012-03-NLP-Text-Processing-solution

April 18, 2024

1 012-03 - NLP Text Processing - Solution Notebook

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

1.1 About

Context :

We are gonna implement our 1st NLP tool !

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
```

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

[ ]:
```
# ls
```

Install various Librairies :

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

1.2.2 Import

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

[ ]:
```
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
```

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 = list(set(stopwords.words("english")))
# stop_words[:10]
```

[ ]:
```
punctuation = list(set(string.punctuation))
punctuation[:10]
```

[ ]:
```
word_dict = words.words()
word_dict[:10]
```

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
```

And to load spacy model :

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

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

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)
```

If needed let's take just a specific % of the dataframe :

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

[ ]:
```
df.cat_1.value_counts()
```

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

Keep a copy of the df :

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

1.3 First tour

1.3.1 Display

Sample 10 :

[ ]:
```
df.sample(10)
```

1.3.2 Structure

Info :

[ ]:
```
df.info()
```

Value counts :

[ ]:
```
df.dtypes.value_counts()
```

Specific data types :

[ ]:
```
df.select_dtypes(exclude=np.number).nunique()
```

1.3.3 Nan & Dupliacted

Any missing values :

[ ]:
```
tmp = df.isna().mean(axis=0)
tmp
```

[ ]:
```
tmp = df.isna().mean(axis=1)
tmp
```

Any duplicated :

[ ]:
```
df.duplicated().sum()
```

1.3.4 Data Inspection

Some numerical stats :

[ ]:
```
df.describe()
```

Other stats :

[ ]:
```
df.describe(include=object)
```

Select numeric :

[ ]:
```
df.select_dtypes(np.number).columns
```

Select non numeric :

[ ]:
```
df.select_dtypes(object).columns
```

1.4 Text Exploration

1.4.1 Display All

[ ]:
```
[print(i + "\n\n") for i in df.description.head().values]
```

[ ]:
```
[print(i + "\n\n") for i in df.description.tail().values]
```

[ ]:
```
[print(i + "\n\n") for i in df.description.sample(10).values]
```

1.4.2 Display by cat

[ ]:
```
lim = 200
```

[ ]:
```
i = 0


key = df.cat_1.unique()[i]
key
```

[ ]:
```
print("-------------------------------------------")
print(f"-------------- {key} --------------- ")
print("-------------------------------------------")
print("\n\n")


tmp = df.loc[df.cat_1 == key, :]
[print(i[:lim] + "\n\n") for i in tmp.description.head(5).values]
[print(i[:lim] + "\n\n") for i in tmp.description.sample(5).values]
[print(i[:lim] + "\n\n") for i in tmp.description.tail(5).values]
```

[ ]:
```
def print_categ(i):

    key = df.cat_1.unique()[i]

    print("-------------------------------------------")
    print(f"-------------- {key} --------------- ")
    print("-------------------------------------------")
    print("\n\n")

    tmp = df.loc[df.cat_1 == key, :]
    [print(i[:lim] + "\n\n") for i in tmp.description.head(5).values]
    [print(i[:lim] + "\n\n") for i in tmp.description.sample(5).values]
    [print(i[:lim] + "\n\n") for i in tmp.description.tail(5).values]
```

Print Categ 1 :

[ ]:
```
print_categ(1)
```

Print Categ 2 :

[ ]:
```
print_categ(2)
```

1.5 From Text To Vector

1.5.1 Tokenize with NLTK

Create doc from 1st description :

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

Tokenize :

[ ]:
```
tokens = nltk.word_tokenize(doc)
tokens
```

How Many Tokens?

[ ]:
```
len(tokens)
```

Our Stop words :

[ ]:
```
stop_words
```

Our punctuation :

[ ]:
```
punctuation
```

English dictionnary (lower) :

[ ]:
```
word_dict
word_dict = [i.lower() for i in word_dict]
word_dict[10000:10010]
```

Lets build a function :

[ ]:
```
def nltk_tokenizer(
    doc: str,
    len_min_word: int = 3,
    force_lower: bool = True,
    remove_stop_words=True,
    remove_punct=True,
    remove_all_digit=True,
    remove_any_digit=False,
    list_dict_word=None,
    list_extra_stop_word=None,
    remove_duplicate=False,
) -> str:

    if force_lower:
        doc = doc.lower()  # if force_lower else doc

    doc = doc.strip()

    tokens = nltk.word_tokenize(doc)

    if remove_stop_words:
        tokens = [t for t in tokens if t not in stop_words]

    if remove_punct:
        tokens = [t for t in tokens if t not in punctuation]

    if len_min_word > 0:
        tokens = [t for t in tokens if len(t) >= len_min_word]

    if remove_all_digit:
        tokens = [t for t in tokens if not t.isdigit()]

    if remove_any_digit:
```

```
        def has_a_digit(i):
            for char in i:
                if char.isdigit():
                    return True
            return False

        tokens = [
            t for t in tokens if not has_a_digit(i)
        ]  # any(map(str.isdigit, list(t)))]

    if list_dict_word:
        tokens = [t for t in tokens if t in list_dict_word]

    if list_extra_stop_word:
        tokens = [t for t in tokens if t not in list_extra_stop_word]

    if remove_duplicate:
        tokens = list(set(tokens))

    return " ".join(tokens)
```

[ ]:
```
res = nltk_tokenizer(doc)
print(res)
print(len(res))
```

1.5.2 Tokenize With Spacy

Same with spacy :

[ ]:
```
doc = "I'm so happy to live here, because this will be the most beautiful place␣
 ↪on earth!!!"
tokens = nlp(doc)
tokens
```

Part of speech :

[ ]:
```
for t in tokens:
    print(f"{t} => {t.pos_}")
```

Name Entity recognition :

[ ]:
```
for t in tokens:
    print(f"{t} => {t.ent_type_}")
```

Try it with Paris :

[ ]:
```
for t in nlp("i live in Paris"):
    print(f"{t} => {t.ent_type_}")
```

[ ]:
```
for t in nlp("i am in love with Paris Hilton"):
    print(f"{t} => {t.ent_type_}")
```

Type of tokens :

[ ]:
```
type(tokens)
```

Stop words :

[ ]:
```
doc = "I'm so happy to live here because this will be the most beautiful place␣
 ↪on earth"
tokens = nlp(doc)
tokens = [t for t in tokens if not t.is_stop]
tokens
```

Punctuation :

[ ]:
```
tokens[2].is_punct
```

[ ]:
```
doc = "I'm so happy to live here because this will be the most beautiful place␣
 ↪on earth"
tokens = nlp(doc)
tokens = [t for t in tokens if not t.is_punct]
tokens
```

Is Digit :

[ ]:
```
doc = "I'm so happy to live here because this will be the most beautiful place␣
 ↪on earth"
tokens = nlp(doc)
tokens = [t for t in tokens if not t.text.isdigit()]
tokens
```

Part of speech :

[ ]:
```
doc = "I'm so happy to live here because this will be the most beautiful place␣
 ↪on earth"
tokens = nlp(doc)

pos_list = ["NOUN", "VERB", "ADJ", "ADV"]

tokens = [t for t in tokens if t.pos_ in pos_list]
tokens
```

Lemmentization :

[ ]:
```
doc = "I'm so happy to live here because this will be the most beautiful place␣
 ↪on earth"
tokens = nlp(doc)
```

```
tokens = [t.lemma_ for t in tokens]
tokens
```

Let's create a function :

[ ]:
```
def spacy_tokenizer(
    doc,
    len_min_word=3,
    force_lower=True,
    remove_stop_words=True,
    remove_punct=True,
    remove_digit_token=True,
    remove_all_digit=True,
    pos_list=["NOUN", "VERB", "ADJ", "ADV"],
    lemmentize=True,
    list_dict_word=None,
    list_extra_stop_word=None,
):
    doc = doc.lower() if force_lower else doc

    doc = doc.strip()

    tokens = nlp(doc)

    if remove_stop_words:
        tokens = [t for t in tokens if not t.is_stop]

    if remove_punct:
        tokens = [t for t in tokens if not t.is_punct]

    tokens = [t for t in tokens if len(t) >= len_min_word]

    if remove_digit_token:
        tokens = [t for t in tokens if not t.text.isdigit()]

    if remove_all_digit:
        tokens = [t for t in tokens if not any(map(str.isdigit, list(t.text)))]

    if pos_list:
        tokens = [t for t in tokens if t.pos_ in pos_list]

    if lemmentize:
        tokens = [t.lemma_ for t in tokens]

    if list_dict_word:
        tokens = [t for t in tokens if t.text in list_dict_word]
```

```
    if list_extra_stop_word:
        tokens = [t for t in tokens if t.text not in list_extra_stop_word]

    return " ".join(tokens)
```

[ ]:
```
doc = df.description.iloc[0]
res = spacy_tokenizer(doc)
print(res)
print(len(res))
```

1.5.3 Count Vectorizer and TFIDF Vectorizer

Let's create an artificial corpus :

[ ]:
```
corpus = [
    "my cat is red",
    "my cat is blue",
    "my cat is yellow, i know that is wierd but he is yellow, yellow, yellow",
]
corpus
```

Building a pd.Series :

[ ]:
```
corpus = pd.Series(corpus, name="text")
corpus
```

Init a Count Vectorizer :

[ ]:
```
cv = CountVectorizer()
```

Fit :

[ ]:
```
# X = cv.fit_transform(corpus).toarray()

X = cv.fit(corpus)
X = cv.transform(corpus).toarray()
X.shape
```

Usefull dataframe :

[ ]:
```
X = pd.DataFrame(X, columns=cv.get_feature_names_out())
X
```

Same with TFIDF :

[ ]:
```
tf = TfidfVectorizer()
```

[ ]:
```
X = tf.fit_transform(corpus).toarray()
X.shape
```

[ ]:
```
X = pd.DataFrame(X, columns=tf.get_feature_names_out())
X
```

1.6 Modelisation

1.6.1 By Hand

[ ]:
```
df.head()
```

[ ]:
```
tf = TfidfVectorizer()

X = tf.fit_transform(df.description).toarray()
X.shape
```

[ ]:
```
X = pd.DataFrame(X, columns=tf.get_feature_names_out())
X
```

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

We can use a much more advanced cross validation tool :

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


cv()
```

Our grid Search :

[ ]:
```
grid = GridSearchCV(
    LogisticRegression(),
    {},
    cv=CV,
    n_jobs=N_JOBS,
    verbose=1,
    return_train_score=True,
)
grid.fit(X, y)
```

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

    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"])
```

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

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

1.6.2 Using Basic Pipeline

[ ]:
```
pipeline = Pipeline(
    [
        ("preprocessor", TfidfVectorizer()),
        # ("scaler", StandardScaler()),
        # ("reductor", TruncatedSVD(n_components=100)),
        ("estimator", RandomForestClassifier()),
    ]
)

pipeline
```

[ ]:
```
param_grid = {
    "estimator": [
        RandomForestClassifier(),
        # # KNeighborsClassifier(),
        # LGBMClassifier(),
        # XGBClassifier(),
        # XGBRFClassifier(),
        LogisticRegression(),
    ]
}
```

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

[ ]:
```
grid.fit(df.description, y)
```

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

1.6.3 Benchmark Pipelines

[ ]:
```
pst = "passthrough"

pipeline = Pipeline(
    [
```

```
        ("preprocessor", TfidfVectorizer()),
        ("imputer", pst),
        ("scaler", pst),
        ("reductor", pst),
        ("estimator", DummyClassifier()),
    ]
)
pipeline
```

[ ]:
```
param_grid = {
    "preprocessor": [CountVectorizer(), TfidfVectorizer()],
    "scaler": [
        StandardScaler(),
        QuantileTransformer(n_quantiles=100),
        # MinMaxScaler(),
        Normalizer(),
        # RobustScaler(),
    ],
    "imputer": [
        pst,
    ],
    "reductor": [
        pst,
    ],
    "estimator": [LogisticRegression(), RandomForestClassifier()],
}

param_grid
```

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

[ ]:
```
grid.fit(df.description.values, y)
```

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

1.6.4 Add Reductor

[ ]:
```
param_grid = {
    "preprocessor": [TfidfVectorizer(), CountVectorizer()],  #
    "scaler": [
        pst,
        StandardScaler(),
        QuantileTransformer(n_quantiles=100),
        Normalizer(),
    ],  # MinMaxScaler() RobustScaler()
    "imputer": [pst],
    "reductor": [TruncatedSVD(n_components=100)],
    "estimator": [
        # KNeighborsClassifier(),
        # XGBRFClassifier(),
        # LGBMClassifier(),
        # XGBClassifier(),
        LogisticRegression(),
        RandomForestClassifier(),
    ],
}

param_grid
```

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

[ ]:
```
grid.fit(df.description, y)
```

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

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

1.6.5 n_components

[ ]:
```
param_grid = {
    "preprocessor": [TfidfVectorizer(), CountVectorizer()],  #
    "scaler": [
        pst,
        StandardScaler(),
        QuantileTransformer(n_quantiles=100),
```

```
        Normalizer(),
    ],  # MinMaxScaler() RobustScaler()
    "imputer": [pst],
    "reductor": [TruncatedSVD()],
    "reductor__n_components": np.linspace(10, 1_000, 10).astype(int),
    "estimator": [
        # KNeighborsClassifier(),
        # # XGBRFClassifier(),
        # LGBMClassifier(),
        # XGBClassifier(),
        LogisticRegression(),
        RandomForestClassifier(),
    ],
}

param_grid
```

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

[ ]:
```
grid.fit(df.description, y)
```

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

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

1.6.6 Using Advanced Pipelines

[ ]:
```
class NltkTokenizer(BaseEstimator, TransformerMixin):
    def __init__(
        self,
        len_min_word=3,
        force_lower=True,
        remove_stop_words=True,
        remove_punct=True,
        remove_digit_token=True,
        remove_all_digit=True,
        list_dict_word=None,
        list_extra_stop_word=None,
    ):
```

```
        self.len_min_word = len_min_word
        self.force_lower = force_lower
        self.remove_stop_words = remove_stop_words
        self.remove_punct = remove_punct
        self.remove_digit_token = remove_digit_token
        self.remove_all_digit = remove_all_digit
        self.list_dict_word = list_dict_word
        self.list_extra_stop_word = list_extra_stop_word

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

    def transform(self, X, y=None):
        f = lambda i: nltk_tokenizer(
            i,
            len_min_word=self.len_min_word,
            force_lower=self.force_lower,
            remove_stop_words=self.remove_stop_words,
            remove_punct=self.remove_punct,
            # remove_digit_token=self.remove_digit_token,
            remove_all_digit=self.remove_all_digit,
            list_dict_word=self.list_dict_word,
            list_extra_stop_word=self.list_extra_stop_word,
        )

        return X.apply(f)
```

[ ]:
```
X = NltkTokenizer().fit_transform(df.description)
len(X)
```

[ ]:
```
pipeline = Pipeline(
    [
        ("tokenizer", NltkTokenizer()),
        ("preprocessor", TfidfVectorizer()),
        ("estimator", LogisticRegression()),
    ]
)
```

[ ]:
```
param_grid = {
    "tokenizer__force_lower": [True, False],
    "tokenizer__len_min_word": [1, 2, 3, 4, 5],
}
```

[ ]:
```
grid = GridSearchCV(
    pipeline,
    param_grid,
    cv=CV,
```

```
    n_jobs=N_JOBS,
    verbose=1,
    return_train_score=True,
)
display(grid)
```

[ ]:
```
grid.fit(df.description, y)
```

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

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

[ ]:
```
class SpacyTokenizer(BaseEstimator, TransformerMixin):
    def __init__(
        self,
        len_min_word=3,
        force_lower=True,
        remove_stop_words=True,
        remove_punct=True,
        remove_digit_token=True,
        remove_all_digit=True,
        list_dict_word=None,
        list_extra_stop_word=None,
    ):
        self.len_min_word = len_min_word
        self.force_lower = force_lower
        self.remove_stop_words = remove_stop_words
        self.remove_punct = remove_punct
        self.remove_digit_token = remove_digit_token
        self.remove_all_digit = remove_all_digit
        self.list_dict_word = list_dict_word
        self.list_extra_stop_word = list_extra_stop_word

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

    def transform(self, X, y=None):
        f = lambda i: spacy_tokenizer(
            i,
            len_min_word=self.len_min_word,
            force_lower=self.force_lower,
            remove_stop_words=self.remove_stop_words,
            remove_punct=self.remove_punct,
            remove_digit_token=self.remove_digit_token,
            remove_all_digit=self.remove_all_digit,
            list_dict_word=self.list_dict_word,
            list_extra_stop_word=self.list_extra_stop_word,
```

```
        )

        return X.apply(f)
```

[ ]:
```
X = SpacyTokenizer().fit_transform(df.description)
len(X)
```

[ ]:
```
pipeline = Pipeline(
    [
        ("tokenizer", SpacyTokenizer()),
        ("preprocessor", TfidfVectorizer()),
        ("estimator", LogisticRegression()),
    ]
)
```

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

[ ]:
```
# grid.fit(df.description, y)
```

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

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

[ ]:
```

```