Estrazione automatica di informazioni da SMS: Progetto IA

Mattia d'Argenio, *Alessandro* Aquino e *Alberto* Montefusco

Abstract. Questo progetto si propone di sviluppare un sistema di Estrazione Automatica di Informazioni da SMS in lingua inglese utilizzando la libreria spaCy v.3.7.2 che contiene al suo interno modelli pre-addestrati per problemi di Natural Language Processing. L'obiettivo del progetto è stato quello di allenare un modello pre-addestrato (RoBERTa) che andasse ad associare dei tag in posizione specifiche di ogni testo di un SMS ed, infine, analizzarne le prestazioni finali.

1 Introduzione

1.1 Problema

Sempre più persone si scambiano SMS mediante l'utilizzo dei social media e l'analisi delle informazioni può essere utilizzata per effettuare statistiche nel comportamento e nella psicologia della gente. Utilizzando il Natural Language Processing (NLP), possiamo estrapolare da ogni messaggio parole chiave che ci permettono di raggiungere gli obiettivi proposti. Il seguente documento tratta lo sviluppo di un sistema di Estrazione Automatica di Informazioni da SMS in lingua inglese mediante l'uso della libreria spaCy che fornisce una serie di modelli pre-addestrati che usano la tecnica NER. La Named-entity recognition (acronimo NER) è un processo di estrazione di informazioni che cerca di trovare e classificare ogni singolo elemento presente in un testo all'interno di categorie predefinite come ad esempio persone, organizzazioni, luoghi, eventi, quantità, valute monetarie, percentuali e molto altro. Nel seguente caso, il modello preso in esame è RoBERTa che andremo ad analizzare nei paragrafi successivi.

1.2 Workflow

La prima operazione effettuata è stata individuare il dataset da utilizzare per il task introdotto nel paragrafo precedente. Il dataset utilizzato è stato: SMS-NER-Dataset-165-Annotations reperito su *kaggle* al seguente link. Successivamente è stato effettuata una data cleaning sul dataset in modo da garantire uniformità nella rappresentazione dei dati. Dopodiché il dataset pulito è stato diviso in training e testing set e convertito in formato `.spacy` in modo che potesse essere computato dal modello scelto. Successivamente, è stato generato il file `config.cfg` che non è altro un file di configurazione con tutti gli iperparametri e le impostazioni che il modello deve rispettare. Dopodiché, la parte di training è stata data in input al modello pre-addestrato e in output sono stati salvati due modelli:

1. `model-last`: il modello addestrato nell'ultima iterazione (si potrebbe utilizzarlo per riprendere l'addestramento in un secondo momento);
2. `model-best`: il modello che ha ottenuto il punteggio più alto sul dataset di test;

Infine, sono state riportate le metriche di Precision, Recall e F1-Score.

Per svolgere al meglio l'operazione di estrazione di informazioni, sono stati utilizzati 3 modelli pre-addestrati differenti nell'accuracy per la predizione dei tags e comparati tra di loro; in questo modo dai grafici riportati nel Paragrafo 3. si può capire quale modello ha ottenuto i risultati migliori. I modelli utilizzati sono stati:

1. `en_core_web_sm`;
2. `en_core_web_md`;
3. `en_core_web_lg`.
4. `en_core_web_trf`.

2 Implementazione

In questa sezione andremo a trattare le parti implementative. In particolare, parleremo della struttura del dataset, dei file di configurazione e quali modelli sono stati usati per l'estrazione delle informazioni dai messaggi SMS.

2.1 Dataset

Il dataset preso in esame è in formato `json` ed è strutturato nel seguente modo:

1. `"classes"`: contiene la lista dei tags che devono essere individuati all'interno dei messaggi: "MONEY", "TITLE", "OTP", "TRANSAC", "TIME", "PURPOSE".
2. `"annotations"`: contiene la lista dei messaggi e la classe entità per ogni messaggio;

(a) `"entities"`: ogni entità è un array di tuple dove ogni tupla ha al suo interno due interi e un tag (gli interi sono le coordinate del tag associato ad una specifica frase, es. `[19,26,"TRANSAC"]`).

Successivamente il dataset viene diviso in due parti: train e test set.
Se un messaggio ha la classe entità associata vuota, allora questa viene riempita con la tupla `[(0, 0, 'PEARSON')]`.

```
    for i in train_data:
    if not i[1]['entities']:
        i[1]['entities'] = [(0, 0, 'PERSON')]
    else:
        for j in range(len(i[1]['entities'])):
            i[1]['entities'][j] = tuple(i[1]['
    entities'][j])
```

Infine, i due mini dataset vengono ulteriolmente "puliti" eliminando i testi più lunghi di 512 caratteri e poi trasformati in formato `.spacy`, in modo tale da essere compatibili con il modello pre-addesstrato:

```

def make_doc_for_data(data):
    nlp = spacy.load("en_core_web_sm")

    db = DocBin() # create a DocBin object
    for text, annot in tqdm(data): # data in
    previous format
        if len(text) > 512:
            continue
        doc = nlp.make_doc(text) # create doc
    object from text
        ents = []
        for start, end, label in annot["entities
    "]: # add character indexes
            span = doc.char_span(start, end,
    label=label, alignment_mode="contract")
            if span is None:
                print("Skipping entity")
            else:
                ents.append(span)
        doc.ents = ents # label the text with
    the ents
        db.add(doc)

    return db

make_doc_for_data(train_data).to_disk("train.
    spacy") # save the docbin object
make_doc_for_data(test_data).to_disk("test.spacy
    ") # save the docbin object
```

2.2 File di Configurazione

All'interno della cartella `SMS-NER-Dataset-165-Annotations` troviamo il file di configurazione `base_config.cfg` utilizzato per impostare il modello che verrà addestrato sul dataset precedente.

Per impostare la struttura del modello eseguiamo il comando:

```
!python -m spacy init fill-config dataset/SMS-
    NER-Dataset-165-Annotations/base_config.cfg
    config.cfg
```

Dopodichè, partirà la fase di training e infine quella di testing lanciando il comando:

```
!python -m spacy train config.cfg --output ./
    output --paths.train train.spacy --paths.dev
     test.spacy
```

Per concludere, stampiamo le metriche prodotte dal modello migliore eseguendo il comando:

```
python -m spacy benchmark accuracy model/large/
    model-best model/large/test.spacy --output
    --code --gold-preproc --gpu-id 0 --displacy-
    path model/large
```

2.3 Iperparametri

Nel file di configurazione sono stati specificati gli iperparametri che abbiamo scelto al termine di alcune prove. Di seguito descriveremo la scelta di quelli più caratterizzanti:

- Modello Transformer:
  - name = "roberta-base": specifica che il modello transformer utilizzato è "roberta-base". Questo modello è una variante del modello BERT e ha dimostrato buone prestazioni nel nostro caso.
  - mixed_precision = false: imposta l'uso di precisione mista (mixed precision) a false. La precisione mista può essere utilizzata per accelerare l'addestramento utilizzando calcoli in precisione ridotta, ma in questo caso non è stata scelta perchè andava a degradare ulteriormente le metriche.
- Tokenizer:
  - use_fast = true: indica l'uso della versione veloce del tokenizzatore. I tokenizzatori veloci di spaCy sono implementati in Cython e offrono prestazioni migliori rispetto alla versione non veloce.
- Componente NER (Named Entity Recognition):
  - ubatch_size = 128: la dimensione del batch influisce sulla quantità di dati processati contemporaneamente durante l'addestramento. Batch più grandi possono accelerare l'addestramento ma richiedono più memoria. In base alle nostre possibilità il valore 128 ha dato un buon equilibrio tra velocità di addestramento ed efficienza di memoria;
  - dropout = 0.1: impedisce al modello di apprendere troppo dai dati di addestramento e di diventare troppo dipendente da particolari esempi. Il valore di 0.1 indica che durante l'addestramento il 10% dei nodi sarà "dropout" ad ogni passaggio, contribuendo a evitare il sovradattamento (overfitting);

  - hidden_width = 64: un valore più alto può fornire al modello una maggiore capacità di rappresentazione, ma può richiedere più risorse computazionali. La dimensione di 64 ha fornito un compromesso a livello di tempo accettabile;
  - use_upper = false: specifica se utilizzare le informazioni in maiuscolo durante l'addestramento.

- @optimizers = "Adam.v1":
  - Adam (Adaptive Moment Estimation) è un ottimizzatore che adatta il tasso di apprendimento per ciascun parametro. È spesso efficace nella pratica e richiede pochi iperparametri da regolare manualmente. Sono stati settati alcuni valori dell'optimizer per bilanciare la velocità di adattamento del tasso di apprendimento per garantire una convergenza stabile.

- Pianificazione del Tasso di Apprendimento: definisce come il tasso di apprendimento deve cambiare durante l'addestramento del modello.
  - @schedules = "warmup_linear.v1": lo scheduler di apprendimento warm-up linear è stato utilizzato per evitare problemi iniziali durante l'addestramento, soprattutto quando si utilizzano grandi modelli di trasformatori. Il warm-up gradualmente aumenta il tasso di apprendimento dai livelli bassi a quelli desiderati, consentendo al modello di stabilizzarsi in modo più efficace;
  - warmup_steps = 250: questo è il numero di passi di addestramento durante i quali il tasso di apprendimento viene gradualmente aumentato. Durante questa fase, il modello può adattarsi lentamente ai dati senza rischiare di saltare troppo velocemente in una fase di addestramento più intensiva;
  - total_steps = 20000: questo è il numero totale di passi di addestramento. Dopo il periodo di warm-up, il tasso di apprendimento può seguire un regime diverso, come una diminuzione graduale o altri schemi più complessi. La scelta di 20.000 passi è basata su prove empiriche, cercando di bilanciare il tempo di addestramento e la convergenza del modello.

2.4 RoBERTa

Il modello utilizzato per il tagging delle frasi è stato RoBERTa: un modello preaddestrato in modo autosupervisionato su un ampio set di dati inglesi. Questo significa che è stato pre-addestrato solo sui testi grezzi, senza che gli umani li etichettassero in alcun modo (motivo per cui può utilizzare moltissimi dati disponibili al pubblico) con un processo automatico per generare input ed etichette da tali testi.
Più precisamente, è stato pre-addestrato con l'obiettivo Masked Language Modeling (MLM): prendendo una frase, il modello maschera in modo casuale il 15% delle parole nell'input, quindi esegue l'intera frase mascherata attraverso il modello e deve prevedere le parole mascherate. Questo è diverso dalle tradizionali reti neurali ricorrenti (RNN) che di solito vedono le parole una dopo l'altra, o da modelli autoregressivi come GPT che mascherano internamente i futuri token. Permette al modello di apprendere una rappresentazione bidirezionale della frase.
In questo modo, il modello apprende una rappresentazione interna della lingua inglese che può poi essere utilizzata per estrarre funzionalità utili, ad esempio, se si dispone di un dataset di frasi etichettate, è possibile addestrare un classificatore che utilizza come input le caratteristiche prodotte dal modello BERT.

3 Analisi dei risultati

In questa sezione andremo a mostrare e a discutere dei risultati ottenuti dai diversi modelli stabilendo quale caso ha prodotto metriche migliori. Per una visione completa andare al GitHub del progetto.
Tra le metriche riportiamo la Precision, la Recall e l'F1-Score del modello generale ma anche le metriche che il modello ha ottenuto su ogni tag specifico. In particolare, la libreria spaCy usa delle metriche ad hoc che non sono altro quelle tradizionali, infatti:

- `NER P`: Precision del modello;
- `NER R`: Recall del modello;
- `NER F`: F1-Score del modello.

```
================================== Results ==================================

TOK     100.00
NER P   72.92
NER R   80.46
NER F   76.50
SPEED   406


============================== NER (per type) ==============================

              P       R       F
OTP       72.73   88.89   80.00
PURPOSE   62.50   65.22   63.83
TITLE     72.73   80.00   76.19
MONEY     69.23   81.82   75.00
TRANSAC  100.00  100.00  100.00
TIME      83.33  100.00   90.91
```

Figure 1. Modello small

```
================================== Results ==================================

TOK     100.00
NER P   76.09
NER R   80.46
NER F   78.21
SPEED   496


============================== NER (per type) ==============================

              P       R       F
OTP       70.00   77.78   73.68
PURPOSE   62.50   65.22   63.83
TITLE     78.79   86.67   82.54
MONEY     81.82   81.82   81.82
TRANSAC  100.00   88.89   94.12
TIME      83.33  100.00   90.91
```

Figure 2. Modello medium

```
================================== Results ==================================

TOK     100.00
NER P   75.82
NER R   79.31
NER F   77.53
SPEED   478


============================== NER (per type) ==============================

               P       R       F
OTP        87.50   77.78   82.35
PURPOSE    64.00   69.57   66.67
TITLE      71.88   76.67   74.19
MONEY      75.00   81.82   78.26
TRANSAC   100.00  100.00  100.00
TIME      100.00  100.00  100.00
```

Figure 3. Modello large

```
================================== Results ==================================

TOK     100.00
NER P   73.91
NER R   78.16
NER F   75.98
SPEED   483


============================== NER (per type) ==============================

               P       R       F
OTP        87.50   77.78   82.35
PURPOSE    60.00   65.22   62.50
TITLE      71.88   76.67   74.19
MONEY      75.00   81.82   78.26
TRANSAC   100.00  100.00  100.00
TIME       83.33  100.00   90.91
```

Figure 4. Modello transformer

Il modello che ha avuto un punteggio maggiore sul nostro datatset è stato il modello '`medium`' in termini di Precision, Recall e F1-Score.

4 Problemi riscontrati

Durante lo sviluppo del progetto abbiamo incontrato varie difficoltà che andremo ora ad analizzare nel dettaglio.
Innanzitutto, le metriche del modello NLP riportate non raggiungono il livello ottimale, rimanendo al di sotto dell'80%. Questo dato è in linea con le sfide comuni affrontate anche da altri ricercatori e sviluppatori nel campo, come evidenziato da diversi paper online. Il raggiungimento di prestazioni superiori nell'ambito dell'elaborazione del linguaggio naturale è una sfida complessa, influenzata da diversi fattori, tra cui la dimensione e la qualità del dataset di addestramento, l'architettura del modello e i parametri di ottimizzazione.
Infatti, un altro problema è stato quello della dimensione del dataset: abbiamo allenato il modello su 165 sample dove 132 sono stati utilizzati per il training e 33 per il testing. Avevamo anche un dataset con circa 100k sample di messaggi SMS ma la difficoltà era nel taggare a mano ogni singolo messaggio producendo un effort non indifferente soprattutto in termini di tempo.
Un'altra sfida che abbiamo affrontato è stata la limitatezza delle risorse hardware. Il tempo di addestramento per ciascun modello richiedeva approssimativamente 3/4 ore per completarsi, e su alcune macchine, l'addestramento si è rivelato impossibile a causa della pesantezza del modello che provocava il crash del sistema.

5 Data availability

Codice Workflow: GitHub.