Building NLP model using different text representation techniques like :

- Count vectorizer
- Word2Vec
- TF-IDF

In [5]:
```
# importing require packages
import pandas as pd
import numpy as np
import spacy
import string

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn import metrics

np.random.seed(42)
```

In [6]:
```
train_data_path = r"C:\Users\ASUS\Desktop\Ml_DL\code\Nlp2llm\toxic_comment_classifier\jigsaw-toxic-comment-classification-challer
test_data_path = r"C:\Users\ASUS\Desktop\Ml_DL\code\Nlp2llm\toxic_comment_classifier\jigsaw-toxic-comment-classification-challeng
```

In [7]:
```
train_df = pd.read_csv(train_data_path)
test_df = pd.read_csv(test_data_path)
```

In [79]:
```
train_df.head()
```

Out[79]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

In [9]:
```
test_df.head()
```

Out[9]:

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

In [13]:
```
# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# stopwords doesnot have much meaning so removing them makes data more small and efficient
stop_words = nlp.Defaults.stop_words
print(stop_words)
```

```
{'full', '’m', '‘ve', 'third', 'towards', 'therein', 'less', 'each', 'formerly', 'she', 'would', 'be', 'forty', 'hereby', 'nowh
ere', 'anything', 'nothing', 'using', 'we', 'had', 'none', 'well', 'their', 'elsewhere', 'throughout', 'its', 'does', 'seem',
'hence', 'nobody', 'amount', 'that', 'somehow', 'seeming', 'other', 'last', 'hereupon', 'could', "'re", 'now', 'alone', 'meanwh
ile', 'side', 'give', 'cannot', 'behind', 'the', 'ever', 'no', 'of', 'first', 'upon', 'someone', 'yours', 'make', 'they', 'an
y', 'fifty', 'while', 'almost', 'hers', 'everyone', "n't", 'hundred', 'some', 'due', 'regarding', 'nor', 'i', 'yourselves', 'ge
t', 'others', 'twenty', 'whereas', 'another', 'during', 'hereafter', 'already', 'herself', 'once', 'top', 'whoever', 'unless',
"'ll", 'along', '‘m', 'her', 'around', 'please', 'neither', 'wherever', '‘d', 'otherwise', 'whom', 'say', 'those', 'too', 'your
self', 'somewhere', 'part', '’d', 'perhaps', 'without', 'our', 'an', 'become', 'him', "'s", 'two', 'all', 'same', 'just', 'anyw
here', 'been', 'can', 'until', 'more', 'whereby', 'bottom', 'much', 'go', 'am', 'herein', 'itself', 'will', 'he', 'never', 'the
m', 'there', 'five', 'few', 'n‘t', 'a', 'own', '’ll', 'seems', 'do', 'may', 'made', 'sixty', 'where', 'about', 'has', 'everywhe
re', 'really', 'something', 'here', 'but', 'even', 'themselves', 'toward', 'is', 'being', 'how', 'though', 'did', "'d", 'to',
'thereupon', 'out', 'twelve', '’re', 'beforehand', 'his', 'several', 'amongst', 'former', 'should', 'although', 'used', 'my',
'eleven', 'himself', 'anyway', 'yet', 'for', 'or', 'these', 'sometimes', 'whither', 'also', 'either', 'namely', 'must', 'whic
h', 'every', 'so', 'below', 'onto', 'still', 'ten', 'doing', 'down', 'see', 'and', 'because', 'thence', 'before', 'thereby', 't
herefore', 'becomes', 'mine', 'done', 'further', 'across', 'moreover', 'myself', 'than', '’s', 'in', 'are', 'have', 'whether',
'whole', 'became', 'call', 'off', 'keep', 'wherein', 'on', 'above', 'front', 'this', 're', 'serious', 'who', '‘ll', 'fifteen',
'up', 'among', 'ourselves', 'various', 'might', 'between', 'nine', 'together', 'such', 'least', 'you', 'whereafter', 'quite',
'becoming', 'over', 'whenever', 'most', 'thus', 'anyone', 'under', 'within', 'show', 'enough', 'was', 'after', 'via', 'n’t', 's
eemed', 'name', 'rather', 'whose', 'four', 'from', 'always', 'everything', 'next', 'whereupon', 'one', 'often', 'ca', 'put', 'e
xcept', 'since', "'m", 'against', 'why', 'beyond', 'empty', 'however', "'ve", 'per', 'indeed', 'it', 'nevertheless', 'take', 't
hru', '‘re', 'thereafter', 'if', 'latterly', '’ve', 'latter', 'when', 'back', 'by', 'three', 'six', 'move', 'else', 'mostly',
'many', 'then', 'beside', 'at', 'your', 'sometime', 'as', 'again', 'besides', 'afterwards', 'only', 'eight', '‘s', 'into', 'any
how', 'very', 'were', 'with', 'us', 'ours', 'through', 'whatever', 'me', 'not', 'both', 'what', 'noone', 'whence'}
```

In [17]:
```
punctuation = string.punctuation
print(punctuation)
```

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

text preprocessing

In [18]:
```
def spacy_preprocessing(sentence):
    ' custom preprocessing function'
    # tokenization
    doc = nlp(sentence)

    # Lemmatization : more accurate but slower , convert word to baseform    & normalization
    lemma_word = [word.lemma_.lower().strip() for word in doc]

    # removing stopwords and punctation
    my_token = [word for word in lemma_word if word not in stop_words and word not in punctuation]

    return my_token
```

In [19]:
```
# testing the preprocessing function we built
pre = spacy_preprocessing("Hellow guys @ how are you in london !")
print(pre)
```

```
['hellow', 'guy', 'london']
```

text representation

In [20]:
```
# using count_vectorizer
count_vector = CountVectorizer(tokenizer = spacy_preprocessing)     # using count vectorizer
tfidf_vector = TfidfVectorizer(tokenizer = spacy_preprocessing)     # using Term frequency Inverse document frequency vectorizer
```

In [33]:
```
count_vector.fit_transform(["hello I am very good  and how are you feeling "]).toarray() # represent on basis of frequency
```

Out[33]:
```
array([[1, 1, 1]], dtype=int64)
```

In [34]:
```
tfidf_vector.fit_transform(["hello here I am doing nlp what about you"]).toarray()   # represent on basis of term_frequency*inver
```

Out[34]:
```
array([[0.70710678, 0.70710678]])
```

In [35]:
```
count_vector.get_feature_names_out()
```

Out[35]:
```
array(['feel', 'good', 'hello'], dtype=object)
```

In [36]:
```
count_vector.vocabulary_
```

Out[36]:
```
{'hello': 2, 'good': 1, 'feel': 0}
```

prepare datasets

In [40]:
```
from sklearn.model_selection import train_test_split

X = train_df['comment_text']         # feature representing comment
y_label = train_df['toxic']          # represent label either toxic or not toxic
X_train, X_test, y_train, y_test = train_test_split(X, y_label, test_size = 0.2, stratify=y_label)
```

In [41]:
```
# logistic regression model for classifying toxic or not toxic
from sklearn.linear_model import LogisticRegression
classifer = LogisticRegression()
```

In [42]:
```
# converting training and test data in representation vectors
X_train_vector = count_vector.fit_transform(X_train)
X_test_vector = count_vector.transform(X_test)
```

In [43]:
```
# training the logistic regression model
classifer.fit(X_train_vector,y_train)
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[43]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
# evaluating model performance using count vectorizer as text representation
predicted = classifer.predict(X_test_vector)
print("Accuracy score : ",metrics.accuracy_score(predicted,y_test))
print("Precision score : ",metrics.precision_score(predicted,y_test))
print("Recall score : ",metrics.recall_score(predicted,y_test))
```

```
Accuracy score :  0.9571048096506345
Precision score :  0.6698267407649559
Recall score :  0.8509136212624585
```

In [49]:
```
X_test[5]        # not toxic
```

Out[49]:
```
'"\n\nCongratulations from me as well, use the tools well. \xa0· talk "'
```

In [56]:
```
classifer.predict(X_test_vector[5]) == 0                    # correct prediction
```

Out[56]:
```
array([ True])
```

In [63]:
```
# function to input message and find if it is toxic or not
def get_answer(message):
    message_vector = count_vector.transform([message])
    prediction = classifer.predict(message_vector)
    if prediction == 0:
        print("Not toxic")
    else:
        print("Toxic")
```

In [64]:
```
get_answer("Hello you are doing great")
```

```
Not toxic
```

In [78]:
```
get_answer("Stupid peace of shit stop deleting my stuff")
```

```
Toxic
```