Cómo construir e implementar un modelo de NLP

La implementación de modelos es una de las habilidades más importantes que debes tener si vas a trabajar con modelos de NLP.

La implementación del modelo es el proceso de integrar su modelo en un entorno de producción existente. El modelo recibirá información y predecirá una salida para la toma de decisiones para un caso de uso específico.

Hay diferentes formas de implementar su modelo de NLP en producción, puede usar Flask, Django, Bottle, etc. Pero en el artículo de hoy, aprenderá cómo construir e implementar su modelo de NLP con FastAPI.

En esta serie de artículos, aprenderá:

- Cómo construir un modelo de NLP que clasifique las reseñas de películas de IMDB en diferentes sentimientos.
- Qué es FastAPI y cómo instalarlo.
- Cómo implementar su modelo con FastAPI.
- Cómo utilizar su modelo de PNL implementado en cualquier aplicación Python.
- En la parte 1, nos centraremos en crear un modelo de NLP que pueda clasificar las reseñas de películas en diferentes sentimientos. ¡Entonces empecemos!
-

Cómo construir el modelo de PNL

Primero, necesitamos construir nuestro modelo de PNL. Usaremos el conjunto de datos de películas IMDB para construir un modelo simple que pueda clasificar si la reseña sobre la película es positiva o negativa. Estos son los pasos que debes seguir para hacerlo.
Dataset : https://www.kaggle.com/c/word2vec-nlp-tutorial/data?ref=hackernoon.com

Paquetes importantes

Primero, importamos paquetes importantes de Python para cargar datos, limpiarlos, crear un modelo de aprendizaje automático (clasificador) y guardar el modelo para su implementación.

```
# import important modules
import numpy as np
import pandas as pd

# sklearn modules
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB # classifier

from sklearn.metrics import (
```

```
    accuracy_score,
    classification_report,
    plot_confusion_matrix,
)
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# text preprocessing modules
from string import punctuation

# text preprocessing modules
from nltk.tokenize import word_tokenize

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re #regular expression

# Download dependency
for dependency in (
    "brown",
    "names",
    "wordnet",
    "averaged_perceptron_tagger",
    "universal_tagset",
):
    nltk.download(dependency)

import warnings
warnings.filterwarnings("ignore")
# seeding
np.random.seed(123)

# load data
data = pd.read_csv("../data/labeledTrainData.tsv", sep='\t')
```

Mostrar muestra del conjunto de datos.

```
# show top five rows of data
data.head()
```

Nuestro conjunto de datos tiene 3 columnas.

    Id: esta es la identificación de la revisión.
    Sentimiento: positivo(1) o negativo(0)
    Reseña - comentario sobre la película.

Verifique la forma del conjunto de datos.

```
# check the shape of the data
data.shape
```

Necesitamos verificar si al conjunto de datos le faltan valores.

```
# check missing values in data
data.isnull().sum()
```

Cómo evaluar la distribución de clases

Podemos usar el método value_counts() del paquete pandas para evaluar la distribución de clases de nuestro conjunto de datos.

```
# evalute news sentiment distribution
data.sentiment.value_counts()
```

Cómo procesar los datos

Después de analizar el conjunto de datos, el siguiente paso es preprocesarlo en el formato correcto antes de crear nuestro modelo de aprendizaje automático.

Las revisiones de este conjunto de datos contienen muchas palabras y caracteres innecesarios que no necesitamos al crear un modelo de aprendizaje automático.

Limpiaremos los mensajes eliminando palabras vacías, números y puntuación. Luego convertiremos cada palabra a su forma base utilizando el proceso de lematización en el paquete NLTK.

La función text_cleaning() se encargará de todos los pasos necesarios para limpiar nuestro conjunto de datos.

```
stop_words =  stopwords.words('english')

def text_cleaning(text, remove_stop_words=True, lemmatize_words=True):
    # Clean the text, with the option to remove stop_words and to lemmatize
word

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9]", " ", text)
    text = re.sub(r"\'s", " ", text)
    text =  re.sub(r'http\S+',' link ', text)
    text = re.sub(r'\b\d+(?:\.\d+)?\s+', '', text) # remove numbers

    # Remove punctuation from text
    text = ''.join([c for c in text if c not in punctuation])

    # Optionally, remove stop words
    if remove_stop_words:
        text = text.split()
        text = [w for w in text if not w in stop_words]
        text = " ".join(text)

    # Optionally, shorten words to their stems
    if lemmatize_words:
```

```
        text = text.split()
        lemmatizer = WordNetLemmatizer()
        lemmatized_words = [lemmatizer.lemmatize(word) for word in text]
        text = " ".join(lemmatized_words)

    # Return a list of words
    return(text)
```

Ahora podemos limpiar nuestro conjunto de datos usando la función text_cleaning().

```
#clean the review
data["cleaned_review"] = data["review"].apply(text_cleaning)
```

Luego divida los datos en variables de características y de destino.

```
#split features and target from  data
X = data["cleaned_review"]
y = data.sentiment.values
```

Nuestra característica para la capacitación es la variable clean_review y el objetivo es la variable de sentimiento.

Luego dividimos nuestro conjunto de datos en datos de entrenamiento y de prueba. El tamaño de la prueba es el 15% de todo el conjunto de datos.

```
# split data into train and validate

X_train, X_valid, y_train, y_valid = train_test_split(
    X,
    y,
    test_size=0.15,
    random_state=42,
    shuffle=True,
    stratify=y,
)
```

Cómo crear realmente nuestro modelo de NLP

Entrenaremos el algoritmo Multinomial Naive Bayes para clasificar si una reseña es positiva o negativa. Este es uno de los algoritmos más comunes utilizados para la clasificación de texto.

Pero antes de entrenar el modelo, debemos transformar nuestras revisiones limpias en valores numéricos para que el modelo pueda comprender los datos. En este caso, usaremos el método TfidfVectorizer de scikit-learn. TfidfVectorizer nos ayudará a convertir una colección de documentos de texto en una matriz de funciones TF-IDF.

Para aplicar esta serie de pasos (preprocesamiento y entrenamiento), usaremos una clase Pipeline de scikit-learn que aplica secuencialmente una lista de transformaciones y un estimador final.

```
Create a classifier in pipeline
sentiment_classifier = Pipeline(steps=[

('pre_processing',TfidfVectorizer(lowercase=False)),
                                   ('naive_bayes',MultinomialNB())
                                   ])
```

Luego entrenamos a nuestra clasificadora.

```
# train the sentiment classifier

sentiment_classifier.fit(X_train,y_train)
```

```
# test model performance on valid data
y_preds = sentiment_classifier.predict(X_valid)
```

El rendimiento del modelo se evaluará utilizando la métrica de evaluación precision_score. Usamos precision_score porque tenemos el mismo número de clases en la variable de sentimiento.

```
accuracy_score(y_valid,y_preds)
```

Guardar canalización del modelo

La canalización del modelo se guardará en el directorio del modelo utilizando el paquete joblib python.

```
#save model
import joblib

joblib.dump(sentiment_classifier, '../models/sentiment_model_pipeline.pkl')
```

Felicitaciones 👏👏, has llegado al final de esto.