021-02-NLP-Embedding-solution

April 18, 2024

1 021-02 - NLP Embedding - Solution Notebook

- Written by Alexandre Gazagnes
- Last update: 2024-02-01

1.1 About

Context :

Let's get the party started !

Data :

You can find the dataset here.

1.2 Preliminaries

1.2.1 System

These commands will display the system information:

Uncomment theses lines if needed.

[ ]:
```
# pwd
```

[ ]:
```
# cd ..
```

[ ]:
```
# ls
```

Install various Librairies :

[ ]:
```
# !pip install -r requirements.txt >> pip.log
# !pip freeze >> pip.freeze
```

1.2.2 Import

[ ]:
```
import os, sys, warnings, secrets, datetime
import pickle
from IPython.display import display

# from joblib import dump, load
```

[ ]:

```
import pandas as pd
import numpy as np
```

[ ]:

```
# import matplotlib.pyplot as plt
# import seaborn as sns
import plotly.express as px
```

[ ]:

```
from sklearn.base import *
from sklearn.preprocessing import *
from sklearn.impute import *
from sklearn.model_selection import *
from sklearn.decomposition import *
from sklearn.ensemble import *
from sklearn.model_selection import *
from sklearn.pipeline import *
from sklearn.feature_extraction import *
from sklearn.dummy import *
from sklearn.feature_extraction.text import *

# from lightgbm import *
# from xgboost import *

from sklearn.linear_model import *
from sklearn.ensemble import *
from sklearn.neighbors import *
```

[ ]:

```
# import nltk

# import wordcloud

# from nltk.corpus import stopwords
# from nltk.corpus import words
# from nltk.tokenize import wordpunct_tokenize

# import string

import spacy

# from spacy.lang.en.stop_words import STOP_WORDS
```

[ ]:

```
import gensim

from gensim.models import KeyedVectors
from gensim.downloader import load

from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from gensim.parsing.preprocessing import preprocess_string
```

[ ]:

```
# import transformers
```

[ ]:

```
from openai import OpenAI
```

[ ]:

```
import requests  # HTTP client
```

1.2.3 Graphs and Settings

[ ]:

```
# sns.set()
```

[ ]:

```
warnings.filterwarnings("ignore")
# warnings.filterwarnings(action="once")
```

If needed we can use a TEST_MODE to run the notebook to have a very fast execution :

[ ]:

```
TEST_MODE = True
```

[ ]:

```
CV = 10  # number of folds for the  cross val
N_JOBS = 7  # number of cpu to use for computations
FRAC = 1.0  # we keep 100% of the dataframe
DISPLAY = True  # display complex viz
TEST_SIZE = 0.25  # Train vs Test %

if TEST_MODE:
    CV = 3
    N_JOBS = -1
    FRAC = 0.3
    DISPLAY = False
    TEST_SIZE = 0.5
```

1.2.4 Thrid Parties Tools

We need some Third parties :

[ ]:

```
# nltk.download("punkt")
# nltk.download("stopwords")
# nltk.download("words")
```

Some string assets :

[ ]:

```
# stop_words = set(stopwords.words("english"))
# punctuation = set(string.punctuation)
# word_dict = words.words()
```

We need to download spacy :

[ ]:

```
# !python -m spacy download en_core_web_sm
# !python -m spacy download en_core_web_md
# !python -m spacy download en_core_web_lg
```

Word2vect :

[ ]:
```
w2c = load("word2vec-google-news-300")
```

And to load spacy model :

[ ]:
```
# nlp = spacy.load("en_core_web_sm")

nlp = spacy.load("en_core_web_lg")
```

If you have an error please run the download command for spacy :

[ ]:
```
# !python -m spacy download en_core_web_lg
```

1.2.5 Data

url of the dataset :

[ ]:
```
url = "https://gist.githubusercontent.com/AlexandreGazagnes/
  ↪cabe445634a092d308d17a883a305a75/raw/
  ↪d2014e8a34bba3c1be3ec8936bb338fb42888f24/nlp.csv"
```

Download the dataset :

[ ]:
```
df = pd.read_csv(url)
df.head(5)
```

Keep a copy of the df :

[ ]:
```
DF = df.copy()
```

[ ]:
```
if TEST_MODE:
    df = df.sample(frac=FRAC)
```

1.3 King - Men + Woman

1.3.1 With Spacy

Tokenize 'King' :

[ ]:
```
king = nlp("king")
king
```

[ ]:
```
type(king)
```

Extract the vector :

[ ]:
```
king_v = king.vector
king_v
```

Length ?

[ ]:

```
len(king.vector)
```

Same for Man :

[ ]:

```
man = nlp("man")
man_v = man.vector
man_v
```

[ ]:

```
len(man_v)
```

Same for wooman :

[ ]:

```
woman = nlp("woman")
woman_v = woman.vector
woman_v
```

Fancy calculation !

[ ]:

```
res = king_v - man_v + woman_v
res
```

Length ?

[ ]:

```
len(res)
```

Reshape new vector :

[ ]:

```
res = res.reshape(1, -1)
res
```

Compute Similarity :

[ ]:

```
vectors = nlp.vocab.vectors.most_similar(res, n=20)
vectors
```

v1 is :

[ ]:

```
v1 = vectors[0][0][0]
v1
```

vect is :

[ ]:

```
vect = nlp.vocab[v1]
vect
```

text is :

[ ]:

```
vect.text
```

[ ]:

```
v2 = vectors[0][0][1]
vect = nlp.vocab[v2]
vect.text
```

[ ]:

```
v3 = vectors[0][0][2]
vect = nlp.vocab[v3]
vect.text
```

Whoooo .... not so good !

Lets do the same with a “huge” model :

[ ]:

```
# !python -m spacy download en_core_web_sm
# !python -m spacy download en_core_web_md
!python -m spacy download en_core_web_lg
# !python -m spacy download en_core_web_trf
```

[ ]:

```
nlp = spacy.load("en_core_web_lg")
```

Good ? ...

Just re-run previous cells with this code.

What are your conclusions ?

Let’s try another last trick :

[ ]:

```
doc = "He is one of the most famous kings:  Richard III was the last king of England to die in battle"
doc = nlp(doc)
king = doc[-7]
king
```

[ ]:

```
king_v = king.vector
```

[ ]:

```
doc = "Fifteen months after the death of King George VI, his daughter Elizabeth is crowned Queen of England"
doc = nlp(doc)
queen = doc[-3]
queen
```

[ ]:

```
queen_v = queen.vector
```

[ ]:

```
doc = "a female, it's a woman, or a lady, a human of female sex."
doc = nlp(doc)
woman = doc[6]
woman
```

[ ]:

```
woman_v = woman.vector
```

[ ]:

```
king_v = king.vector
```

[ ]:

```
doc = "a boy, a guy, or a man, it's a human being of male sex."
doc = nlp(doc)
```

```
man = doc[8]
man
```

[ ]:
```
man_v = man.vector
```

[ ]:
```
out = nlp.vocab.vectors.most_similar(queen_v.reshape(1, -1), n=20)
out
```

[ ]:
```
for t_id in out[0][0]:
    print(nlp.vocab[t_id].text)
```

1.3.2 With Doc2Vect

Let's do the same with Pretrained Doct2Vect :

[ ]:
```
result = w2c.most_similar(positive=["woman", "king"], negative=["man"], topn=10)
result
```

1.4 Using Gensim

1.4.1 Prepare Data

Create y vector :

[ ]:
```
y = df.cat_1
y
```

Create X :

[ ]:
```
X = df.description
X
```

Cross validation :

[ ]:
```
def cv():
    return StratifiedShuffleSplit(n_splits=CV, test_size=TEST_SIZE)


cv()
```

1.4.2 By Hand

Our documents :

[ ]:
```
documents = df.description
documents[:10]
```

Init spacy :

[ ]:
```
nlp = spacy.load("en_core_web_lg")
```

If you have an error please download en_core_web_lg model for spacy

Preprocess (clean) the corpus :

[ ]:

```
tokenized_docs = [
    [
        token.lemma_
        for token in nlp(doc.lower())
        if not token.is_stop and not token.is_punct
    ]
    for doc in documents
]
tokenized_docs[:10]
```

[ ]:

```
tokenized_docs[0]
```

Key concept here is a tagged document => Token + id

[ ]:

```
tagged_docs = [
    TaggedDocument(words=doc, tags=[i]) for i, doc in enumerate(tokenized_docs)
]
tagged_docs[:10]
```

1.5 Train the Doc2Vec model

1.5.1 Building Gensim Models

sm :

[ ]:

```
# 5s
model_sm = Doc2Vec(
    tagged_docs,
    vector_size=50,  # size of output vect
    window=2,  # nb words before and after a target word
    min_count=1,  # minimum frequency count of words. ,
    workers=4,  # number of cpu
    epochs=100,  # number of iterations (passes over the entire dataset)
)

model_sm
```

md :

[ ]:

```
# 10s
model_md = Doc2Vec(
    tagged_docs,
    vector_size=100,
    window=4,
    min_count=1,
    workers=4,
    epochs=200,
)
model_md
```

lg :

[ ]:

```
# 30s
model_lg = Doc2Vec(
    tagged_docs,
    vector_size=500,
    window=10,
    min_count=1,
    workers=4,
    epochs=500,
)
model_lg
```

xl :

[ ]:

```
# # 15m => 1h
# model_xxl = Doc2Vec(
#     tagged_docs,
#     vector_size=1_000,
#     window=10,
#     min_count=3,
#     workers=6,
#     epochs=2_000,
# )
# model_xxl
```

[ ]:

```
# 1m => 3m
model_xl = Doc2Vec(
    tagged_docs,
    vector_size=1_000,
    window=10,
    min_count=1,
    workers=4,
    epochs=1_000,
)
model_xl
```

Get the vectors :

[ ]:

```
# 2 => 5 mins with xl

doc_vectors = []
for doc in tokenized_docs:
    vector = model_xl.infer_vector(doc)
    doc_vectors.append(vector)
```

```python

doc_vectors
```

A more pythonic implementation

[ ]:
```python
doc_vectors = [model_xl.infer_vector(doc) for doc in tokenized_docs]
doc_vectors
```

Data Type :

[ ]:
```python
type(doc_vectors)
```

[ ]:
```python
type(doc_vectors[0])
```

Length ? :

[ ]:
```python
len(doc_vectors)
```

[ ]:
```python
df.shape
```

[ ]:
```python
len(doc_vectors[0])
```

[ ]:
```python
len(df)
```

Rebuild a ‘special’ X :

[ ]:
```python
X = pd.DataFrame(doc_vectors)
X
```

[ ]:
```python
X.to_csv("df_from_doc2vect_model_xl.csv", index=False)
```

Shape :

[ ]:
```python
X.shape
```

Grid :

[ ]:
```python
grid = GridSearchCV(
    RandomForestClassifier(),
    {},
    cv=CV,
    n_jobs=N_JOBS,
    return_train_score=True,
    verbose=1,
)
grid
```

[ ]:
```python
grid.fit(X, y)
```

[ ]:
```python
display(grid.best_estimator_)
```

Resultize :

[ ]:
```
def resultize(grid, head=20, export=False, token=None):

    res = grid.cv_results_
    res = pd.DataFrame(res)

    cols = [i for i in res.columns if "split" not in i]
    res = res.loc[:, cols]

    res = res.drop(columns=["mean_score_time", "std_score_time"])

    res = res.round(2).sort_values("mean_test_score", ascending=False)

    if export:
        _res = res.copy().head(head)
        _res["token"] = token
        _res = _res.astype(str)
        now = str(datetime.datetime.now())[:19].replace(" ", "_")
        _res.to_csv(f"results__{token}__{now}.csv", index=False)

    return res.head(head)
```

[ ]:
```
resultize(grid)
```

1.5.2 Using a pipeline

[ ]:
```
pipeline = Pipeline(
    [
        ("preprocessor", "passthrough"),
        ("scaler", "passthrough"),
        ("reductor", "passthrough"),
        ("estimator", LogisticRegression()),
    ]
)

pipeline
```

What is “passthrough” :

[ ]:
```
pst = "passthrough"

pst
```

Param grid :

[ ]:
```
param_grid = {
    "scaler": [
        "passthrough",
        StandardScaler(),
        QuantileTransformer(n_quantiles=100),
        Normalizer(),
    ],
    "reductor": [PCA()],
    "reductor__n_components": [0.7, 0.85, 0.9, 0.95, 0.99],
    "estimator": [RandomForestClassifier(), LogisticRegression()],
}
param_grid
```

New grid :

[ ]:

```
grid = GridSearchCV(
    pipeline,
    param_grid=param_grid,
    cv=CV,
    n_jobs=N_JOBS,
    return_train_score=True,
    verbose=1,
)
grid
```

[ ]:

```
grid.fit(X, y)
```

Results

[ ]:

```
display(grid.best_estimator_)
```

[ ]:

```
resultize(grid)
```

1.5.3 Using a custom transformer

Our Transformer :

[ ]:

```
class Doc2VecTransformer(BaseEstimator, TransformerMixin):
    """ """

    def __init__(
        self,
        model=None,
        vector_size=500,
        window=5,
        min_count=5,
        epochs=100,
    ):

        self.vector_size = vector_size
        self.window = window
        self.min_count = min_count
        self.epochs = epochs
        self.model = model

    def fit(self, X, y=None):

        if not isinstance(X, list):
            _X = X.values.tolist()
        else:
            _X = X

        if self.model:
            return self

        tagged_docs = [
            TaggedDocument(words=preprocess_string(doc), tags=[i])
            for i, doc in enumerate(_X)
        ]
        model = Doc2Vec(
            vector_size=self.vector_size, min_count=self.min_count, epochs=self.
↪epochs
        )
        model.build_vocab(tagged_docs)
        model.train(tagged_docs, total_examples=model.corpus_count,␣
↪epochs=model.epochs)
        self.model = model

        return self

    def transform(self, X, y=None):

        if not isinstance(X, list):
            _X = X.values.tolist()
        else:
            _X = X

        vectors = [self.model.infer_vector(preprocess_string(i)) for i in X]
        return vectors
```

Original df :

[ ]:
```
df
```

Init d2f :

[ ]:
```
d2v = Doc2VecTransformer()
d2v
```

Fit :

[ ]:

```
d2v.fit(df.description)
```

Transform :

[ ]:

```
text = ["my new watch is a very funny flic flac digital chronometer"]
vector_list = d2v.transform(text)
vector_list
```

To be sure :

[ ]:

```
vector_list[0]
```

With pretrained model :

[ ]:

```
text = ["my new watch is a very funny flic flac digital chronometer"]
d2v = Doc2VecTransformer(model=model_xl)
d2v.transform(text)
```

New param grid :

[ ]:

```
param_grid = {"preprocessor": [Doc2VecTransformer()]}
```

[ ]:

```
pipeline
```

New Grid :

[ ]:

```
grid = GridSearchCV(
    pipeline,
    param_grid,
    cv=CV,
    n_jobs=N_JOBS,
    return_train_score=True,
    verbose=2,
)
grid
```

Fit :

[ ]:

```
grid.fit(df.description.values, y)
```

[ ]:

```
grid.best_estimator_
```

Results :

[ ]:

```
resultize(grid)
```

Just as a remainder :

[ ]:

```
# # 30s
# model_lg = Doc2Vec(
#     tagged_docs,
#     vector_size=500,
#     window=10,
#     min_count=1,
#     workers=4,
#     epochs=500,
# )
# model_lg
```

Testing various transformers params :

[ ]:
```
param_grid = {
    "preprocessor": [Doc2VecTransformer()],
    "preprocessor__vector_size": [500],  # 100, 200, # 1000
    "preprocessor__window": [5],  # 5, 10, 5, 10,   # 20, 25, 30
    "preprocessor__epochs": [500],
    "preprocessor__min_count": [1, 3],  # 100, 300,  # 1000
    "estimator": [LogisticRegression(), RandomForestClassifier()],
    # preprocessor__model = [model_sm, model_md, model_lg, model_xl]
}
```

[ ]:
```
# param_grid = {
#     "preprocessor": [Doc2VecTransformer()],
#     "preprocessor__vector_size": [500, 1000],  # 100, 200, # 1000
#     "preprocessor__window": [5, 10, 15],  # 5, 10, 5, 10,   # 20, 25, 30
#     "preprocessor__epochs": [500, 1000],
#     "preprocessor__min_count": [1, 3, 5],  # 100, 300,  # 1000
#     # preprocessor__model = [model_sm, model_md, model_lg, model_xl]
# }
```

[ ]:
```
# param_grid = {
#     "preprocessor": [Doc2VecTransformer()],
#     "preprocessor__vector_size": [500],  # 100, 200, # 1000
#     "preprocessor__window": [5, 10],  # 5, 10, 5, 10,   # 20, 25, 30
#     "preprocessor__epochs": [500],
#     "preprocessor__min_count": [1, 3, 5],  # 100, 300,  # 1000
#     # preprocessor__model = [model_sm, model_md, model_lg, model_xl]
# }
```

[ ]:
```
# param_grid = {
#     "preprocessor": [Doc2VecTransformer()],
#     "preprocessor__vector_size": [500, 1000],  # 100, 200, # 1000
#     "preprocessor__window": [10, 15],  # 5, 10, 5, 10,   # 20, 25, 30
#     "preprocessor__epochs": [500, 1000],
#     "preprocessor__min_count": [1, 3, 5],  # 100, 300,  # 1000
#     # preprocessor__model = [model_sm, model_md, model_lg, model_xl]
# }
```

Grid :

[ ]:

```
grid = GridSearchCV(
    pipeline,
    param_grid,
    cv=CV,
    n_jobs=N_JOBS,
    return_train_score=True,
    verbose=2,
)
```

Fit :

[ ]:

```
grid.fit(df.description, y)
```

[ ]:

```
grid.best_estimator_
```

Results :

[ ]:

```
res = resultize(grid, head=30)
res
```

1.5.4 Post mortem analysis

Our problem is :

[ ]:

```
px.scatter_3d(
    res,
    x="param_preprocessor__vector_size",
    y="mean_test_score",
    z="param_preprocessor__window",
)
```

With box plots :

[ ]:

```
px.box(
    res,
    x="param_preprocessor__vector_size",
    y="mean_test_score",
    # color="param_preprocessor__window",
)
```

[ ]:

```
px.scatter(
    res.loc[res.param_preprocessor__window == 15],
    x="mean_score_time",
    y="mean_test_score",
    # color="param_preprocessor__window",
)
```

1.5.5 Exporting our model

[ ]:
```
best_pipeline = grid.best_estimator_
best_pipeline
```

Unique id for our model :

[ ]:
```
token = secrets.token_hex(4)
token
```

Date :

[ ]:
```
now = str(datetime.datetime.now())[:19].replace(" ", "_")
now
```

Filename :

[ ]:
```
fn = f"gensim_model__{token}__{now}.pk"
fn
```

Saving with pickle :

[ ]:
```
with open(fn, "wb") as f:
    pickle.dump(best_pipeline, f)
```

Size of our model :

[ ]:
```
sys.getsizeof(best_pipeline)
```

[ ]:
```
sys.getsizeof(grid)
```

With a function :

[ ]:
```
def save(model, base_fn, token=None, score=None):
    """Saving our model"""

    if not token:
        token = secrets.token_hex(4)

    now = str(datetime.datetime.now())[:19].replace(" ", "_")

    fn = f"{base_fn}__{token}__{now}"
    if score:
        fn += "__" + str(round(score, 2))

    with open(fn + ".pk", "wb") as f:
        pickle.dump(model, f)

    return fn, sys.getsizeof(model)
```

Save our model :

[ ]:

```
out = save(best_pipeline, "gensim_model")
out
```

Save our Transformer :

[ ]:

```
out = save(Doc2VecTransformer, "d2v_trasnformer")
out
```

Just to be sure :

[ ]:

```
best_pipeline.predict(["i have a beautiful analogic watch rolex"])
```

[ ]:

```
grid.predict(["i have a beautiful analogic watch rolex"])
```

1.5.6 Load

[ ]:

```
def load(fn):

    if not os.path.isfile(fn):
        raise AttributeError(f"The File {fn} do not exists!")

    with open(fn, "rb") as f:
        model = pickle.load(f)

    return model, sys.getsizeof(model)
```

List of .pk files :

[ ]:

```
fn_list = os.listdir()
fn_list
```

[ ]:

```
[i for i in fn_list if ".pk" in i]
```

Loading a model :

[ ]:

```
fn = "-- THE FILE NAME OF YOUR MODEL --"

loaded_pipeline, _ = load(fn)
loaded_pipeline
```

Testing our loaded model :

[ ]:

```
loaded_pipeline.predict(["i have a beautiful analogic watch rolex"])
```

1.6 Using OpenAI GPT Emedding

Init your client :

[ ]:

```
client = OpenAI()
client
```

Doc :

[ ]:

```
doc = df.description.iloc[0]
doc
```

Just a try :

[ ]:

```
response = client.embeddings.create(
    input=doc,
    model="text-embedding-3-small",
)
```

What is response :

[ ]:

```
response
```

The vector :

[ ]:

```
vector = response.data[0].embedding
vector
```

Size?

[ ]:

```
len(vector)
```

List of Vectors ?

[ ]:

```
# li = []

# for doc in df.description.values :
#     # print(i)

#     response = client.embeddings.create(
#         input=doc,
#         model="text-embedding-3-small",
#     )
#     vector = response.data[0].embedding
#     li.append(vector)

# li = pd.DataFrame(li)
# li.to_csv("df_from_gpt.csv", index=False)
# li
```

Or load this file :

[ ]:

```
li = pd.read_csv("df_from_gpt.csv")
li
```

With a custom transformer :

[ ]:

```
class OpenAIVecTransformer(BaseEstimator, TransformerMixin):

    def __init__(self, model="text-embedding-3-small"):

        self.model = model
        self.client = OpenAI()

    def fit(self, X, y=None):

        return self

    def transform(self, X, y=None):

        if not isinstance(X, list):
            _X = X.values.tolist()
        else:
            _X = X

        get_vect = (
            lambda i: self.client.embeddings.create(
                input=i,
                model=self.model,
            )
            .data[0]
            .embedding
        )
        X_ = [get_vect(i) for i in X]

        return X_
```

Let's build a very basic Pipeline / grid search :

[ ]:

```
pipeline = Pipeline(
    [
        ("preprocessor", "passthrough"),
        ("scaler", "passthrough"),
        ("reductor", "passthrough"),
        ("estimator", RandomForestClassifier()),
    ]
)

pipeline
```

What is “passthrough” :

[ ]:

```
pst = "passthrough"
pst
```

Param grid :

[ ]:

```
param_grid = {
    "scaler": [
        "passthrough",
        StandardScaler(),
        QuantileTransformer(n_quantiles=100),
        Normalizer(),
    ],
    # "reductor": [PCA()],
    # "reductor__n_components": [0.7, 0.85, 0.9, 0.95, 0.99],
    "estimator": [RandomForestClassifier()],  # LogisticRegression()
}
param_grid
```

New grid :

[ ]:

```
grid = GridSearchCV(
    pipeline,
    param_grid=param_grid,
    cv=CV,
    n_jobs=N_JOBS,
    return_train_score=True,
    verbose=1,
)

grid.fit(li, y)
```

Results

[ ]:

```
display(grid.best_estimator_)
```

[ ]:

```
resultize(grid)
```

1.7 Using our own API

[ ]:

```
url = "https://centrale-casa-api.onrender.com"


route = "/predict/"

data = "watches"
response = requests.get(url + route + data)

response.json()
```