UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

BERT: addestramento e test di un modello per il riconoscimento del Sentiment dalle recensioni di film

Studenti:
Daniele Pallini 1107326
Matteo Abbruzzetti 1108842

Docenti:
Domenico Ursino
Gianluca Bonifazi
Michele Marchetti

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

1.1 BERT

BERT, acronimo di *Bidirectional Encoder Representations from Transformers*, è un framework open source di machine learning per il Natural Language Processing (NLP).

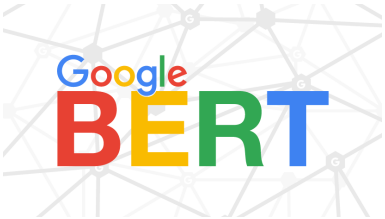

Figura 1.1: Logo di BERT

Si basa sui *Transformer*, un modello di deep learning in cui ogni elemento di output è collegato a ogni elemento di input e i pesi tra di essi sono calcolati dinamicamente in base alla loro connessione. Ciò che contraddistingue BERT dagli altri modelli di NLP è la *bidirezionalità*, ovvero la caratteristica che gli permette di leggere contemporaneamente da sinistra a destra e viceversa, operazione che i classici modelli di NLP non riuscivano a fare. Questo fattore, nonostante sembri una sottile differenza, in realtà è un vero e proprio stravolgimento, perché permette al modello una comprensione molto più accurata del contesto e, di conseguenza, della semantica di parole che possono creare *ambiguità*.

I modelli NLP basati sul deep learning necessitano di quantità di dati molto grandi: sono state sviluppate varie tecniche per addestrare modelli di rappresentazione linguistica generici utilizzando le enormi pile di testo non annotato presenti sul web (questa operazione è nota come pre-training). Questi modelli pre-addestrati per scopi generici possono poi essere raffinati (fine-tuned) su insiemi di dati più piccoli e specifici per le attività, ad esempio quando si lavora con problemi come il query answering e la sentiment analysis.

È proprio quest'ultimo il nostro caso: verrà addestrato un modello su un dataset contenente recensioni di film, al fine di distinguere i commenti positivi da quelli negativi. Si effettuerà, quindi, un'operazione di fine tuning per preparare il modello a riconoscere il sentiment e poi lo si testerà su dati di prova, registrando e commentando i risultati ottenuti.

1.2 Dataset

Il dataset scelto per il task (https://huggingface.co/datasets/rotten_tomatoes) contiene 5331 recensioni positive e 5331 recensioni negative prese da Rotten Tomatoes, un sito specializzato nella raccolta di recensioni cinematografiche. Di queste 10662 recensioni, l'80% (corrispondente a 8530) verrà utilizzato per addestrare il modello; il restante 20% (pari a 2132) sarà a sua volta suddiviso in due parti di uguale dimensione, che verranno utilizzate, rispettivamente, per l'operazione di *validation* e per testare il modello.

Ogni recensione del dataset è associata ad una label (Figura 1.2), che può assumere i valori "1" o "0". Questi valori stanno ad indicare se la recensione è positiva (valore "1" o "pos") o negativa (valore "0" o "neg") e saranno utilizzati dal modello come tag in fase di *training* e *validation*. Inoltre, in fase di *test*, le label saranno usate per verificare l'operato del modello su recensioni di cui non conosce il sentiment associato.

text (string)	label (class label)
"steers turns in a snappy screenplay that curls at the edges ; it's so clever you want to hate it . but he somehow pulls it off ."	1 (pos)
"take care of my cat offers a refreshingly different slice of asian cinema ."	1 (pos)
"this is a film well worth seeing , talking and singing heads and all ."	1 (pos)
"what really surprises about wisegirls is its low-key quality and genuine tenderness ."	1 (pos)
"(wendigo is) why we go to the cinema : to be fed through the eye , the heart , the mind ."	1 (pos)
"one of the greatest family-oriented , fantasy-adventure movies ever ."	1 (pos)
"ultimately , it ponders the reasons we need stories so much ."	1 (pos)

Figura 1.2: Esempi di recensioni e relative label

Capitolo 2

Preprocessing

2.1 Preparazione del dataset

Per poter effettuare la Sentiment Analysis, c'è bisogno di eseguire dei passaggi preliminari di pulizia dei dati. In questa sezione si vedranno in dettaglio le operazioni eseguite riguardo la preparazione del dataset.

2.1.1 Import del dataset

La prima operazione da compiere è installare il pacchetto *datasets* del sito *Hugging Face* e importare il dataset d'interesse (Listato 2.1). In particolare, il dataset *rotten_tomatoes* è strutturato nel modo visibile in Figura 2.1.

```
pip install datasets
from datasets import load_dataset
dataset = load_dataset('rotten_tomatoes')
rotten_train = load_dataset('rotten_tomatoes',split='train')
rotten_validation = load_dataset('rotten_tomatoes',split='validation')
rotten_test = load_dataset('rotten_tomatoes',split='test')
```

Listato 2.1: Import del dataset

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 8530
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
})
```

Figura 2.1: Struttura del dataset

2.1.2 Rimozione delle stopwords

Per poter avere un addestramento più efficiente, un'altra operazione da eseguire è la rimozione delle stopwords, ovvero delle parole che hanno la sola funzione di collegare i vari elementi di una frase, senza dare informazioni in merito al contesto o al significato della frase stessa. Questo riduce al minimo indispensabile il numero di parole processate e, contemporaneamente, permette di concentrarsi sui termini che, realmente, forniscono informazione. Nello Listato 2.2 si fa proprio questo; dopo aver importato le stopwords comuni della lingua Inglese si applica la funzione *clear_ text()* che va a rimuovere da ogni recensione le stopwords presenti.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
sw = stopwords.words('english')
df = pd.DataFrame(rotten_train["text"])
df["text"] = df["text"].apply(lambda x: clean_text(x))
df["label"] = pd.DataFrame(rotten_train["label"])
```

Listato 2.2: Rimozione delle stopwords

2.2 Preprocessing dell'input

Come accennato nel capitolo introduttivo, BERT si basa sui Transformer. Un transformer di base consiste in un encoder per leggere il testo in ingresso e in un decoder per produrre una predizione per il task di interesse. L'input dell'encoder di BERT è una sequenza di token, che vengono prima convertiti in vettori e poi elaborati dalla rete neurale. Un esempio di assegnazione di ID token (Listato 2.3) è rappresentato in Figura 2.2.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=
    True)
table = np.array([tokenizer.tokenize(reviews[0]),
        tokenizer.convert_tokens_to_ids(tokenizer.tokenize(reviews[0]))]).T
print(tabulate(table,headers = ['Tokens', 'Token IDs'],tablefmt = 'fancy_grid'))
```

Listato 2.3: Assegnazione di ID token

Tokens	Token IDs
rock	2600
destined	16036
st	2358
century	2301
new	2047

Figura 2.2: Esempio di assegnazione di ID token

2.2.1 Calcolo del max length

BERT ha bisogno di metadati aggiuntivi per l'elaborazione. Per far questo, si aggiunge un token [CLS] ai token delle parole in ingresso all'inizio della prima frase e inserendo un token [SEP] alla fine di ogni frase. Al fine di ottimizzare il processo, in questa fase viene calcolata anche la lunghezza massima della frase all'interno del dataset, che fungerà da upper bound (Listati 2.4, 2.5 e Figura 2.3).

```
for sent in reviews:
    input_ids = tokenizer.encode(sent, add_special_tokens=True)
    max_len = max(max_len, len(input_ids))
print('Max sentence length: ', max_len)
```

Listato 2.4: Calcolo della lunghezza massima delle frasi

```
Max sentence length:  69
```

Figura 2.3: Lunghezza massima della frase nel dataset

2.2.2 Indicizzazione e aggiunta dei token speciali

A questo punto, si effettua una serie di operazioni per preparare le frasi:

- Suddivisione delle frasi in token.
- Inserimento del token '[CLS]' all'inizio della frase.
- Inserimento del token '[SEP]' alla fine di ogni frase.
- Mappatura dei token con i rispettivi ID.
- Aggiunta di padding nella frase per rispettare la 'max_length'.
- Creazione di attention masks per i token [PAD].
- Conversione delle liste in tensori.

```
input_ids = []
attention_masks = []
for review in reviews:
    encoded_dict = tokenizer.encode_plus(
                        review, add_special_tokens = True,
                        max_length = max_len, truncation=True,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt')
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)
```

Listato 2.5: Assegnazione e aggiunta dei token speciali

In Figura 2.4 è rappresentato un esempio di mappatura di una frase del dataset[1] con i relativi ID token.

failed connections , divine secrets ya ya sisterhood nurturing , gauzy , dithering way

Tokens	Token IDs	Attention Mask
[CLS]	101	1
failed	3478	1
connections	7264	1
,	1010	1
divine	7746	1
secrets	7800	1
ya	8038	1
ya	8038	1
sister	2905	1
##hood	9021	1
nur	27617	1
##turing	16037	1
,	1010	1
ga	11721	1
##uz	17040	1
##y	2100	1
,	1010	1
di	4487	1
##ther	12399	1
##ing	2075	1
way	2126	1
[SEP]	102	1
[PAD]	0	0
[PAD]	0	0

Figura 2.4: Esempio di frase e token associati

[1] In questa fase il dataset comprende sia i dati di train sia quelli di validation

Capitolo 3

Sentiment Analysis e Fine Tuning

3.1 Scelta dei parametri

3.1.1 Split Train Validation

Come già affermato, il dataset *rotten_ tomatoes* è diviso in 3 parti: *train*, *validation* e *test*. Queste sezioni costituiscono rispettivamente l'80%, il 10% e il 10% della totalità del dataset. Si è scelto di rispettare e non alterare questa suddivisione per tutti i successivi task. Si utilizzeranno, quindi, le 8530 recensioni presenti in *train* per addestrare il modello, le 1066 recensioni in *validation* per l'operazione di validazione e le 1066 recensioni in *test* per testare il modello stesso.

3.1.2 Creazione del modello BERT

A questo punto, si procede con l'addestramento del modello. Si vanno a creare i DataLoaders per i set di addestramento e validazione (Listato 3.1). Dal momento che l'ordine in cui si prendono i dati incide sull'addestramento, verranno presi campioni del set di training in ordine casuale. Nella validazione si seguirà, invece, l'ordine sequenziale, non essendo, quest'ultimo, un fattore di disturbo.

Batch size

Il DataLoader necessita di conoscere la dimensione del batch per l'addestramento. Solitamente, è consigliato un valore pari a 16 o 32. Dopo una serie di test, il valore più adatto al task è risultato essere 32.

```
batch_size = 32
train_dataloader = DataLoader(train_dataset,
            sampler = RandomSampler(train_dataset),
            batch_size = batch_size )
validation_dataloader = DataLoader(val_dataset,
            sampler = SequentialSampler(val_dataset),
            batch_size = batch_size )
```

Listato 3.1: Creazione dei DataLoader

Learning rate e epsilon

Si va a caricare BertForSequenceClassification, un modello di BERT preaddestrato con un unico strato di classificazione lineare. Essendo una classificazione binaria (la recensione può essere positiva o negativa), il parametro *num_labels* è pari a 2. L'optimizer selezionato, denominato Adam, necessita di conoscere due parametri: *learning rate* e *epsilon*. Solitamente, è consigliato un valore di *learning rate* tra $5e^{-5}$, $3e^{-5}$ o $2e^{-5}$. Dopo una serie di test, il valore più adatto al task è risultato essere $2e^{-5}$. Per epsilon, invece, si è scelto di adottare il valore standard, ovvero $1e^{-8}$ (Listato 3.2).

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 2,
    output_attentions = False,
    output_hidden_states = False,)
model = model.to(device)

optimizer = AdamW(model.parameters(),
                  lr = 2e-5,
                  eps = 1e-8)
```

Listato 3.2: Creazione dei DataLoader

Numero di epoche

Altro parametro, da impostare prima di addestrare il modello, è rappresentato dal numero di epoche. È consigliato un numero tra 2, 3 e 4. Infatti, un numero troppo elevato di epoche può comportare over-fitting. Dopo accurati test, si è scelto di procedere con il numero 3, in quanto risultato essere il migliore nelle prestazioni (Listato 3.3).

```
epochs = 3

total_steps = len(train_dataloader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0,
                                            num_training_steps = total_steps)
```

Listato 3.3: Configurazione del numero di epoche

3.2 Addestramento del modello

Terminata la fase di scelta dei parametri, si procede con l'addestramento vero e proprio del modello. Verranno effettuate, quindi, le operazioni di training e validation con i dataset relativi.

3.2.1 Training

Il sistema, epoca per epoca, procede sequenzialmente con le operazioni di training e validation. Si avranno, di conseguenza, 3 sequenze training-validation, corrispondenti alle 3 epoche definite in precedenza (Listato 3.4). Alla fine di ogni ciclo si misurano e registrano i valori di *accuracy, precision, recall, specificity* e *train loss*.

```
for _ in trange(epochs, desc = 'Epoch'):

    # ========== Training ==========

    model.train()
    tr_loss = 0
    nb_tr_examples, nb_tr_steps = 0, 0

    for step, batch in enumerate(train_dataloader):
        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch
        optimizer.zero_grad()

        train_output = model(b_input_ids,
                             token_type_ids = None,
                             attention_mask = b_input_mask,
                             labels = b_labels)

        train_output.loss.backward()
        optimizer.step()

        tr_loss += train_output.loss.item()
        nb_tr_examples += b_input_ids.size(0)
        nb_tr_steps += 1

    # ========== Validation ==========

    model.eval()

    val_accuracy = []
    val_precision = []
    val_recall = []
    val_specificity = []

    for batch in validation_dataloader:
        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch
        with torch.no_grad():
          eval_output = model(b_input_ids,
                              token_type_ids = None,
                              attention_mask = b_input_mask)
        logits = eval_output.logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        b_accuracy, b_precision, b_recall, b_specificity = b_metrics(logits,
    label_ids)
        val_accuracy.append(b_accuracy)
```

Listato 3.4: Training e validation del modello

3.2.2 Risultati della validazione

I risultati ottenuti, suddivisi per epoca, sono riportati in figura 3.1.

```
Epoch:  33%|███       | 1/3 [01:44<03:29, 104.50s/it]
        - Train loss: 0.2005
        - Validation Accuracy: 0.8105
        - Validation Precision: 0.4954
        - Validation Recall: 0.7709
        - Validation Specificity: 0.8439

Epoch:  67%|██████    | 2/3 [03:28<01:44, 104.13s/it]
        - Train loss: 0.1108
        - Validation Accuracy: 0.8040
        - Validation Precision: 0.4963
        - Validation Recall: 0.7304
        - Validation Specificity: 0.8715

Epoch: 100%|████████| 3/3 [05:12<00:00, 104.11s/it]
        - Train loss: 0.0721
        - Validation Accuracy: 0.8048
        - Validation Precision: 0.4951
        - Validation Recall: 0.7920
        - Validation Specificity: 0.8123
```

Figura 3.1: Metriche di validazione per ogni epoca

3.3 Test del modello

Terminato l'addestramento del modello, si procede con la fase di test. In maniera analoga a quanto fatto nel Capitolo 2, innanzitutto si effettua il Preprocessing sul dataset di test. Successivamente, si esegue il test vero e proprio del modello (Listato 3.5). Un confronto tra la predizione del modello di alcune recensioni del dataset e l'effettivo sentiment delle stesse è riportato in Figura 3.2. In Figura 3.3 è rappresentata, invece, la matrice di confusione associata, che evidenzia il rapporto tra veri e falsi positivi e negativi. Nella matrice è riportata, anche, l'accuracy misurata su tutto il dataset. In generale, si registra una precisione del modello superiore alle 4 recensioni predette correttamente su 5.

```
model.eval()
predictions , true_labels = [], []
for batch in prediction_dataloader:
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask, b_labels = batch
    with torch.no_grad():
      outputs = model(b_input_ids, token_type_ids=None,
                      attention_mask=b_input_mask)
    logits = outputs[0]
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()
    predictions.append(logits)
    true_labels.append(label_ids)
```

Listato 3.5: Test del modello

	reviews	label	truelabel
0	lovingly photographed in the manner of a golde...	1	1
1	consistently clever and suspenseful .	1	1
2	it's like a " big chill " reunion of the baade...	1	1
3	the story gives ample opportunity for large-sc...	1	1
4	red dragon " never cuts corners .	0	1
...	...	...	...
1060	it's so downbeat and nearly humorless that it ...	0	0
1061	a terrible movie that some people will neverth...	1	0
1062	there are many definitions of 'time waster' bu...	0	0
1063	as it stands , crocodile hunter has the hurrie...	0	0
1064	the thing looks like a made-for-home-video qui...	0	0

1065 rows × 3 columns

Figura 3.2: Confronto della predizione con il sentiment effettivo

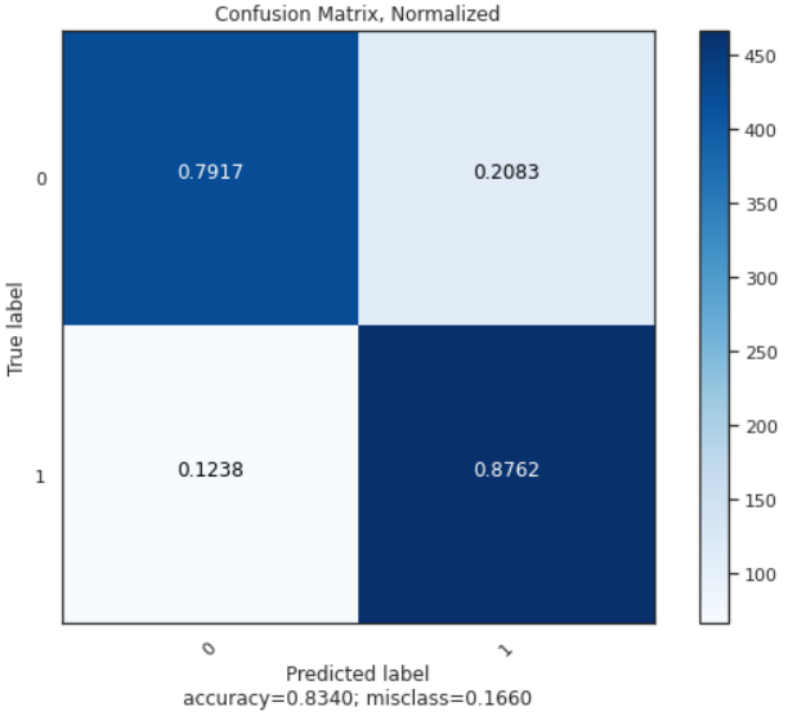

Figura 3.3: Matrice di confusione

Capitolo 4

Conclusioni

Si è utilizzato il framework BERT per un task di Sentiment Analysis su recensioni di film. Si è partiti dalla preparazione del dataset, con la rimozione delle stopwords e l'assegnazione dei token alle parole. Si è proseguito, poi, con la scelta dei parametri più adatti per il dataset di studio e con l'addestramento vero e proprio del modello. Sono stati, quindi, valutati i risultati ottenuti ed effettuati test di verifica delle prestazioni del modello.
In generale, BERT è risultato essere di semplice utilizzo ma molto potente. Inoltre, la caratteristica di essere open source lo rende molto appetibile come strumento per l'elaborazione del linguaggio naturale (Natural Language Processing).

Elenco delle figure

Lista dei codici