```
import pandas as pd
import·numpy·as·np
from tqdm import tqdm
from keras.preprocessing.text import Tokenizer
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
from keras_preprocessing.sequence import pad_sequences
import gensim
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
```

```
df = pd.read_csv('data.csv',delimiter=',',encoding='latin-1')
df = df[['Category','Message']]
df = df[pd.notnull(df['Message'])]
df.rename(columns = {'Message':'Message'}, inplace = True)
df.head()
```

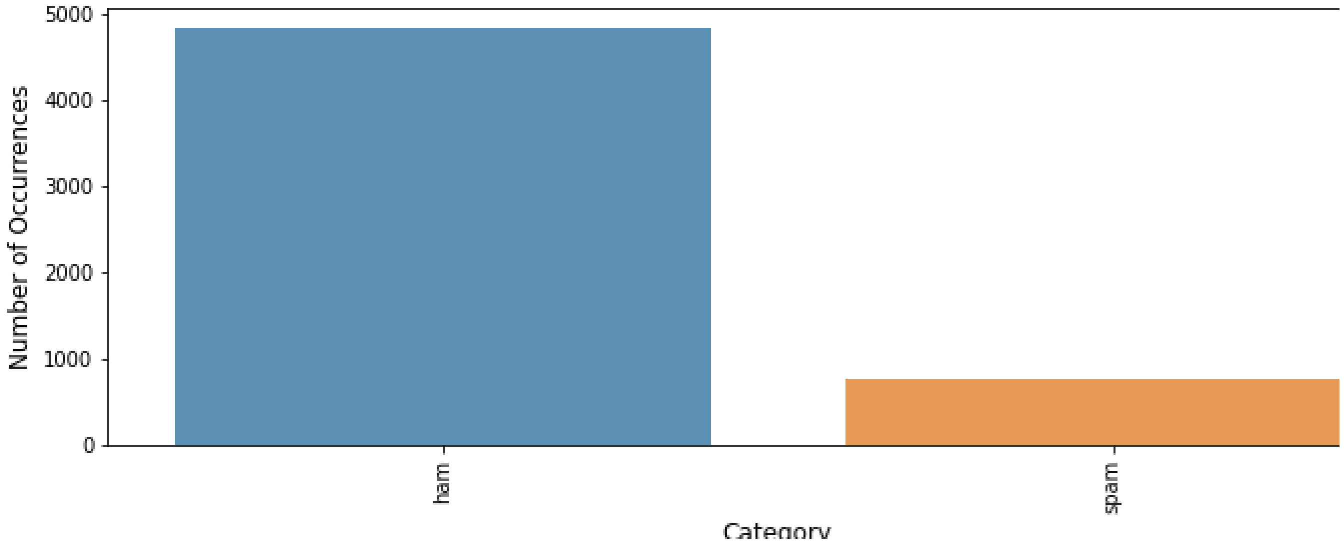
	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df.shape
```

```
(5572, 2)
```

```
cnt_pro = df['Category'].value_counts()
plt.figure(figsize=(12,4))
sns.barplot(cnt_pro.index, cnt_pro.values, alpha=0.8)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Category', fontsize=12)
plt.xticks(rotation=90)
plt.show();
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```



```
from bs4 import BeautifulSoup
def cleanText(text):
    text = BeautifulSoup(text, "lxml").text
    text = re.sub(r'\|\|\|', r' ', text)
    text = re.sub(r'http\S+', r'<URL>', text)
    text = text.lower()
    text = text.replace('x', '')
    return text
df['Message'] = df['Message'].apply(cleanText)
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
train, test = train_test_split(df, test_size=0.000001 , random_state=42)
import nltk
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            #if len(word) < 0:
            if len(word) <= 0:
                continue
            tokens.append(word.lower())
    return tokens
train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['Message']), tags=[r.Category]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['Message']), tags=[r.Category]), axis=1)
```

```
# The maximum number of words to be used. (most frequent)
max_fatures = 500000

# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 100

#tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer = Tokenizer(num_words=max_fatures, split=' ', filters='!"#$%&()*+,-./:;<=>?@[\]^_`{
tokenizer.fit_on_texts(df['Message'].values)
X = tokenizer.texts_to_sequences(df['Message'].values)
X = pad_sequences(X)
print('Found %s unique tokens.' % len(X))
```

```
Found 5572 unique tokens.
```

```
X = tokenizer.texts_to_sequences(df['Message'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (5572, 100)
```

```
d2v_model = Doc2Vec(dm=1, dm_mean=1, size=20, window=8, min_count=1, workers=1, alpha=0.065,
d2v_model.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
/usr/local/lib/python3.8/dist-packages/gensim/models/doc2vec.py:570: UserWarning: The pa
  warnings.warn("The parameter `size` is deprecated, will be removed in 4.0.0, use `vect
100%|██████████| 5571/5571 [00:00<00:00, 1407025.21it/s]
```

```
for epoch in range(30):
    d2v_model.train(utils.shuffle([x for x in tqdm(train_tagged.values)]), total_examples=len
    d2v_model.alpha -= 0.002
    d2v_model.min_alpha = d2v_model.alpha
```

```
100%|██████████| 5571/5571 [00:00<00:00, 1463239.25it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1510730.43it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1481703.71it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1554551.77it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1524927.73it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1234426.94it/s]
100%|██████████| 5571/5571 [00:00<00:00, 600973.94it/s]
100%|██████████| 5571/5571 [00:00<00:00, 744012.85it/s]
100%|██████████| 5571/5571 [00:00<00:00, 774981.51it/s]
100%|██████████| 5571/5571 [00:00<00:00, 651039.75it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1344600.51it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1081530.55it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1406855.78it/s]
100%|██████████| 5571/5571 [00:00<00:00, 2793361.34it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1192531.77it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1544277.81it/s]
```

```
100%|██████████| 5571/5571 [00:00<00:00, 1238089.74it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1268744.51it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1092299.34it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1448813.71it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1497562.49it/s]
100%|██████████| 5571/5571 [00:00<00:00, 2555108.54it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1099857.26it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1252826.53it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1117852.35it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1339974.06it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1581272.76it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1443087.18it/s]
100%|██████████| 5571/5571 [00:00<00:00, 2773138.81it/s]
100%|██████████| 5571/5571 [00:00<00:00, 1473109.80it/s]
```

```
# save the vectors in a new matrix
embedding_matrix = np.zeros((len(d2v_model.wv.vocab)+ 1, 20))

for i, vec in enumerate(d2v_model.docvecs.vectors_docs):
    while i in vec <= 1000:
    #print(i)
    #print(model.docvecs)
          embedding_matrix[i]=vec
    #print(vec)
    #print(vec[i])
```

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Flatten, SimpleRNN
vocab_size = 100

model = Sequential()
model.add(Embedding(len(d2v_model.wv.vocab)+1,20,input_length=X.shape[1],weights=[embedding_m
model.add(layers.SimpleRNN(32, input_shape=(32, 100)))
model.add(Dense(2, activation='softmax'))
model.add(Flatten())
```

```
model.compile(optimizer="adam",loss="binary_crossentropy",metrics=['acc'])
#model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
```

```
Y = pd.get_dummies(df['Category']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.15, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(4736, 100) (4736, 2)
(836, 100) (836, 2)
```

```
batch_size = 32
history=model.fit(X_train, Y_train, epochs = 50, batch_size=batch_size, verbose = 1)
```

```
Epoch 1/50
148/148 [==============================] - 4s 21ms/step - loss: 0.2123 - acc: 0.9339
Epoch 2/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0446 - acc: 0.9882
Epoch 3/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0209 - acc: 0.9943
Epoch 4/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0093 - acc: 0.9981
Epoch 5/50
148/148 [==============================] - 3s 20ms/step - loss: 0.0051 - acc: 0.9989
Epoch 6/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0039 - acc: 0.9987
Epoch 7/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0046 - acc: 0.9989
Epoch 8/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0017 - acc: 0.9994
Epoch 9/50
148/148 [==============================] - 3s 21ms/step - loss: 9.6735e-04 - acc: 0.9
Epoch 10/50
148/148 [==============================] - 3s 21ms/step - loss: 5.4663e-04 - acc: 1.0
Epoch 11/50
148/148 [==============================] - 3s 21ms/step - loss: 3.8773e-04 - acc: 1.0
Epoch 12/50
148/148 [==============================] - 3s 21ms/step - loss: 2.8860e-04 - acc: 1.0
Epoch 13/50
148/148 [==============================] - 6s 38ms/step - loss: 2.2674e-04 - acc: 1.0
Epoch 14/50
148/148 [==============================] - 3s 22ms/step - loss: 1.8175e-04 - acc: 1.0
Epoch 15/50
148/148 [==============================] - 3s 21ms/step - loss: 1.5132e-04 - acc: 1.0
Epoch 16/50
148/148 [==============================] - 3s 22ms/step - loss: 1.2944e-04 - acc: 1.0
Epoch 17/50
148/148 [==============================] - 3s 22ms/step - loss: 1.0956e-04 - acc: 1.0
Epoch 18/50
148/148 [==============================] - 3s 21ms/step - loss: 0.0068 - acc: 0.9981
Epoch 19/50
148/148 [==============================] - 3s 22ms/step - loss: 0.0011 - acc: 0.9998
Epoch 20/50
148/148 [==============================] - 3s 21ms/step - loss: 4.8189e-04 - acc: 1.0
Epoch 21/50
148/148 [==============================] - 3s 21ms/step - loss: 1.7661e-04 - acc: 1.0
Epoch 22/50
148/148 [==============================] - 3s 21ms/step - loss: 1.2006e-04 - acc: 1.0
Epoch 23/50
148/148 [==============================] - 3s 21ms/step - loss: 9.4192e-05 - acc: 1.0
Epoch 24/50
148/148 [==============================] - 3s 21ms/step - loss: 7.7327e-05 - acc: 1.0
Epoch 25/50
148/148 [==============================] - 3s 21ms/step - loss: 6.5130e-05 - acc: 1.0
Epoch 26/50
148/148 [==============================] - 3s 21ms/step - loss: 5.5753e-05 - acc: 1.0
Epoch 27/50
148/148 [==============================] - 3s 23ms/step - loss: 4.8405e-05 - acc: 1.0
Epoch 28/50
148/148 [==============================] - 3s 21ms/step - loss: 4.2365e-05 - acc: 1.0
```

```
Epoch 29/50
```

```
plt.plot(history.history['acc'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.savefig('model_accuracy.png')

# summarize history for loss
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.savefig('model_loss.png')
```

```
validation_size = 200

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 1, batch_size = batch_size)

print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
20/20 [==============================] - 0s 6ms/step - loss: 0.1387 - acc: 0.9827
score: 0.14
acc: 0.98
```

```
message = ['thanks for accepting my request to connect']
seq = tokenizer.texts_to_sequences(message)

padded = pad_sequences(seq, maxlen=X.shape[1], dtype='int32', value=0)

pred = model.predict(padded)

labels = ['ham','spam']
print(pred, labels[np.argmax(pred)])
```

```
1/1 [==============================] - 0s 185ms/step
[[9.999944e-01 5.604740e-06]] ham
```

```
message = ['Congratulations! you have won a $1,000 Walmart gift card. Go to http://bit.ly/123
seq = tokenizer.texts_to_sequences(message)

padded = pad_sequences(seq, maxlen=X.shape[1], dtype='int32', value=0)

pred = model.predict(padded)

labels = ['ham','spam']
print(pred, labels[np.argmax(pred)])
```

```
1/1 [==============================] - 0s 34ms/step
[[1.3108779e-04 9.9986887e-01]] spam
```