Raaghav_94_NLP4_Hate

February 14, 2024

1 Hate Speech Identification

1.1 Pipeline

- Loading the input data.
- Cleaning and Preprocessing the data.
  - Lowercasing the text data.
  - Removing punctuations and stopwords.
  - Tokenizing the sentences.
- Generate representations for the features.
  - Word2Vec, FastText
    - Get the mean vectors for the sentences.
  - CNN, RNN
- Feeding the representation into a classifier model.
- Evaluating the model using test data with accuracy as the metric.

1.2 Import Libraries

[ ]:
```
import numpy as np
import pandas as pd
import regex as re
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import remove_stopwords
from nltk.stem.wordnet import WordNetLemmatizer
```

1.3 Load Data

[ ]:
```
tweet = pd.read_csv("labeled_data.csv")
tweet = tweet[['class', 'tweet']]
tweet.head()
```

[ ]:

	class	tweet
0	2	!!! RT @mayasolovely: As a woman you shouldn't…
1	1	!!!!! RT @mleew17: boy dats cold…tyga dwn ba…
2	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby…
3	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo…
4	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you…

[ ]:
```
X = tweet['tweet']
y = tweet['class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state=0, stratify=y)
```

1.4 Preprocess Data

[ ]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
lemma = WordNetLemmatizer()

def preprocess(text):
    text = re.sub('@[\w]+', '', text)
    text = simple_preprocess(remove_stopwords(text))
    return [lemma.lemmatize(str(word)) for word in text if word != 'rt']

train_sentences = [preprocess(text) for text in X_train]
test_sentences = [preprocess(text) for text in X_test]
```

1.5 Word Embeddings

[ ]:
```
from gensim.models.word2vec import Word2Vec
from gensim.models import FastText
```

1.5.1 Word2Vec

[ ]:
```
cbow_train = Word2Vec(train_sentences, vector_size=100, window=5, min_count=2,␣
 ↪sg=0)
cbow_test = Word2Vec(test_sentences, vector_size=100, window=5, min_count=2,␣
 ↪sg=0)
```

[ ]:
```
train_vocab = cbow_train.wv.index_to_key
test_vocab = cbow_test.wv.index_to_key

def get_mean_vector(model, sentence, vocab):
    words = [word for word in sentence if word in vocab]
    if len(words) >= 1:
        return np.mean(model.wv[words], axis=0)
    return np.zeros((100,))
```

[ ]:
```
cbow_array_train = []
cbow_array_test = []

for sentence in train_sentences:
    mean_vec = get_mean_vector(cbow_train, sentence, train_vocab)
    cbow_array_train.append(mean_vec)

for sentence in test_sentences:
    mean_vec = get_mean_vector(cbow_test, sentence, test_vocab)
    cbow_array_test.append(mean_vec)

cbow_array_train = np.array(cbow_array_train)
cbow_array_test = np.array(cbow_array_test)
```

1.5.2 FastText

[ ]:
```
fasttext_train = FastText(train_sentences, sg=1, workers=4, vector_size=100,␣
↪min_count=2,window=5)
fasttext_test = FastText(test_sentences, sg=1, workers=4, vector_size=100,␣
↪min_count=2,window=5)
```

[ ]:
```
fasttext_array_train = []
fasttext_array_test = []

for sentence in train_sentences:
    mean_vec = get_mean_vector(fasttext_train, sentence, train_vocab)
    fasttext_array_train.append(mean_vec)

for sentence in test_sentences:
    mean_vec = get_mean_vector(fasttext_test, sentence, test_vocab)
    fasttext_array_test.append(mean_vec)

fasttext_array_train = np.array(fasttext_array_train)
fasttext_array_test = np.array(fasttext_array_test)
```

1.6 Model Building

[ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

[ ]:
```
cbow_tree = SVC()
cbow_tree.fit(cbow_array_train, y_train)
```

[ ]:
```
SVC()
```

[ ]:
```
cbow_pred = cbow_tree.predict(cbow_array_test)
print(f"Accuracy for Word2Vec: {accuracy_score(y_test, cbow_pred):.4f}")
```

```
Accuracy for Word2Vec: 0.7660
```

[ ]:
```
fasttext_tree = SVC()
fasttext_tree.fit(fasttext_array_train, y_train)
```

[ ]:
```
SVC()
```

[ ]:
```
fasttext_pred = fasttext_tree.predict(fasttext_array_test)
print(f"Accuracy for FastText: {accuracy_score(y_test, fasttext_pred):.4f}")
```

```
Accuracy for FastText: 0.7751
```

1.7 CNN

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv1D, MaxPooling1D, Embedding,␣
 ↪Bidirectional, LSTM
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
```

[ ]:
```
top_words = 10000

tokenizer = Tokenizer()
train_sequences = tokenizer.texts_to_sequences(train_sentences)
train_pad_sequences = sequence.pad_sequences(train_sequences, maxlen=100)

test_sequences = tokenizer.texts_to_sequences(test_sentences)
test_pad_sequences = sequence.pad_sequences(test_sequences, maxlen=100)
```

[ ]:
```
model = Sequential([
    Embedding(top_words, 32, input_length=100),
    Conv1D(32, 3, padding='same', activation='relu'),
    MaxPooling1D(),
    Flatten(),
    Dense(250, activation='relu'),
    Dense(1, activation='softmax')
])

model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 embedding_8 (Embedding)     (None, 100, 32)           320000

 conv1d_4 (Conv1D)           (None, 100, 32)           3104

 max_pooling1d_4 (MaxPoolin  (None, 50, 32)            0
 g1D)

 flatten_4 (Flatten)         (None, 1600)              0

 dense_12 (Dense)            (None, 250)               400250

 dense_13 (Dense)            (None, 1)                 251

=================================================================
Total params: 723605 (2.76 MB)
Trainable params: 723605 (2.76 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
model.fit(train_pad_sequences, y_train, validation_data=(test_pad_sequences,␣
↪y_test), epochs=2, batch_size=128, verbose=2)
```

```
Epoch 1/2
155/155 - 7s - loss: -1.0494e+03 - accuracy: 0.7743 - val_loss: -5.8186e+03 -
val_accuracy: 0.7743 - 7s/epoch - 42ms/step
Epoch 2/2
155/155 - 6s - loss: -4.9316e+04 - accuracy: 0.7743 - val_loss: -1.3855e+05 -
val_accuracy: 0.7743 - 6s/epoch - 37ms/step
```

[ ]:
```
<keras.src.callbacks.History at 0x7b6358a3f910>
```

[ ]:
```
scores = model.evaluate(test_pad_sequences, y_test)
print(f"Accuracy for CNN: {(scores[1]*100):.4f}")
```

```
155/155 [==============================] - 1s 7ms/step - loss: -138546.8906 -
accuracy: 0.7743
Accuracy for CNN: 77.4259
```

1.8 RNN

[ ]:
```
rnn = Sequential([
    Embedding(top_words, 64, input_length=100),
    Bidirectional(LSTM(64)),
    Dense(1, activation='softmax')
])
rnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

[ ]:
```
history = rnn.fit(train_pad_sequences,␣
↪y_train,validation_data=(test_pad_sequences, y_test), epochs=2,␣
↪batch_size=128)
```

```
Epoch 1/2
155/155 [==============================] - 60s 345ms/step - loss: -2.1117 -
accuracy: 0.7743 - val_loss: -3.7022 - val_accuracy: 0.7743
Epoch 2/2
155/155 [==============================] - 53s 343ms/step - loss: -4.8768 -
accuracy: 0.7743 - val_loss: -6.0184 - val_accuracy: 0.7743
```

[ ]:
```
scores = rnn.evaluate(test_pad_sequences, y_test)
print(f"Accuracy for RNN: {(scores[1]*100):.4f}")
```

```
155/155 [==============================] - 7s 47ms/step - loss: -6.0184 -
accuracy: 0.7743
Accuracy for RNN: 77.4259
```

1.9 Result

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, cbow_pred, zero_division=True))
```

```
              precision    recall  f1-score   support

           0       1.00      0.00      0.00       286
           1       0.84      0.88      0.86      3838
           2       0.43      0.49      0.46       833

    accuracy                           0.77      4957
   macro avg       0.76      0.46      0.44      4957
weighted avg       0.78      0.77      0.75      4957
```

- Predicting the 'neutral' class is very difficult for the models.
- It classifies all the tweets only as either positive or negative.