

Word embedding in NLP

Lecture 10

Prepared by

Dr. Hager Saleh

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import re
import nltk
```

Function to evaluate model

```
def evaluate_model(model_name, y_true, y_pred):  
  
    # Calculate metrics  
    accuracy = accuracy_score(y_true, y_pred)  
    precision = precision_score(y_true, y_pred, average='weighted')  
    recall = recall_score(y_true, y_pred, average='weighted')  
    f1 = f1_score(y_true, y_pred, average='weighted')  
    cm = confusion_matrix(y_true, y_pred)  
  
    # Create a report  
    report = classification_report(y_true, y_pred)  
  
    # Output results  
    metrics = {  
        'Model Name': model_name,  
        'Accuracy': accuracy,  
        'Precision': precision,  
        'Recall': recall,  
        'F1 Score': f1,  
  
        'Classification Report': report  
    }  
  
    return metrics
```

Word embedding

Word embedding is a technique in Natural Language Processing (NLP) where words or phrases from a vocabulary are mapped to dense vectors of real numbers in a continuous vector space. These vectors capture semantic meaning

Why use word embeddings?

Traditional methods like one-hot encoding treat words as independent and don't capture meaning or relationships. Word embeddings solve this by placing similar words close together in the vector space.

Key Characteristics:

- Low-dimensional (e.g., 50 to 300 dimensions)
- Continuous: Unlike sparse one-hot vectors
- Context-aware (in modern embeddings)

Different types of word embedding

Prediction-Based Embeddings Word2Vec is a neural network-based algorithm that converts words into dense vectors of real numbers — embedding the semantic meaning of words into numerical space.

Words that appear in similar contexts will have similar vectors, which allows the model to capture deep semantic relationships like:

- CBOW
 - predicts the context words from a target word
 - Goal: Given context words, predict the target (center) word.
 - Context: ["The", "sat", "on", "the"] Target: "cat"
- Skip-gram
 - predicts the target word from context words.
 - Example: "The cat sat on the mat."
 - Target: "cat"
 - Context (window size=2): ["The", "sat", "on", "the"]

Cbow

يتعلم التنبؤ بالكلمة المركزية باستخدام CBOW: هذه هي البيانات التي تستخدم لتدريب نموذج الكلمات المحيطة بها

```
window_size = 2

def generate_cbow_data(words, window_size):
    cbow_data = []
    for idx in range(window_size, len(words) - window_size):
        context = []
        for i in range(-window_size, window_size + 1):
            if i != 0:
                context.append(words[idx + i])
        target = words[idx]
        cbow_data.append((context, target))
    return cbow_data
```

```
cbow_data = generate_cbow_data(words, window_size)
```

```
print("CBOW training samples:")
pprint(cbow_data)
```

```
the quick brown fox jumps over the lazy dog
```

```
CBOW training samples:
```

```
[(['the', 'quick', 'fox', 'jumps'], 'brown'),
 ([ 'quick', 'brown', 'jumps', 'over'], 'fox'),
 ([ 'brown', 'fox', 'over', 'the'], 'jumps'),
 ([ 'fox', 'jumps', 'the', 'lazy'], 'over'),
 ([ 'jumps', 'over', 'lazy', 'dog'], 'the')]
```

Skipgram

```
def generate_skipgram_data(words, window_size):
    skipgram_data = []
    for idx in range(len(words)):
        for i in range(-window_size, window_size + 1):
            if i != 0 and 0 <= idx + i < len(words):
                target = words[idx]
                context = words[idx + i]
                skipgram_data.append((target, context))
    return skipgram_data

skipgram_data = generate_skipgram_data(words, window_size)
print("\nSkip-gram training samples:")
pprint(skipgram_data[:5])
```

Skip-gram training samples:

```
[('the', 'quick'),
 ('the', 'brown'),
 ('quick', 'the'),
 ('quick', 'brown'),
 ('quick', 'fox')]
```

Download library

```
##pip install gensim
## pip install tensorflow
## pip install keras

import nltk
from nltk.tokenize import word_tokenize
import gensim
import numpy as np
import random
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
import pandas as pd
```

Reading dataset

```
df = pd.read_csv("clean_dataset.csv",encoding="ISO-8859-1")
df=df.dropna()

df.head()
```

	sentiment	clean_text
0	negative	sooo sad miss san diego
1	negative	boss bulli

```

2 negative                interview leav alon
3 negative  son couldnt put releas alreadi bought
4 positive                feed babi fun smile coo

df['sentiment'].value_counts()

sentiment
positive    8579
negative    7777
neutral     3179
Name: count, dtype: int64

def preprocess(text):
    tokens = word_tokenize(text.lower())
    return tokens

df['tokens'] = df['clean_text'].apply(preprocess)

df['tokens']

0                [sooo, sad, miss, san, diego]
1                [boss, bulli]
2                [interview, leav, alon]
3      [son, couldnt, put, releas, alreadi, bought]
4                [feed, babi, fun, smile, coo]
...
19552                [enjoy, ur, night]
19553  [wish, could, come, see, u, denver, husband, l...
19554  [ive, wonder, rake, client, made, clear, net, ...
19555  [yay, good, enjoy, break, probabl, need, hecti...
19556                [worth]
Name: tokens, Length: 19535, dtype: object

#vector_size=100 → every word is represented as a 100-dimensional
vector.

cbow_model = Word2Vec(sentences=df['tokens'], vector_size=100,
window=5, min_count=1, sg=0) # sg=0 for CBOW

```

```
sentences=df['tokens']
```

This is the input data for training.
df['tokens'] is a column in your DataFrame that contains lists of tokens (words).

Example: ['i', 'love', 'this', 'product']

Each row is a sentence, and each sentence is already tokenized (split into words).

vector_size=100: Every word will be represented as a vector of 100 values.

window=5

This is the context window size.

It tells the model how many words before and after the target word to consider when learning context.

For example, if the word is "love", it will look at 5 words before and 5 words after to understand its context.

min_count=1

This sets the minimum number of times a word must appear in the corpus to be included in the model.

min_count=1 means even words that appear only once will be included.

If you set min_count=5, it will ignore all words that appear less than 5 times.

sg=0

This controls the training algorithm:

sg=0: Use CBOW – the model predicts a word based on its surrounding context.

sg=1: Use Skip-Gram – the model predicts context words from a given word.

CBOW is usually faster and works better for large datasets.

Average Word Vectors

Using averaged word vectors for each sentence, which gives you one fixed-size vector per sentence. This is perfectly fine for models like Logistic Regression, Random Fores.

```
# Average Word Vectors
def get_sentence_vector(tokens, model):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(model.vector_size)

df['Vector_Cbow'] = df['tokens'].apply(lambda x:
get_sentence_vector(x, cbow_model))

# Prepare data for classification
```

```
X = np.vstack(df['Vector_Cbow'].values)
y = df['sentiment'].values
```

Train-test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Applying machine learning models

```
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Model Name:

LogisticRegression

Accuracy: 0.6325

Precision: 0.5309

Recall: 0.6325

F1 Score: 0.5758

	precision	recall	f1-score	support
negative	0.59	0.78	0.67	1541
neutral	0.00	0.00	0.00	650
positive	0.68	0.74	0.71	1716
accuracy			0.63	3907
macro avg	0.42	0.51	0.46	3907
weighted avg	0.53	0.63	0.58	3907

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\linear_model\_logistic.py:469: 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(
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this
behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

```
# Predict and evaluate
```

```
y_pred = clf.predict(X_test)
```

```
# Evaluate the model
```

```
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)
```

```
# Print the evaluation results
```

```
for key, value in evaluation_results.items():
```

```

if key == 'Classification Report':
    print(value) # Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

Model Name:

RandomForestClassifier

Accuracy: 0.6243

Precision: 0.6015

Recall: 0.6243

F1 Score: 0.5845

	precision	recall	f1-score	support
negative	0.59	0.74	0.66	1541
neutral	0.46	0.06	0.11	650
positive	0.67	0.74	0.70	1716
accuracy			0.62	3907
macro avg	0.57	0.51	0.49	3907
weighted avg	0.60	0.62	0.58	3907

Skipgram

```

skipgram_model = Word2Vec(sentences=df['tokens'], vector_size=100,
window=5, min_count=1, sg=1) # sg=0 for CBOW

```

```

df['Vector_skipgram'] = df['tokens'].apply(lambda x:
get_sentence_vector(x, skipgram_model))

```

Prepare data for classification

```
X = np.vstack(df['Vector_skipgram'].values)
```

```
y = df['sentiment'].values
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

```

```
clf = LogisticRegression()
```

```
clf.fit(X_train, y_train)
```

Predict and evaluate

```
y_pred = clf.predict(X_test)
```

Evaluate the model

```

evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)

```

Print the evaluation results

```
for key, value in evaluation_results.items():
```

```

if key == 'Classification Report':
    print(value) # Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

Model Name:

LogisticRegression

Accuracy: 0.6726

Precision: 0.6721

Recall: 0.6726

F1 Score: 0.6203

	precision	recall	f1-score	support
negative	0.63	0.81	0.71	1541
neutral	0.66	0.03	0.06	650
positive	0.71	0.79	0.75	1716
accuracy			0.67	3907
macro avg	0.67	0.54	0.51	3907
weighted avg	0.67	0.67	0.62	3907

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\linear_model_logistic.py:469: 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(

```

```

clf = RandomForestClassifier()

```

```

clf.fit(X_train, y_train)

```

```

# Predict and evaluate

```

```

y_pred = clf.predict(X_test)

```

```

# Evaluate the model

```

```

evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)

```

```

# Print the evaluation results

```

```

for key, value in evaluation_results.items():

```

```

    if key == 'Classification Report':

```

```

        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")

```

Model Name:

RandomForestClassifier

Accuracy: 0.6680

Precision: 0.6400

Recall: 0.6680

F1 Score: 0.6327

	precision	recall	f1-score	support
negative	0.64	0.78	0.70	1541
neutral	0.45	0.10	0.17	650
positive	0.71	0.78	0.74	1716
accuracy			0.67	3907
macro avg	0.60	0.55	0.54	3907
weighted avg	0.64	0.67	0.63	3907

Applying Deep Learning

You cannot directly apply LSTM on these averaged vectors because an LSTM is designed to work with sequences — not with one fixed-length vector per sentence.

```

def get_sequence_vectors(tokens, model, max_len=20):
    sequence = [model.wv[word] for word in tokens if word in model.wv]

    # Padding or truncating to max_len
    if len(sequence) < max_len:
        padding = [np.zeros(model.vector_size)] * (max_len -
len(sequence))
        sequence += padding
    else:
        sequence = sequence[:max_len]

    return np.array(sequence)

from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

[0,1,0]
[1,0,0]

# Step 1: Label encode
label_encoder = LabelEncoder()
integer_labels = label_encoder.fit_transform(df['sentiment']) # e.g.,

```

```

'positive' → 2

# Step 2: One-hot encode
y = to_categorical(integer_labels)

print(label_encoder.classes_)

['negative' 'neutral' 'positive']

max_len = 20 # You can choose suitable max length
X = np.stack(df['tokens'].apply(lambda x: get_sequence_vectors(x,
cbow_model, max_len)).values)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Masking

model = Sequential()

model.add(Masking(mask_value=0.0, input_shape=(max_len,
cbow_model.vector_size)))

model.add(LSTM(200))

model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()

# Train
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.1)

```

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\keras\
src\layers\core\masking.py:47: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.

```

```

    super().__init__(**kwargs)

```

```

Model: "sequential_6"

```

Layer (type)	Output Shape
Param #	

masking_4 (Masking)	(None, 20, 100)	
0		
lstm_6 (LSTM)	(None, 200)	
240,800		
dense_6 (Dense)	(None, 3)	
603		

Total params: 241,403 (942.98 KB)

Trainable params: 241,403 (942.98 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

489/489 ————— 8s 14ms/step - accuracy: 0.5394 - loss: 0.9774 - val_accuracy: 0.7256 - val_loss: 0.7598

Epoch 2/10

489/489 ————— 7s 14ms/step - accuracy: 0.5829 - loss: 0.9142 - val_accuracy: 0.7504 - val_loss: 0.7494

Epoch 3/10

489/489 ————— 6s 13ms/step - accuracy: 0.5825 - loss: 0.9082 - val_accuracy: 0.7551 - val_loss: 0.7098

Epoch 4/10

489/489 ————— 7s 15ms/step - accuracy: 0.5879 - loss: 0.8931 - val_accuracy: 0.7540 - val_loss: 0.7254

Epoch 5/10

489/489 ————— 7s 14ms/step - accuracy: 0.5843 - loss: 0.8992 - val_accuracy: 0.7497 - val_loss: 0.7447

Epoch 6/10

489/489 ————— 6s 12ms/step - accuracy: 0.5979 - loss: 0.8805 - val_accuracy: 0.7397 - val_loss: 0.7481

Epoch 7/10

489/489 ————— 6s 12ms/step - accuracy: 0.6056 - loss: 0.8694 - val_accuracy: 0.7586 - val_loss: 0.7223

Epoch 8/10

489/489 ————— 6s 11ms/step - accuracy: 0.6126 - loss: 0.8616 - val_accuracy: 0.7517 - val_loss: 0.7389

Epoch 9/10

489/489 ————— 6s 12ms/step - accuracy: 0.6145 - loss: 0.8605 - val_accuracy: 0.7185 - val_loss: 0.7853

Epoch 10/10

489/489 ————— 6s 12ms/step - accuracy: 0.6038 - loss: 0.8596 - val_accuracy: 0.7346 - val_loss: 0.7380

<keras.src.callbacks.history.History at 0x1dd1a4ab380>

Predict and evaluate

y_pred = model.predict(X_test)

predicted_class_ids = y_pred.argmax(axis=1)

y_true_class_ids = y_test.argmax(axis=1)

Evaluate the model

evaluation_results = evaluate_model('LSTM', y_true_class_ids,
predicted_class_ids)

Print the evaluation results

for key, value in evaluation_results.items():

if key == 'Classification Report':

print(value) *# Print report separately for better readability*

else:

print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

123/123 ————— 1s 5ms/step

Model Name:

LSTM

Accuracy: 0.6383

Precision: 0.6007

Recall: 0.6383

F1 Score: 0.6082

	precision	recall	f1-score	support
0	0.62	0.70	0.66	1541
1	0.32	0.11	0.16	650
2	0.69	0.78	0.73	1716
accuracy			0.64	3907
macro avg	0.54	0.53	0.52	3907
weighted avg	0.60	0.64	0.61	3907