



Open in app

Get started



Published in Towards Data Science



Sultan Al Awar

Follow

Mar 20 · 10 min read · Listen



Save



Sentiment Analysis on News Headlines: Classic Supervised Learning vs Deep Learning Approach

An explanatory guide to develop a binary classifier to detect positive and negative news headlines using classic machine learning and deep learning techniques



[Open in app](#)[Get started](#)



Open in app

Get started

Introduction

*This article will enable you to build a binary classifier that performs sentiment analysis on **unlabelled data** with two different approaches:*

*1- Supervised Learning through **scikit-learn library and nlp packages***

*2- Deep Learning using **TensorFlow and Keras frameworks***

However, the challenge lies that we are dealing with unlabelled data, so we will utilize a **weak supervision technique** namely **Snorkel** to create labels as 0 (negative) or 1 (positive) for our training data points.

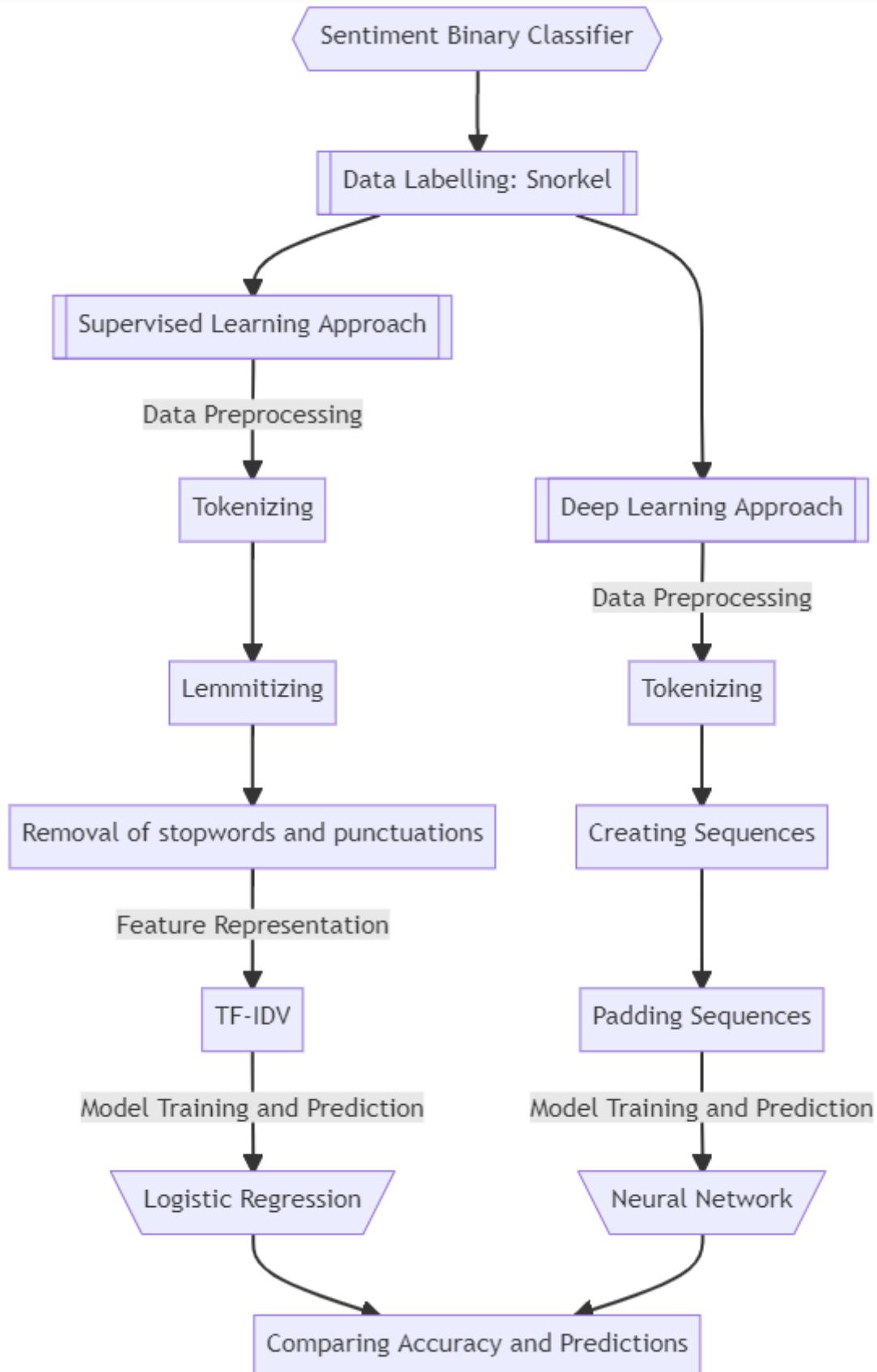
A visual plan of the project is presented in the below chart:





Open in app

Get started





Open in app

Get started

notebook, replicate and run the code on your own.

```
#install needed packages
!pip install snorkel
!pip install textblob
#import libraries and modules
from google.colab import files
import io
import pandas as pd
#Snorkel
from snorkel.labeling import LabelingFunction
import re
from snorkel.preprocess import preprocessor
from textblob import TextBlob
from snorkel.labeling import PandasLFApplier
from snorkel.labeling.model import LabelModel
from snorkel.labeling import LFAalysis
from snorkel.labeling import filter_unlabeled_dataframe
from snorkel.labeling import labeling_function
#NLP packages
import spacy
from nltk.corpus import stopwords
import string
import nltk
import nltk.tokenize
punc = string.punctuation
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
#Supervised learning
from tqdm import tqdm_notebook as tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
##Deep learning libraries and APIs
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

1. Load the data





Open in app

Get started

```
# store the dataset as a Pandas Dataframe
df = pd.read_csv(io.BytesIO(uploaded['data.csv']))

#conduct some data cleaning
df = df.drop(['publish_date', 'Unnamed: 2'], axis=1)
df = df.rename(columns = {'headline_text': 'text'})
df['text'] = df['text'].astype(str)

#check the data info
df.info()
```

Here we can check that our dataset has 63821 instances.

```
#check the data info
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63821 entries, 0 to 63820
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype
---  -
0    text    63821 non-null    object
dtypes: object(1)
memory usage: 498.7+ KB
```

Dataset Info

2. Create Labels: Snorkel Technique

Because the dataset is unlabelled, we will employ Snorkel, to come up with heuristics and programmatic rules using functions which assign labels of two classes that differentiate if the headline is positive (1) or negative (0). For more details about Snorkel Labelling, visit this [link](#).

Below, you can find the generated labelling functions. The first function looks into the input words in a headline, and the second one assigns the proper label according to the predefined words in the positive and negative lists. For example, if the word 'promising' is





Open in app

Get started

```
#define constants to represent the class labels :positive, negative,
and abstain
POSITIVE = 1
NEGATIVE = 0
ABSTAIN = -1
#define function which looks into the input words to represent a
proper label
def keyword_lookup(x, keywords, label):
    if any(word in x.text.lower() for word in keywords):
        return label
    return ABSTAIN
#define function which assigns a correct label
def make_keyword_lf(keywords, label=POSITIVE):
    return LabelingFunction(
        name=f"keyword_{keywords[0]}",
        f=keyword_lookup,
        resources=dict(keywords=keywords, label=label))
#resource: https://www.snorkel.org/use-cases/01-spam-tutorial#3-writing-more-labeling-functions

#these two lists can be further extended
"""positive news might contain the following words' """
keyword_positive = make_keyword_lf(keywords=['boosts', 'great',
'develops', 'promising', 'ambitious', 'delighted', 'record', 'win',
'breakthrough', 'recover', 'achievement', 'peace', 'party', 'hope',
'flourish', 'respect', 'partnership', 'champion', 'positive', 'happy',
'bright', 'confident', 'encouraged', 'perfect', 'complete', 'assured'
])

"""negative news might contain the following words"""
keyword_negative = make_keyword_lf(keywords=['war', 'solidiers',
'turmoil', 'injur', 'trouble', 'aggressive', 'killed', 'coup',
'evasion', 'strike', 'troops', 'dismisses', 'attacks', 'defeat',
'damage', 'dishonest', 'dead', 'fear', 'foul', 'fails', 'hostile',
'cuts', 'accusations', 'victims', 'death', 'unrest', 'fraud',
'dispute', 'destruction', 'battle', 'unhappy', 'bad', 'alarming',
'angry', 'anxious', 'dirty', 'pain', 'poison', 'unfair', 'unhealthy'
], label=NEGATIVE)
```

Another set of labelling functions were implemented through TextBlob tool, a pretrained sentiment analyzer. We will create a Pre-processor that runs TextBlob on our headlines, then extracts the polarity and subjectivity scores.





Open in app

Get started

```
def textblob_sentiment(x):
    scores = TextBlob(x.text)
    x.polarity = scores.sentiment.polarity
    x.subjectivity = scores.sentiment.subjectivity
    return x

#find polarity
@labeling_function(pre=[textblob_sentiment])
def textblob_polarity(x):
    return POSITIVE if x.polarity > 0.6 else ABSTAIN

#find subjectivity
@labeling_function(pre=[textblob_sentiment])
def textblob_subjectivity(x):
    return POSITIVE if x.subjectivity >= 0.5 else ABSTAIN
```

The next step is to combine all the labelling functions and apply it on our dataset. Then, we fit the `label_model` to predict and generate the positive and negative classes.

```
#combine all the labeling functions
lfs = [keyword_positive, keyword_negative, textblob_polarity,
textblob_subjectivity ]

#apply the lfs on the dataframe
applier = PandasLFApplier(lfs=lfs)
L_snorkel = applier.apply(df=df)

#apply the label model
label_model = LabelModel(cardinality=2, verbose=True)
#fit on the data
label_model.fit(L_snorkel)
#predict and create the labels
df["label"] = label_model.predict(L=L_snorkel)
```

We can notice that after dropping the unlabelled data points (as shown below), we have around 12300 positive labels and 6900 negative which will be sufficient to build our sentiment classifier.

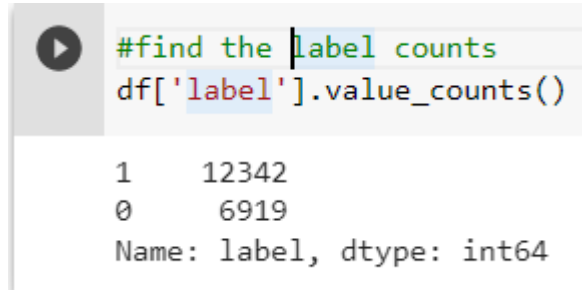




Open in app

Get started

```
#find the label counts  
df['label'].value_counts()
```



```
#find the label counts  
df['label'].value_counts()  
  
1    12342  
0     6919  
Name: label, dtype: int64
```

Positive and Negative Labels

3. Apply Supervised Learning Approach: Logistic Regression

*The first approach that we will use to build the sentiment classifier is the classic supervised one, the **Logistic Regression** which is considered as a powerful binary classifier that estimates the probability of an instance belonging to a certain class and makes predictions accordingly. However, we should first pre-process the data and create vector representations before training the model.*

3.1 Text Pre-processing

Pre-processing is an essential task in Natural Language Processing (NLP) to prepare text data for training. It converts an input of raw text into cleansed tokens as single words or characters. The main pre-processing methods are summarized below:

- 1- **Tokenizing**: which splits words into tokens
- 2- **Lemmatizing**: which breaks words into their root format
- 3- **Removal of stop words**: which takes off unnecessary words such as the, he, she, etc.
- 4- **Removal of punctuations**: which takes off unessential words elements such as comma, period, brackets, parenthesis, etc.





Open in app

Get started

```
#define a function which handles the text preprocessing
def preparation_text_data(data):
    """
    This pipeline prepares the text data, conducting the following
    steps:
    1) Tokenization
    2) Lemmatization
    4) Removal of stopwords
    5) Removal of punctuation
    """
    # initialize spacy object
    nlp = spacy.load('en_core_web_sm')
    # select raw text
    raw_text = data.text.values.tolist()
    # tokenize
    tokenized_text = [[nlp(i.lower().strip()) for i in
    tqdm(raw_text)]]
    #define the punctuations and stop words
    punc = string.punctuation
    stop_words = set(stopwords.words('english'))
    #lemmatize, remove stopwords and punctuation
    corpus = []
    for doc in tqdm(tokenized_text):
        corpus.append([word.lemma_ for word in doc[0] if (word.lemma_
        not in stop_words and word.lemma_ not in punc)])
    # add prepared data to df
    data["text"] = corpus
    return data

#apply the data preprocessing function
data = preparation_text_data(data)
```

We can notice in the figure below that the data has been cleansed properly with being displayed as separated tokens, each token at its original root and without stop words and punctuations.

```
[23] #apply the data preprocessing function
data = preparation_text_data(data)
data
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

```
100% [████████████████████████████████████████] 19261/19261 [03:14<00:00, 104.71it/s]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```





Open in app

Get started

Text Preparation Results

3.2 Text Representation

The second step involves transforming text data into meaningful vectors that can be understood by the ML model. I applied the **TF-IDF (Term Frequency(TF) — Inverse Dense Frequency(IDF))** which creates counted weights for input data based on the word occurrences in the entire corpus.

```
def text_representation(data):
    tfidf_vect = TfidfVectorizer()
    data['text'] = data['text'].apply(lambda text: " ".join(set(text)))
    X_tfidf = tfidf_vect.fit_transform(data['text'])
    print(X_tfidf.shape)
    print(tfidf_vect.get_feature_names())
    X_tfidf = pd.DataFrame(X_tfidf.toarray())
    return X_tfidf

#apply the TFIDV function
X_tfidf = text_representation(data)
```

Below, we can find the results of the text_representation function, whereby we see that words have been transformed to meaningful vectors.

#apply the TFIDV function

```
X_tfidf = text_representation(data)
X_tfidf
```

(19261, 11120)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 11110 | 11111 | 11112 | 11113 | 11114 | 11115 | 11116 | 11117 | 11118 | 11119 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 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 | 0.0 | 0.0 | 0.0 |
| 1 | 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 | 0.0 | 0.0 |
| 2 | 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 | 0.0 | 0.0 |
| 3 | 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 | 0.0 | 0.0 |
| 4 | 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 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 10256 | 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 | 0.0 | 0.0 |

Text Representation TFIDV Results





Open in app

Get started

that the model has an **accuracy score of 92%**.

```
X= X_tfidf
y = data['label']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

#fit Log Regression Model
clf= LogisticRegression()
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
#fit Log Regression Model
clf= LogisticRegression()
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.91 | 0.87 | 0.89 | 2246 |
| 1 | 0.93 | 0.95 | 0.94 | 4111 |
| accuracy | | | 0.92 | 6357 |
| macro avg | 0.92 | 0.91 | 0.92 | 6357 |
| weighted avg | 0.92 | 0.92 | 0.92 | 6357 |

Logistic Regression Classification Report

3.4 Predict on new instance

We can predict on new instance as shown below, we feed the model with a new headline and we predict the label which is **negative** in our example as we are conveying **war and sanctions**






Open in app

Get started

```
tf = TfidfVectorizer()
tfidf = tf.fit_transform(data['text'])
vect = pd.DataFrame(tf.transform(new_data).toarray())
new_data = pd.DataFrame(vect)
logistic_prediction = clf.predict(new_data)
print(logistic_prediction)
```

```
] new_data = ["The US imposes sanctions on Russia because of the Ukrainian war"]
tf = TfidfVectorizer()
tfidf = tf.fit_transform(data['text'])
vect = pd.DataFrame(tf.transform(new_data).toarray())
new_data = pd.DataFrame(vect)
logistic_prediction = clf.predict(new_data)
print(logistic_prediction)
```

 [0]

Approach I: Prediction on New Instance

4. Deep learning Approach: Tensor Flow and Keras

Neural Network is a deep learning algorithm that consists of layers of interconnected neurons powered by activation functions. It considers a weighted sum of each input and then applies a step function to that sum and outputs the results which shows the class of an instance.

In fact, **Keras and TensorFlow** are among the most popular frameworks when it comes to Deep Learning. To define, Keras is a high-level neural networks library that is running on the top of TensorFlow, whereas TensorFlow is an end-to-end open-source platform for machine learning which consists of tools, libraries and other resources that provide workflows with high-level APIs.

*In our project, we will utilize the TensorFlow to pre-process and pad the data using the **tokenizer class** and we will use Keras to load and train the **Sequential model (neural***





Open in app

Get started

```
##store headlines and labels in respective lists
text = list(data['text'])
labels = list(data['label'])

##sentences
training_text = text[0:15000]
testing_text = text[15000:]

##labels
training_labels = labels[0:15000]
testing_labels = labels[15000:]
```

4.2 Set up the tokenizer from Tensor to pre-process the data.

In this step, we use the **word tokenizer from tensorflow.keras** to create *word encodings* (dictionary with key-value pairs) and *sequences using texts_to_sequences instance*, and then we *pad these sequences* to make it of equal length using the **pad_sequences instance**.

```
#preprocess
tokenizer = Tokenizer(num_words=10000, oov_token= "<OOV>")
tokenizer.fit_on_texts(training_text)

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_text)
training_padded = pad_sequences(training_sequences, maxlen=120,
padding='post', truncating='post')

testing_sequences = tokenizer.texts_to_sequences(testing_text)
testing_padded = pad_sequences(testing_sequences, maxlen=120,
padding='post', truncating='post')

# convert lists into numpy arrays to make it work with TensorFlow
training_padded = np.array(training_padded)
training_labels = np.array(training_labels)
testing_padded = np.array(testing_padded)
testing_labels = np.array(testing_labels)
```





Open in app

Get started

outputs probabilities between 0 or 1. You can simply play with the hyperparameters within each layer to increase model performance. Then, we compile the model with an optimizer and metric performance and we train it on our dataset.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(10000, 16, input_length=120),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

##compile the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=
['accuracy'])

model.summary()
```

We can check in the following figure that we have 4 layers, max_length of 120, dense layers nodes of 16 and 24 and 160,433 trainable parameters.

`model.summary()`

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|-----------------|---------|
| embedding (Embedding) | (None, 120, 16) | 160000 |
| global_average_pooling1d (GlobalAveragePooling1D) | (None, 16) | 0 |
| dense (Dense) | (None, 24) | 408 |
| dense_1 (Dense) | (None, 1) | 25 |

=====
 Total params: 160,433
 Trainable params: 160,433
 Non-trainable params: 0





Open in app

Get started

```
num_epochs = 10
history = model.fit(training_padded,
                    training_labels,
                    epochs=num_epochs,
                    validation_data=(testing_padded, testing_labels),
                    verbose=2)
```

We can further check that our built neural network model with 10 running epochs has a **very good accuracy of 99%** , **decreasing validation loss and increasing validation accuracy** which assure a powerful predictive performance and a low risk of generalisation (overfitting) error.

```
469/469 - 5s - loss: 0.6509 - accuracy: 0.6434 - val_loss: 0.6518 - val_accuracy: 0.6273 - 5s/epoch - 10ms/step
Epoch 2/10
469/469 - 3s - loss: 0.6081 - accuracy: 0.6547 - val_loss: 0.5392 - val_accuracy: 0.6968 - 3s/epoch - 6ms/step
Epoch 3/10
469/469 - 3s - loss: 0.3890 - accuracy: 0.8479 - val_loss: 0.3065 - val_accuracy: 0.8747 - 3s/epoch - 7ms/step
Epoch 4/10
469/469 - 3s - loss: 0.2085 - accuracy: 0.9382 - val_loss: 0.2062 - val_accuracy: 0.9247 - 3s/epoch - 7ms/step
Epoch 5/10
469/469 - 3s - loss: 0.1344 - accuracy: 0.9618 - val_loss: 0.1576 - val_accuracy: 0.9486 - 3s/epoch - 6ms/step
Epoch 6/10
469/469 - 3s - loss: 0.0974 - accuracy: 0.9728 - val_loss: 0.1489 - val_accuracy: 0.9427 - 3s/epoch - 6ms/step
Epoch 7/10
469/469 - 3s - loss: 0.0752 - accuracy: 0.9787 - val_loss: 0.1219 - val_accuracy: 0.9552 - 3s/epoch - 6ms/step
Epoch 8/10
469/469 - 3s - loss: 0.0593 - accuracy: 0.9843 - val_loss: 0.1151 - val_accuracy: 0.9559 - 3s/epoch - 6ms/step
Epoch 9/10
469/469 - 3s - loss: 0.0480 - accuracy: 0.9863 - val_loss: 0.1068 - val_accuracy: 0.9606 - 3s/epoch - 6ms/step
Epoch 10/10
469/469 - 2s - loss: 0.0380 - accuracy: 0.9900 - val_loss: 0.1022 - val_accuracy: 0.9625 - 2s/epoch - 5ms/step
```

Neural Network Training Results

4.4 Predict on new instance

Now, we will use this particular model to predict on the same headline. Again the output is closed to zero which also indicates that this headline is negative.

```
new_headline = ["The US imposes sanctions on Rassia because of the
Ukranian war"]
```





Open in app

Get started

```
new_headline = ["The US imposes sanctions on Rassia because of the Ukranian war"]

##prepare the sequences of the sentences in question
sequences = tokenizer.texts_to_sequences(new_headline)
padded_seqs = pad_sequences(sequences, maxlen=120, padding='post', truncating='post')

print(model.predict(padded_seqs))

[[0.09581235]]
```

Approach II: Prediction on New Instance

5. Conclusion

In this article, we built a binary classifier to detect the sentiment of news headlines. However, we first employed some heuristic rules using Snorkel method to create labels that classify negative and positive headlines. We created sentiment predictions using supervised ML and deep learning approaches. Both methods were successful in predicting the correct headline on a new given instance and they both had reasonable high accuracy scores of 92% and 96% for Logistic Regression and deep neural network respectively.

You might ask yourself, which approach is better or easier to use in my next data science prediction task; however, the answer totally depends on the scope and complexity of the project as well as the availability of data, sometimes we might opt for simple solutions using the well-known algorithms from scikit-learn whereby the predictive system deploys mathematical intuitions to assign a desired output value for a given input. On the other hand, deep learning tries to mimic the functioning of human brain with functions that enforces rules (as outputs) from a given input. We should keep in mind that neural networks usually require a huge amount of data and high computational power to achieve the task.

I hope you enjoyed reading this article, and look forward for more enriching ones which cascade my knowledge about data science and machine learning topics.



[Open in app](#)[Get started](#)

[1] A Million News Headlines, News headlines published over a period of 18 Years,
License CCO: Public Domain, [Kaggle](#)

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

[Get this newsletter](#)

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

