Sentiment Analysis of Product Review

Content

Introduction

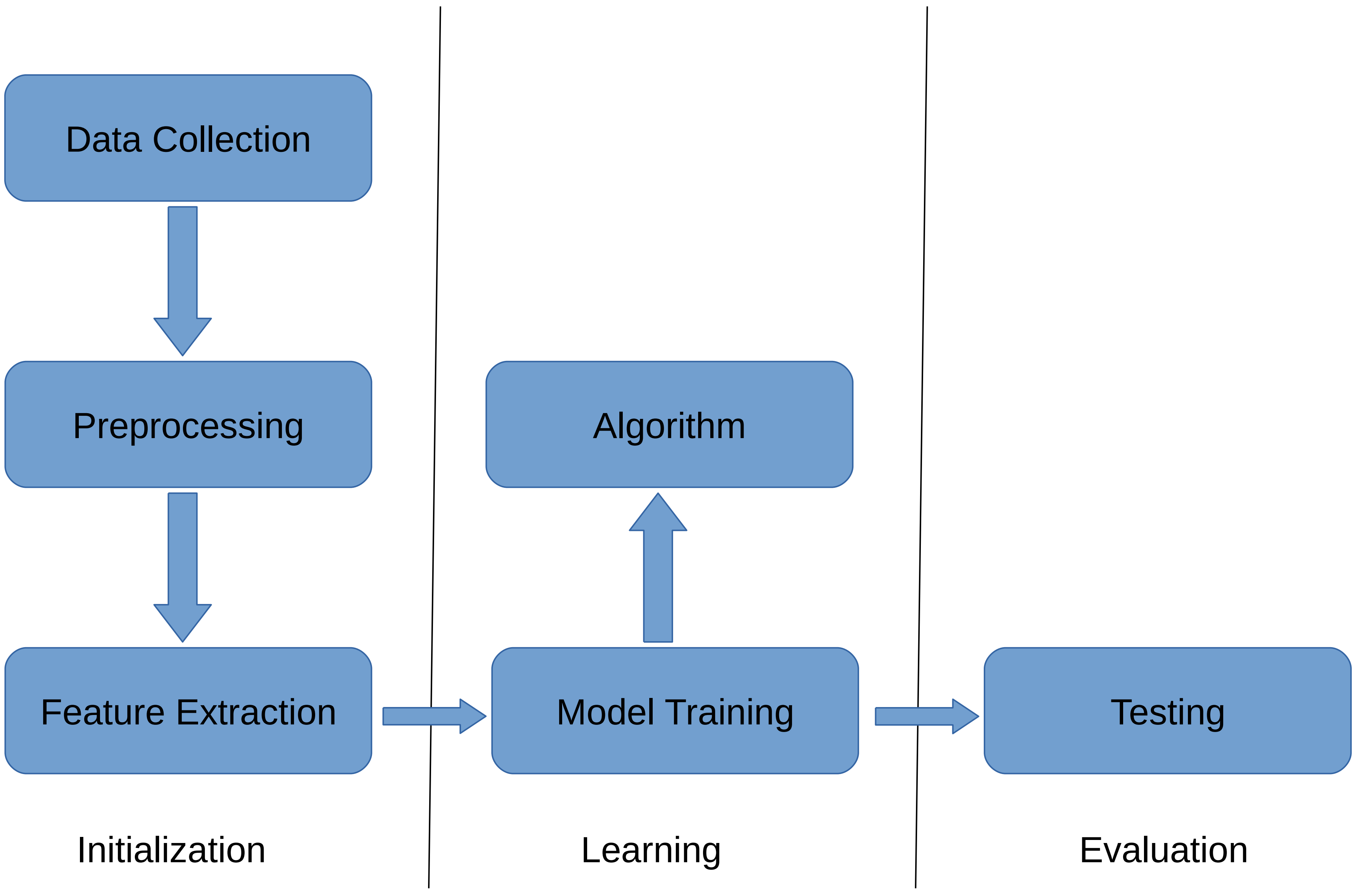
- A large amount of data on the internet is **unstructured** which requires processing to generate insights
- To analyze the natural language and make sense out of the unstructed data we employ **Natural Language Processing** techniques
- Some examples of unstructed data are news articles, posts on social media, search history, etc.

Contd..

- **Sentiment Analysis** is one such common NLP task
- It involves classifying texts or parts of texts into a pre-defined sentiment
- Its an automated extraction of subjective content from text and predicting the subjectivity such as **positive** or **negative**

Why Sentiment Analysis is important?

- Time is money therefore instead of spending time in manually reading and evaluating sentiment use automated algorithms
- Everybody is interested in what others think – Which Mobile? Which Movie? Which Policy? Which Place? Etc
- Surveys – Good / Poor / Fair / Excellent



Implementation Steps for Sentiment Analysis

Preprocessing Steps

- **Case Normalization** – Change the case of all the words to lower
- **Word Tokenization** – Split the text into word tokens using pre-built tokenizer
- **Stemming/Lemmatization** – Convert the word into its base form. Example: Studying -> Study
- **Stopword Removal** – Remove unwanted words like if, the, was, is, etc.

Stemming vs Lemmatization

- **Stemming**
  - Stemming is a process which works by cutting off the end or the beginning of the word
  - It takes into account a list of common prefixes and suffixes that can be found in an inflected word.
  - Example : **studying -> studi, crying -> cri, waiting -> wait**
  - Stemming indiscriminately cuts off the ends which might be useful in some occassions but not always
  - Hence this process has its own limitation and generally it is less preferred.
  - Available tool/library -> **Porter Stemmer** module from Python’s NLTK library

Contd..

- **Lemmatization**
  - Lemmatization takes into account the morphological analysis of the words
  - It takes into account the detailed dictionary of the word where algorithm can look through to link the form back to its lemma
  - Example: **studies -> {Third person, singular number, present tense of the verb ‘study’} -> study**
  - Lemma is a base form of all its inflection forms whereas stem isn’t. For example – ‘studying’, ‘studies’ all has base form has ‘study’ but the same is not true for its stemmed forms
  - Available tool/library -> **WordNet Lemmatizer** module from Python’s NLTK library

Word Embeddings

- Word Embeddings are text converted into numbers
- It tries to map a word using a dictionary to a vector
- Why Word Embeddings are required?
  - ML or DL algorithms are incapable of processing raw or plain text
  - They require numbers as input to perform any job like Logistic Regression, Classification,etc
- Types of Word Embeddings -
  - Frequency Based
  - Prediction Based

Frequency Based Word Embeddings

- There are generally 3 types of vectors under this category
  - Count Vector
  - TF-IDF Vector
  - Co-Occurence Vector
- Let us understand these vectors by example

Count Vectorizer

- Lets consider 2 simple statements -
  - S1: He is a good coder. She is also good with coding.
  - S2: Hitesh is a good coder.
- As a next step we create a dictionary of unique words/tokens -
  - D : [‘He’, ‘She’, ‘good’, ‘coder’, ‘coding’, ‘Hitesh’]
- Given the above 2 sentence, we can create count matrix -

	He	She	good	coder	coding	Hitesh
S1	1	1	2	1	1	0
S2	0	0	1	1	0	1

Tf-Idf Vectorizer

- Another frequency based method but different than Count Vectorizer
- It takes into account not just occurence of a word in single document but in the entire corpus i.e. list of documents
- But why?
  - A document on ‘TensorFlow’ will have more occurences of the word ‘Tensorflow’ but it will also have common words like ‘the’ etc in higher frequency
  - We would want to down weight the common terms and give more importance to words appear in subset of documents
  - It penalizes most frequent words like - ‘is’, ‘the’, ‘if’, etc in the document
  - Gives more weightage to terms like ‘TensorFlow’
- How does it work? ->

Document 1		Document 2	
Lets	1	Lets	1
talk	1	talk	2
about	2	about	1
TensorFlow	4	Tf-Idf	1

	Term Frequency (TF)	**Inverse Document Frequency (IDF)**
Formula	Count(t) / Total count	log(N/n)
Terms meaning	‘t’ -> token under consideration	‘N’ -> Number of documents in the corpus ‘n’ -> Number of documents term ‘t’ has appeared in
Example	TF(‘Lets’,Document 1) = 1/8	IDF(‘Lets’) = log(2/2) = 0
Explanation and Reasoning	It denotes the contribution of the word to the document i.e. the word relevant to the document will be frequent	If a word has appeared in all the document, then probably that word is not relevant to a particular document But if it has appeared in a subset of documents then probably that word is of some relevance to the documents it is present in.

- Computing the TF-IDF for common words like - ‘Lets’ and ‘TensorFlow’
  - TF-IDF(‘Lets’, Document 1) = (1/8) * (0) = **0**
  - TF-IDF(‘Lets’,Document 2) = (1/5) * (0) = **0**
  - TF-IDF(‘TensorFlow’, Document 1) = (4/8) * log(2/1) = 0.5 * 0.301 = **0.15**
- As we can see, TF-IDF method heavily penalizes the word ‘Lets’ but assigns greater weight to ‘TensorFlow’
- So this can be interpreted as ‘TensorFlow’ is an important word for Document 1 from the context of the entire corpus

Prediction Based Word Embeddings

- Deterministic based word vectors proved to be limited in their word representations
- Hence a new probabilistic based approach called **word2vec** emerged, developed by ‘Tomas Mitolov’
- It is based on 2 techniques -
  - Continuous Bag of Words (CBOW)
  - SkigGram Model
- Both of these methods are based on shallow neural network which maps word(s) to the target word(s)
- Both of these models learns weight which act as word vector representation
- CBOW tries to predict the probability of a word given a context. Here the context can be a word or a group of words
- SkipGram tries achieve exactly opposite. It tries to predict the probability of a context given a word

Model Building

- Once we have the vector representations ready we can apply any classification algorithm
- We can apply Logistic Regression or Tree based method
- Alternatively we can look for Neural Network based methods like LSTM or CNN.
- Try different combination of hyperparameters for better accuracy

Thank You!