

FAKE DISASTER CLASSIFICATION

Jarir Salame Younis

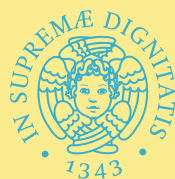
Marta Fioravanti

Corso di Text Analytics

proff. Attardi G., Esuli A.

Università di Pisa

aa. aa. 2019 - 2020



UNIVERSITÀ DI PISA

Il Progetto

Il progetto si propone di sperimentare diverse tecniche di natural language processing per la classificazione di tweet riguardanti disastri reali e falsi. Il dataset è stato preso da una competizione [Kaggle](#). La ricerca è avvenuta seguendo i seguenti passaggi:

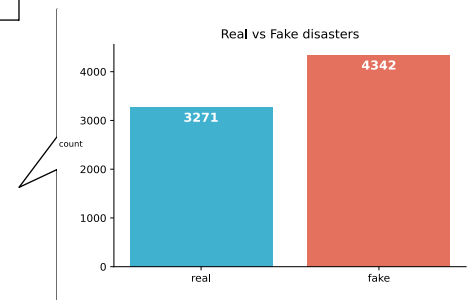
1. data exploration
2. data preprocessing
3. modellazione e validation
4. risultati

Data Exploration

Il dataset è costituito da un training di 7613 record composto dai seguenti attributi:

id	keyword	location	text	target
chiave del record	parola associata al contenuto del tweet	Città, Stato	contenuto del tweet	1: disastro reale, 0: altrimenti

Sono stati considerati non-null i record il cui attributo 'text' non fosse nullo e perciò non è stato necessario eliminare dati. Il dataset non è perfettamente bilanciato, ma non si è ritenuto necessario preprocessarlo per ottenere la stessa quantità di record appartenenti alla classe **target=0** e **target=1**.



Le prime osservazioni sono state sulla natura linguistica del materiale di studio e sono state svolte sia sull'intero insieme di testi, sia dividendo i dati tra **real_disaster** e **fake_disaster**, rimuovendo le stopwords.

#	real_disaster		fake_disaster	
	word	frequency	word	frequency
1	fire	271	like	294
2	kill	156	amp	193
3	news	146	new	172
4	disaster	121	body	118
5	california	115	good	118
6	bomb	112	love	117
7	suicide	112	come	108
8	year	112	day	104
9	crash	112	video	102
10	amp	110	know	102

Parole più comuni per le categorie **fake_disaster** e **real_disaster**

L'SVM i vettori sono stati ottenuti attraverso la media (aritmetica o pesata sugli indici TF-IDF) dei word embedding dei token dei vari tweet. Sono stati utilizzati i Glove embedding per tweet di dimensione 100. Le keyword e gli hashtag sono stati trattati attraverso un one-hot-encoding in quanto meno variegate e più facilmente utilizzabili come dato categorico.

Per quanto riguarda lo split dei dati, il dataset di training della competizione è stato inizialmente diviso in development set (80%) per la validation e il restante per l'internal test set (20%) .

Modellazione e Validation

Per ogni modello è stata svolta una grid-search con 5-fold cross validation sul development set.

Naïve Bayes

È stato utilizzato il Multinomial Naïve Bayes della libreria scikit-learn, considerando come feature il bag of words dei tweet, gli one hot encoding delle keywords e degli hashtag. Nella fase di validation è stato indagato il parametro alpha, il quale definisce la complessità del modello. Sono stati presi in considerazione i valori compresi tra 0.025 e 2.0 (80 valori). Il miglior valore ottenuto è 0.575, con F1 media pari a 0.753 nel validation set.

Naïve Bayes

alpha = 0.575

training

accuracy = 0.8835
f1 = 0.8558

validation

accuracy = 0.8047
f1 = 0.7589

Support Vector Machines

Il classificatore utilizzato è stato il Support Vector Machine Classifier di scikit-learn. Dopo una serie di prove con diversi tipi di input è emerso che la migliore prestazione si otteneva utilizzando solo i testi vettorializzati e con media aritmetica sugli embedding, anziché pesata sul TF-IDF. Il miglior modello è stato ottenuto trovando innanzitutto il kernel (rbf) e il gamma (scale) corretti e in seguito effettuando un'ulteriore cross validation fissando i parametri già menzionati e trovando il valore ideale dell'indice di penalizzazione C (5.6).

Support Vector Machines

kernel = rbf
C = 5.6
gamma = scale

training

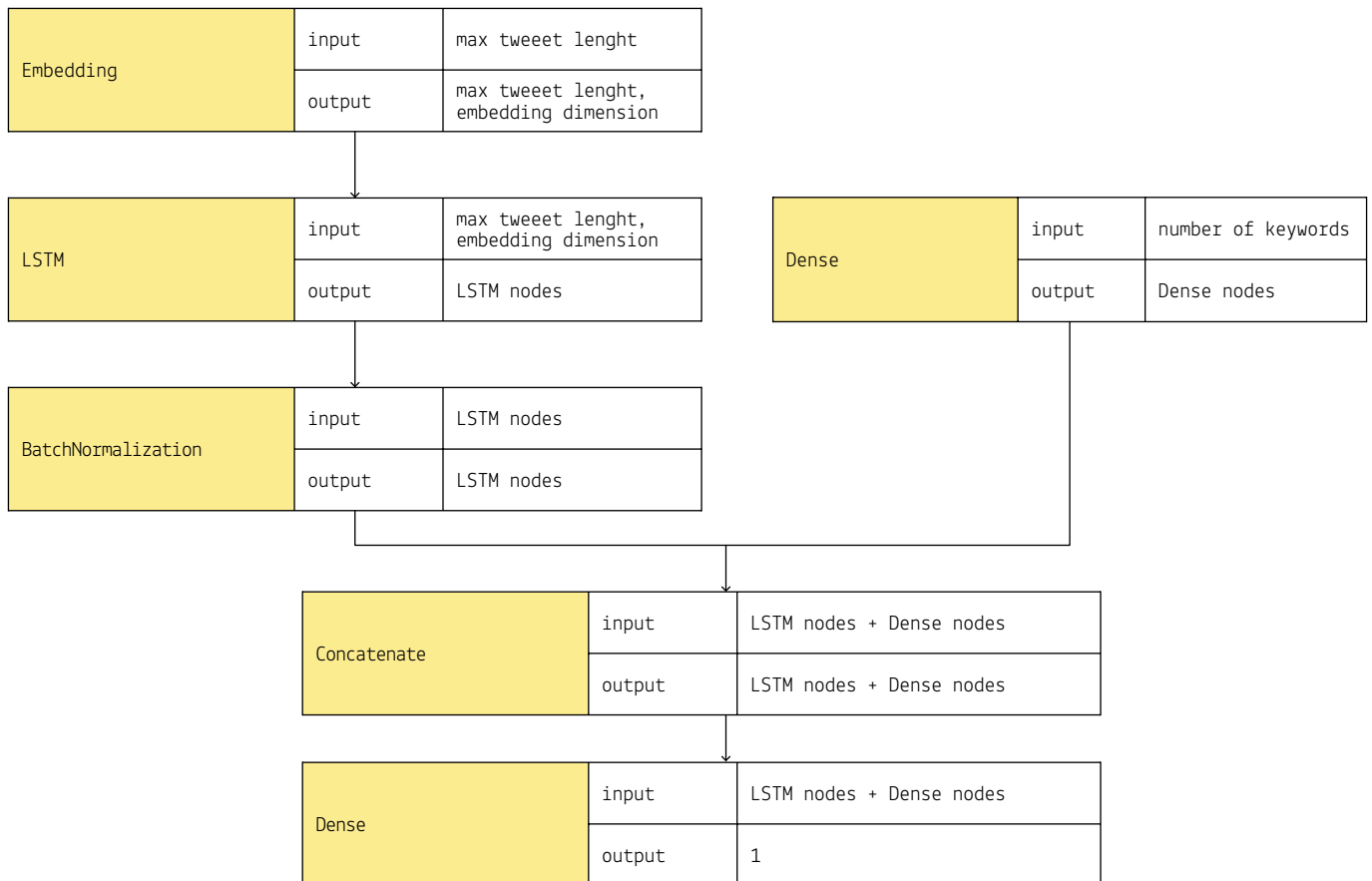
accuracy = 0.8419
F1 = 0.8011

validation

accuracy = 0.8170
F1 = 0.7706

Deep Learning con Glove

La classificazione è in seguito stata testata su una rete neurale tramite Keras che prende in input i tweet preprocessati e, separatamente, le keyword. Nel primo caso le parole passano attraverso un pretrained layer di embedding (Glove) senza fine tuning e un LSTM con



attivazione relu. Le keyword invece passano attraverso un dense layer sempre con attivazione relu. Un layer finale concatena i due output e li passa attraverso una sigmoid. La rete neurale utilizza il dropout per regolarizzare il modello e prevenire l'overfitting; inoltre ha un early stopping con patience pari a 10 per velocizzare la computazione ed estrarre i pesi che portano a un risultato migliore. Inizialmente la rete prendeva in input anche l'attributo location, ma esso si è rivelato troppo sparso per contribuire in modo utile al modello.

Per quanto riguarda la model selection, si sono indagati gli iperparametri di dropout, di learning rate e di numero di hidden layer. Il numero di epoche è stato fissato a 200.

Bert

Si è infine deciso di utilizzare Albert, la versione lite di Bert. Questo modello ha un'architettura più semplice e meno parametri, pur avendo la stessa prestazione di Bert, nel task di classificazione. È stato utilizzato albert-base-v1 fornito da Transformers di Huggingface tramite la libreria ktrain che permette di usare questi modelli in keras. In questo caso il testo dato in input non è stato preprocessato, salvo per quanto riguarda la pulizia iniziale, in quanto il modello contiene quanto necessario per

LSTM Neural Network

learning rate = 10^{-3}
 dropout = 0.1
 LSTM hidden layer = 40
 DENSE keyword hidden layer = 5

training

accuracy = 0.8609
 F1 = 0.8258

validation

accuracy = 0.8272
 F1 = 0.7829

Bert

learning rate = 10^{-5}
 learning rate decay = 0.8

training

accuracy = 0.8544
 F1 = 0.8206

validation

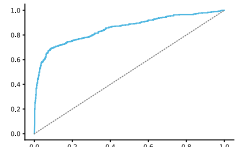
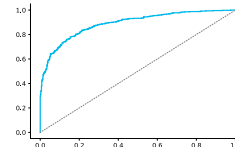
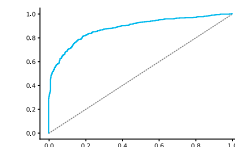
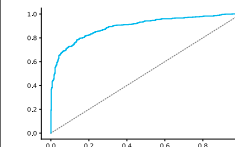
accuracy = 0.8333
 F1 = 0.7947

trattare i dati grezzi. Per la fase di validation sono stati indagati gli iperparametri learning rate e learning rate decay (che indica la frazione con cui il learning rate diminuisce a ogni iterazione). Per trovare il range di valori ideale per il primo iperparametro si è utilizzato il metodo lr_find fornito da ktrain.

I migliori risultati sono stati ottenuti da un learner con learning rate pari a 10^{-5} e learning rate decay da 0.8.

Risultati

Una volta individuata la migliore combinazione di iperparametri i modelli sono stati riallenati sull'intero development set e testati sull'internal test set; per i modelli di rete neurale si estrae un 10% del development set e si usa come validation set per early stopping. Si riporta di seguito un confronto delle prestazioni dei 4

	Naïve Bayes	SVM	LSTM	Albert
Accuracy	0.8047	0.8170	0.8272	0.8333
F1	0.7589	0.7706	0.7829	0.7947
AUC	0.8517	0.8902	0.8801	0.8944
ROC				

modelli.

Il modello con i migliori risultati è Albert, anche se con poco scarto rispetto al LSTM. Lo stesso procedimento è

	Naïve Bayes	SVM	LSTM	Albert
F1	0.80573	0.80163	0.81288	0.81083

stato fatto utilizzando poi l'intero training set di Kaggle per ottenere le predizioni per il blind test set.

Dai primi risultati su Kaggle sembra che LSTM abbia ottenuto migliori performance in termini di F1. Questo score tuttavia è calcolato solo sul 30% del blind test set. Per sapere il risultato finale bisognerà attendere il termine della competizione. Come ci si aspettava i modelli di deep learning ottengono migliori performance rispetto a Naive Bayes ed SVM.

