Natural Language Processing

02: Sentiment Analysis with Logistic Regression

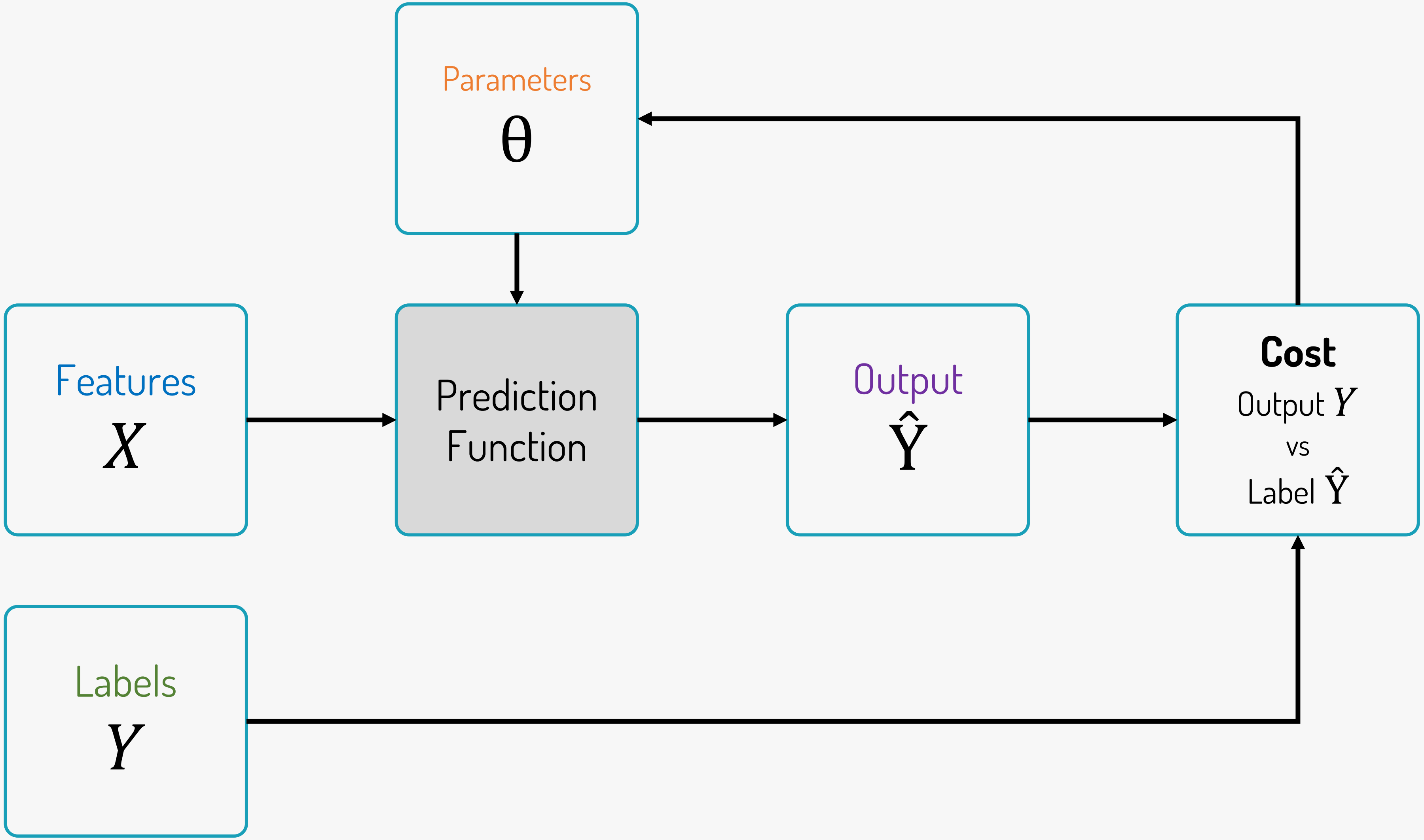
1. Sentiment Analysis & Feature Extraction
2. Preprocessing
3. Logistic Regression
4. Preprocessing using Python
5. Visualizing Word Frequencies
6. Logistic Regression Model

Dr. Imran Ihsan
Ph.D. in Knowledge Engineering
Associate Professor

01-01
Sentiment Analysis & Feature Extraction

02 Sentiment Analysis with Logistic Regression

Supervised Machine Learning (Training)



Sentiment Analysis

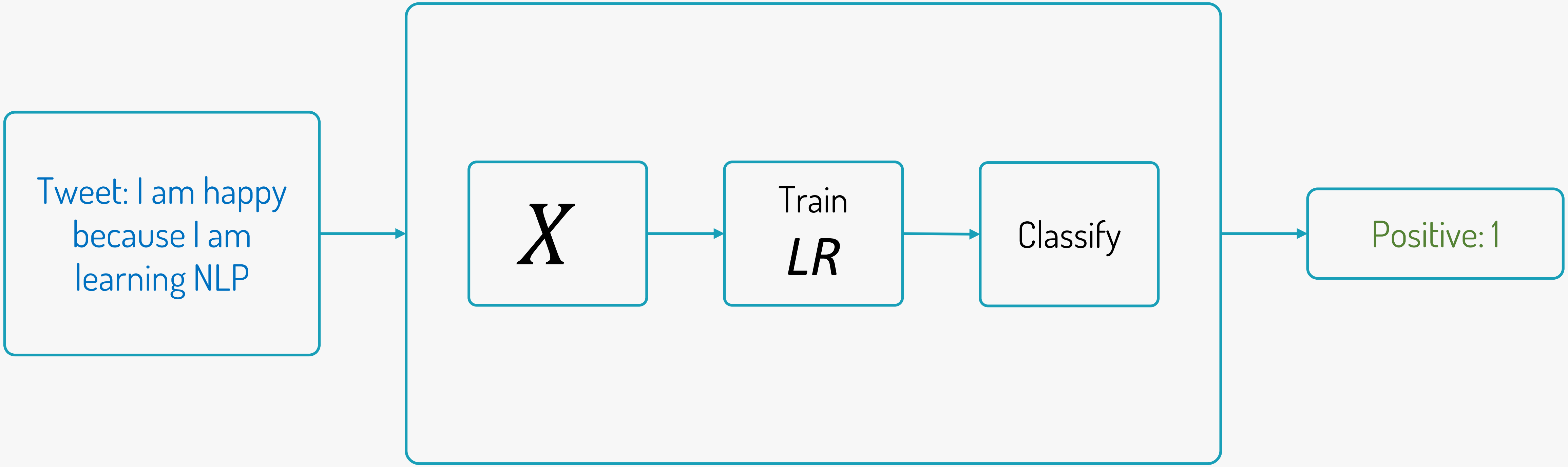
Tweet: I am happy because I am learning NLP

Positive: 1

Negative: 0

Logistic Regression

Supervised ML & Sentiment Analysis



Vocabulary

Tweets:
[tweet_1, tweet_2, ... , tweet_m]

I am happy because I am learning NLP
...
...
...
I hated the movie

$$V = [\, I, am, happy, because, learning, NLP, \dots, hated, the, movie \,]$$

Feature Extraction

I am happy because I am learning NLP

$$[\ I, am, happy, because, learning, NLP, \ldots, hated, the, movie\ ]$$

$$\downarrow$$

$$[\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ \ldots,\ 0,\ 0,\ 0\ ]$$

A lot of Zeros! That's a sparse representation.

Problems with Sparse Representation

I am happy because I am learning NLP

$$[\ 1, 1, 1, 1, 1, 1, \ldots, 0, \ldots, 0, 0, 0\ ]$$

$1 \qquad\qquad |V|$

All Zeros!

$$[\theta_0, \theta_1, \theta_2, \cdots \theta_n\ ]$$
$$n = |V|$$

1: Large Training Time
2: Large Prediction Time

Positive and Negative Frequencies

Corpus

I am happy because I am learning NLP

I am happy

I am sad, I am not learning NLP

I am sad

Vocabulary
I
am
happy
because
learning
NLP
sad
not

Positive and Negative Frequencies

Positive Tweets

I am happy because I am learning NLP

I am happy

Negative Tweets

I am sad because I am not learning NLP

I am sad

Positive and Negative Frequencies

Positive Tweets

I am happy because I am learning NLP

I am happy

Vocabulary	PosFreq (1)
I	
am	
happy	2
because	
learning	
NLP	
sad	
not	

Positive and Negative Frequencies

Positive Tweets

I am happy because I am learning NLP

I am happy

Vocabulary	PosFreq (1)
I	3
am	3
happy	2
because	1
learning	1
NLP	1
sad	0
not	0

Positive and Negative Frequencies

NegFreq (0)	Vocabulary
	I
3	am
	happy
	because
	learning
	NLP
	sad
	not

Negative Tweets

I am sad, I am not learning NLP

I am sad

Positive and Negative Frequencies

NegFreq (0)	Vocabulary
3	I
3	am
0	happy
0	because
1	learning
1	NLP
2	sad
1	not

Negative Tweets

I am sad, I am not learning NLP

I am sad

...Word Frequencies in Classes

Corpus

I am happy because I am learning NLP

I am happy

I am sad, I am not learning NLP

I am sad

Vocabulary	PosFreq (1)	NegFreq (0)
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

Word Frequencies in Classes

Vocabulary	PosFreq (1)	NegFreq (0)
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

$freqs$:

dictionary mapping from (word, class) to frequency

Feature Extraction

$freqs$: dictionary mapping from (word, class) to frequency

$$X_m = \left[1, \sum_w freqs(w,1), \sum_w freqs(w,0)\right]$$

Feature of tweet m

Bias

Sum Pos. Frequencies

Sum Neg. Frequencies

Feature Extraction

Vocabulary	PosFreq (1)	NegFreq (0)
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

I am sad, I am not learning NLP

$$X_m = \left[1, \sum_w freqs(w,1), \sum_w freqs(w,0)\right]$$

8

11

$$X_m = [1, 8, 11]$$

01-02
Preprocessing

02 Sentiment Analysis with Logistic Regression

…Preprocessing: Stop Words and Punctuations

Tweet

@Ilhsan and @AndrewYNg are tuning a GREAT AI model at https://au.edu.pk!!!

Stop Words
and
is
are
at
has
for
a
…

Punctuations
,
.
:
!
“
‘
;
…

Preprocessing: Stop Words and Punctuations

Tweet

@Ilhsan ~~and~~ @AndrewYNg ~~are~~ tuning ~~a~~ GREAT AI model ~~at~~ https://au.edu.pk!!!

@Ilhsan @AndrewYNg tuning
GREAT AI model https://au.edu.pk!!!

Stop Words
~~and~~
is
~~are~~
~~at~~
has
for
~~a~~
…

Punctuations
,
.
:
!
“
‘
;
…

Preprocessing: Stop Words and Punctuations

Tweet

@Ilhsan and @AndrewYNg are tuning a GREAT AI model at https://au.edu.pk!!!

↓

@Ilhsan @AndrewYNg tuning GREAT AI model https://au.edu.pk~~!!!~~

↓

@Ilhsan @AndrewYNg tuning GREAT AI model https://au.edu.pk

Stop Words
and
is
are
at
has
for
a
…

Punctuations
,
.
:
~~!~~
"
'
;
…

…Preprocessing: Handles and URLs

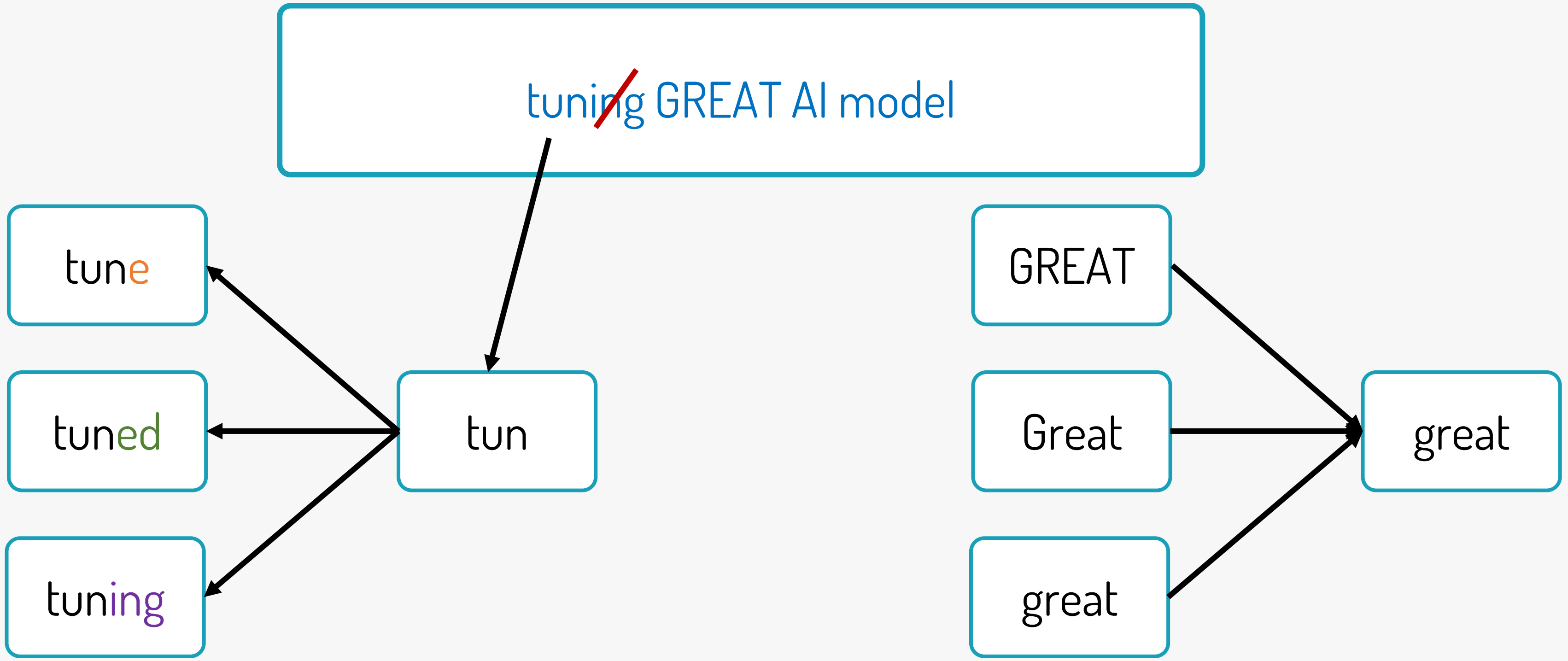
Tweet

~~@lIhsan~~ ~~@AndrewYNg~~ tuning GREAT AI model ~~https://au.edu.pk~~

↓

tuning GREAT AI model

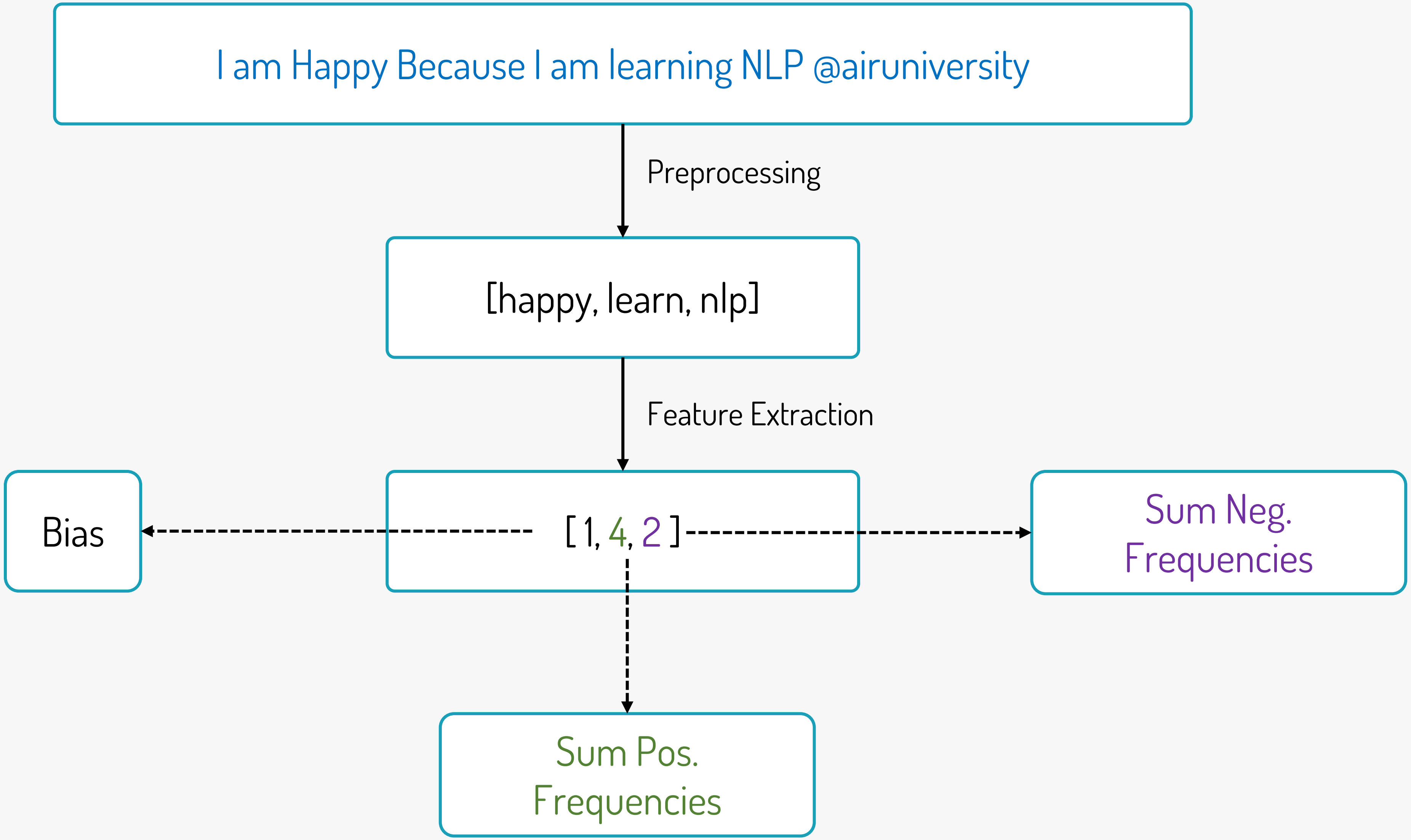
Preprocessing: Stemming and Lowercasing



Preprocessed Tweet

[tun, great, ai, model]

Feature Vector from Single Tweet



…Feature Vectors after Preprocessing

I am happy because I am learning NLP

@airuniversity

I am sad, I am not learning NLP

…

I am sad ☹

[happy, learn, nlp]

[sad, not, learn, nlp]

…

[sad]

[[1, 40, 20],

[1, 20, 50],

…

[1, 5, 35]]

Feature Vector Matrix

$$\begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix} \longleftrightarrow$$

```
[[1, 40, 20],
 [1, 20, 50],
 ...
 [1, 5, 35]]
```

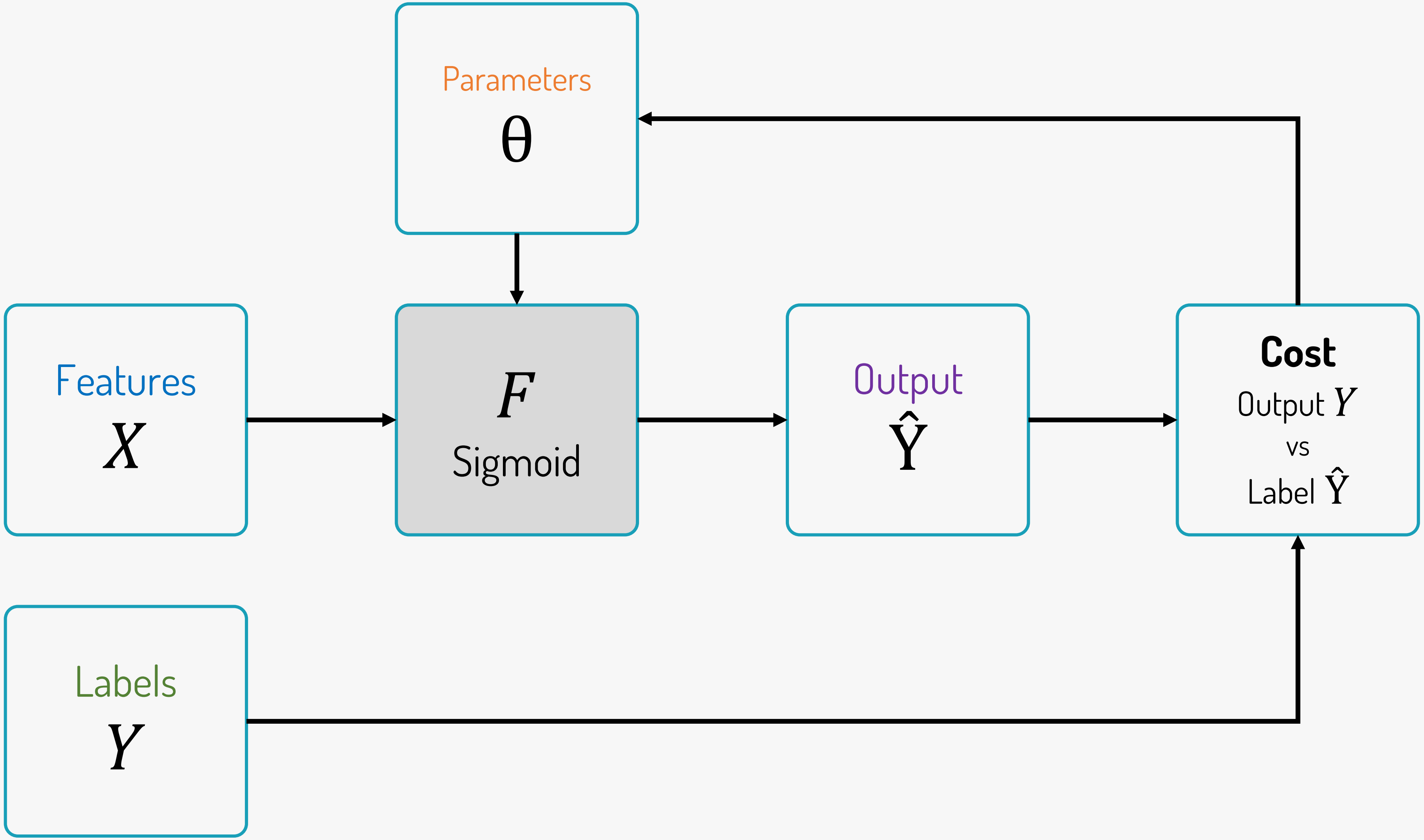
⋯General Implementation

```
freqs = build_freqs(tweets,labels) #Build frequencies dictionary

X = np.zeros((m,3)) #Initialize matrix X

for i in range(m): #For every tweet

    p_tweet = process_tweet(tweets[i]) #Process tweet

    X[i,:] = extract_features(p_tweet,freqs) #Extract Features
```

02-03
Logistic Regression

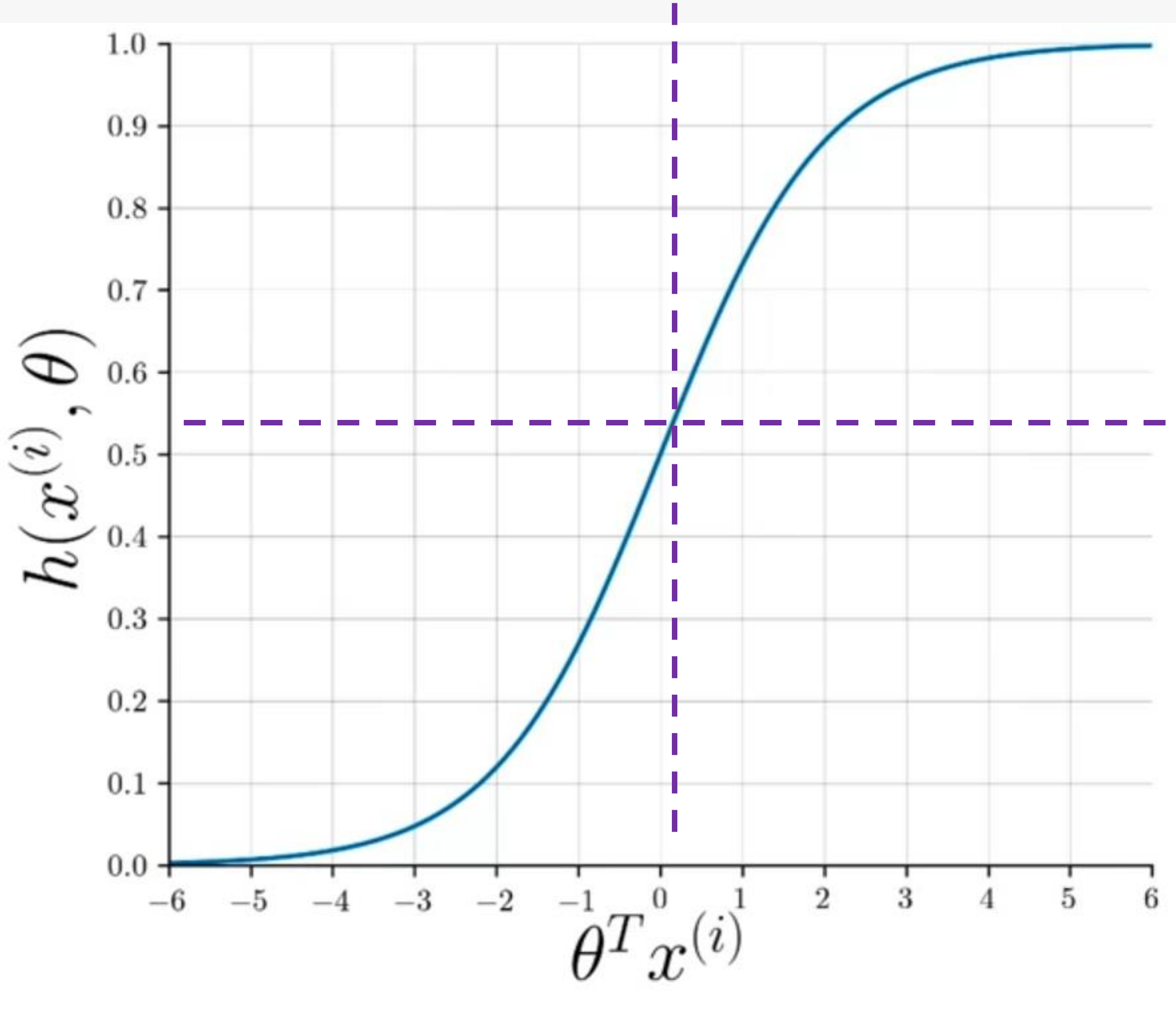
02 Sentiment Analysis with Logistic Regression

Logistic Regression



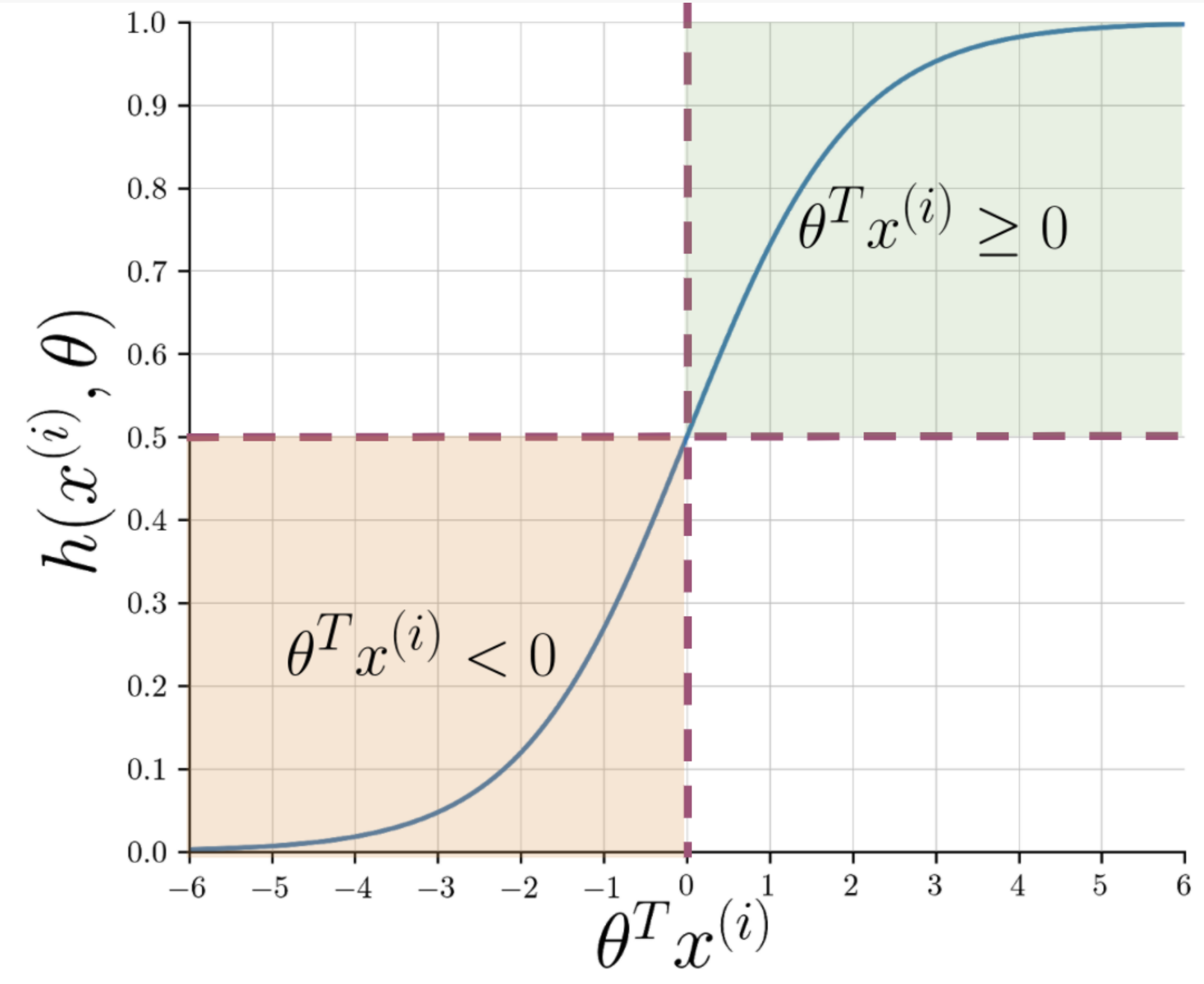
Sigmoid Function

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$



Sigmoid Function

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

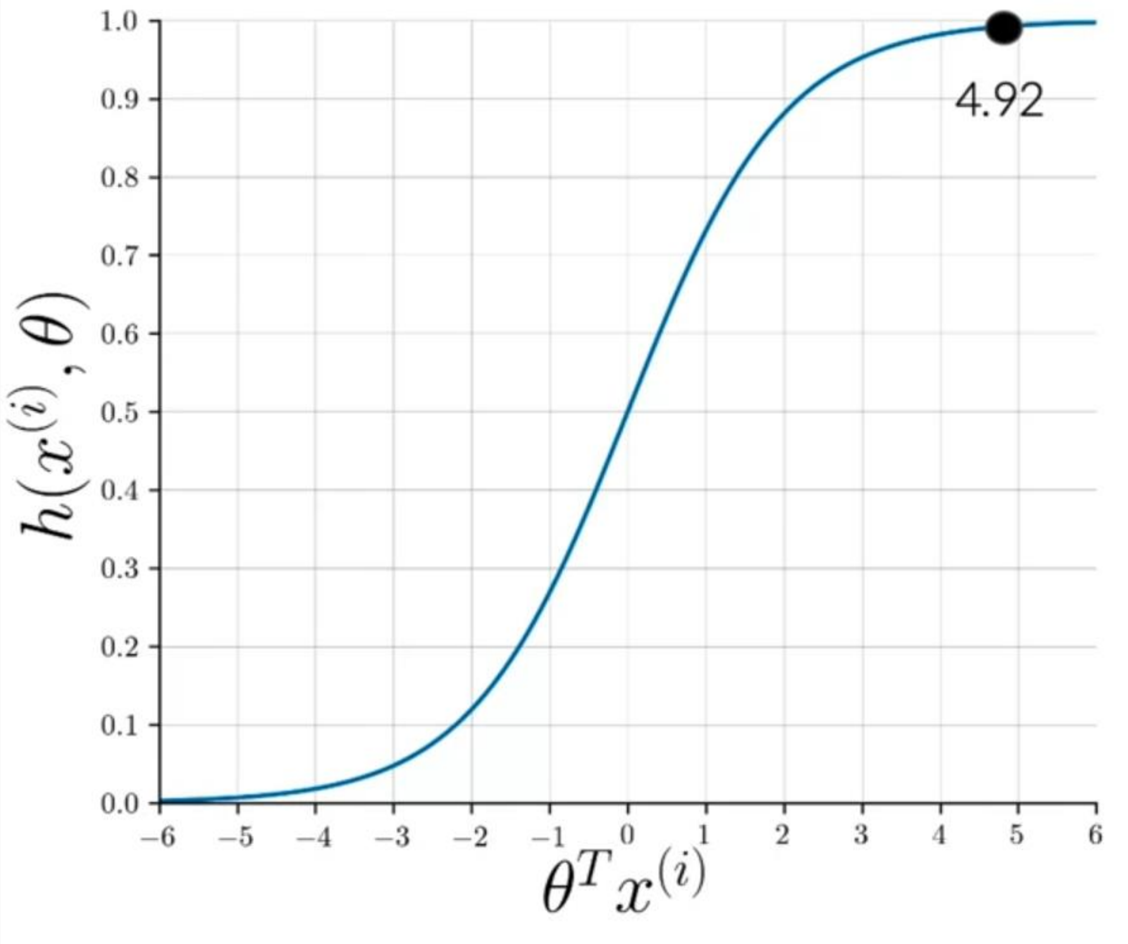


…Logistic Regression

@Ilhsan @AndrewYNg tuning GREAT AI model at https://au.edu.pk

[tun, ai, great, model]

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.0012 \end{bmatrix}$$

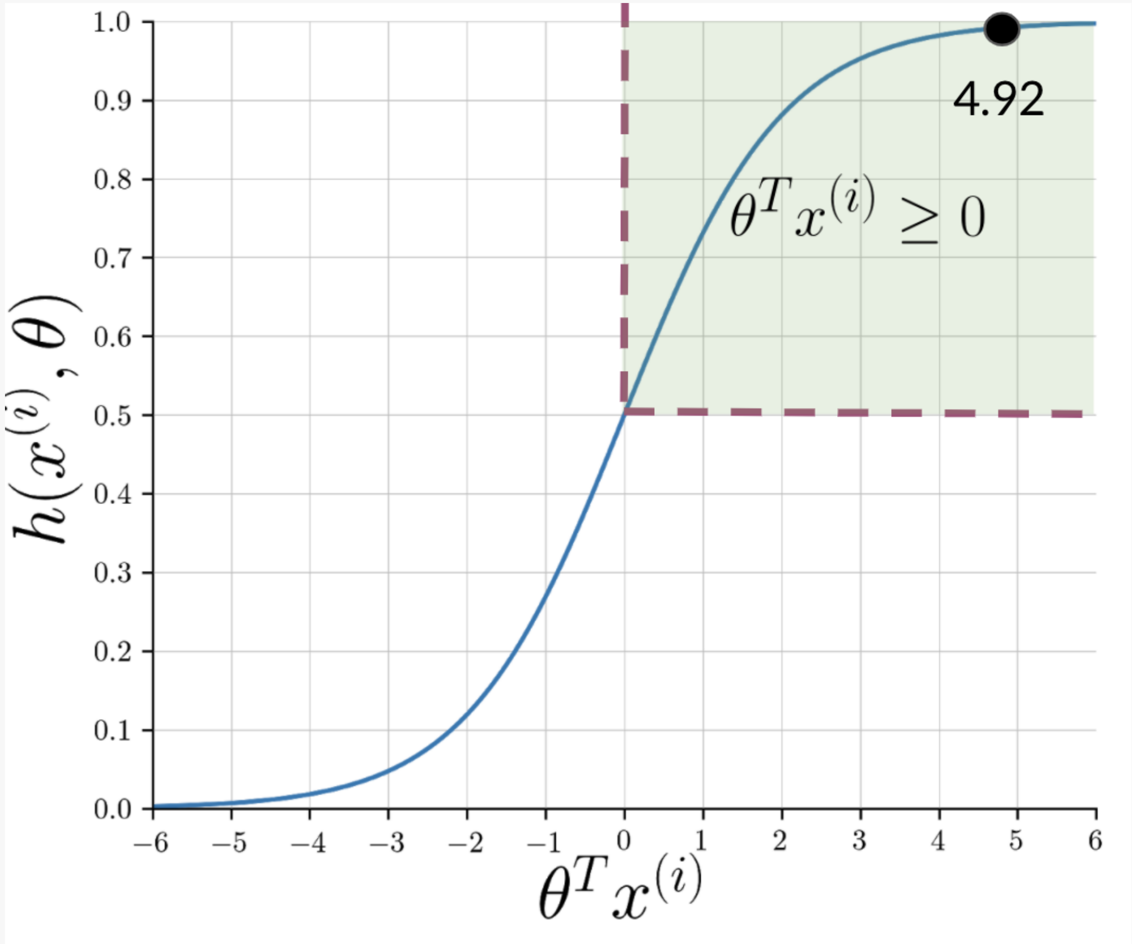


…Logistic Regression

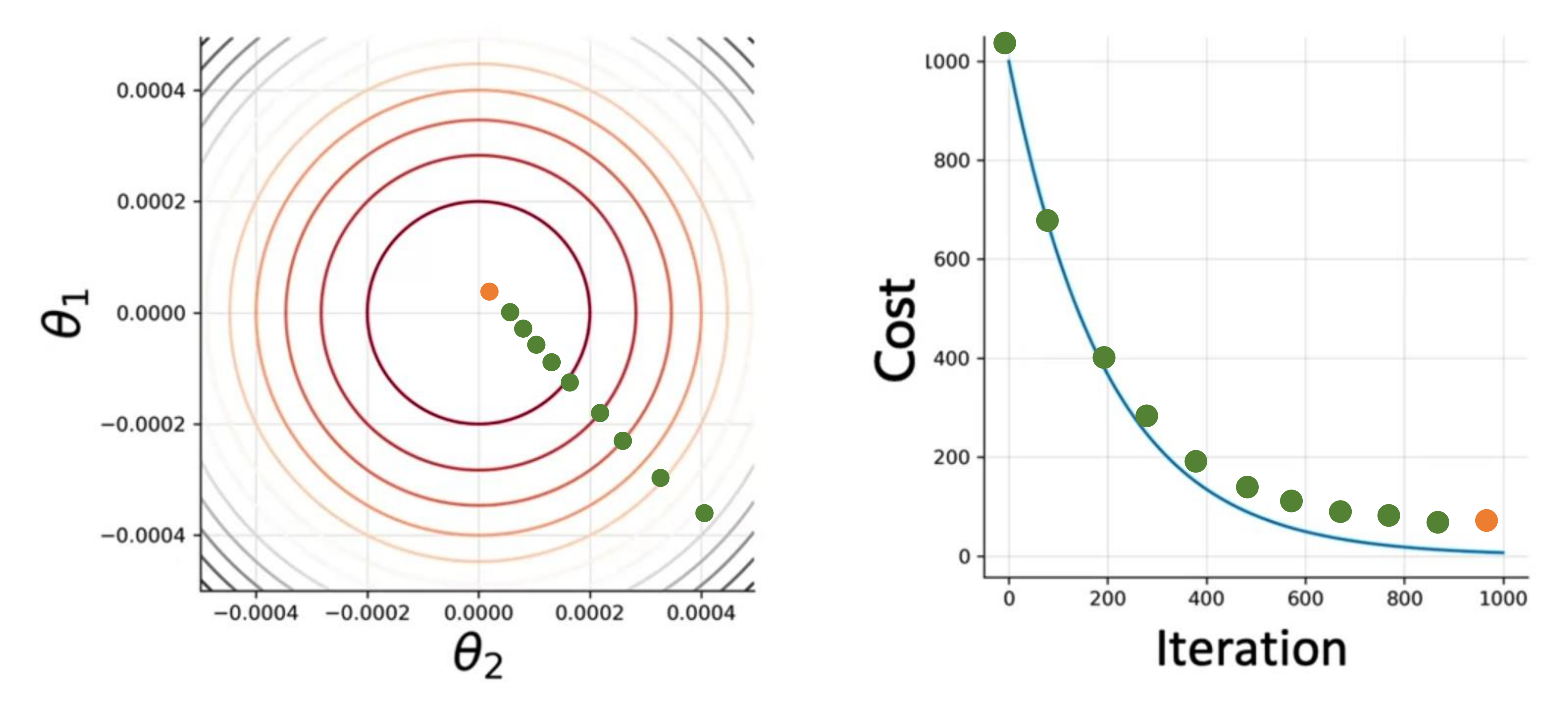
@Ilhsan @AndrewYNg tuning GREAT AI model at https://au.edu.pk

[tun, ai, great, model]

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.0012 \end{bmatrix}$$



Logistic Regression: Training



Logistic Regression: Training

Gradient Descent Algorithm

$$\theta$$

$$h = h(X, \theta)$$

$$\nabla = \frac{1}{m} X^T (h - y)$$

$$\theta = \theta - \alpha \nabla$$

$$J(\theta)$$

- Initialize Parameters
- Classify / Predict
- Get Gradient
- Update
- Get Loss

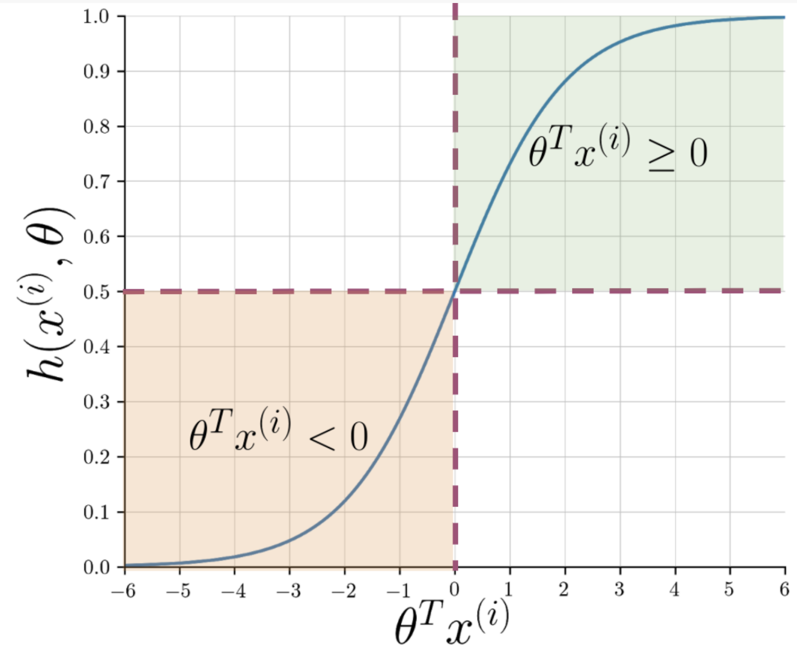
Until Good Enough

…Logistic Regression: Testing

X_{val} Y_{val} θ

$h(X_{val}, \theta)$

$pred = h(X_{val}, \theta) \geq 0.5$



$$\begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 \qquad \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ h_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

Logistic Regression: Testing

$$\sum_{i=1}^{m} \frac{\left(pred^{(i)} == y_{val}^{(i)}\right)}{m}$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m == Y_{val_m} \end{bmatrix}$$

Logistic Regression: Testing

$$Y_{val} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad pred = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (Y_{val} == pred) = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\boxed{accuracy = \frac{4}{5} = 0.8}$$

02-04
Preprocessing using Python

02 Sentiment Analysis with Logistic Regression

Natural Language Toolkit – NLTK (nltk.org)

NLTK is a leading platform for building Python programs to work with human language data.

It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing.

Setup NLTK and Twitter Corpus

On Google Colab, import libraries we will be using.

```
import nltk                                      # Python library for NLP
from nltk.corpus import twitter_samples    # sample Twitter dataset from NLTK
import matplotlib.pyplot as plt            # library for visualization
import random                                    # pseudo-random number generator
```

Twitter Dataset

The sample dataset from NLTK is separated into positive and negative tweets. It contains 5000 positive tweets and 5000 negative tweets exactly. The exact match between these classes is not a coincidence. The intention is to have a balanced dataset. That does not reflect the real distributions.

```
# downloads sample twitter dataset. uncomment the line below if running on a local machine.
nltk.download('twitter_samples')
        [nltk_data] Downloading package twitter_samples to /root/nltk_data...
        [nltk_data] Unzipping corpora/twitter_samples.zip.
        True
```

Load the text fields of the positive and negative tweets by using the module's strings() method:

```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

Positive and Negative Tweets

Report with the number of positive and negative tweets to know the data structure of the datasets.

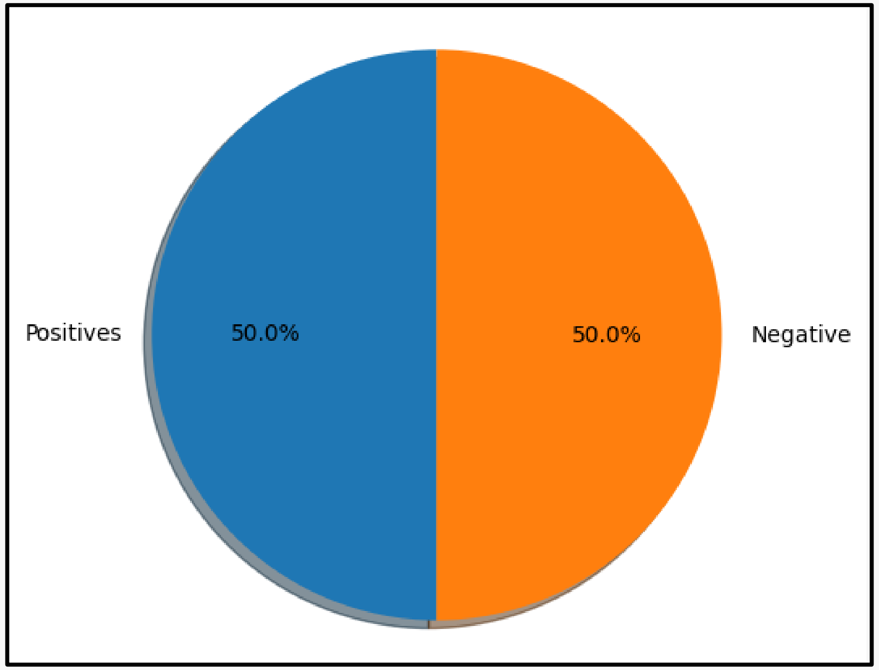
```
print('Number of positive tweets: ', len(all_positive_tweets))
print('Number of negative tweets: ', len(all_negative_tweets))

print('\nThe type of all_positive_tweets is: ', type(all_positive_tweets))
print('The type of a tweet entry is: ', type(all_negative_tweets[0]))
```

```
Number of positive tweets: 5000
Number of negative tweets: 5000
The type of all_positive_tweets is: <class 'list'>
The type of a tweet entry is: <class 'str'>
```

Visualizing Tweets

Use Matplotlib's pyplot library to create a pie chart to visualize of this kind of data.

```
# Declare a figure with a custom size
fig = plt.figure(figsize=(5, 5))

# labels for the two classes
labels = 'Positives', 'Negative'

# Sizes for each slide
sizes = [len(all_positive_tweets), len(all_negative_tweets)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise:
plt.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show( )
```



Raw Text

```
# print positive in greeen
print('\033[92m' + all_positive_tweets[random.randint(0,5000)])

# print negative in red
print('\033[91m' + all_negative_tweets[random.randint(0,5000)])
```

@Bacon_is_life @marcin360 same here, ofc .. I am glad it influenced so many other to create so many awesome RTSs :) Cheers

@seunjinbing @NGVMelbourne I can’t, thesis :(

Select a Sample Tweet

```
# Our selected sample. Complex enough to exemplify each step
tweet = all_positive_tweets[2277]
print(tweet)
```

My beautiful sunflowers on a sunny Friday morning off :)
#sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i

Libraries for Preprocessing

```
# download the stopwords from NLTK
nltk.download('stopwords')
        [nltk_data] Downloading package stopwords to /root/nltk_data...
        [nltk_data] Unzipping corpora/stopwords.zip.
        True

import re          # library for regular expression operations
import string      # for string operations

from nltk.corpus import stopwords          # module for stop words that come with NLTK
from nltk.stem import PorterStemmer        # module for stemming
from nltk.tokenize import TweetTokenizer   # module for tokenizing strings
```

Remove Hyperlink, Twitter Marks and Styles

```
print('\033[92m' + tweet)
print('\033[94m')

# remove old style retweet text "RT"
tweet2 = re.sub(r'^RT[\s]+', '', tweet)

# remove hyperlinks
tweet2 = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet2)

# remove hashtags
# only removing the hash # sign from the word
tweet2 = re.sub(r'#', '', tweet2)

print(tweet2)
```

My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i

My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…

Tokenize String

```
print( )
print('\033[92m' + tweet2)
print('\033[94m')

# instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                           reduce_len=True)

# tokenize tweets
tweet_tokens = tokenizer.tokenize(tweet2)

print()
print('Tokenized string:')
print(tweet_tokens)
```

My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…

Tokenized string:

['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', 'sunflowers', 'favourites', 'happy', ...]

Stop Words and Punctuations

```
#Import the english stop words list from NLTK
stopwords_english = stopwords.words('english')

print('Stop words\n')
print(stopwords_english)

print('\nPunctuation\n')
print(string.punctuation)
```

```
Stop words
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',... ]

Punctuation
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Remove Stop Words and Punctuations

```
print()
print('\033[92m')
print(tweet_tokens)
print('\033[94m')

tweets_clean = [ ]

for word in tweet_tokens: # Go through every word in your tokens list
   if (word not in stopwords_english and  # remove stopwords
       word not in string.punctuation):  # remove punctuation
         tweets_clean.append(word)

print('removed stop words and punctuation:')
print(tweets_clean)
```

```
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', 'sunflowers', 'favourites', 'happy', ...
removed stop words and punctuation:
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites', 'happy', 'friday', '…']
```

Stemming

```
print()
print('\033[92m')
print(tweets_clean)
print('\033[94m')

stemmer = PorterStemmer( )                    # Instantiate stemming class
tweets_stem = [ ]                             # Create an empty list to store the stems
for word in tweets_clean:
    stem_word = stemmer.stem(word)  # stemming word
    tweets_stem.append(stem_word)  # append to the list

print('stemmed words:')
print(tweets_stem)
```

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites', 'happy', 'friday', '…']
stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit', 'happi', 'friday', '…']
```

Process Tweet

```
def process_tweet(tweet):
  stemmer = PorterStemmer()
  stopwords_english = stopwords.words('english')
  tweet = re.sub(r'\$\w*', '', tweet)
  tweet = re.sub(r'^RT[\s]+', '', tweet)
  tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
  tweet = re.sub(r'#', '', tweet)
  tokenizer = TweetTokenizer(preserve_case=False,
                             strip_handles=True,reduce_len=True)
  tweet_tokens = tokenizer.tokenize(tweet)

  tweets_clean = [ ]
  for word in tweet_tokens:
    if (word not in stopwords_english and
          word not in string.punctuation):
        stem_word = stemmer.stem(word)  # stemming word
        tweets_clean.append(stem_word)

  return tweets_clean
```

```
# choose the same tweet
tweet = all_positive_tweets[2277]
print( )
print('\033[92m')
print(tweet)
print('\033[94m')

# call the imported function
tweets_stem = process_tweet(tweet); # Preprocess a given tweet
print('preprocessed tweet:')
print(tweets_stem) # Print the result
```

My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off... https://t.co/3tfYom0N1i

preprocessed tweet: **['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit', 'happi', 'friday', '…']**

02-05
Visualizing Word Frequencies

02 Sentiment Analysis with Logistic Regression

Visualizing Word Frequencies

```
import re
import string
import numpy as np

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
```

Process Tweet

```
def process_tweet(tweet):
    """Process tweet function.
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet

    """
    stemmer = PorterStemmer( )
    stopwords_english = stopwords.words('english')
    # remove stock market tickers like $GE
    tweet = re.sub(r'\$\w*', '', tweet)
    # remove old style retweet text "RT"
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    # remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    # remove hashtags
    # only removing the hash # sign from the word
    tweet = re.sub(r'#', '', tweet)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False,
                               strip_handles=True, reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)

    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and  # remove stopwords
                word not in string.punctuation):  # remove punctuation
            # tweets_clean.append(word)
            stem_word = stemmer.stem(word)  # stemming word
            tweets_clean.append(stem_word)

    return tweets_clean
```

Word Frequency Dictionary

```
def build_freqs(tweets, ys):
    """Build frequencies.
    Input:
        tweets: a list of tweets
        ys: an m x 1 array with the sentiment label of each tweet
            (either 0 or 1)
    Output:
        freqs: a dictionary mapping each (word, sentiment) pair to its
        frequency
    """

    # Convert np array to list since zip needs an iterable.
    # The squeeze is necessary, or the list ends up with one element.
    # Also note that this is just a NOP if ys is already a list.
    yslist = np.squeeze(ys).tolist()

    # Start with an empty dictionary and populate it by looping over all
    # tweets and over all processed words in each tweet.
    freqs = { }
    for y, tweet in zip(yslist, tweets):
        for word in process_tweet(tweet):
            pair = (word, y)
            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1

    return freqs
```

Word Frequency Dictionary

```
# create frequency dictionary
freqs = build_freqs(tweets, labels)
# check data type
print(f'type(freqs) = {type(freqs)}')
# check length of the dictionary
print(f'len(freqs) = {len(freqs)}')
```

```
type(freqs) = <class 'dict'>
len(freqs) = 13065
```

```
print(freqs)
```

```
{('followfriday', 1.0): 25, ('top', 1.0): 32, ('engag', 1.0): 7, ('member', 1.0): 16, ('commun', 1.0): 33, ('week', 1.0): 83, ...
```

Table of Word Counts

```
# select some words to appear in the report. we will assume that each word is unique
keys = ['happi', 'merri', 'nice', 'good', 'bad', 'sad', 'mad', 'best', 'pretti',  '❤️', ':)', ':(', '😒', '😬',
        '😄', '😍', '👑', 'song', 'idea', 'power', 'play', 'magnific']

# list representing our table of word counts.
# each element consist of a sublist with this pattern: [<word>, <pos_count>, <neg_count>]
data = [ ]

for word in keys:           # loop through our selected words
   pos = 0                  # initialize positive and negative counts
   neg = 0

   if (word, 1) in freqs:             # retrieve number of positive counts
      pos = freqs[(word, 1)]

   if (word, 0) in freqs:             # retrieve number of negative counts
      neg = freqs[(word, 0)]

   data.append([word, pos, neg])      # append the word counts to the table

data
```

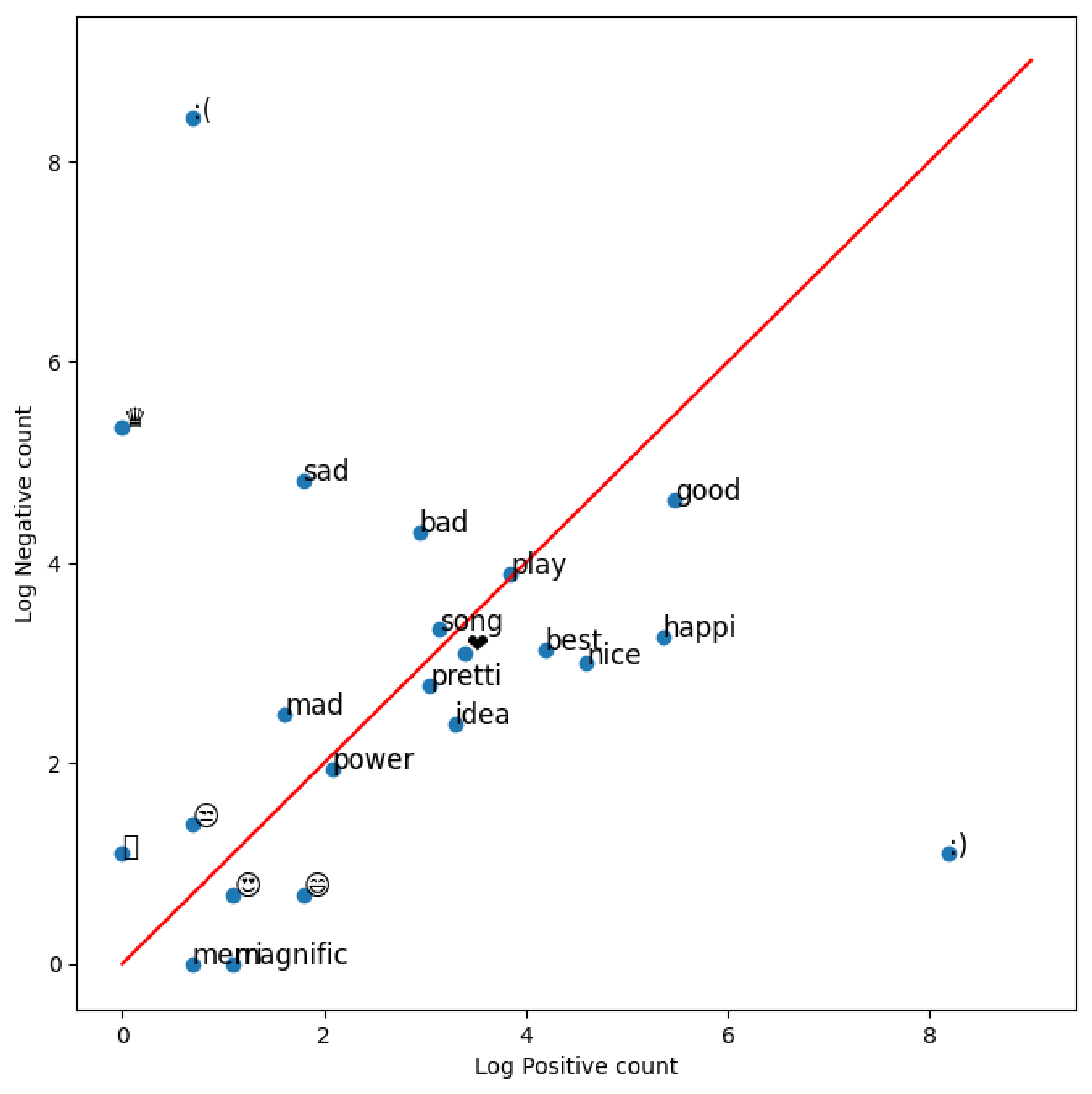
```
[['happi', 211, 25],
 ['merri', 1, 0],
 ['nice', 98, 19],
 ['good', 238, 101],
 ['bad', 18, 73],
 ['sad', 5, 123],
 ['mad', 4, 11],
 ['best', 65, 22],
 ['pretti', 20, 15],
 ['💙', 29, 21],
 [':)', 3568, 2],
 [':(', 1, 4571],
 ['😒', 1, 3],
 ['😬', 0, 2],
 ['😄', 5, 1],
 ['😍', 2, 1],
 ['👑', 0, 210],
 ['song', 22, 27],
 ['idea', 26, 10],
 ['power', 7, 6],
 ['play', 46, 48],
 ['magnific', 2, 0]]
```

Scatter Plot

```
fig, ax = plt.subplots(figsize = (8, 8))

# convert positive raw counts to logarithmic scale.
# we add 1 to avoid log(0)
x = np.log([x[1] + 1 for x in data])

# do the same for the negative counts
y = np.log([x[2] + 1 for x in data])

# Plot a dot for each pair of words
ax.scatter(x, y)

# assign axis labels
plt.xlabel("Log Positive count")
plt.ylabel("Log Negative count")
```

```
# Add the word as the label at the same position as you added the points just before
for i in range(0, len(data)):
    ax.annotate(data[i][0], (x[i], y[i]), fontsize=12)

ax.plot([0, 9], [0, 9], color = 'red')
# Plot the red line that divides the 2 areas.

plt.show( )
```

Scatter Plot



02-06
Logistic Regression Model

02 Sentiment Analysis with Logistic Regression

Import the Required Libraries

```
import nltk                          # NLP toolbox
from os import getcwd
import pandas as pd                  # Library for Dataframes
from nltk.corpus import twitter_samples
import matplotlib.pyplot as plt              # Library for visualization
import numpy as np                   # Library for math functions

#from utils import process_tweet, build_freqs # Our functions for NLP
# download the stopwords and twitter_samples for the process_tweet function
nltk.download('stopwords')
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Unzipping corpora/twitter_samples.zip.
True
```

NLTK Twitter Sample Dataset

```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')

tweets = all_positive_tweets + all_negative_tweets ## Concatenate the lists.
labels = np.append(np.ones((len(all_positive_tweets),1)), np.zeros((len(all_negative_tweets),1)), axis = 0)

# split the data into two pieces, one for training and one for testing (validation set)
train_pos  = all_positive_tweets[:4000]
train_neg  = all_negative_tweets[:4000]

train_x = train_pos + train_neg

print("Number of tweets: ", len(train_x))
```

```
Number of tweets:  8000
```

Extracted Features

```
data = pd.read_csv('logistic_features.csv'); # Load a 3 columns csv file using pandas function
data.head(10) # Print the first three data entries
```

	bias	positive	negative	sentiment
0	1.0	3020.0	61.0	1.0
1	1.0	3573.0	444.0	1.0
2	1.0	3005.0	115.0	1.0
3	1.0	2862.0	4.0	1.0
4	1.0	3119.0	225.0	1.0
5	1.0	2955.0	119.0	1.0
6	1.0	3934.0	538.0	1.0
7	1.0	3162.0	276.0	1.0
8	1.0	628.0	189.0	1.0
9	1.0	264.0	112.0	1.0

Data Frame to Numpy Arrays

```
# Each feature is labeled as bias, positive and negative
X = data[['bias', 'positive', 'negative']].values        # Get only the numerical values of the dataframe
Y = data['sentiment'].values;                # Put in Y the corresponding labels or sentiments

print(X.shape)        # Print the shape of the X part
print(X)              # Print some rows of X
```

```
(8000, 3)
[[1.000e+00 3.020e+03 6.100e+01]
 [1.000e+00 3.573e+03 4.440e+02]
 [1.000e+00 3.005e+03 1.150e+02]
 ...
 [1.000e+00 1.440e+02 7.830e+02]
 [1.000e+00 2.050e+02 3.890e+03]
 [1.000e+00 1.890e+02 3.974e+03]]
```

Pre-Trained LR Model

A Logistic regression model must be trained.

The next code contains the resulting model from such training.

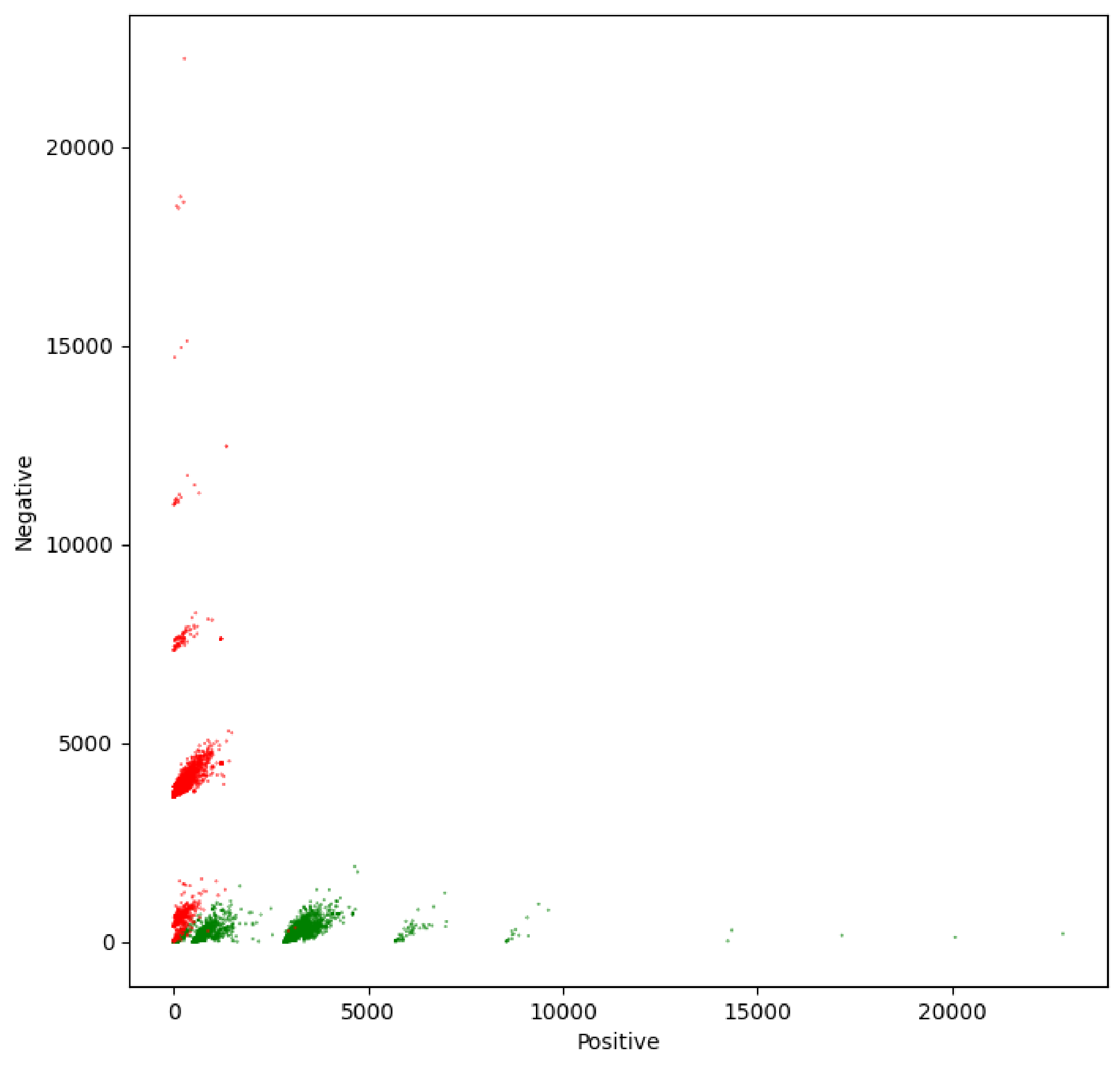
Notice that a list of 3 numeric values represents the whole model, that we have called theta θ .

```
theta = [7e-08, 0.0005239, -0.00055517]
```

Sample Scatter Plot

```
# Plot the samples using columns 1 and 2 of the matrix
fig, ax = plt.subplots(figsize = (8, 8))

colors = ['red', 'green']

# Color based on the sentiment Y
ax.scatter(X[:,1], X[:,2], c=[colors[int(k)] for k in Y], s = 0.1)
# Plot a dot for each pair of words
plt.xlabel("Positive")
plt.ylabel("Negative")
```

```
Text(0, 0.5, 'Negative')
```



Plot the Model Alongside the Data

```python
# Equation for the separation plane
# It give a value in the negative axe as a function of a positive value
# f(pos, neg, W) = w0 + w1 * pos + w2 * neg = 0
# s(pos, W) = (w0 - w1 * pos) / w2
def neg(theta, pos):
    return (-theta[0] - pos * theta[1]) / theta[2]
```

$$z = \theta * x = 0$$
$$x = [1, pos, neg]$$
$$z(\theta, x) = \theta_0 + \theta_1 * pos + \theta_2 * neg = 0$$
$$neg = (-\theta_0 - \theta_1 * pos)/\theta_2$$

```python
# Equation for the direction of the sentiments change
# We don't care about the magnitude of the change. We are only interested
# in the direction. So this direction is just a perpendicular function to the
# separation plane
# df(pos, W) = pos * w2 / w1
def direction(theta, pos):
    return    pos * theta[2] / theta[1]
```

$$direction = pos * \theta_2/\theta_1$$

Plot the Model Alongside the Data

```
fig, ax = plt.subplots(figsize = (8, 8))          # Plot the samples using columns 1 and 2 of the matrix
colors = ['red', 'green']
# Color base on the sentiment Y
ax.scatter(X[:,1], X[:,2], c=[colors[int(k)] for k in Y], s = 0.1)  # Plot a dot for each pair of words
plt.xlabel("Positive")
plt.ylabel("Negative")
maxpos = np.max(X[:,1])          # Now lets represent the logistic regression model in this chart.
offset = 5000                    # The pos value for the direction vectors origin

ax.plot([0,  maxpos], [neg(theta, 0),   neg(theta, maxpos)], color = 'gray')       # Plot a gray line that divides the 2 areas.
# Plot a green line pointing to the positive direction
ax.arrow(offset, neg(theta, offset), offset, direction(theta, offset), head_width=500, head_length=500, fc='g', ec='g')
# Plot a red line pointing to the negative direction
ax.arrow(offset, neg(theta, offset), -offset, -direction(theta, offset), head_width=500, head_length=500, fc='r', ec='r')
plt.show( )
```

Plot the Model Alongside the Data