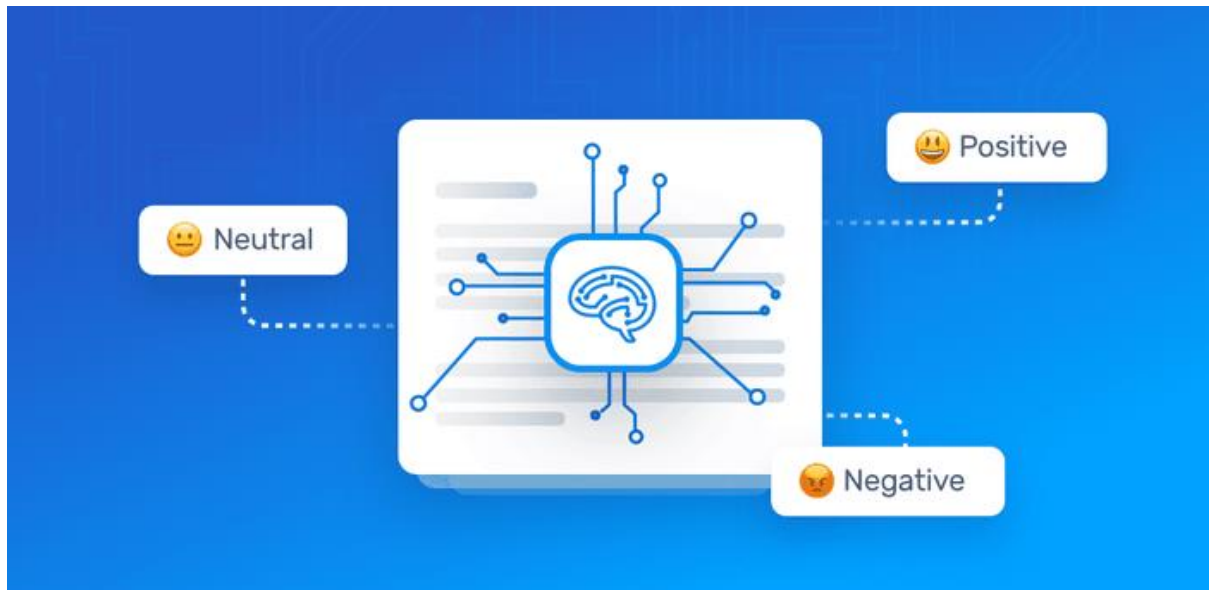


Sentiment Analysis & Machine Learning



[Sentiment analysis](#) is a [machine learning](#) tool that analyzes texts for polarity, from positive to negative. By training machine learning tools with examples of emotions in text, machines automatically learn how to detect sentiment without human input.

To put it simply, machine learning allows computers to learn new tasks without being expressly programmed to perform them. Sentiment analysis models can be trained to read beyond mere definitions, to understand things like, context, sarcasm, and misapplied words. For example:

"Super user-friendly interface. Yeah right. An engineering degree would be helpful."

Out of context, the words 'super user-friendly' and 'helpful' could be read as positive, but this is clearly a negative comment. Using sentiment analysis, computers can automatically process text data and understand it just as a human would, saving hundreds of employee hours.

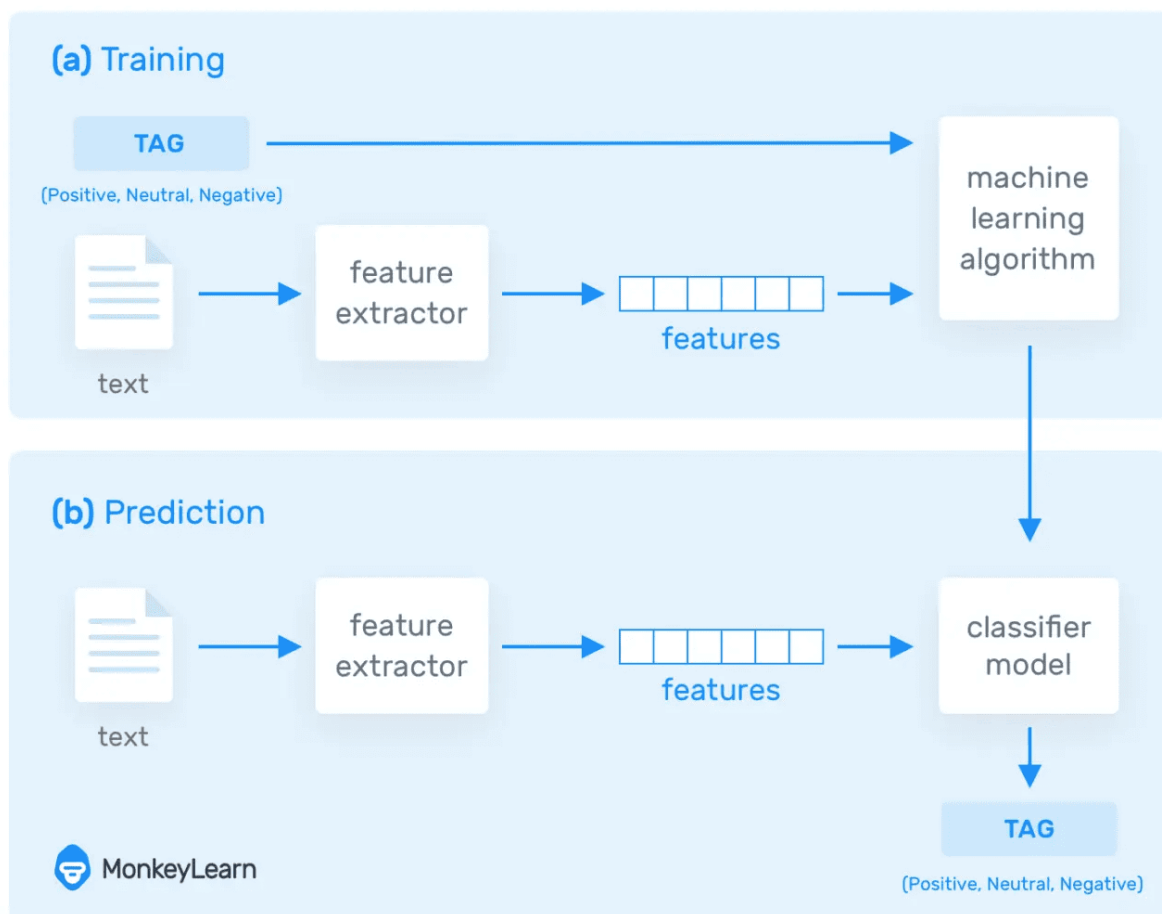
Imagine using machine learning to process customer service tickets, categorize them in order of urgency, and automatically route them to the correct department or employee. Or, to analyze thousands of product reviews and social media posts to [gauge brand sentiment](#).

Read on to learn more about how machine learning works and how it can help your business

How Does Sentiment Analysis with Machine Learning Work?

There are a number of techniques and complex algorithms used to command and train machines to perform sentiment analysis. There are pros and cons to each. But, used together, they can provide exceptional results. Below are some of the most used algorithms.

How Does Sentiment Analysis Work?



Naive Bayes

Naive Bayes is a fairly simple group of probabilistic algorithms that, for sentiment analysis classification, assigns a probability that a given word or phrase should be considered positive or negative.

Essentially, this is how Bayes' theorem works. *The probability of A, if B is true, is equal to the probability of B, if A is true, times the probability of A being true, divided by the probability of B being true:*

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

But that's a lot of math! Basically, Naive Bayes calculates words against each other. So, with machine learning models trained for word polarity, we can calculate the likelihood that a word, phrase, or text is positive or negative.

When techniques like lemmatization, stopwords removal, and [TF-IDF](#) are implemented, Naive Bayes becomes more and more predictively accurate.

Linear Regression

Linear regression is a statistical algorithm used to predict a *Y* value, given *X* features. Using machine learning, the data sets are examined to show a relationship. The relationships are then placed along the *X/Y* axis, with a straight line running through them to predict further relationships.

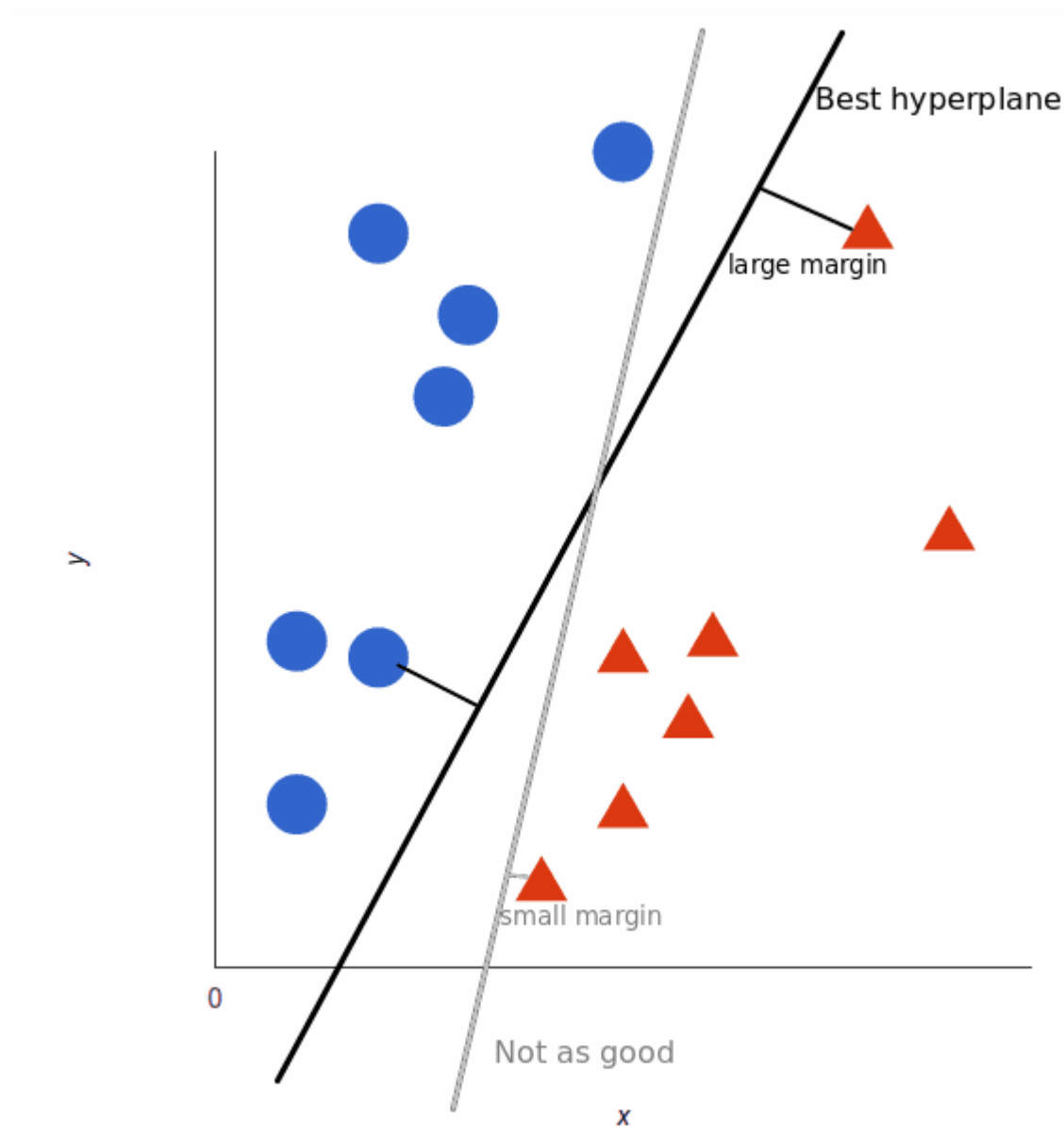
Linear regression calculates how the *X* input (words and phrases) relates to the *Y* output (polarity). This will determine where words and phrases fall on a scale of polarity from "really positive" to "really negative" and everywhere in between.

Support Vector Machines (SVM)

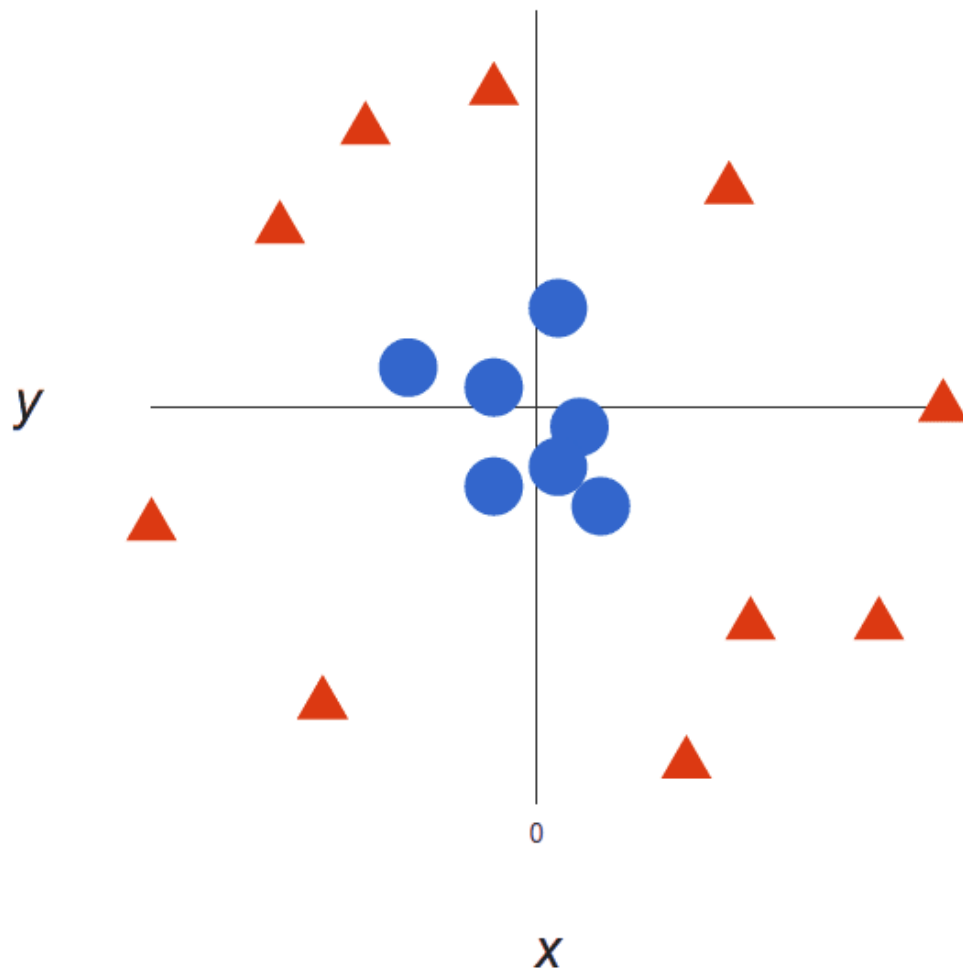
A support vector machine is another supervised machine learning model, similar to linear regression but more advanced. SVM uses algorithms to train and classify text within our sentiment polarity model, taking it a step beyond *X/Y* prediction.

For a simple visual explanation, we'll use two tags: *red* and *blue*, with two data features: *X* and *Y*. We'll train our classifier to output an *X/Y* coordinate as either *red* or *blue*.

In order to maximize machine learning, the best hyperplane is the one with the largest distance between each tag:

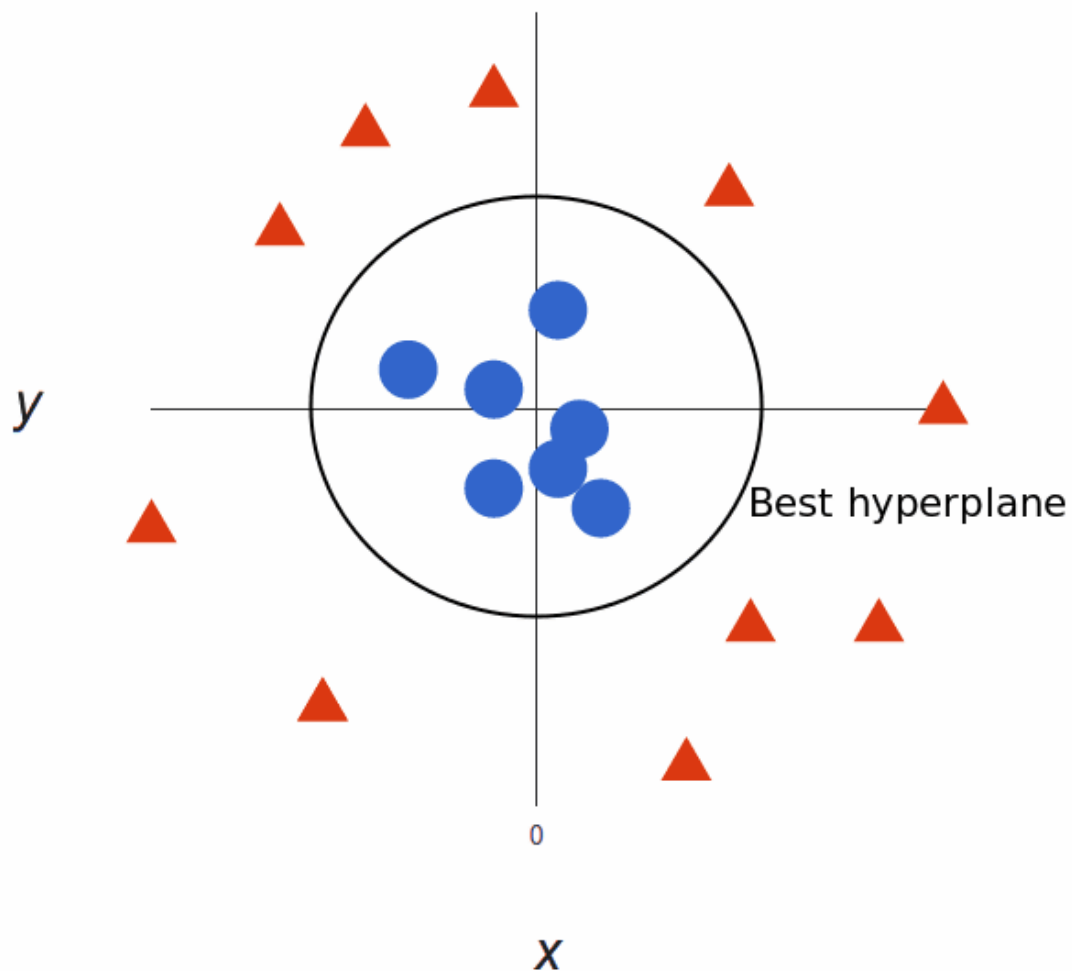


However, as data sets become more complex, it may not be possible to draw a single line to classify the data into two camps:



Using SVM, the more complex the data, the more accurate the predictor will become. Imagine the above in three dimensions, with a Z axis added, so it becomes a circle.

Mapped back to two dimensions with the best hyperplane, it looks like this:



Very simply put, SVM allows for more accurate machine learning because it's multidimensional.

Deep Learning

Deep learning is a subfield of machine learning that aims to calculate data as the human brain does using "artificial neural networks."

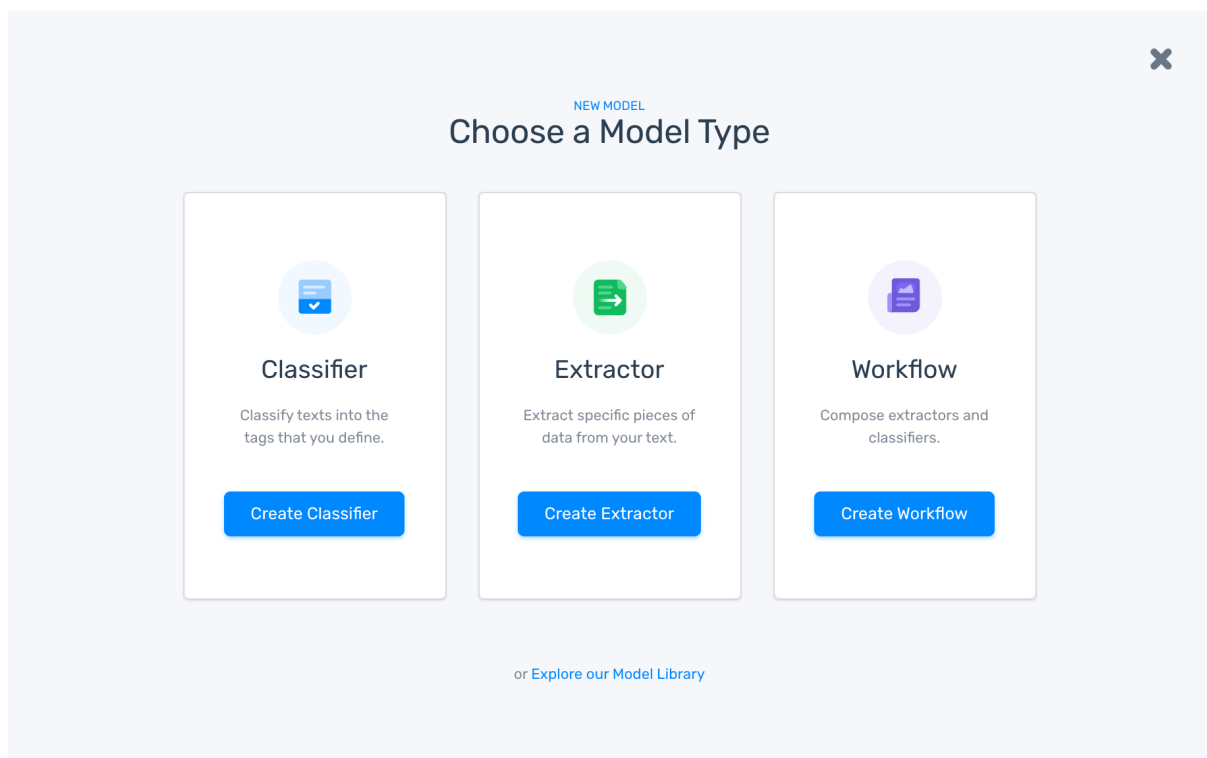
Deep learning is *hierarchical* machine learning. In other words, it's multi-level, and allows a machine to automatically 'chain' a number of human-created processes together. By allowing multiple algorithms to be used progressively, while moving from step to step, deep learning is able to solve complex problems in the same way humans do.

Sentiment Analysis with Machine Learning Tutorial

As you can see from the above, the calculations and algorithms involved in sentiment analysis are quite complex. But with user-friendly tools, sentiment analysis with machine learning is accessible to everyone, whether you have a computer science background or not.

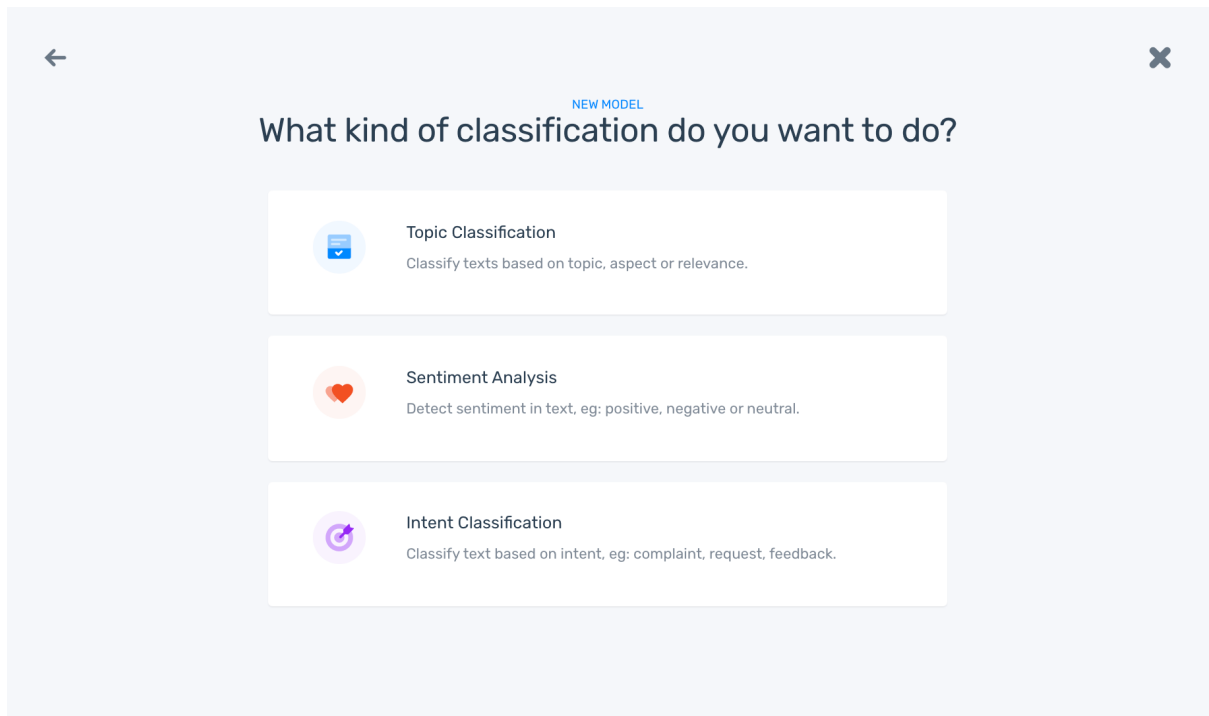
1. Choose your model

Once you've signed up, go to the dashboard and choose 'Create a model', then click 'Classifier,':



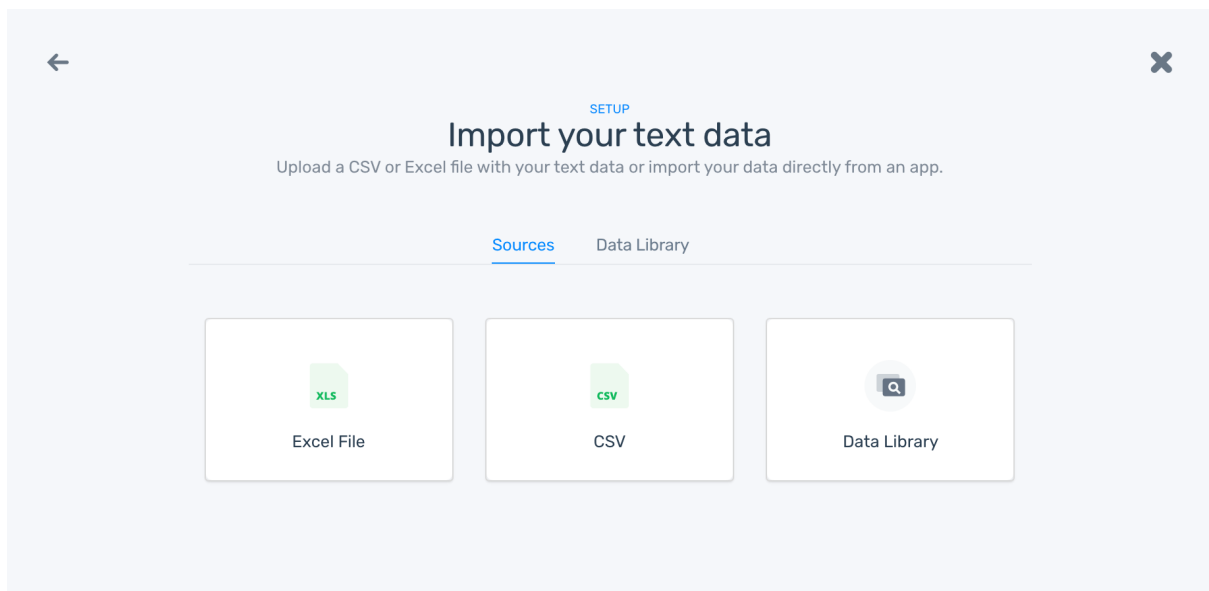
2. Choose your classifier

We want to show how machine learning works on customer opinions, so click on 'Sentiment Analysis':



3. Import your data

You can import data from an app or upload a CSV or Excel file. This will be used to train your sentiment analysis model.

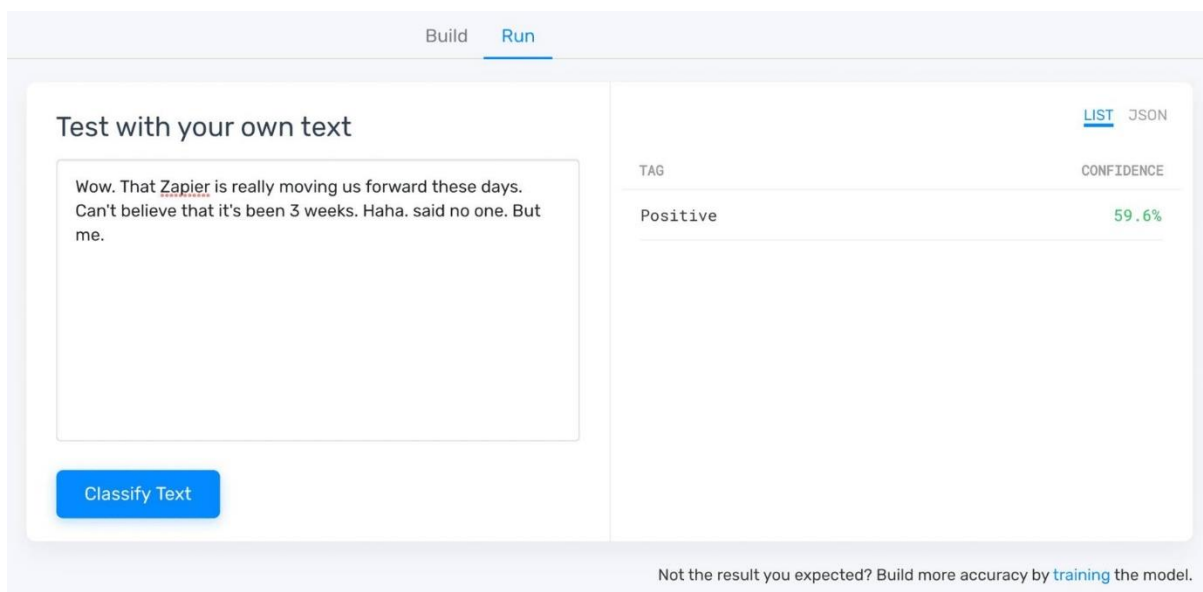


4. Tag tweets to train your sentiment analysis classifier

Here's where we see machine learning at work. Tag each tweet as *Positive*, *Negative*, or *Neutral* to train your model based on the opinion within the text. Once you tag a few, the model will begin making its own predictions. Correct them, if the model has tagged them wrong:

5. Test your classifier

Once the model has been trained with some examples, you can paste your own text to see how they're classified. If it's not tagging correctly, you can keep training. The more you train the model, the better its predictions will become:



The screenshot shows the MonkeyLearn interface in the 'Run' tab. On the left, under 'Test with your own text', there is a text input area containing the tweet: 'Wow. That Zapier is really moving us forward these days. Can't believe that it's been 3 weeks. Haha. said no one. But me.' Below the text is a blue button labeled 'Classify Text'. On the right, the results are displayed in a table with two columns: 'TAG' and 'CONFIDENCE'. The table shows a single result: 'Positive' with a confidence of '59.6%'. At the top right of the results section are links for 'LIST' and 'JSON'. At the bottom of the interface, a message reads: 'Not the result you expected? Build more accuracy by training the model.'

TAG	CONFIDENCE
Positive	59.6%

MonkeyLearn shows a number of [sentiment analysis statistics](#) to help understand how well machine learning is working: *Precision* and *Recall* are tag level statistics, and *Accuracy* and *F1 Score* are statistics on the overall model. The keyword cloud helps visualize the most used words.

In the example below more tags are needed for *Negative*.

[API](#): easy programming for quick plug-in analysis:

Code Examples

Request Example

Curl Python Ruby PHP Node.js Java

```
1 curl --data '{"data": [{"@Balgev @Zendesk @Zendesk is changing the game! @Balgev are you using any of their other offerings?}"]}' \
2 -H "Authorization:Token " \
3 -H "Content-Type: application/json" \
4 -D - \
5 "https://api.monkeylearn.com/v3/classifiers/cl_g0m22u2h/classify/"
```

Put Machine Learning to Work for You

Sentiment analysis using machine learning can help any business analyze public opinion, improve customer support, and automate tasks with fast turnarounds. Not only saving you time, but also money. Sentiment analysis results will also give you real actionable insights, helping you make the right decisions.

While machine learning can be complex, SaaS tools make it simple for everyone to use.

Tools are also completely scalable, and can be effortlessly configured to your specific needs.

Example:

A Twitter sentiment analysis determines negative, positive, or neutral emotions within the text of a tweet using NLP and ML models. Sentiment analysis or opinion mining refers to identifying as well as classifying the sentiments that are expressed in the text source. Tweets are often useful in generating a vast amount of sentiment data upon analysis. These data are useful in understanding the opinion of people on social media for a variety of topics.

What is Twitter Sentiment Analysis?

Twitter sentiment analysis analyzes the sentiment or emotion of tweets. It uses natural language processing and machine learning algorithms to classify tweets automatically as positive, negative, or neutral based on their content. It can be done for individual tweets or a larger dataset related to a particular topic or event.

Why is Twitter Sentiment Analysis Important?

1. **Understanding Customer Feedback:** By analyzing the sentiment of customer feedback, companies can identify areas where they need to improve their products or services.
2. **Reputation Management:** Sentiment analysis can help companies monitor their brand reputation online and quickly respond to negative comments or reviews.
3. **Political Analysis:** Sentiment analysis can help political campaigns understand public opinion and tailor their messaging accordingly.
4. **Crisis Management:** In the event of a crisis, sentiment analysis can help organizations monitor social media and news outlets for negative sentiment and respond appropriately.
5. **Marketing Research:** Sentiment analysis can help marketers understand consumer behavior and preferences, and develop targeted advertising campaigns.

How to Do Twitter Sentiment Analysis?

In this article, we aim to analyze Twitter sentiment analysis using machine learning algorithms, the sentiment of tweets provided from the **Sentiment140 dataset** by developing a machine learning pipeline involving the use of three classifiers (**Logistic Regression, Bernoulli Naive Bayes, and SVM**) along with using **Term**

Frequency- Inverse Document Frequency (TF-IDF). The performance of these classifiers is then evaluated using **accuracy** and **F1 Scores**.

For data preprocessing, we will be using Natural Language Processing's (NLP) NLTK library.

Twitter Sentiment Analysis: Problem Statement

In this project, we try to implement an NLP **Twitter sentiment analysis model** that helps to overcome the challenges of sentiment classification of tweets. We will be classifying the tweets into positive or negative sentiments. The necessary details regarding the dataset involving the Twitter sentiment analysis project are:

The dataset provided is the **Sentiment140 Dataset** which consists of **1,600,000 tweets** that have been extracted using the Twitter API. The various columns present in this Twitter data are:

- **target:** the polarity of the tweet (positive or negative)
- **ids:** Unique id of the tweet
- **date:** the date of the tweet
- **flag:** It refers to the query. If no such query exists, then it is NO QUERY.
- **user:** It refers to the name of the user that tweeted
- **text:** It refers to the text of the tweet

Twitter Sentiment Analysis: Project Pipeline

The various steps involved in the **Machine Learning Pipeline** are:

- Import Necessary Dependencies
- Read and Load the Dataset
- Exploratory Data Analysis

- Data Visualization of Target Variables
- Data Preprocessing
- Splitting our data into Train and Test sets.
- Transforming Dataset using TF-IDF Vectorizer
- Function for Model Evaluation
- Model Building
- Model Evaluation

Let's get started,

Step-1: Import the Necessary Dependencies

```
# utilities
import re
import numpy as np
import pandas as pd
# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
# nltk
from nltk.stem import WordNetLemmatizer
# sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

Step-2: Read and Load the Dataset

```
# Importing the dataset
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('Project_Data.csv', encoding=DATASET_ENCODING,
names=DATASET_COLUMNS)
df.sample(5)
```

Output:

	target	ids	date	flag	user	text
305165	0	1999924339	Mon Jun 01 21:04:24 PDT 2009	NO_QUERY	twentyred25	man my b-day is coming up but i dont know what...
673186	0	2217413241	Fri Jun 19 19:03:37 PDT 2009	NO_QUERY	belenneleb	@officiallut_ i need they to come back here. ...
573387	0	2208824277	Wed Jun 17 10:50:29 PDT 2009	NO_QUERY	TeenieWehine	@krystyn13 Sorry to hear that
246882	0	1982346860	Sun May 31 11:01:36 PDT 2009	NO_QUERY	BrianWCollins	@TheJoeLynch I've only seen 3 (Leon, 5th Eleme...
669112	0	2246153162	Fri Jun 19 17:10:15 PDT 2009	NO_QUERY	heidioftheopera	kinda feels bad for missing out on the Solistic...

Step-3: Exploratory Data Analysis

3.1: Five top records of data

```
df.head()
```

Output:

	target	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	TheSpecialOne	@switchfoot http://twitpic.com/2y1zi - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all...

3.2: Columns/features in data

```
df.columns
```

Output:

```
Index(['target', 'ids', 'date', 'flag', 'user', 'text'], dtype='object')
```

3.3: Length of the dataset

```
print('length of data is', len(df))
```

Output:

```
length of data is 1048576
```

3.4: Shape of data

```
df.shape
```

Output:

```
(1048576, 6)
```

3.5: Data information

```
df.info()
```

Output:


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048576 entries, 0 to 1048575
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   target      1048576 non-null    int64
1   ids         1048576 non-null    int64
2   date        1048576 non-null    object
3   flag        1048576 non-null    object
4   user        1048576 non-null    object
5   text        1048576 non-null    object
dtypes: int64(2), object(4)
memory usage: 48.0+ MB
```

3.6: Datatypes of all columns

```
df.dtypes
```

Output:

```
target      int64
ids         int64
date        object
flag        object
user        object
text        object
dtype: object
```

3.7: Checking for null values

```
np.sum(df.isnull().any(axis=1))
```

Output:

```
0
```

3.8: Rows and columns in the dataset

```
print('Count of columns in the data is: ', len(df.columns))
print('Count of rows in the data is: ', len(df))
```

Output:

```
Count of columns in the data is: 6
Count of rows in the data is: 1048576
```

3.9: Check unique target values

```
df['target'].unique()
```

Output:

```
array([0, 4], dtype=int64)
```

3.10: Check the number of target values

```
df['target'].nunique()
```

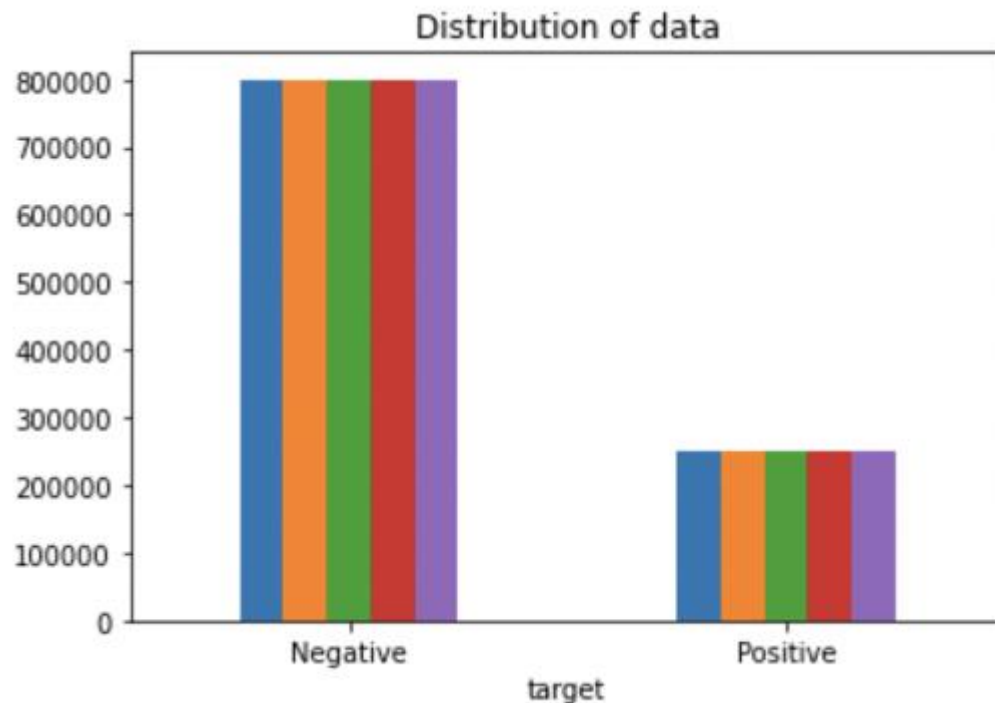
Output:

```
2
```

Step-4: Data Visualization of Target Variables

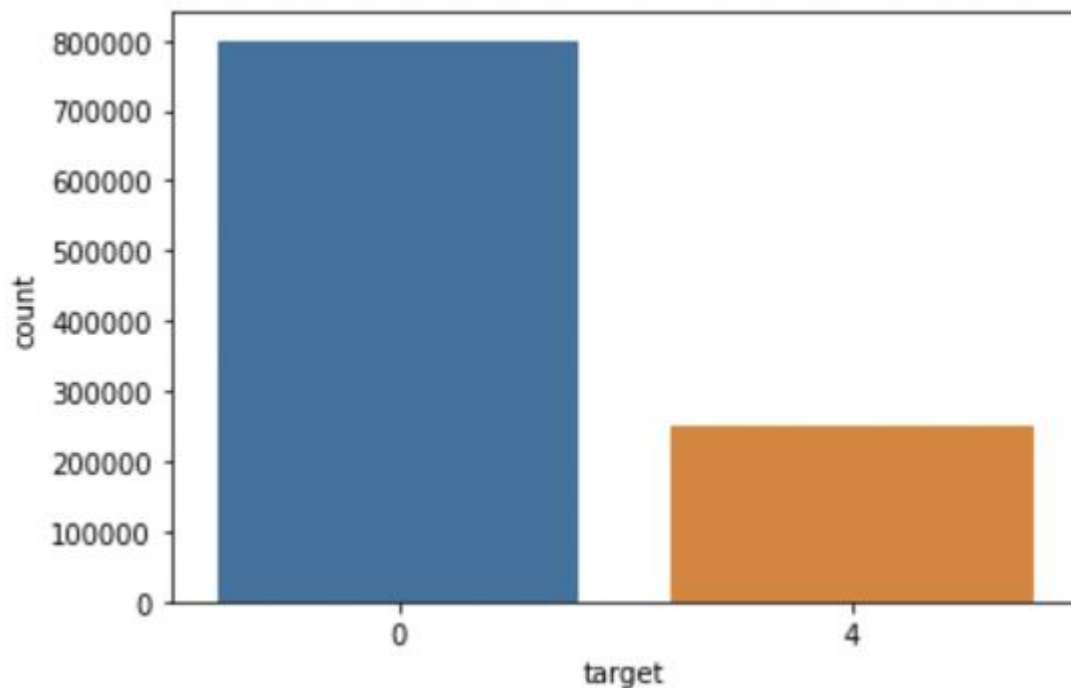
```
# Plotting the distribution for dataset.  
ax = df.groupby('target').count().plot(kind='bar', title='Distribution of  
data', legend=False)  
ax.set_xticklabels(['Negative', 'Positive'], rotation=0)  
# Storing data in lists.  
text, sentiment = list(df['text']), list(df['target'])
```

Output:



```
import seaborn as sns  
sns.countplot(x='target', data=df)
```

Output:



Step-5: Data Preprocessing

In the above-given problem statement, before training the model, we performed various pre-processing steps on the dataset that mainly dealt with removing stopwords, removing special characters like emojis, hashtags, etc. The text document is then converted into lowercase for better generalization.

Subsequently, the punctuations were cleaned and removed, thereby reducing the unnecessary noise from the dataset. After that, we also removed the repeating characters from the words along with removing the URLs as they do not have any significant importance.

At last, we then performed **Stemming(reducing the words to their derived stems)** and **Lemmatization(reducing the derived words to their root form, known as lemma)** for better results.

5.1: Selecting the text and Target column for our further analysis

```
data=df[['text','target']]
```

5.2: Replacing the values to ease understanding. (Assigning 1 to Positive sentiment 4)

```
data['target'] = data['target'].replace(4,1)
```

5.3: Printing unique values of target variables

```
data['target'].unique()
```

Output:

```
array([0, 1], dtype=int64)
```

5.4: Separating positive and negative tweets

```
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
```

5.5: Taking one-fourth of the data so we can run it on our machine easily

```
data_pos = data_pos.iloc[:int(20000)]
data_neg = data_neg.iloc[:int(20000)]
```

5.6: Combining positive and negative tweets

```
dataset = pd.concat([data_pos, data_neg])
```

5.7: Making statement text in lowercase

```
dataset['text']=dataset['text'].str.lower()
dataset['text'].tail()
```

Output:

```
19995    not much time off this weekend, work trip to m...
19996                                one more day of holidays
19997    feeling so down right now .. i hate you damn h...
19998    geez,i hv to read the whole book of personalit...
19999    i threw my sign at donnie and he bent over to ...
Name: text, dtype: object
```

5.8: Defining set containing all stopwords in English.

```
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
                'and', 'any', 'are', 'as', 'at', 'be', 'because', 'been', 'before',
                'being', 'below', 'between', 'both', 'by', 'can', 'd', 'did', 'do',
                'does', 'doing', 'down', 'during', 'each', 'few', 'for', 'from',
                'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
                'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
                'into', 'is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
                'me', 'more', 'most', 'my', 'myself', 'now', 'o', 'of', 'on', 'once',
```

```

'only', 'or', 'other', 'our', 'ours', 'ourselves', 'out', 'own',
're', 's', 'same', 'she', "shes", 'should', "shouldve", 'so', 'some', 'such',
't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was',
'we', 'were', 'what', 'when', 'where', 'which', 'while', 'who', 'whom',
'why', 'will', 'with', 'won', 'y', 'you', "youd", "youll", "youre",
"youve", 'your', 'yours', 'yourself', 'yourselves']

```

5.9: Cleaning and removing the above stop words list from the tweet text

```

STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()

```

Output:

```

800000          love @health4uandpets u guys r best!!
800001    im meeting one besties tonight! cant wait!! - ...
800002    @darealsunisakim thanks twitter add, sunisa! g...
800003    sick really cheap hurts much eat real food plu...
800004          @lovesbrooklyn2 effect everyone
Name: text, dtype: object

```

5.10: Cleaning and removing punctuations

```

import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()

```

Output:

```

19995    not much time off weekend work trip malmið fr...
19996          one day holidays
19997          feeling right  hate damn humprey
19998    geezi hv read whole book personality types emb...
19999    threw sign donnie bent over get but thingee ma...
Name: text, dtype: object

```

5.11: Cleaning and removing repeating characters

```

def cleaning_repeating_char(text):
    return re.sub(r'(.){1,}', r'1', text)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'].tail()

```

Output:

```
19995    not much time of wekend work trip malmið½ fris...
19996                                           one day holidays
19997                                           feling right hate damn humprey
19998    gezi hv read whole bok personality types embar...
19999    threw sign donie bent over get but thinge made...
Name: text, dtype: object
```

5.12: Cleaning and removing URLs

```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^\s]+))', ' ', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()
```

Output:

```
19995    not much time of wekend work trip malmið½ fris...
19996                                           one day holidays
19997                                           feling right hate damn humprey
19998    gezi hv read whole bok personality types embar...
19999    threw sign donie bent over get but thinge made...
Name: text, dtype: object
```

5.13: Cleaning and removing numeric numbers

```
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```

Output:

```
19995    not much time of wekend work trip malmið½ fris...
19996                                           one day holidays
19997                                           feling right hate damn humprey
19998    gezi hv read whole bok personality types embar...
19999    threw sign donie bent over get but thinge made...
Name: text, dtype: object
```

5.14: Getting tokenization of tweet text

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'w+')
dataset['text'] = dataset['text'].apply(tokenizer.tokenize)
dataset['text'].head()
```


Output:

```
800000      [love, healthuandpets, u, guys, r, best]
800001      [im, meting, one, besties, tonight, cant, wait...
800002      [darealsunisakim, thanks, twiter, ad, sunisa, ...
800003      [sick, realy, cheap, hurts, much, eat, real, f...
800004      [lovesbrooklyn, efect, everyone]
Name: text, dtype: object
```

5.15: Applying stemming

```
import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
```

Output:

```
800000      [love, healthuandpets, u, guys, r, best]
800001      [im, meting, one, besties, tonight, cant, wait...
800002      [darealsunisakim, thanks, twiter, ad, sunisa, ...
800003      [sick, realy, cheap, hurts, much, eat, real, f...
800004      [lovesbrooklyn, efect, everyone]
Name: text, dtype: object
```

5.16: Applying lemmatizer

```
lm = nltk.WordNetLemmatizer()
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
dataset['text'].head()
```

Output:

```
800000      [love, healthuandpets, u, guys, r, best]
800001      [im, meting, one, besties, tonight, cant, wait...
800002      [darealsunisakim, thanks, twiter, ad, sunisa, ...
800003      [sick, realy, cheap, hurts, much, eat, real, f...
800004      [lovesbrooklyn, efect, everyone]
Name: text, dtype: object
```

5.17: Separating input feature and label

```
X=data.text
y=data.target
```

5.18: Plot a cloud of words for negative tweets

```
data_neg = data['text'][:800000]
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
```

Output:



5.19: Plot a cloud of words for positive tweets

```
data_pos = data['text'][800000:]
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_pos))
plt.figure(figsize = (20,20))
plt.imshow(wc)
```

Output:

- Confusion Matrix with Plot
- ROC-AUC Curve

```
def model_Evaluate(model):
# Predict values for Test dataset
y_pred = model.predict(X_test)
# Print the evaluation metrics for the dataset.
print(classification_report(y_test, y_pred))
# Compute and plot the Confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
categories = ['Negative', 'Positive']
group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() /
np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names, group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues', fmt = '',
xticklabels = categories, yticklabels = categories)
plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)
plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Step-9: Model Building

In the problem statement, we have used three different models respectively :

- Bernoulli Naive Bayes Classifier
- SVM (Support Vector Machine)
- Logistic Regression

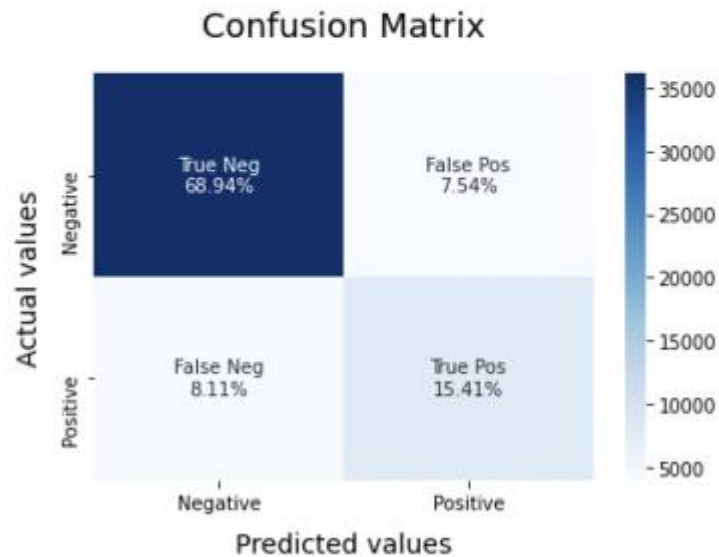
The idea behind choosing these models is that we want to try all the classifiers on the dataset ranging from simple ones to complex models, and then try to find out the one which gives the best performance among them.

8.1: Model-1

```
BNBmodel = BernoulliNB()
BNBmodel.fit(X_train, y_train)
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

Output:

	precision	recall	f1-score	support
0	0.89	0.90	0.90	40097
1	0.67	0.66	0.66	12332
accuracy			0.84	52429
macro avg	0.78	0.78	0.78	52429
weighted avg	0.84	0.84	0.84	52429



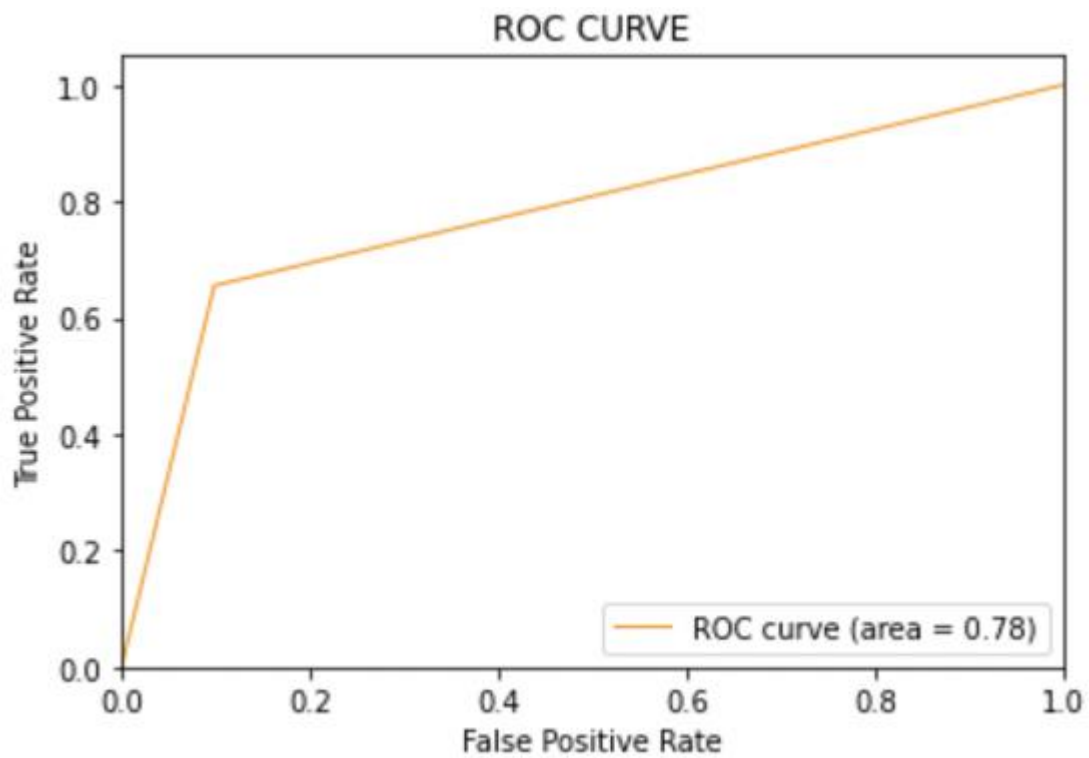
8.2: Plot the ROC-AUC Curve for model-1

```

from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred1)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```

Output:



8.3: Model-2:

```
SVCmodel = LinearSVC()  
SVCmodel.fit(X_train, y_train)  
model_Evaluate(SVCmodel)  
y_pred2 = SVCmodel.predict(X_test)
```

Output:

	precision	recall	f1-score	support
0	0.89	0.93	0.91	40097
1	0.74	0.63	0.68	12332
accuracy			0.86	52429
macro avg	0.81	0.78	0.80	52429
weighted avg	0.86	0.86	0.86	52429



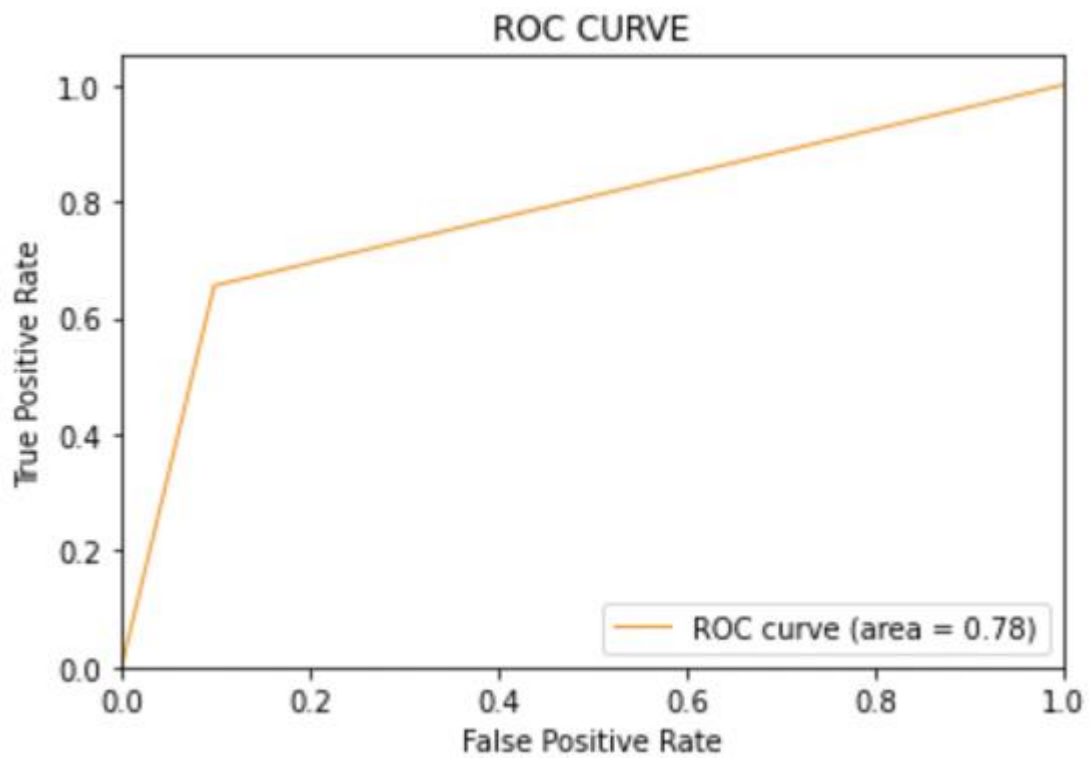
8.4: Plot the ROC-AUC Curve for model-2

```

from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```

Output:

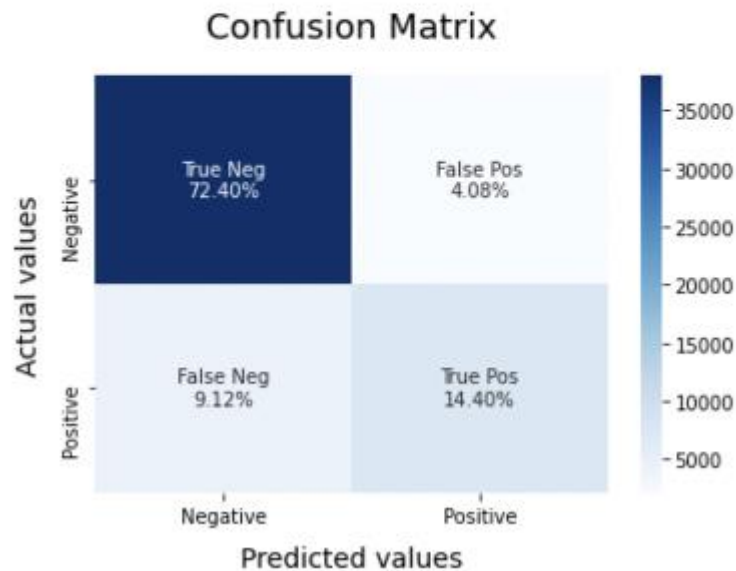


8.5: Model-3

```
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
```

Output:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	40097
1	0.78	0.61	0.69	12332
accuracy			0.87	52429
macro avg	0.83	0.78	0.80	52429
weighted avg	0.86	0.87	0.86	52429



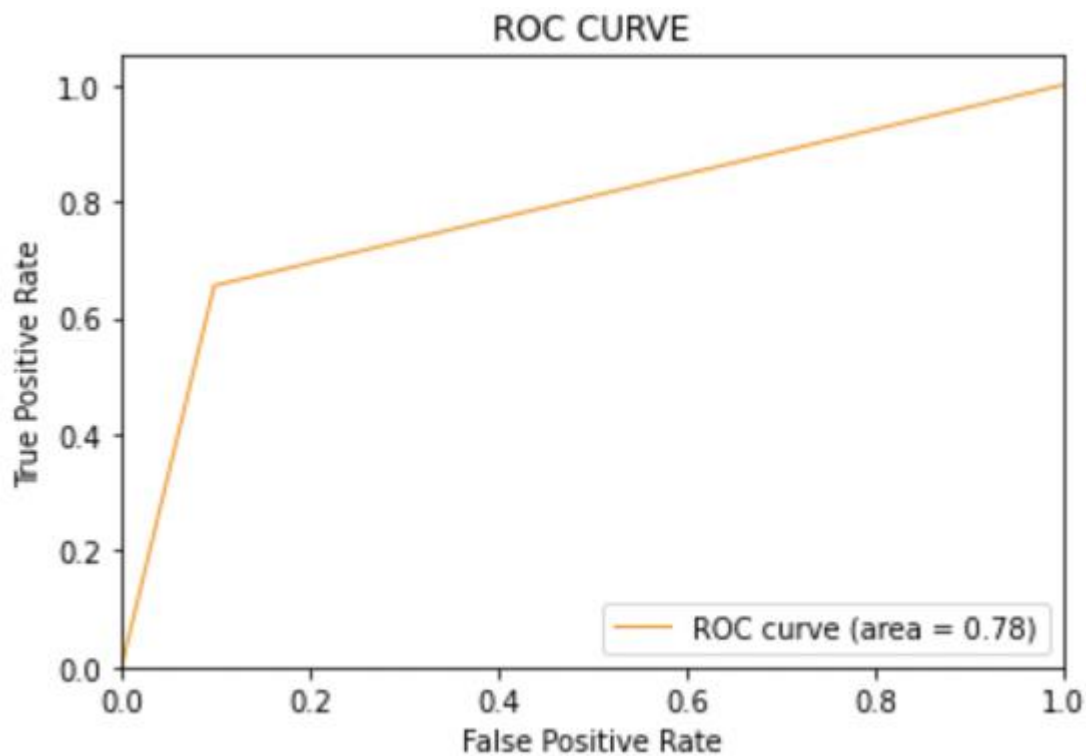
8.6: Plot the ROC-AUC Curve for model-3

```

from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```

Output:



Step-10: Model Evaluation

Upon evaluating all the models, we can conclude the following details i.e.

Accuracy: As far as the accuracy of the model is concerned, Logistic Regression performs better than SVM, which in turn performs better than Bernoulli Naive Bayes.

F1-score: The F1 Scores for class 0 and class 1 are :

- (a) For class 0: Bernoulli Naive Bayes (accuracy = 0.90) < SVM (accuracy = 0.91) < Logistic Regression (accuracy = 0.92)
- (b) For class 1: Bernoulli Naive Bayes (accuracy = 0.66) < SVM (accuracy = 0.68) < Logistic Regression (accuracy = 0.69)

AUC Score: All three models have the same ROC-AUC score.

We, therefore, conclude that the Logistic Regression is the best model for the above-given dataset.

In our problem statement, **Logistic Regression** follows the principle of **Occam's Razor**, which defines that for a particular problem statement, if the data has no assumption, then the simplest model works the best. Since our dataset does not have any assumptions and Logistic Regression is a simple model. Therefore, the concept holds true for the above-mentioned dataset.

Conclusion

We hope through this article, you got a basic of how Sentimental Analysis is used to understand public emotions behind people's tweets. As you've read in this article, Twitter Sentimental Analysis helps us preprocess the data (tweets) using different methods and feed it into ML models to give the best accuracy.

Key Takeaways

- Twitter Sentimental Analysis is used to identify as well as classify the sentiments that are expressed in the text source.
- Logistic Regression, SVM, and Naive Bayes are some of the ML algorithms that can be used for Twitter Sentimental Analysis.