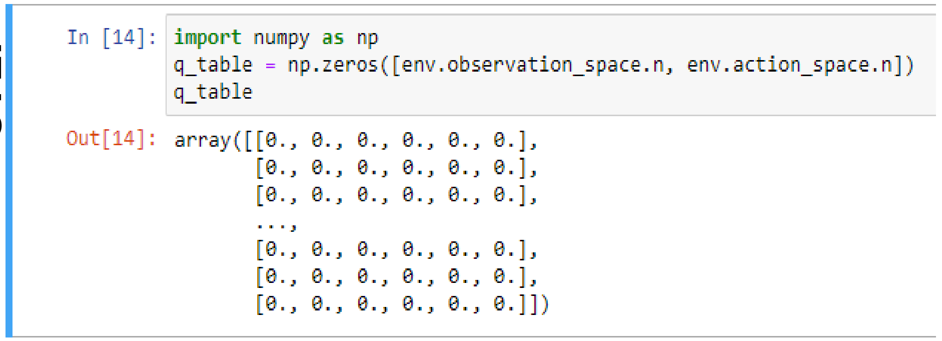
Q-LEARNING PROCESS STEPS:

1. Initialize the Q-table by all zeros.
2. Start exploring actions: For each state, select any one among all possible actions for the current state (S).
3. Travel to the next state (S') as a result of that action (a).
4. For all possible actions from the state (S') select the one with the highest Q-value.
5. Update Q-table values using the equation.

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\left(reward + \gamma \max_{a} Q(next\ state, all\ actions)\right)$$

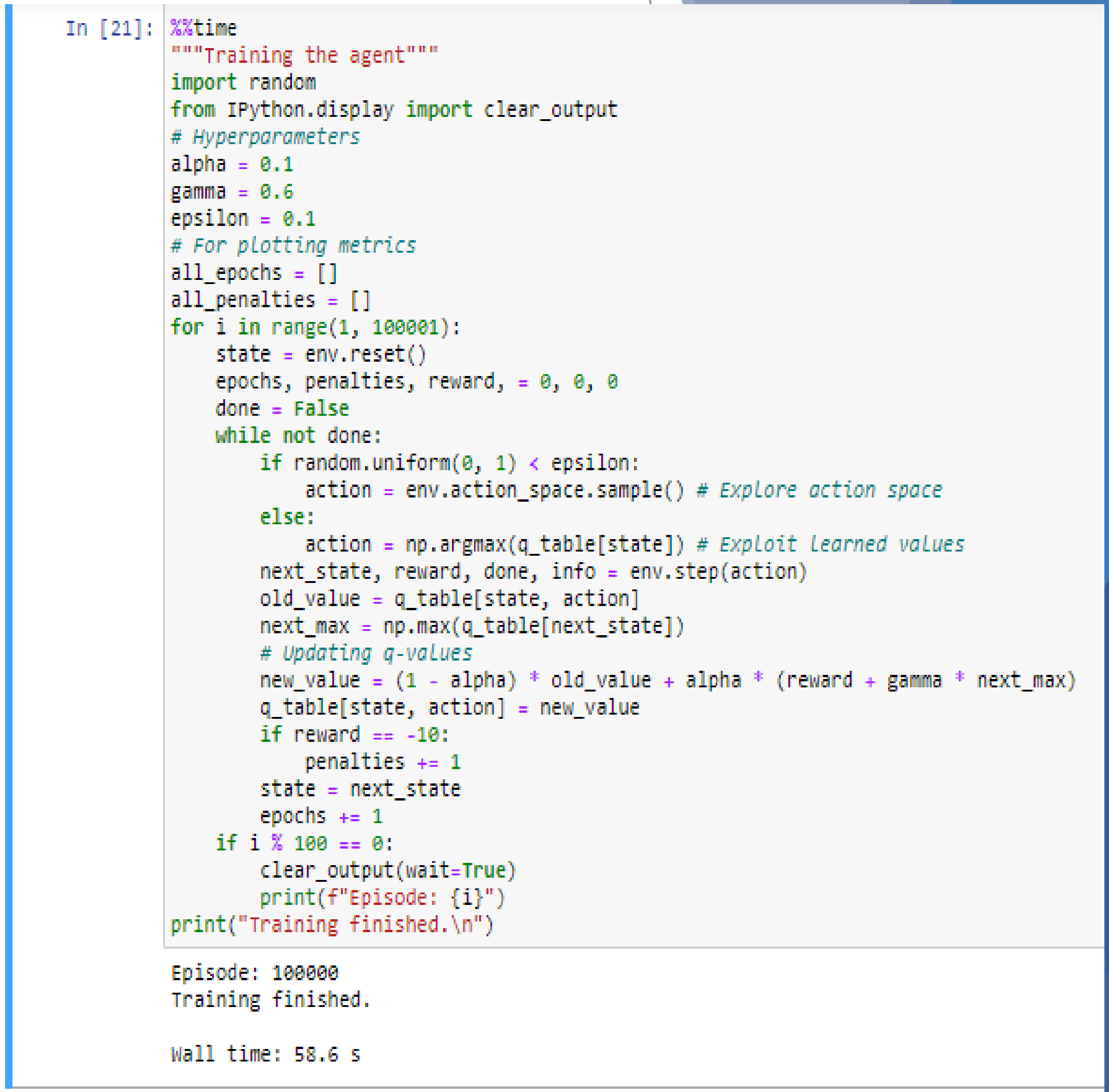
6. Set the next state as the current state.
7. If goal state is reached, then end and repeat the process.

STEP 1



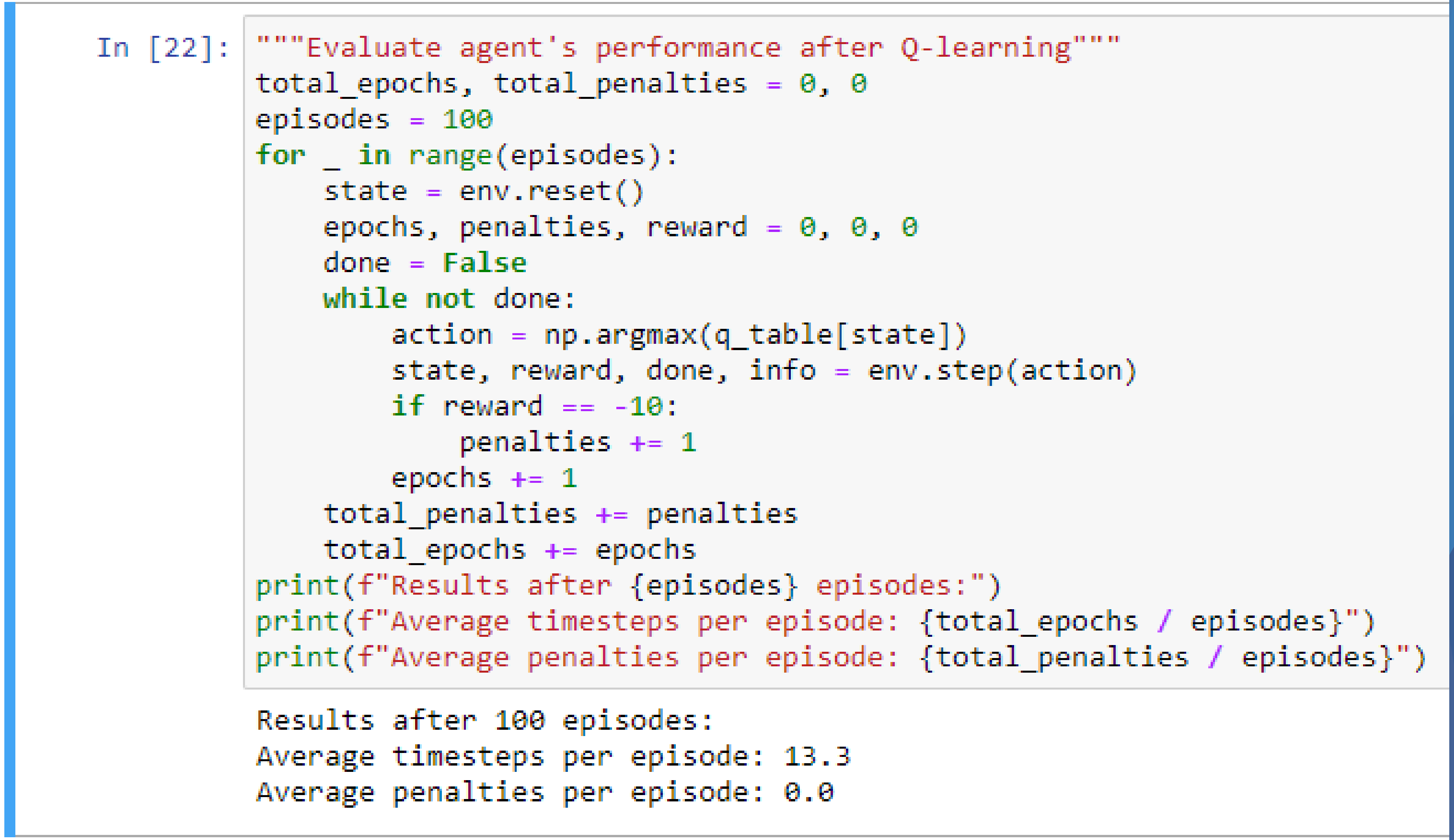
```
In [14]: import numpy as np
         q_table = np.zeros([env.observation_space.n, env.action_space.n])
         q_table

Out[14]: array([[0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.],
                ...,
                [0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.]])
```

STEP 2 to 7



```
In [21]: %%time
         """Training the agent"""
         import random
         from IPython.display import clear_output
         # Hyperparameters
         alpha = 0.1
         gamma = 0.6
         epsilon = 0.1
         # For plotting metrics
         all_epochs = []
         all_penalties = []
         for i in range(1, 100001):
             state = env.reset()
             epochs, penalties, reward, = 0, 0, 0
             done = False
             while not done:
                 if random.uniform(0, 1) < epsilon:
                     action = env.action_space.sample() # Explore action space
                 else:
                     action = np.argmax(q_table[state]) # Exploit learned values
                 next_state, reward, done, info = env.step(action)
                 old_value = q_table[state, action]
                 next_max = np.max(q_table[next_state])
                 # Updating q-values
                 new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
                 q_table[state, action] = new_value
                 if reward == -10:
                     penalties += 1
                 state = next_state
                 epochs += 1
             if i % 100 == 0:
                 clear_output(wait=True)
                 print(f"Episode: {i}")
         print("Training finished.\n")

         Episode: 100000
         Training finished.

         Wall time: 58.6 s
```

We can see from the evaluation, the **agent's performance improved significantly** and it incurred no penalties, which means it performed the **correct pickup/dropoff actions with 100 different passengers.**



```
In [22]: """Evaluate agent's performance after Q-learning"""
         total_epochs, total_penalties = 0, 0
         episodes = 100
         for _ in range(episodes):
             state = env.reset()
             epochs, penalties, reward = 0, 0, 0
             done = False
             while not done:
                 action = np.argmax(q_table[state])
                 state, reward, done, info = env.step(action)
                 if reward == -10:
                     penalties += 1
                 epochs += 1
             total_penalties += penalties
             total_epochs += epochs
         print(f"Results after {episodes} episodes:")
         print(f"Average timesteps per episode: {total_epochs / episodes}")
         print(f"Average penalties per episode: {total_penalties / episodes}")
```

```
Results after 100 episodes:
Average timesteps per episode: 13.3
Average penalties per episode: 0.0
```

HYPERPARAMETERS AND OPTIMIZATIONS

- The values of `alpha`, `gamma`, and `epsilon` were mostly based on intuition and some "hit and trial", but there are better ways to come up with good values.

- Ideally, all three should decrease over time because as the agent continues to learn.

- α: (the learning rate) should decrease as you continue to gain a larger and larger knowledge base.
- γ: as you get closer and closer to the deadline, your preference for near-term reward should increase, as you won't be around long enough to get the long-term reward, which means your gamma should decrease.
- ϵ: as we develop our strategy, we have less need of exploration and more exploitation to get more utility from our policy, so as trials increase, epsilon should decrease.

Natural Language Processing

Introduction to NLP – NLP Implementation in Python

NATURAL LANGUAGE PROCESSING

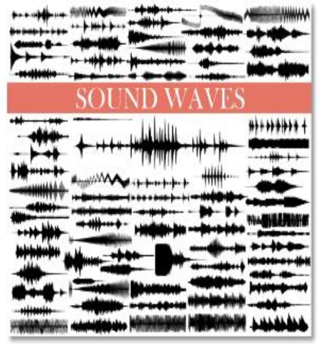
Natural Language Processing (or NLP) is applying Machine Learning models to text and language. Teaching machines to understand what is said in spoken and written word is the focus of Natural Language Processing.

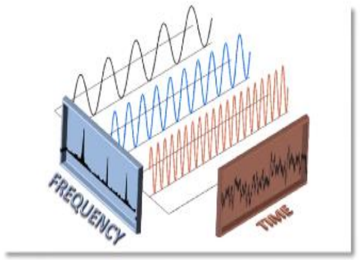
NLP can be used:

- ❑ on a text review to predict if the review is a good one or a bad one
- ❑ an article to predict some categories of the articles you are trying to segment.
- ❑ on a book to predict the genre of the book.
- ❑ to build a machine translator or a speech recognition system
- ❑ Apply Classification algorithms to classify language. Classification models include Logistic Regression, Naive Bayes, CART, Decision Trees, Hidden Markov Models etc.
- ❑ A very well-known model in NLP is the Bag of Words model. It is a model used to preprocess the texts to classify before fitting the classification algorithms on the observations containing the texts.

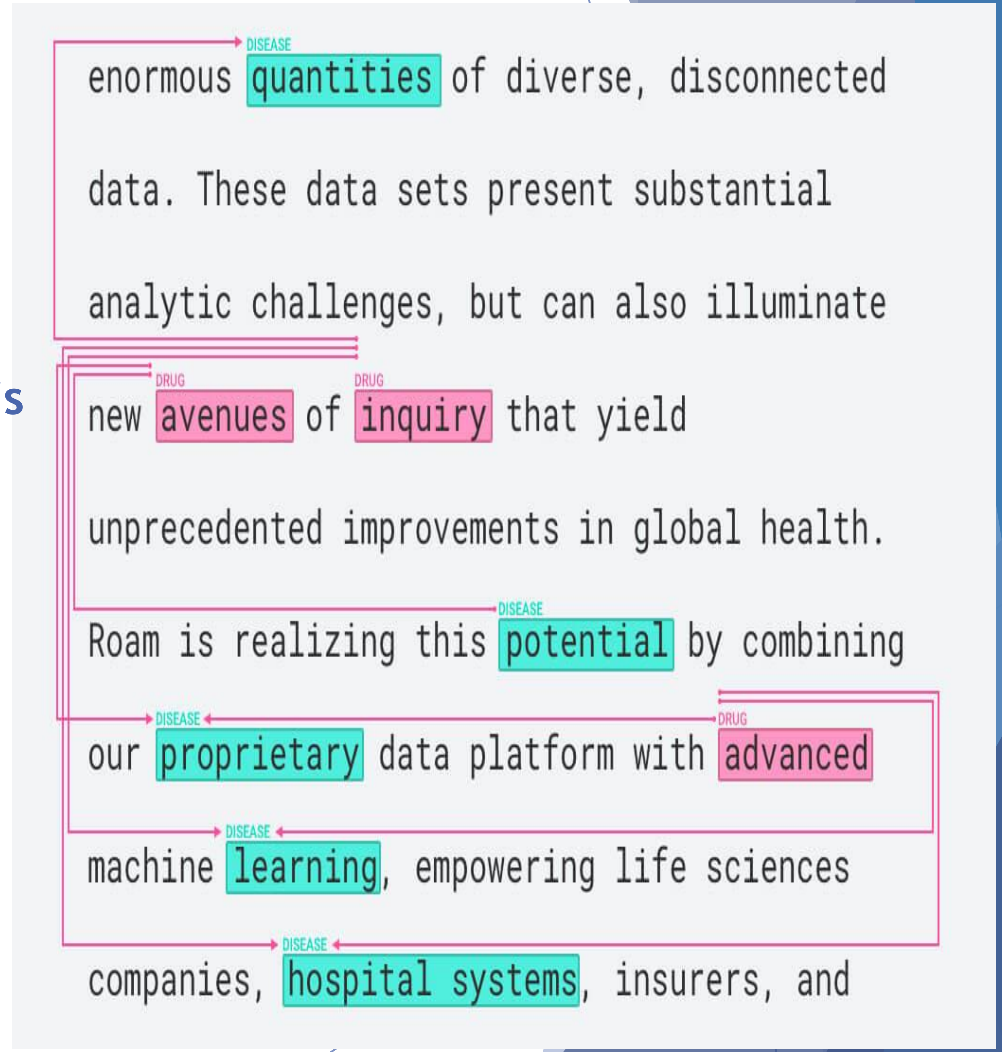
NLP IN ACTION - EXAMPLES

If / Else Rules
(Chatbot)

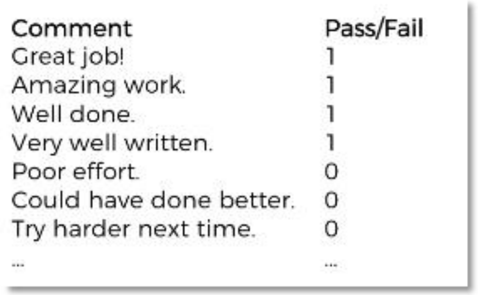




Auto frequency components analysis
(Speech Recognition)



enormous quantities of diverse, disconnected data. These data sets present substantial analytic challenges, but can also illuminate new avenues of inquiry that yield unprecedented improvements in global health. Roam is realizing this potential by combining our proprietary data platform with advanced machine learning, empowering life sciences companies, hospital systems, insurers, and

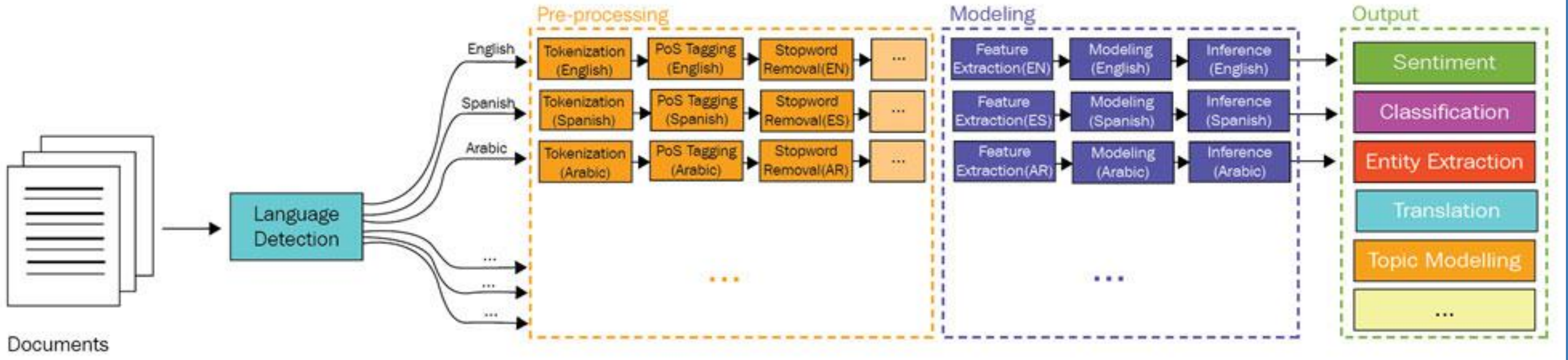


Comment	Pass/Fail
Great job!	1
Amazing work.	1
Well done.	1
Very well written.	1
Poor effort.	0
Could have done better.	0
Try harder next time.	0
...	...

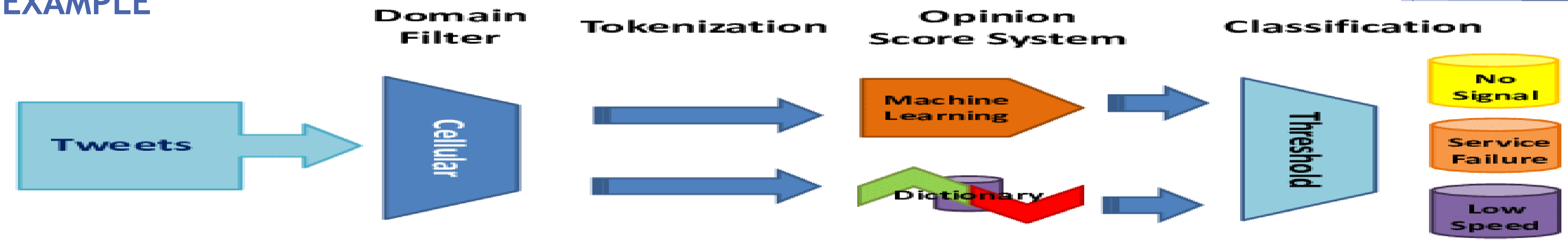
Bag-of-words model
(Classification)

Text Recognition (Classification)

CLASSICAL NLP



EXAMPLE



Lexicon Example:
@AVOID_TalkTalk we've had no service for 4 days. Is this the worst company in the UK?

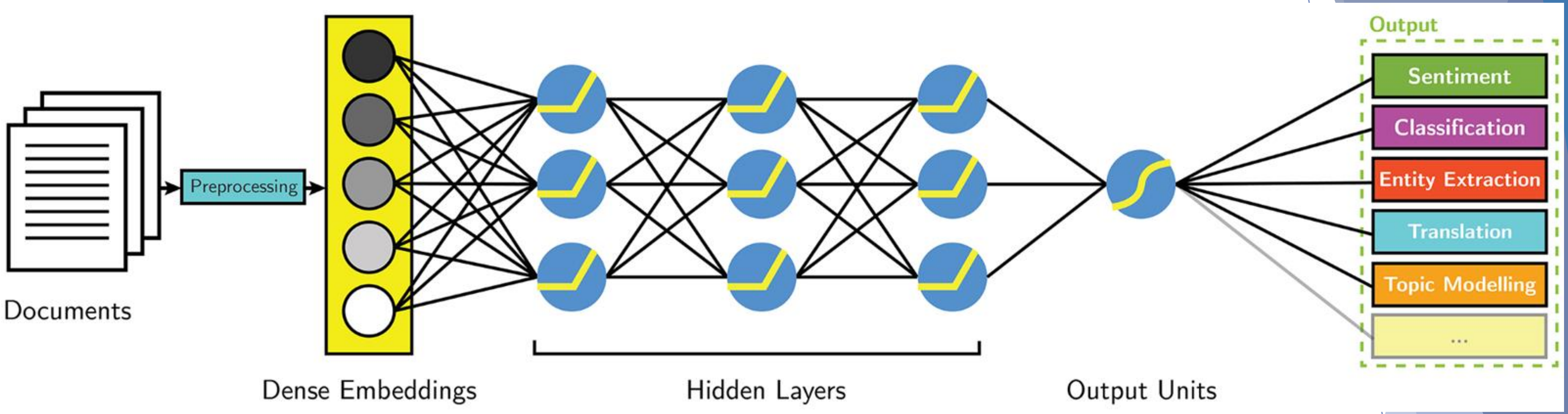
Domain Keywords:
[TalkTalk] + [no service]

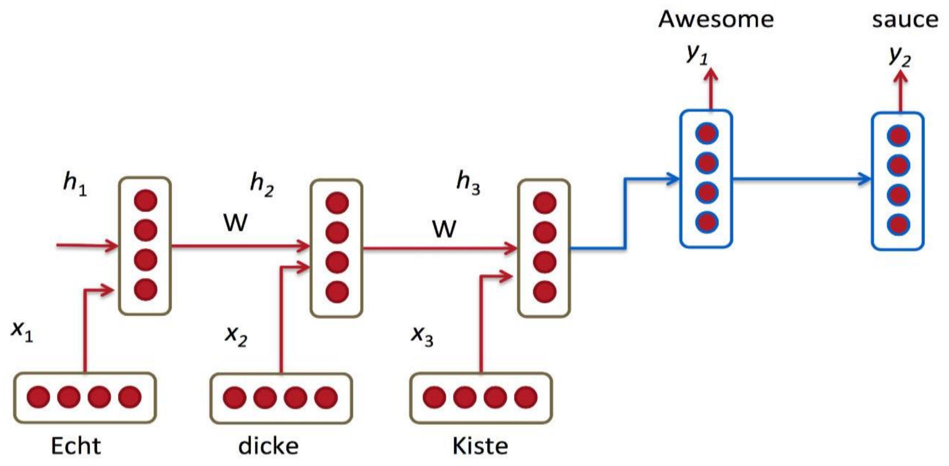
Tokenization:
avoid talktalk we have had no service for 4 days is this the worst company in the UK

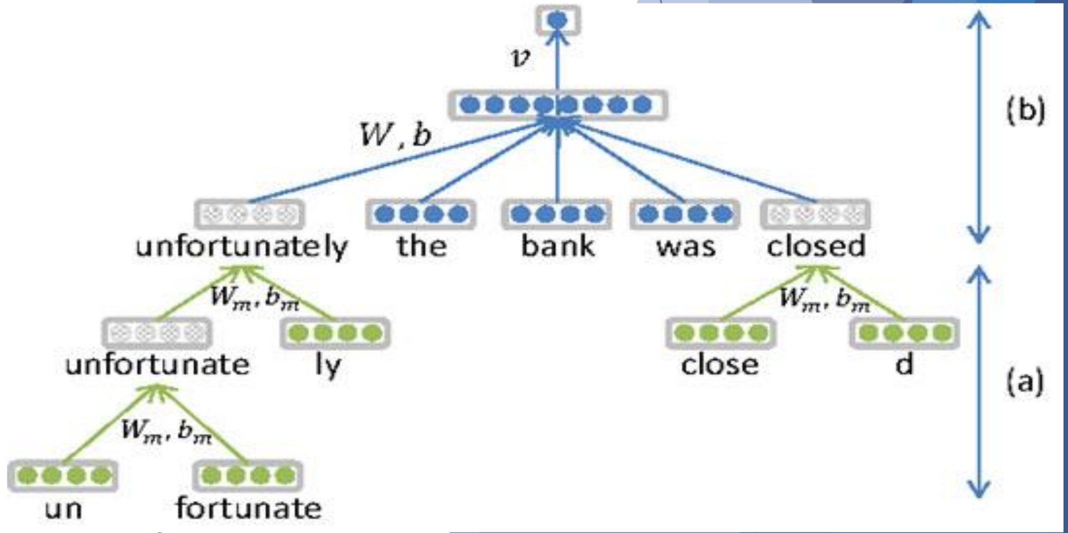
Opinion Score (-3):
avoid
no
worst

Classification:
Type: No Service
Opinion Score: -3
GPS: 51.54, -0.11
Location: St Mary Magdalene Academy, N7 8PG, London

DEEP NLP

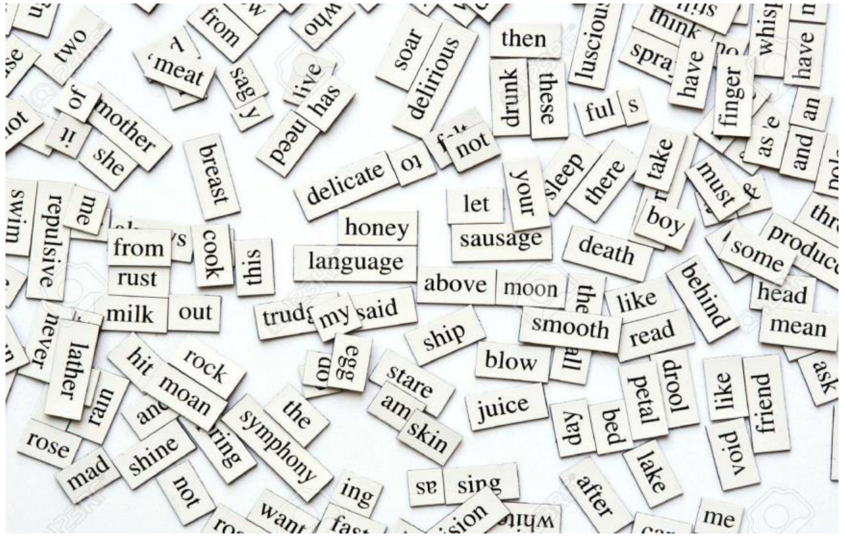

Documents
Preprocessing
Dense Embeddings
Hidden Layers
Output Units
Output
Sentiment
Classification
Entity Extraction
Translation
Topic Modelling
...


Awesome
y_1
sauce
y_2
h_1
h_2
h_3
W
W
x_1
x_2
x_3
Echt
dicke
Kiste


v
W, b
unfortunately
the
bank
was
closed
W_m, b_m
unfortunate
ly
close
d
W_m, b_m
un
fortunate
(b)
(a)

IMPORTANT TERMS

Corpus: Large collection of words or phrases - can come from different sources: documents, web sources, database. Either compiled as written texts or as a transcription of recorded speech



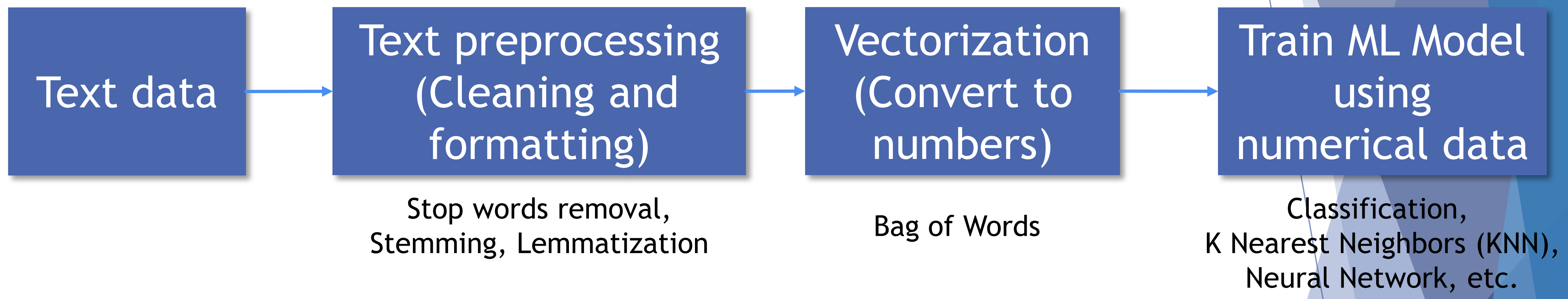
Token: Words or phrases extracted from documents

Feature vector: A numeric array that ML models use for different tasks such as training and prediction

	royalty ⇩	femininity ⇩	intelligence ⇩
king	0.9	-0.9	0.5
queen	0.9	0.9	0.5
man	0.1	-0.9	0.5
woman	0.1	0.9	0.5
smart	0.5	0	0.87
intelligent	0.5	0	0.9

MACHINE LEARNING WITH TEXT DATA

ML models need **well-defined numerical data.**


Text data
Text preprocessing (Cleaning and formatting)
Stop words removal, Stemming, Lemmatization
Vectorization (Convert to numbers)
Bag of Words
Train ML Model using numerical data
Classification, K Nearest Neighbors (KNN), Neural Network, etc.

TEXT PREPROCESSING

Tokenization

Splits text/document into small parts by white space and punctuation.
“I don’t like eggs.” -> “I”, “do”, “n’t”,“like”, “eggs”, “.”

Stop words

Some words that frequently appear in texts, but they don’t contribute too much to the overall meaning.
Common stop words: “a”, “the”, “so”, “is”, “it”, “at”, “in”, “this”, “there”, “that”, “my” etc…..
“There is a tree near the house” -> “tree near house”

Stemming

Set of rules to slice a string to a substring that usually refers to a more general meaning.
The goal is to remove word affixes (particularly suffixes) such as “s”, “es”, “ing”, “ed”, etc.
The issue: It doesn’t usually work with irregular forms such as irregular verbs: “taught”, “brought”, etc.
“playing”, “played”, “plays” -> ”play”

Lemmatization

Similar to stemming, but more advanced. It uses a look-up dictionary. Handles more situations and usually works better than stemming.
“taught”, “teaching”, “teaches”” -> “teach”
am”, ”is”, "are" -> “be”

BAG OF WORDS (BOW)

Bag of Words method **converts text data into numbers**

It is done by:

creating **vocabulary** from all the words in the document

calculating the **occurrences** of words: **binary (present or not), word counts, frequencies**

Sentence	like	don’t	I	pizza	ice	cream	sweets	cakes
“I like pizza”	**1**	0	**1**	**1**	0	0	0	0
“I don’t like ice cream”	**1**	**1**	**1**	0	**1**	**1**	0	0
“I like sweets, I don’t like cakes”	**2**	**1**	**2**	0	0	0	**1**	**1**

TERM FREQUENCY (TF)

Term Frequency: **Increases the weight** for common words in a document.

$$tf(term, doc) = \frac{number\ of\ times\ the\ term\ occurs\ in\ the\ doc}{total\ number\ of\ terms\ in\ the\ doc}$$

Sentence	like	don't	I	pizza	ice	cream	sweets	cakes
"I like pizza"	**0.33**	0	**0.33**	**0.33**	0	0	0	0
"I don't like ice cream"	**0.2**	**0.2**	**0.2**	0	**0.2**	**0.2**	0	0
"I like sweets, I don't like cakes"	**0.28**	**0.14**	**0.28**	0	0	0	**0.14**	**0.14**

INVERSE TERM FREQUENCY (IDF)

Inverse document frequency (IDF): Decreases the weights for **commonly** used words and **increases** weights for **rare** words in the vocabulary.

$$idf(term) = log\left(\frac{n_{documents}}{n_{documents\ containing\ the\ term} + 1}\right) + 1$$

Term	IDF
like	log(3/4)+1=**0.87**
don't	log(3/3)+1=**1**
I	log(3/4)+1=**0.87**
pizza	log(3/2)+1=**1.18**
ice	log(3/2)+1=**1.18**
cream	log(3/2)+1=**1.18**
sweets	log(3/2)+1=**1.18**
cakes	log(3/2)+1=**1.18**

Term Freq. - Inverse Doc. Freq (TF-IDF):
Combines term frequency and inverse document frequency.

$$tf_{idf}(term, doc) = tf(term, doc) * idf(term)$$

Sentence	like	don't	I	pizza	ice	cream	sweets	cake s
"I like pizza"	**0.29**	0.00	**0.29**	**0.39**	0.00	0.00	0.00	0.00
"I don't like ice cream"	**0.17**	**0.20**	**0.17**	0.00	**0.24**	**0.24**	0.00	0.00
"I like sweets, I don't like cakes"	**0.24**	**0.14**	**0.24**	0.00	0.00	0.00	**0.17**	**0.17**

N-GRAMS

An **n-gram** is a sequence of **n tokens** from a given sample of text or speech.

We can include n-grams in our term frequencies too.

Sentence	1-gram (uni-gram):	2-gram (bi-gram):
It is not a dog, it is a wolf	“it”, “is”, “not”, “a”, “dog”, “it”, “is”, “a”, “wolf”	“it is”, “is not”, “not a”, “a dog”, “dog it”, “it is”, “is a”, “a wolf”

HANDS-ON PRACTICE : ANALYZING REVIEWS

Lets Try:

- **NLP Implementation**
- Airline Tweets:
  - A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

Airline.csv