**Web Platform per il training di modelli di NLP: DOCUMENTO TECNICO**

Introduzione:

Questo progetto rappresenta una Web Platform realizzata in Python che offre diversi servizi, tra cui la gestione di Intents, Entities e Training Phrases salvati come collezioni di documenti in un Database NoSQL, l’addestramento, il download, la cancellazione di tre modelli di NLP (Natural Language Processing), così come la visualizzazione dei risultati degli addestramenti e il loro testing.

Implementazione del server:

Il server è realizzato in Python utilizzando:

* Flask come WSGI Web App Framework, modulo Python che permette di sviluppare applicazioni web;
* Flask-PyMongo strettamente integrato con Flask e PyMongo; quest’ultimo rappresenta il Driver MongoDB ufficiale per Python che permette la gestione di Database NoSQL;
* Flask-SocketIO che fornisce alle applicazioni Flask l’accesso alle comunicazioni bidirezionali a bassa latenza tra il client e il server.

Per cui l’implementazione del server, che rappresenta il cuore dell’applicazione, è contenuta nel file **main.py**.

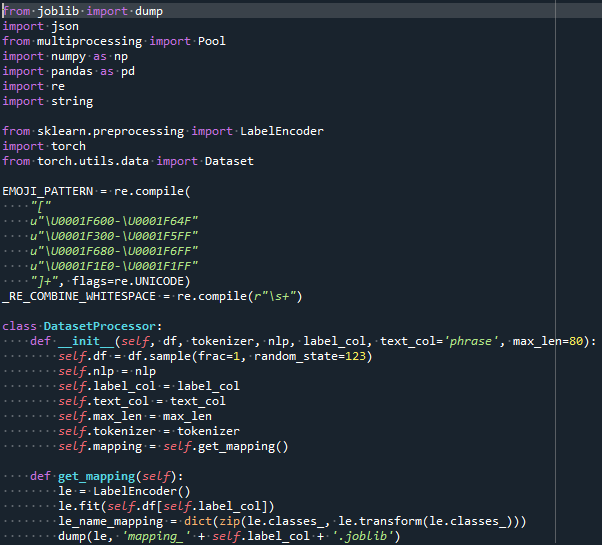
Addestramento dei modelli di IR e SA:

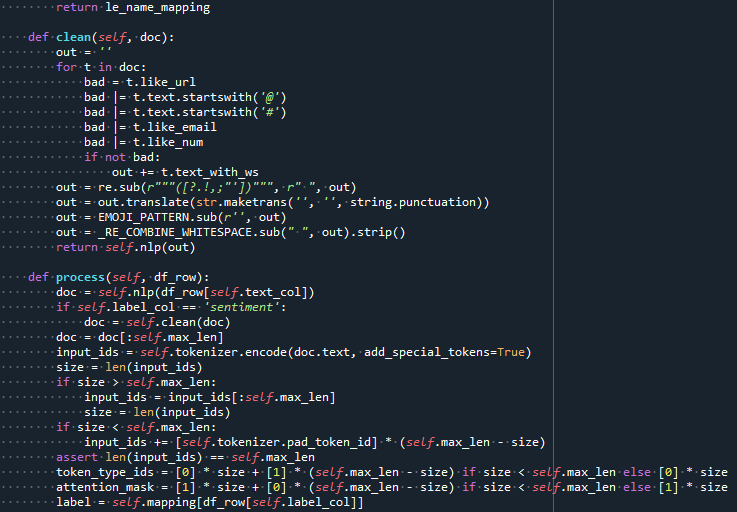
Il training di questi modelli è effettuato su thread paralleli (quindi è possibile avviare l’addestramento dei modelli in sequenza, anche EE) in Python utilizzando principalmente:

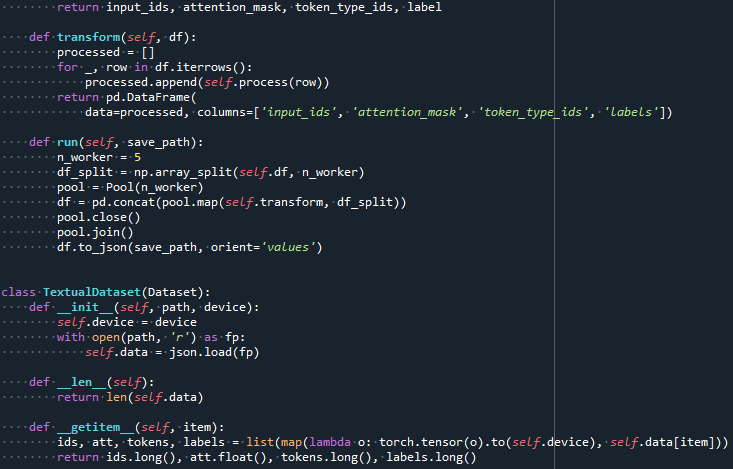
* spaCy, che rappresenta una libreria open-source progettata per lo sviluppo di applicazioni NLP;
* PyTorch, come pacchetto Python che fornisce Tensor Computation e Deep Neural Networks costruite su un sistema di autograd basato su nastro;
* Transformers che fornisce architetture di uso generale per NLU (Natural Language Understanding) e NLG (Natural Language Generation).

L’addestramento avviene attraverso due classi e diversi file elencati di seguito:

**dataset.py:**

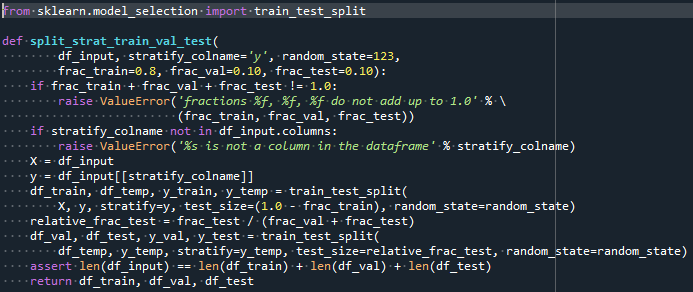






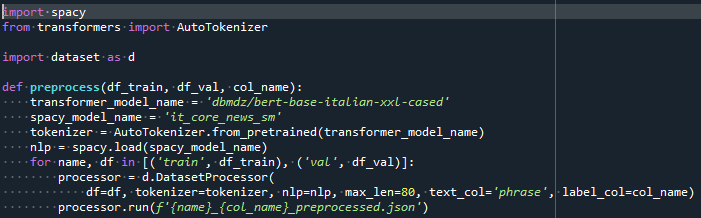
Nel file **dataset.py** ci sono le classi DatasetProcessor e TextualDataset che permettono il processing del dataset per l’addestramento dei modelli di IR (Intent Recognition) e di SA (Sentiment Analysis). Queste due classi vengono utilizzate principalmente in **preprocessing.py** e in **bert\_training.py**.

**splitting.py:**

****

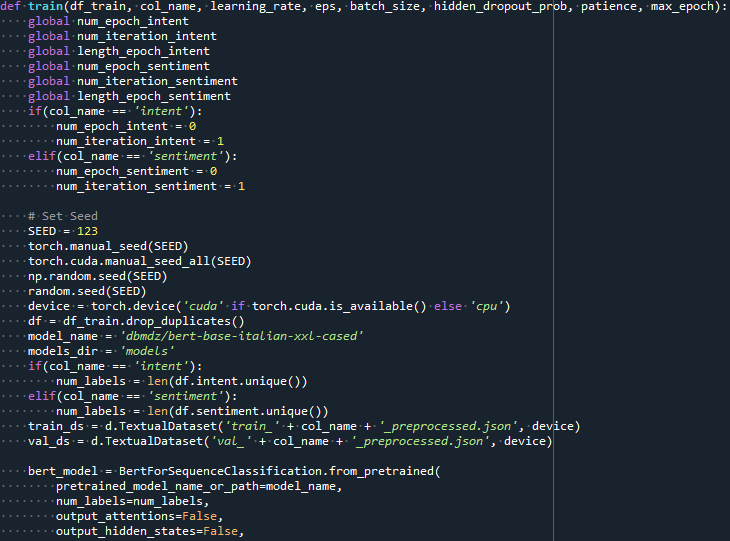
Nel file **splitting.py** avviene il processo dello splitting, cioè la divisione del testo in frasi.

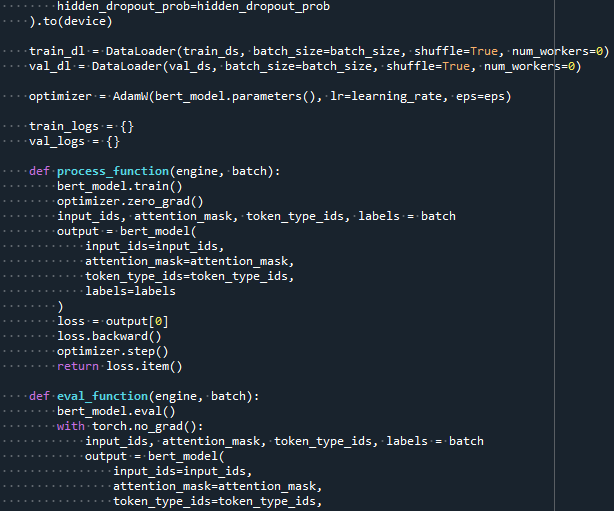
**preprocessing.py:**

****

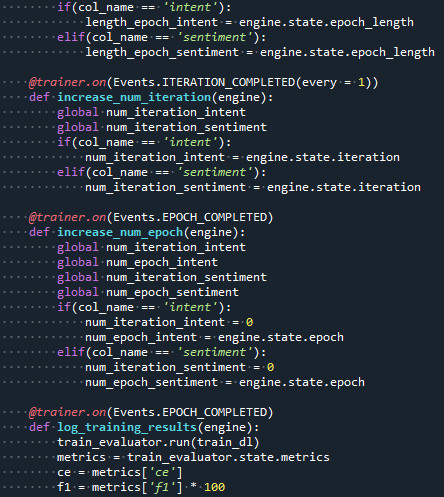
Nel file **preprocessing.py** avviene il preprocessamento del testo, applicato trasversalmente sia al corpus sul quale è svolto l'addestramento, sia agli input da sottoporre al modello addestrato, il cui obiettivo è semplificare e trasformare il dato testuale a livello di singola parola o token*.*

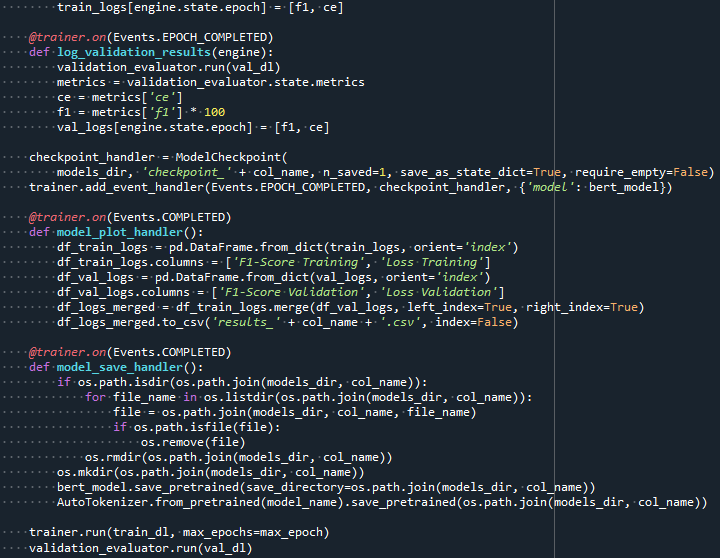
**bert\_training.py:**

****

****

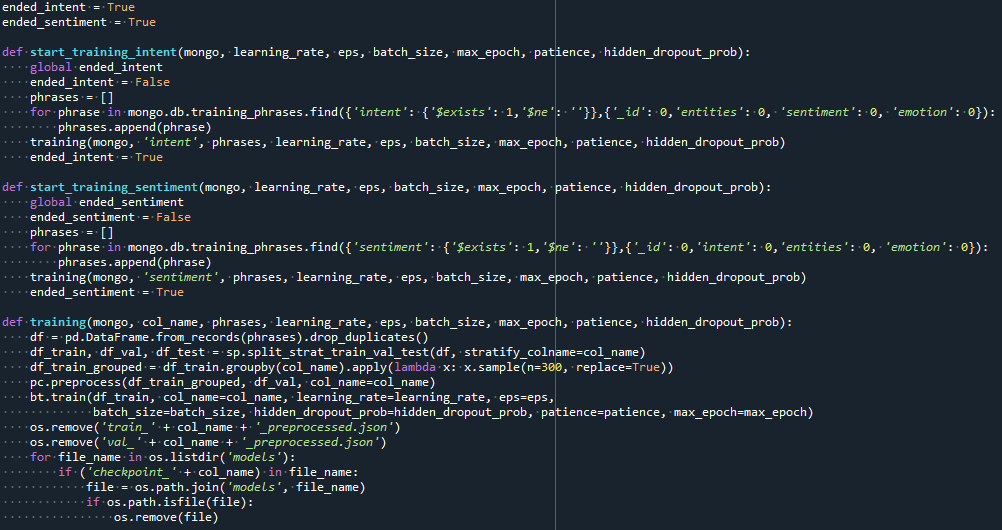
****

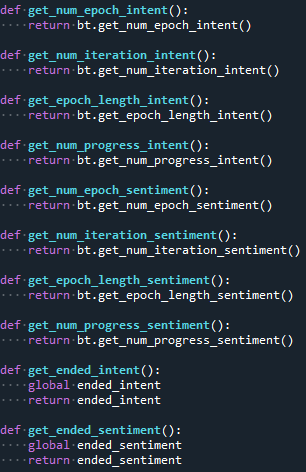
****

****

Nel file **bert\_training.py** la funzione principale è la funzione di train, che utilizza un algoritmo open source attualmente utilizzato da Google nel proprio motore di ricerca, cioè BERT(Bidirectional Encoder Representations from Transformers). BERT può comprendere parti di parlato come pronomi, fornendo un meccanismo per analizzare in maniera più efficace query scritte in linguaggio naturale.

**training.py**





Nel file **training.py** ci sono le funzioni che iniziano l’addestramento del modello su un thread parallelo, di IR o di SA, cioè start\_training\_intent e start\_training\_sentiment. Entrambe richiamano la funzione training, tra i cui parametri si ritrovano gli iperparametri utilizzati per l’addestramento del modello, cioè: learning rate, eps, batch size, max epoch, patience, hidden dropout. Essa si occupa dello splitting, del preprocessing e del bert training.

Infine ci sono funzioni get che permettono di visualizzare lo stato dell’addestramento in esecuzione nella pagina status relativa all’addestramento.