Table of Contents

Introduction

Implementation of a sentiment analysis tool that can determine whether customer sentiment is positive, neutral, or negative.

The Goal :

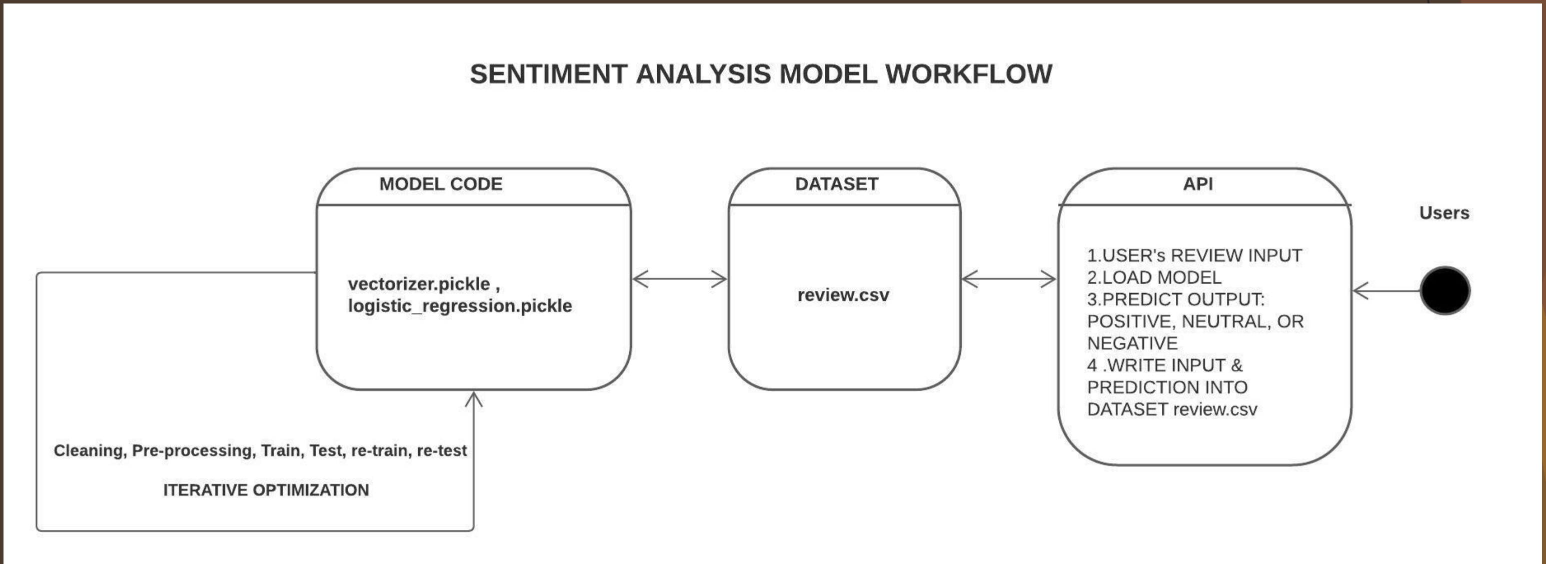
- 1. To use Python to make a simple Sentiment Analysis App that can tell whether customer reviews are positive, neutral, or negative.
- 2. To use pre-trained Vader to fit ‘neutral’ sentiment to dataset.
- 3. As pre-processing steps, clean, stem, and lemmatize the dataset.
- 4. Training, validating, and testing the model using Logistic Regression.
- 5. To vectorize/ turn a dataset into a vector using Tf-IDF (Term frequency – inverse document frequency).
- 6. To save the models with pickle, a Python’s library.
- 7. To perform sentiment analysis of user’s input through Streamlit API, and show the prediction.
- 8. To add new rows to an existing dataset with data collected from the user's input.
- 9. As part of daily iterative optimization, the training of a daily-updated dataset should be triggered to automate the process.
- 10. To deploy an application onto the Streamlit Cloud.

Streamlit Cloud Link: https://hennypurwadi-sentiment-analysis-streamlit-app-xdpl7j.streamlitapp.com/

Github Link: https://github.com/hennypurwadi/Sentiment_Analysis

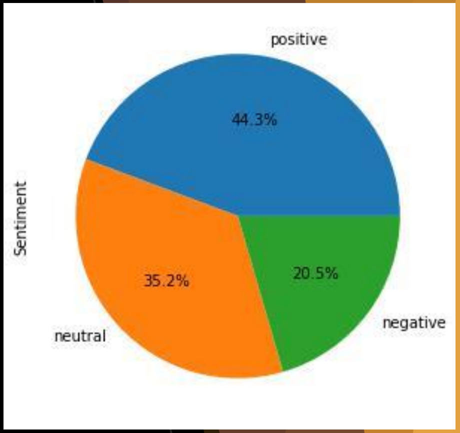
Dataset Link: https://www.kaggle.com/datasets/d4rklucif3r/restaurant-reviews

UML Activity Diagram



Vader is used to change two emotions in a dataset into three emotions.

Since the existing dataset only had two emotions ("positive" and "negative"), a "neutral" emotion was found by using a "pre-trained" version of Vader.



VADER (Valence Aware Dictionary and sEntiment Reasoner), which was made in 2014, is a model that has already been trained and uses rule-based values that are tuned to sentiments from social media. It looks at the text of a message and rates not only the positive and negative emotions, but also how strong those emotions are. (Hutto, C.J., and Gilbert, E.E. (2014).

A positive sentiment is when the value is greater than zero.The name for something that has no value is "neutral sentiment."A negative sentiment is when the value is less than zero.

```
df1.head(3)
```

	Review	Liked	Clean_Reviews	Cleaned_Reviews	Score	compound	Sentiment
0	Wow... Loved this place.	positive	wow love thi place	wow love thi place	{'neg': 0.0, 'neu': 0.2, 'pos': 0.8, 'compound...	0.8402	positive
1	Crust is not good.	negative	crust is not good	crust good	{'neg': 0.445, 'neu': 0.555, 'pos': 0.0, 'comp...	-0.3412	negative
2	Not tasty and the texture was just nasty.	negative	not tasti and the textur wa just nasti	tasti textur wa nasti	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	0.0000	neutral

Replace value become "neutral" in "Liked" column, if compound == 0

```
df1['Liked'].mask(df1['compound'] ==0 ,'neutral', inplace=True)
```

```
df1.head(3)
```

	Review	Liked	Clean_Reviews	Cleaned_Reviews	Score	compound	Sentiment
0	Wow... Loved this place.	positive	wow love thi place	wow love thi place	{'neg': 0.0, 'neu': 0.2, 'pos': 0.8, 'compound...	0.8402	positive
1	Crust is not good.	negative	crust is not good	crust good	{'neg': 0.445, 'neu': 0.555, 'pos': 0.0, 'comp...	-0.3412	negative
2	Not tasty and the texture was just nasty.	neutral	not tasti and the textur wa just nasti	tasti textur wa nasti	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	0.0000	neutral

Pre-processing steps: clean, stem, and lemmatize the dataset.

- CLEANING,
- STEMMING
- LEMMATIZE

```
def lemmatize_process(preprocessedtext):
    # Create Lemmatizer
    lem = WordNetLemmatizer()
    finalprocessedtext = []
    for word in preprocessedtext:
        text_pos = pos_tag(word_tokenize(word))
        words = [x[0] for x in text_pos]
        pos = [x[1] for x in text_pos]
        word_stm = " ".join([lem.lemmatize(a,get_jvnr(b)) for a,b in zip(words,pos)])
        finalprocessedtext.append(word_stm)
    return finalprocessedtext

def stemming_process(preprocessedtext):
    # Create stemming
    stm = PorterStemmer()
    finalprocessedtext = []
    for word in preprocessedtext:
        text_pos = pos_tag(word_tokenize(word))
        words = [x[0] for x in text_pos]
        pos = [x[1] for x in text_pos]
        word_stm = " ".join([stm.stem(a,get_jvnr(b)) for a,b in zip(words,pos)])
        finalprocessedtext.append(word_stm)
    return finalprocessedtext

def preprocess(preprocessedtext):
    processedText = []
    for Review_data in preprocessedtext:
        Review_data = str(Review_data).lower()
        Review_data = re.sub(r'#[A-Za-z0-9]+', '', Review_data) #remove hashtags
        Review_data=re.sub(r'@[A-Za-z0-9]+', '',Review_data) #remove usernames
        Review_data=re.sub(r'@\w+', ' ', Review_data) #remove usernames
        Review_data= re.sub(r'\b\w{1}\b', '', Review_data) #remove stopwords
        Review_data = re.sub(r'&(?![A-Za-z]+[0-9]*;|#[0-9]+;|#x[0-9a-fA-F]+;)', '', Review_data)
        Review_data = re.sub(r'&amp', '', Review_data)
        Review_data = re.sub('\n', '', Review_data) #Remove line breaks.
        Review_data = re.sub('[%s]' % re.escape(string.punctuation), '', Review_data) #remove punctuation
        Review_data = re.sub('\[.*?\]', '', Review_data)
        Review_data=re.sub(r'http\S+', ' ', Review_data) #remove all Url
        Review_data = re.sub(r'https?:\/\/.*[\r\n]*', '', Review_data) #remove website
        Review_data = re.sub('https?://\S+|www\.\S+', '', Review_data)  #remove all websites
        Review_data = re.sub(r' +', ' ', Review_data) #remove extra space
        Review_data = re.sub('<.*?>+', '', Review_data)
        Review_data = re.sub('\w*\d\w*', '', Review_data)
        Review_data = re.sub(r'^RT[\s]+', '', Review_data)
        Review_data = re.sub(r'[^a-z A-Z]', ' ',Review_data) #Remove all not characters
        processedText.append(Review_data)
    return processedText
```

Using Tf-IDF to vectorize a dataset

The TF-idf weight is made up of the normalized Term Frequency (TF), which is the number of times a word appears in a document, divided by the total number of words in that document. The Inverse Document Frequency (IDF) gives less weight to words that are used often and more weight to words that are used rarely. (H.Wu, K. Wong, K. Kwok, and R. Luk (2008).

Vectorize cleaned texts with Tf-IDF with ngram_range

ngram_range of (1, 1) means only unigrams. (1, 2) means unigrams and bigrams. (1, 3) means unigrams, bigrams, and trigrams. (2, 2) means only bigrams.

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

```
tfidf = TfidfVectorizer(min_df=1 ,max_df=0.95, ngram_range=(1,3),stop_words='english', lowercase=True, smooth_idf=True)
tfidf_text = tfidf.fit_transform(df1['Clean_Reviews'])
tfidf_text
```

```
<1000x9063 sparse matrix of type '<class 'numpy.float64'>'
	with 13710 stored elements in Compressed Sparse Row format>
```

Split dataset into Training and Testing data: 70% training data, 30% testing data.

Split the Data become train data and test data

```
# Splitting dataset into train and test, test size of 3%
X_train, X_test, y_train, y_test = train_test_split(df1.Clean_Reviews, df1.Liked, test_size = 0.03, random_state = 0)
print('Done Data Split')
```

```
Done Data Split
```

```
X_train = tfidf.transform(X_train)
X_test  = tfidf.transform(X_test)
print(f'Data Transformed.')
```

```
Data Transformed.
```

Compare accuracy of the Linear, Bernoulli, and Logistic Regression algorithms. Save most precise model (Logistic Regression).

Using Logistic Regression to train, validate, and evaluate the model.
Split dataset into Training and Testing data:70% training data, 30% testing data.

Develop Models : LinearSVC, Bernoulli NB, Logistic Regression

```
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))

    #return y_pred
```

Logistic Regression

```
LRmodel = LogisticRegression(C = 1, max_iter = 1000, n_jo
LogReg = LRmodel.fit(X_train, y_train)
y_test_pred = model_Evaluate(LRmodel)
```

```
              precision    recall  f1-score   support

    negative       0.86      0.60      0.71        10
     neutral       0.73      0.89      0.80         9
    positive       0.83      0.91      0.87        11

    accuracy                           0.80        30
   macro avg       0.81      0.80      0.79        30
weighted avg       0.81      0.80      0.79        30
```

Logistic Regression accuracy is around 80% accuracy

LinearSVC

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

```
              precision    recall  f1-score   support

    negative       0.88      0.70      0.78        10
     neutral       0.73      0.89      0.80         9
    positive       0.82      0.82      0.82        11

    accuracy                           0.80        30
   macro avg       0.81      0.80      0.80        30
weighted avg       0.81      0.80      0.80        30
```

BernoulliNB

```
BNBmodel = BernoulliNB(alpha = 2)
BNBmodel.fit(X_train, y_train)
model_Evaluate(BNBmodel)
```

```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00        10
     neutral       0.38      1.00      0.55         9
    positive       1.00      0.55      0.71        11

    accuracy                           0.50        30
   macro avg       0.46      0.52      0.42        30
weighted avg       0.48      0.50      0.42        30
```

Using the Model

```
def load_models():

    # Load the vectorizer.
    file = open('vectorizer.pickle', 'rb')
    vectorizer = pickle.load(file)
    file.close()

    # Load the LR Model.
    file = open('sentimentanalysis_LR.pickle', 'rb')
    LRmodel = pickle.load(file)
    file.close()

    return vectorizer, LRmodel
```

```
vectorizer, LRmodel = load_models()
dfmessage = predict_message(vectorizer, LRmodel, message)
dfmessage.head(10)
```

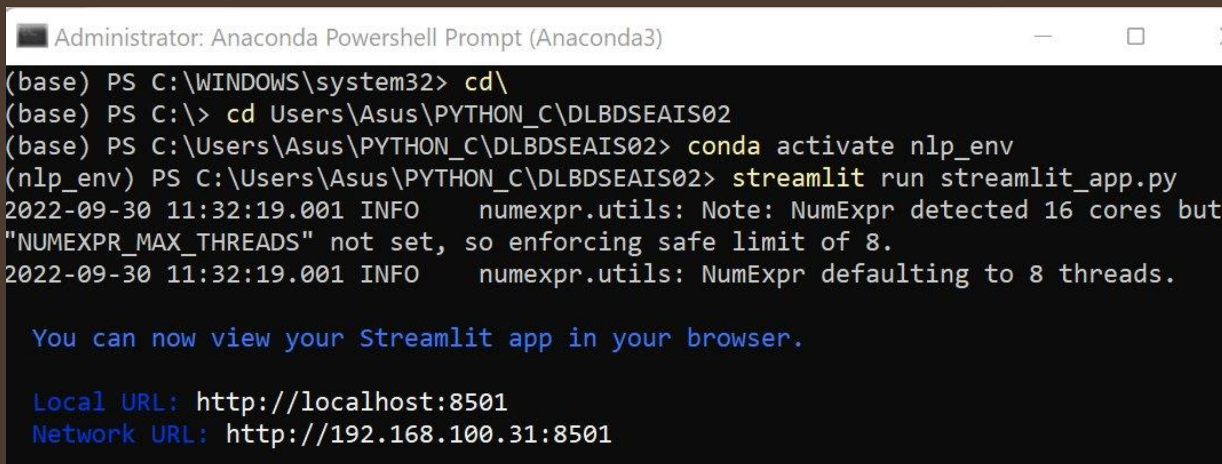
	sentences	sentiment	Probability(Confidence Level)
0	The food is good	positive	8.882%
1	The juice is too sour	neutral	44.101%
2	I feel disappointed	negative	10.595%

```
dfmessage_b = dfmessage[['sentences','sentiment']]
dfmessage_b
```

	sentences	sentiment
0	The food is good	positive
1	The juice is too sour	neutral
2	I feel disappointed	negative

Use the models to predict the sentiment of text.

Execute Sentiment Analysis App on the local system to determine the positive, neutral, or negative feelings of customer evaluations.



Determine if the user's input is automatically added to the dataset as new rows.

Dataset Prior to Running the application

997	I think food should have flavor and texture and both were lacking.,-1
998	Appetite instantly gone.,0
999	Overall I was not impressed and would not go back.,-1
1000	The whole experience was underwhelming and I think we'll just go to Ninja Sushi next time.,-1
1001	Then as if I hadn't wasted enough of my life there they poured salt in the wound by drawing out the time it took to bring the check.,0

Dataset after the application is launched.

997	I think food should have flavor and texture and both were lacking.,-1
998	Appetite instantly gone.,0
999	Overall I was not impressed and would not go back.,-1
1000	The whole experience was underwhelming and I think we'll just go to Ninja Sushi next time.,-1
1001	Then as if I hadn't wasted enough of my life there they poured salt in the wound by drawing out the time it took to bring the check.,0
1002	The food is good,1
1003	The juice is too sour,0
1004	I feel disappointed,-1

new rows added into dataframe

Automate daily training using schedule

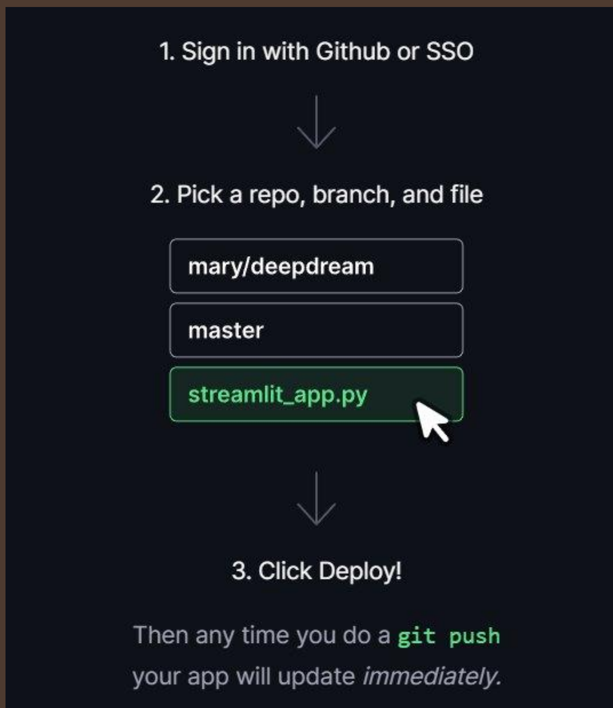
Iterative optimization of the system

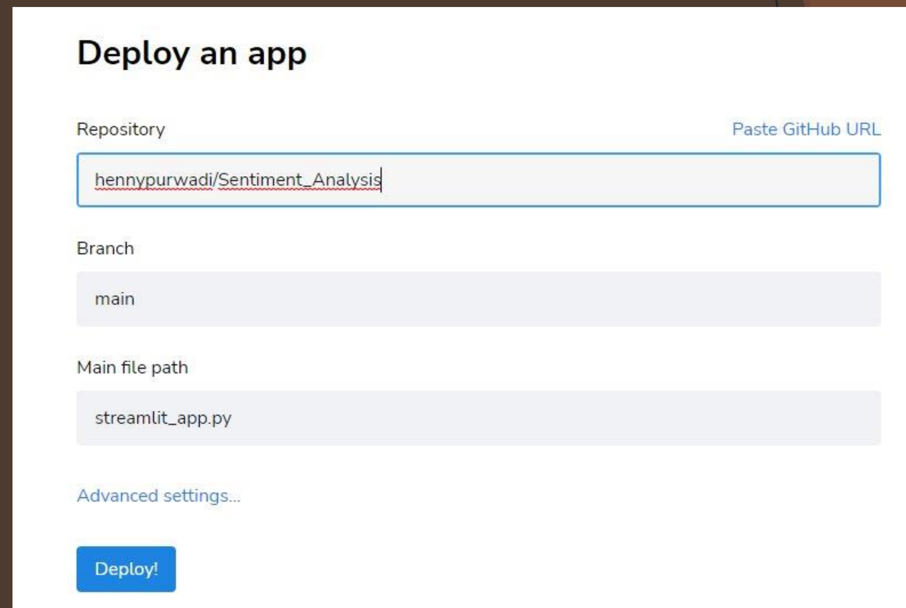
In [52]:
```
#Automate scheduled training

train_models()
schedule.every(24).hours.do(train_models)
print("training done")
```

```
Models saved
training done
```

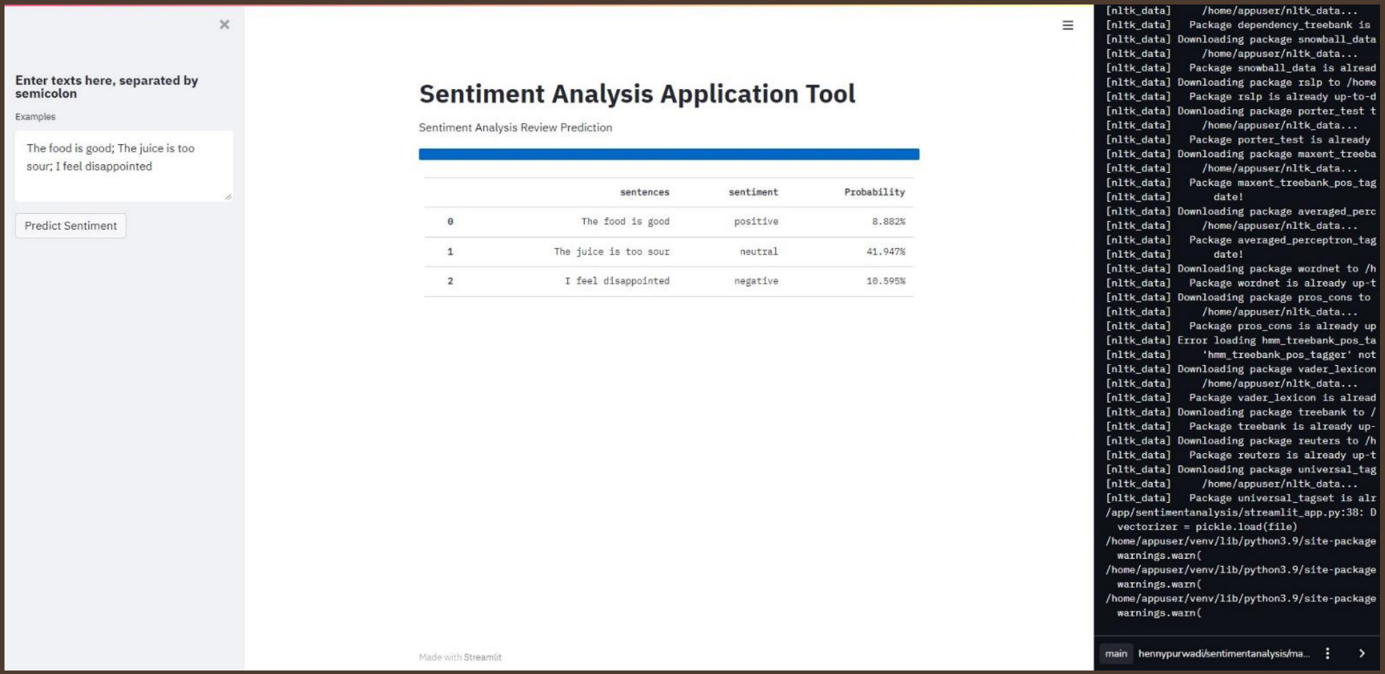
Application deployment on Streamlit Cloud

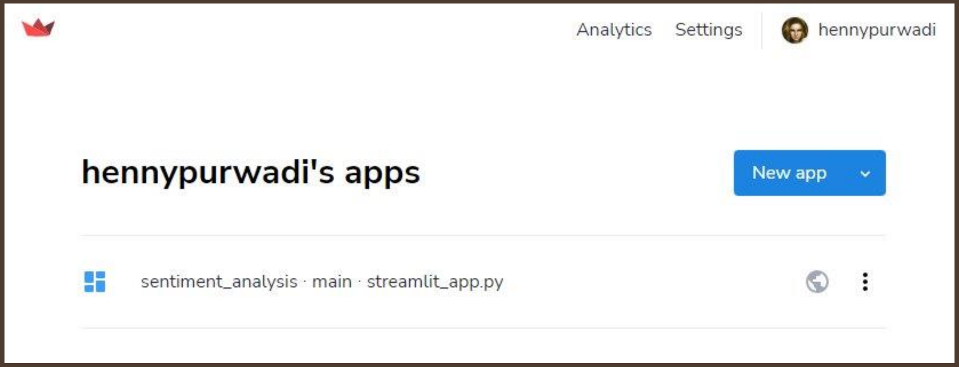
Connecting Streamlit to a repository on GitHub.





Cloud deployment





Conclusion

Natural Language Processing's Artificial Intelligence (AI) makes it possible for the Customer review application to predict the sentiment of customer reviews without human intervention.

These predictions can be made regardless of whether the review was po sitive, negative, or neutral.

Literature:

- Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

- H.Wu and R. Luk and K. Wong and K. Kwok.(2008).Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems.

- Hutto, C.J. & Gilbert, E.E. (2014).vaderSentiment. https://github.com/cjhutto/vaderSentimentt

- Scikit-learn 1.1.2. (2022). https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

- What does Tf-idf means.(2021). http://www.tfidf.com/

- Streamlit Community Cloud.(2021). https://streamlit.io/cloud