DIGITIAMO: test di valutazione tecnica

Bacchetta Giacomo
e-mail: giacomo.bacchetta@gmail.com

***Abstract* – Questo documento si propone di descrivere il processo di sviluppo e utilizzo delle tecniche di machine learning per la classificazione di testi. In particolare, l'obiettivo è creare un processo automatico per la classificazione dei testi al fine di riconoscere la casa editrice di un dato testo, utilizzando le informazioni fornite nel dataset Kaggle. Il percorso inizia esplorando vari algoritmi di machine learning, tra cui Support Vector Machines (SVM), Random Tree e Random Forest, e la loro implementazione utilizzando la libreria scikit-learn. Vengono analizzate le fasi di pre-elaborazione dei dati testuali, come la rimozione dei caratteri speciali e della punteggiatura e l'estrazione delle caratteristiche utilizzando CountVectorizer. La performance del modello viene ottimizzata attraverso una Grid Search per trovare la migliore combinazione di iperparametri. L'efficacia del modello finale viene valutata su un set di dati di validation, e vengono analizzate metriche come l'accuratezza. Gli insights acquisiti da questo documento saranno utili per creare un sistema di classificazione dei testi robusto ed efficiente per identificare le case editrici in diversi dati testuali.**

I. Introduzione

Iniziamo questo lavoro con la fase di acquisizione dei dati, scaricando i dataset necessari dal provider ***Kaggle***. Una volta ottenuti i due file CSV contenenti informazioni riguardanti testi sia in inglese che in giapponese, procediamo alla loro lettura utilizzando la libreria ***Pandas***.

Concentrandoci inizialmente sul dataset in inglese, notiamo come questo sia composto da 36889 righe e 5 colonne, tra cui *Casa editrice*, *Autore*, *Titolo*, *Data*, e *Testo*. Prima di procedere con l'analisi, utilizziamo le funzioni *info()* e *describe()* di *Pandas* per verificare la presenza di valori nulli nel dataset. Notiamo che la colonna "Testo"presenta alcuni valori nulli. Poiché il testo è la variabile indipendente su cui costruiremo ed eseguiremo i nostri algoritmi, procediamo rimuovendo le righe che contengono tali valori nulli. Di conseguenza il numero di righe, e quindi di testi da analizzare, diminuisce passando da 36889 a 36766.
Una volta rimosse le righe con valori *null* nella colonna dei testi, procediamo a verificare se il dataset è bilanciato o meno. Utilizzando Pandas in collaborazione con un'altra libreria, come ***matplotlib***, notiamo che il dataset è altamente sbilanciato. Esistono solo due classi: 'The Japan Times' e 'Manichi Shimbum', con rispettivamente 36692 e 74 osservazioni. Questo sbilanciamento potrebbe causare problemi durante l'implementazione futura degli algoritmi di classificazione.

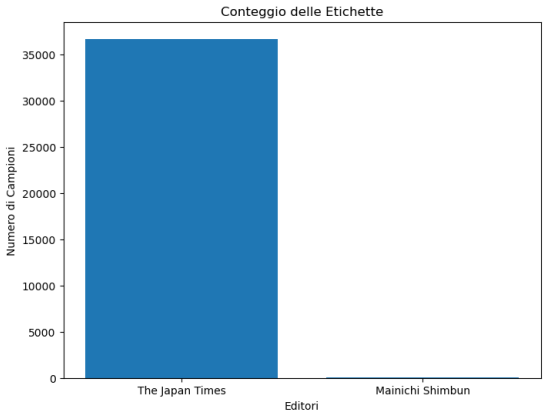

Fig. 1. Sbilanciamento delle classi

Per evitare tali problemi, è necessario adottare le giuste azioni di bilanciamento delle classi. Ci sono diverse strategie che possono essere utilizzate per affrontare il problema dello sbilanciamento dei dati, come l'**oversampling** della classe minoritaria, l'**undersampling** della classe maggioritaria o l'utilizzo di **pesi** delle classi durante l'addestramento. La scelta dell'undersampling come tecnica per affrontare lo sbilanciamento delle classi è una decisione valida. L'undersampling comporta la riduzione del numero di campioni, estraendoli casualmente, nella classe maggioritaria per renderla equamente distribuita rispetto alla classe minoritaria. Ciò aiuta a mitigare il problema dello sbilanciamento, consentendo agli algoritmi di apprendimento di gestire meglio le classi durante il training.
Dopo aver completato il bilanciamento del training set, procediamo alla conversione dei testi in ingresso, sia quelli del training set che del validation set (composto esclusivamente dai testi della classe maggioritaria), in vettori utilizzabili da un algoritmo. Per svolgere questa operazione, facciamo uso di ***CountVectorizer***, una classe fornita dalla libreria *scikit-learn* che svolge un ruolo fondamentale nel pre-processamento dei dati testuali.
CountVectorizer opera in modo semplice ma efficace. Prendendo in input una lista di testi, crea un vocabolario contenente tutte le parole uniche presenti nei testi stessi. Le parole vengono quindi convertite in vettori di conteggio, dove ogni elemento del vettore rappresenta il numero di occorrenze di una particolare parola nel testo.

II. Scelta e implementazione dei modelli

Nella seguente sezione, descriveremo le tecniche di *supervised learning* scelte per affrontare il problema di classificazione delle case editrici dei testi e le modalità con cui verranno applicate. È importante sottolineare come i parametri ottimali di ogni metodo descritto di seguito sono ricercati tramite l'implementazione di una **Grid Search**, assicurandoci di selezionare le configurazioni migliori per la classificazione

dei testi in base alle case editrici. Questa metodologia ci consentirà di ottenere modelli di classificazione altamente performanti e accurati.

A. **Support Vector Machine (SVM)**

Il modello di Support Vector Machine (SVM) è una potente tecnica di classificazione utilizzata per separare le osservazioni di diverse classi in uno spazio multidimensionale. SVM cerca di trovare l'iperpiano ottimale che massimizza il margine tra le classi, fornendo una classificazione precisa e generalizzabile.
Per ottenere prestazioni ottimali dal classificatore SVM, abbiamo eseguito una grid search per identificare i valori ottimali dei parametri *C*, *kernel* e *gamma*:

- C: 0.1;
- kernel: 'linear';
- gamma: 'scale'.

La decisione di utilizzare un SVM come punto di partenza è stata motivata principalmente da considerazioni accademiche. Durante l'ultimo anno del mio corso di laurea magistrale, ho avuto l'opportunità di partecipare attivamente, insieme a un team, nello sviluppo da zero di un classificatore lineare di questo tipo.

B. **Decision Tree**

L'algoritmo Decision Tree è un modello intuitivo di classificazione che crea un albero di decisione a partire dai dati di addestramento. Il modello prende decisioni sequenziali basate su condizioni sui dati, suddividendoli in sottogruppi sempre più specifici. Il nostro Decision Tree sarà addestrato sul dataset bilanciato, costruendo una struttura di alberi per classificare in modo efficace i testi sulla base delle caratteristiche dei dati.
Per ottenere prestazioni ottimali dal classificatore SVM, abbiamo eseguito una grid search per identificare i valori ottimali dei parametri *criterion*, *max_depth*:

- criterion: 'gini;
- max_depth: 'None'.

C. **Random Forest**

Il modello di Random Forest è una tecnica di ensemble learning che combina diversi alberi decisionali per ottenere una migliore performance di classificazione. Ogni albero è addestrato su un sottoinsieme casuale dei dati e la classificazione finale è ottenuta attraverso il voto di maggioranza dei singoli alberi. Il nostro Random Forest sarà addestrato sul dataset bilanciato, fornendo una soluzione robusta e accurata per la classificazione delle case editrici dei testi.
Per ottenere prestazioni ottimali dal classificatore SVM, abbiamo eseguito una grid search per identificare i valori ottimali dei parametri *criterion*, *max_depth* e *number of estimators*:

- criterion: 'gini;
- max_depth: 'None';
- number of estimators: 50.

III. Valutazione delle performance

Dopo aver implementato i modelli con i parametri ottimali, è essenziale valutare le loro prestazioni sia sul training set che sul validation set. Nel validation set sono presenti solo i testi della classe maggioritaria, poiché è stato precedentemente effettuato un bilanciamento per il training set.
Per valutare la qualità dei modelli, misuriamo l'accuratezza utilizzando *accuracy score* di scikit-learn.

$$Accuracy = \frac{NumerodiPredizioniCorrette}{NumeroTotalediPredizioni} \quad (1)$$

	Training Accuracy	**Validation Accuracy**
SVM	1.00000	0.99798
Decision Tree	1.00000	0.99801
Random Forest	1.00000	0.99801

Inoltre, riportiamo la **matrice di confusione** per valutare ulteriormente la qualità dei modelli. La matrice di confusione mostra il numero di predizioni corrette ed errate per ciascuna classe, consentendoci di identificare i veri positivi, i falsi positivi, i veri negativi e i falsi negativi.
Queste valutazioni ci aiutano a comprendere meglio le prestazioni dei modelli e a prendere decisioni informate sulla loro efficacia nella risoluzione del problema di classificazione.

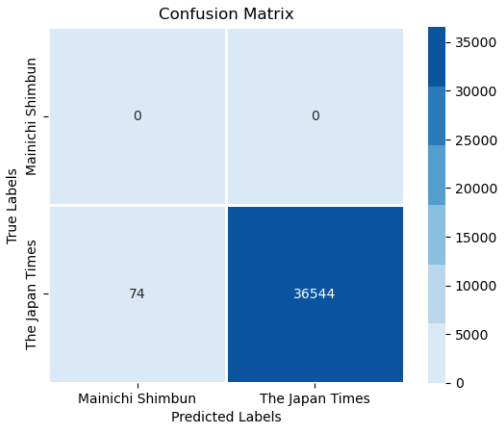

Fig. 2. SVM: matrice di confusione sul validation set

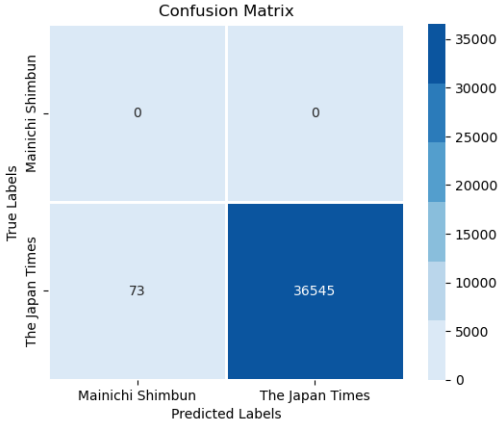

Fig. 3. Decision Tree: matrice di confusione sul validation set

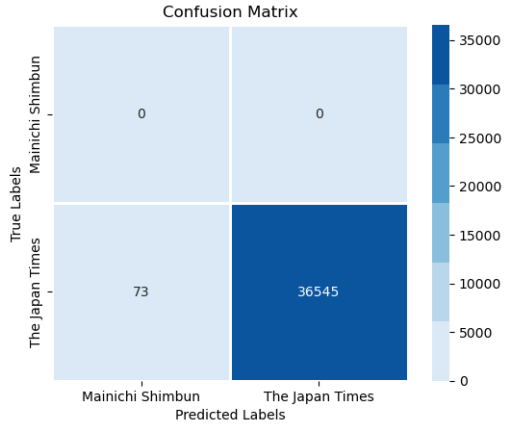

Fig. 4. Random forest: matrice di confusione sul validation set

IV. Text classification: lingua giapponese

Come anticipato nella fase introduttiva, il link di riferimento di *Kaggle* fornisce due diversi dataset: uno contenente testi in lingua inglese e l'altro con testi in lingua giapponese, anch'esso sbilanciato e quindi oggetto di **undersampling**. Finora abbiamo visto come le tecniche descritte funzionino bene per la lingua inglese, ma lo stesso non può dirsi per la lingua giapponese.

Questa lingua è composta da ideogrammi, chiamati "kanji"e "hiragana", i quali possono assumere significati diversi a seconda del contesto e della combinazione con altri ideogrammi. A causa di questa complessità, l'utilizzo del solo *CountVectorizer()* per la tokenizzazione dei testi in giapponese risulta problematico, in quanto non riesce a identificare frasi all'interno del testo e tende a trattare l'intero testo come una singola parola, portando le tecniche di classificazione a problemi di **overfitting**.

Per superare questa difficoltà, abbiamo esaminato diverse soluzioni e optato per l'utilizzo di una libreria chiamata ***MeCab***, che è appositamente progettata per la tokenizzazione degli ideogrammi giapponesi. Tuttavia, abbiamo scoperto che MeCab non è supportata nei sistemi operativi Windows, il sistema su cui abbiamo operato. Questo ha limitato le nostre opzioni e ci ha portato a cercare alternative o approcci diversi per la tokenizzazione dei testi giapponesi.

La complessità della lingua giapponese, unita alla limitazione delle librerie disponibili per il nostro sistema operativo, ci ha posto di fronte a una sfida significativa nel processo di analisi dei testi giapponesi. Ciò ha reso necessario adottare un approccio diverso o cercare ulteriori risorse e strumenti per affrontare in modo efficace la tokenizzazione e l'analisi dei testi giapponesi.

Alla fine, siamo riusciti a individuare una soluzione utilizzando la libreria ***fugashi***. Attraverso la sua implementazione, abbiamo potuto eseguire il processo di tokenizzazione dei testi giapponesi. L'output della tokenizzazione è stato successivamente fornito a CountVectorizer per la vettorizzazione, proprio come fatto in precedenza per i testi in lingua inglese.

Sottolineiamo che, anche per quanto riguarda la text classification in lingua giapponese, abbiamo dedicato particolare attenzione alla ricerca dei migliori iperparametri per ciascun modello utilizzato. Questo approccio è stato fondamentale per ottimizzare le prestazioni dei modelli e garantire risultati accurati nella classificazione dei testi giapponesi.

- Support Vector Machine (SVM)
  - C: 0.1;
  - gamma: 'scale';
  - 'kernel': 'linear'.
- Decision Tree
  - criterion: 'gini';
  - max_depth: 30.
- Random Forest
  - criterion: 'gini';
  - max_depth: 20;
  - n_estimators: 200.

I risultati di accuracy ottenuti sono contenuti nella seguente tabella:

	Training Accuracy	**Validation Accuracy**
SVM	0.98755	0.19357
Decision Tree	0.59414	0.13618
Random Forest	0.98974	0.23300

V. Conclusioni

La conclusione sottolinea che, nonostante l'ampia disponibilità di diverse tecniche per effettuare text classification, tutti e tre gli algoritmi considerati hanno fornito risultati soddisfacenti. In particolare, si è osservato che la qualità del modello è paragonabile, almeno per quanto riguarda i testi in inglese, tra quei metodi che si basano sulla costruzione di alberi decisionali per effettuare il processo di classificazione, ossia il *Decision Tree* e il *Random Forest*.

L'uniformità nella qualità del modello tra i due algoritmi basati su alberi decisionali offre un'interessante prospettiva per la scelta dell'approccio di text classification più adatto in base alle specifiche esigenze del problema e delle risorse disponibili.

Prendendo in considerazione entrambi gli algoritmi, abbiamo deciso di preferire il **Decision Tree** per diversi motivi. In primo luogo, il Decision Tree è più semplice e presenta una struttura più chiara rispetto al Random Forest, consentendoci di comprendere meglio il processo di classificazione e di interpretare facilmente le decisioni prese dal modello.

Inoltre, il Decision Tree ha un tempo computazionale minore (1 secondo tra training e validation prediction per il Decision Tree e 17 secondi per il Random Forest) e gode di una maggiore interpretabilità rispetto al Random Forest.

Discorso diverso per quanto riguarda invece la classificazione dei testi in lingua giapponese dove l'algoritmo di **Random Forest** ha riportato una perfomance superiore rispetto agli altri due algoritmi considerati.