s-in-nlp-using-bilstm-kikuyu-words

March 16, 2024

[91]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Embedding, Bidirectional, LSTM, Dense,Flatten
from sklearn.preprocessing import OneHotEncoder
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
```

[92]:
```python
# Load the dataset
data = pd.read_csv('/content/Kikuyu_Words.csv')
```

[93]:
```python
# Viewing the first 25 words with coresponding POS label
data.head(25)
```

[93]:
```
       Word Label
0     Mũndũ  Noun
1     Mũaki  Noun
2      Ihũa  Noun
3    Kĩrĩma  Noun
4    Gĩtĩri  Noun
5     Ikara  Verb
6      Rehe  Verb
7   Tengera  Verb
8      Rũga  Verb
9      koma  Verb
10   andika  Verb
11     tuma  Verb
12   Cukuru  Noun
13    Handũ  Noun
14   Mũteti  Noun
15    Mũiko  Noun
16    Mũitu  Noun
17  Ndereba  Noun
18   Mũrogi  Noun
```

```
19     Kĩhĩĩ  Noun
20  Mũrutani  Noun
21     Mũirũ  Noun
22     Mũici  Noun
23    Kiratũ  Noun
24   Mũrũthi  Noun
```

[94]:

```
# Preprocessing for easy tokenization
data["Word"] = data["Word"].str.replace("ũ", "u")
data["Word"] = data["Word"].str.replace("ĩ", "i")
data['Word'] = data['Word'].str.lower()
```

[95]:

```
data.head(25)
```

[95]:

```
        Word Label
0      mundu  Noun
1      muaki  Noun
2       ihua  Noun
3     kirima  Noun
4     gitiri  Noun
5      ikara  Verb
6       rehe  Verb
7    tengera  Verb
8       ruga  Verb
9       koma  Verb
10    andika  Verb
11      tuma  Verb
12    cukuru  Noun
13     handu  Noun
14    muteti  Noun
15     muiko  Noun
16     muitu  Noun
17   ndereba  Noun
18    murogi  Noun
19     kihii  Noun
20  murutani  Noun
21     muiru  Noun
22     muici  Noun
23    kiratu  Noun
24   muruthi  Noun
```

[96]:

```
# Encoding labels
label_encoder = LabelEncoder()
data['Label'] = label_encoder.fit_transform(data['Label'])
```

[97]:

```
# viewing words to be used as features
X = data['Word'].values
```

```
X
```

[97]:
```
array(['mundu', 'muaki', 'ihua', 'kirima', 'gitiri', 'ikara', 'rehe',
       'tengera', 'ruga', 'koma', 'andika', 'tuma', 'cukuru', 'handu',
       'muteti', 'muiko', 'muitu', 'ndereba', 'murogi', 'kihii',
       'murutani', 'muiru', 'muici', 'kiratu', 'muruthi', 'thiia',
       'kimbu', 'nugu', 'kingangi', 'kahiu', 'nyungu', 'gakaraku',
       'mutune', 'mweru', 'muiru', 'njau', 'mbakuri', 'twara', 'roga',
       'tura', 'rima', 'enda', 'onja', 'aka', 'endia', 'toga', 'rwara',
       'ria', 'hokeka', 'uma', 'thoma', 'enyuka', 'ora', 'agana', 'raiha',
       'kiga', 'kura', 'mwihokeku', 'mwonju', 'muthomu', 'muumu',
       'nyenyuku', 'njuru', 'njaganu', 'ndaihu', 'ngigu', 'nguru',
       'inyui', 'ithui', 'nii', 'othee', 'wee', 'atia', 'riria', 'nuu',
       'ma', 'umuthi', 'niki', 'tene', 'riu', 'hwaiini'], dtype=object)
```

[98]:
```
# Viewing target lables
y = data['Label'].values
y
```

[98]:
```
array([2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1])
```

[99]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[100]:
```
# Tokenize words and pad sequences
word_to_index = {word: idx + 1 for idx, word in enumerate(set(X))}
X_train_tokens = np.array([[word_to_index[word] for word in sentence.split()]␣
 ↪for sentence in X_train])
X_test_tokens = np.array([[word_to_index[word] for word in sentence.split()]␣
 ↪for sentence in X_test])
# sequence padding
max_sequence_length = max(max(len(x) for x in X_train_tokens), max(len(x) for x␣
 ↪in X_test_tokens))
X_train_padded = pad_sequences(X_train_tokens, maxlen=max_sequence_length,␣
 ↪padding='post')
X_test_padded = pad_sequences(X_test_tokens, maxlen=max_sequence_length,␣
 ↪padding='post')
```

[101]:
```
# Convert labels to one-hot encoding
num_classes = len(label_encoder.classes_)
y_train_one_hot = to_categorical(y_train, num_classes=num_classes)
y_test_one_hot = to_categorical(y_test, num_classes=num_classes)
```

[147]:
```
# Defining the BiLSTM model
model = Sequential()
model.add(Embedding(input_dim=len(word_to_index) + 1, output_dim=100,␣
 ↪input_length=max_sequence_length))
model.add(Bidirectional(LSTM(32, return_sequences=True)))
#Flatten layer to match the output shape
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
```

[148]:
```
# model summary
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 1, 100)            8100

 bidirectional_16 (Bidirect  (None, 1, 64)             34048
 ional)

 flatten_13 (Flatten)        (None, 64)                0

 dense_16 (Dense)            (None, 5)                 325

=================================================================
Total params: 42473 (165.91 KB)
Trainable params: 42473 (165.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[149]:
```
# Defining loss function and the appropriate optimizer
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[150]:
```
# Training the model
model.fit(X_train_padded, y_train_one_hot,
          epochs=10, batch_size=16,
          validation_data=(X_test_padded,
                           y_test_one_hot))
```

```
Epoch 1/10
4/4 [==============================] - 5s 313ms/step - loss: 1.6078 - accuracy:
0.3125 - val_loss: 1.6063 - val_accuracy: 0.2941
Epoch 2/10
4/4 [==============================] - 0s 19ms/step - loss: 1.5965 - accuracy:
0.5469 - val_loss: 1.6024 - val_accuracy: 0.4118
```

```
Epoch 3/10
4/4 [==============================] - 0s 22ms/step - loss: 1.5857 - accuracy:
0.6875 - val_loss: 1.5983 - val_accuracy: 0.4706
Epoch 4/10
4/4 [==============================] - 0s 16ms/step - loss: 1.5737 - accuracy:
0.6875 - val_loss: 1.5951 - val_accuracy: 0.4118
Epoch 5/10
4/4 [==============================] - 0s 19ms/step - loss: 1.5614 - accuracy:
0.7188 - val_loss: 1.5913 - val_accuracy: 0.3529
Epoch 6/10
4/4 [==============================] - 0s 16ms/step - loss: 1.5468 - accuracy:
0.7188 - val_loss: 1.5879 - val_accuracy: 0.2941
Epoch 7/10
4/4 [==============================] - 0s 20ms/step - loss: 1.5299 - accuracy:
0.7344 - val_loss: 1.5844 - val_accuracy: 0.2941
Epoch 8/10
4/4 [==============================] - 0s 20ms/step - loss: 1.5109 - accuracy:
0.7344 - val_loss: 1.5805 - val_accuracy: 0.2941
Epoch 9/10
4/4 [==============================] - 0s 15ms/step - loss: 1.4864 - accuracy:
0.7500 - val_loss: 1.5764 - val_accuracy: 0.2941
Epoch 10/10
4/4 [==============================] - 0s 15ms/step - loss: 1.4591 - accuracy:
0.8125 - val_loss: 1.5721 - val_accuracy: 0.2941
```

[150]:
```
<keras.src.callbacks.History at 0x78f652b2faf0>
```

[151]:
```python
# Evaluating the model
loss, accuracy = model.evaluate(X_test_padded, y_test_one_hot)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 0s 33ms/step - loss: 1.5721 - accuracy:
0.2941
Test Accuracy: 29.41%
```

[143]:
```python
# Function to Test of the model can tag a word to its respective POS
def predict_pos_tags(model, word_to_index, label_encoder):
    while True:
        # User to enter words
        user_input = input("Enter a sentence or a list of words (type 'exit' to␣
↪quit): ")
        if user_input.lower() == 'exit':
            break

        # Tokenize and pad the input
        tokens = [word_to_index[word] for word in user_input.split()]
```

```
        padded_tokens = pad_sequences([tokens], maxlen=max_sequence_length,␣
↪padding='post')

        # Predict POS tags
        predictions = model.predict(padded_tokens)

        # Decode predicted labels
        predicted_labels = label_encoder.inverse_transform(np.
↪argmax(predictions, axis=1))

        # Print the words along with their predicted POS tags
        for word, pos_tag in zip(user_input.split(), predicted_labels):
            print(f"{word}: {pos_tag}")

# Call the predict_pos_tags function
predict_pos_tags(model, word_to_index, label_encoder)
```

```
Enter a sentence or a list of words (type 'exit' to quit): ihua
1/1 [==============================] - 1s 757ms/step
ihua: Noun
Enter a sentence or a list of words (type 'exit' to quit): muaki
1/1 [==============================] - 0s 30ms/step
muaki: Noun
Enter a sentence or a list of words (type 'exit' to quit): rehe
1/1 [==============================] - 0s 32ms/step
rehe: Verb
Enter a sentence or a list of words (type 'exit' to quit): njaganu
1/1 [==============================] - 0s 21ms/step
njaganu: Verb
Enter a sentence or a list of words (type 'exit' to quit): nii
1/1 [==============================] - 0s 21ms/step
nii: Verb
Enter a sentence or a list of words (type 'exit' to quit): mwihokeku
1/1 [==============================] - 0s 23ms/step
mwihokeku: Verb
Enter a sentence or a list of words (type 'exit' to quit): exit
```

The Poor Performance of the model is caused by having few instances in training set ,thus the model is not able to learn alot of context on these words,therefore it will perform very dismally on new data