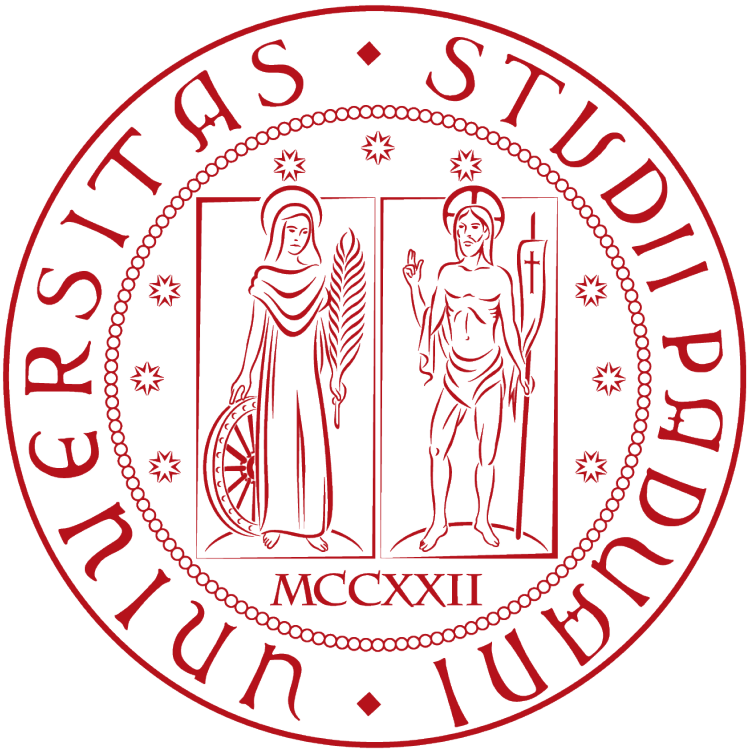

Relazione progetto di Intelligenza Artificiale

Informazioni sul progetto

Redatto	Francesco Corti - 1142525 Giovanni Sorice - 1144558
Referente	Giovanni Sorice - 1144558 giovanni.sorice@studenti.unipd.it

Link al sito

https://github.com/FraCorti/AI_Project

Descrizione

Documento riportante le informazioni relative al progetto di intelligenza artificiale.

Indice

1 Introduzione

1.1 Scopo dell'analisi

Questo progetto consiste nell'applicazione e confronto di due metodi di apprendimento supervisionato, per sottolinearne le potenzialità e i problemi.

1.2 Il problema dello spam

Il problema dello spam affligge da molti anni tutte le persone che dispongano di una casella di posta elettronica oppure di uno smartphone.
In passato è stato constato come l'eliminazione manuale dei messaggi spam presentasse costi di tempo insostenibili.
Questo ha portato ad uno sviluppo di tecniche algoritmiche che permettessero di classificare automaticamente un messaggio ricevuto, come spam o ham.
Si è però scoperto che un approccio di tipo *offline learning*, non era efficiente.
Gli spammer riuscivano a modificare i messaggi in modo da renderli classificati come ham dai sistemi anti-spam presenti.
Questo era possibile in quanto gli algoritmi non evolvevano nel tempo, cambiando la struttura del messaggio di spam questo veniva erroneamente identificato come un messaggio non spam.
Si è quindi passati a un approccio di tipo online che si è visto essere quello più ottimale.
Gli algoritmi in questo modo non smettono di imparare una volta terminato l'input dei dati ma evolvono nel corso del tempo imparando a classificare nuove tipologie di messaggi come spam.
Abbiamo scelto questo problema dato che si adatta molto bene all'applicazione di algoritmi supervisionati.

2 Panoramica

Gli algoritmi presi in considerazione all'interno del nostro progetto sono stati:

- *Logistic Regression*;
- *Neural Network* (con e senza Dropout).

Lo sviluppo di tali algoritmi è stato svolto all'interno di Google Colab.

Gli obiettivi principali che ci siamo prefissati trattano come la fase di progettazione e sviluppo doveva avvenire (cioè in modo corretto, efficace ed efficiente) e dei confronti in termini di velocità di training, precision e recall.
Abbiamo deciso di dividere il progetto in 7 fasi:

- Scelta e studio dell'ambito;
- Scelta e studio del problema;
- Scelta e studio degli algoritmi;
- Progettazione;
- Implementazione;
- Tracciamento dei risultati;
- Confronto tra gli algoritmi.

3 Metodologie utilizzate

3.1 Studio del problema

Lo **spam** è l'invio anche verso indirizzi generici, non verificati o sconosciuti, di messaggi ripetuti ad alta frequenza o a carattere di monotematicità tale da renderli indesiderati. Il problema dello spam nasce quando, con l'avvento di tecnologie informatiche, questa tipologia di messaggi ha iniziato ad invadere le caselle di posta elettronica e gli smartphone di aziende e semplici utilizzatori. Ciò ha portato molte persone ad interessarsi al tema dello spam e di come riuscire a contrastarlo in modo efficace.
In prima battuta, ci sono state molte discussioni riguardanti la tipologia del problema. Le principali controversie furono tra chi sosteneva fosse un problema di *classificazione* e tra chi, invece, sosteneva fosse un problema di *modellazione*. Alcune correnti di pensiero continuano a sostenere che la metodologia migliore da applicare in questo caso sia la modellazione, ma la maggior parte della comunità degli studiosi afferma con convinzione che la tecnica migliore sia quella della classificazione. Noi concordiamo con quest'ultimi, vero che il bisogno iniziale di dati già classificati è uno svantaggio, ma nell'era dei big data, è ormai semplice trovare dei set di dati consistenti e pronti all'uso. Inoltre, c'è da sottolineare la maggiore flessibilità della tecnica che è portata maggiormente ad imparare dai propri errori.
Detto ciò, si può capire perché tra i tanti tipi di approcci al problema abbiamo deciso di utilizzare la *Logistic Regression* e le *Neural Networks*, vediamoli ora in modo più approfondito.

3.2 Neural networks

Le *Neural networks*, o reti neurali, sono ispirate alle reti di neuroni biologici. Sono sistemi che imparano dagli esempi e che inizialmente sono configurati in modo randomizzato.
Il concetto alla base è il neurone artificiale, che poi viene rappresentato in collezioni connesse con altri neuroni, come nelle reti biologiche.

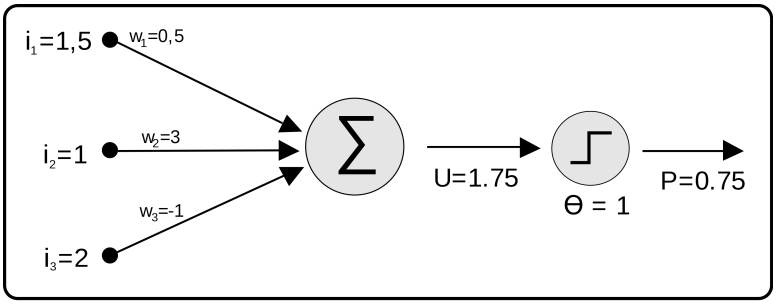

Figura 1: Esempio di neurone artificiale

Queste collezioni vengono comunemente chiamate *layer* e le connessioni tra ogni nodo vengono chiamate *edge*.
Ad ogni neurone e ad ogni *edge* viene assegnato un peso, il quale, ha la funzione di "aggiustamento" del tasso di apprendimento influenzando la il valore dei nodi o delle connessioni.

3.3 Logistic Regression

La *Logistic Regression* è un modello statistico che si basa sull'utilizzo della Logistic Function per modellare una variabile binaria dipendente.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{1}$$

Il fulcro della *Logistic Regression* sta nell'assegnare un valore che va da 0 a 1 della probabilità che un dato elemento sia o meno del tipo A, la somma delle probabilità di appartenere o no alla categoria A deve essere 1. Sottolineiamo quindi che, come dal nome, è una forma di regressione matematica.
Interessante la possibilità di avere un grado maggiore o minore di tolleranza, e quindi una sorta di scelta, spostando in alto o in basso la soglia minima per essere considerati di un certo tipo. Questo ci consente maggiore flessibilità in caso di incertezza del nostro algoritmo o maggiore rigidezza in caso di estrema precisione.

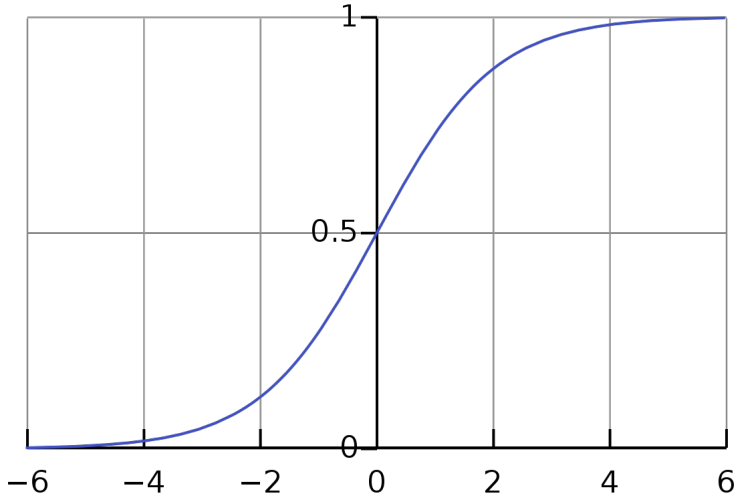

Figura 2: La logistic function standard

4 Preparazione del dataset

Per preparare il dataset abbiamo utilizzato il *Text Preprocessing* presente nelle API di Keras. Questo permette di vettorizzare il corpus del testo, convertendo

ogni parola presente in una sequenza di interi.
In particolare abbiamo scelto di assegnare al parametro *num_words* il valore di 1000. Questo stabilisce il numero massimo di parole da mantenere, in base alla loro frequenza.
Vengono quindi mantenute solo le parole *num_words-1* più comuni.

```
1 tokenizer = Tokenizer(num_words = num_max)
```

Dopo aver definito il nostro Tokenizer abbiamo utilizzato i metodi:

1. *fit_on_texts* per aggiornare il vocabolario interno basato sulla frequenza delle parole e crearne l'indice;
2. *texts_to_matrix* per convertire il testo in un numpy array di forma: `(len(texts), num_words)`.

Maggiori info sui metodi di Keras preprocessing sono presenti al seguente link: Tokenizer Preprocessing.

5 Realizzazione

È stato scelto di suddividere il dataset in 80% e 20%. Dove 80% dei dati è stato destinato al training set e il rimanente 20% è stato utilizzato per il test set. Il dataset che è stato utilizzato è quello fornito dalla piattaforma Kaggle, è disponibile al seguente link: *Spam Collection Dataset*.
Abbiamo utilizzato il metodo *train_test_split* presente nella libreria scikit-learn per suddividere i dati in training set e test set in modo causale.

5.1 Reti neurali

Il primo approccio che abbiamo scelto di utilizzare è stato quello delle reti neurali. Per farlo ci siamo appoggiati alla libreria *TensorFlow* la quale, dalla versione 2.0.0, contiene *Keras* al suo interno.
Maggiori informazioni riguardanti l'integrazione di Keras sono presenti al seguente link *tf.keras*.
Questo ci ha permesso di:

1. Utilizzare TensorFlow(*tf*) come ecosistema;
2. Definire la rete tramite il modulo *tf.keras* presente in *TensorFlow*;

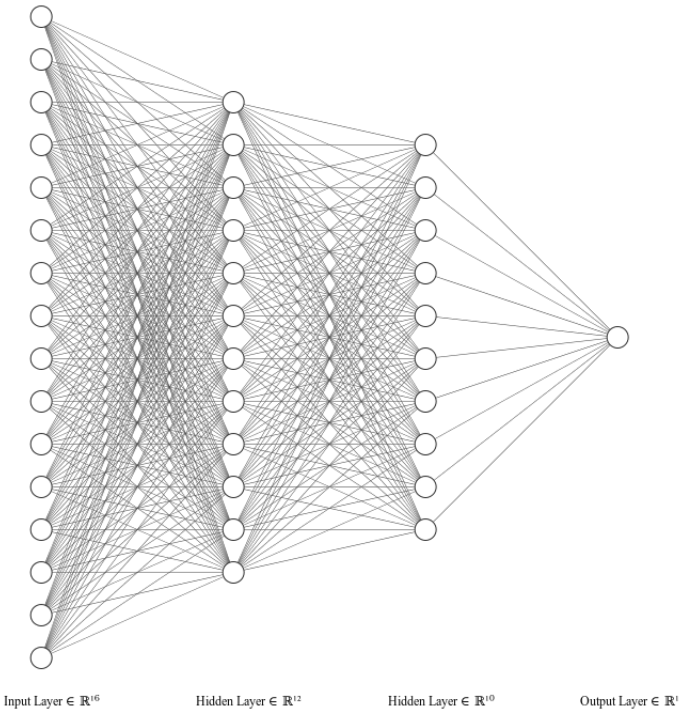

Figura 3: Esempio di rete neurale *sequential*

5.1.1 Struttura

È stata scelto di utilizzare il tipo di rete *Sequential*.
Abbiamo creato due reti, con tre layer di tipo *Dense*. Le funzione di attivazione scelte sono le seguenti:

1. *Relu* per i nodi interni;
2. *Sigmoid* per l'output della rete.

La prima rete presenta la seguente struttura:

```
model = Sequential()
model.add(Dense(512, activation = 'relu')
model.add(Dense(256, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
```

La seconda rete presenta la seguente struttura:

```
model = Sequential()
model.add(Dense(512, activation = 'relu')
model.add(Dropout(0.2))
```

```
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid'))
```

5.1.2 Configurazione

Tramite il metodo *compile()* presente in Keras è possibile stabilire:

1. **Loss functions**: *binary crossentropy* nel nostro caso, maggiori info al seguente link *loss functions*;
2. **Optimizer**: l'ottimizzatore che verrà utilizzato per l'aggiustamento dei pesi e per minimizzare la loss function. Nel nostro caso *Adam*.
3. **Metrics**: lista di metriche che verranno valutate dal modello durante la fase di training e testing. Nel nostro caso è stata scelta la metrica di *binary accuracy*.

5.2 Logistic Regression

Il secondo approccio che abbiamo scelto di utilizzare è stato quello della logistic regression. Per farlo ci siamo appoggiati alla libreria *Sklearn*.

5.2.1 Configurazione

Tramite il metodo *fit()* presente in Sklearn stabiliamo:

1. **precision**: sono tutte quei messaggi classificati nel modo corretto, quindi attribuendogli la giusta etichetta. Maggiori informazioni al seguente link *precision*;
2. **recall**: sono tutti quei messaggi classificati come non spam e che effetivamente fanno parte dei messaggi non spam. Nel nostro caso, maggiori informazioni al seguente link *recall*;
3. **f1**: viene calcolato tramite la media armonica di precisione e recupero. Nel nostro caso, maggiori informazioni al seguente link *f1*.

6 Risultati

Dopo aver visto come abbiamo deciso di implementare gli algoritmi e le metodologie presi in considerazione, passiamo ora a vedere i risultati ottenuti.

6.1 Reti neurali

Per quanto riguarda le reti neurali, i risultati ottenuti sono i seguenti.

6.1.1 Reti neurali con Dropout

Epoch	Loss	Acc	Binary accuracy	Val loss	Val acc	Val binary accuracy
1	0.1800	0.9419	0.9419	0.0555	0.9854	0.9854
2	0.0326	0.9905	0.9905	0.0796	0.9843	0.9843
3	0.0088	0.9978	0.9978	0.0563	0.9865	0.9865
4	0.0039	0.9992	0.9992	0.0729	0.9865	0.9865
5	0.0024	0.9997	0.9997	0.0863	0.9854	0.9854
6	0.0021	0.9997	0.9997	0.0887	0.9865	0.9865
7	0.0018	0.9997	0.9997	0.0945	0.9865	0.9865
8	0.0017	0.9997	0.9997	0.0993	0.9877	0.9877
9	0.0016	0.9997	0.9997	0.1026	0.9877	0.9877
10	0.0017	0.9997	0.9997	0.0965	0.9865	0.9865

Tabella 1: Risultati test rete neurale con dropout

loss	acc	binary accuracy
0.1885	0.9857	0.9857

Tabella 2: Risultati validazione rete neurale con dropout

Dai risultati notiamo come già alla prima iterazione, la rete neurale con dropout riesca ad avere ottimi risultati fino ad arrivare alla decima iterazione del training in cui l'accuratezza e l'errore risultato infinitesimali.
Questo aspetto è osservabile anche nel set di validazione nel quale il risultato è eccellente. Si denota quindi una buona divisione del dataset nelle porzioni di training, validazione e verifica.

6.1.2 senza Dropout

Epoch	Loss	Acc	Binary accuracy	Val loss	Val acc	Val binary accuracy
1	0.4319	0.8628	0.8628	0.4045	0.8778	0.8778
2	0.3404	0.8651	0.8651	0.4793	0.8711	0.8711
3	0.2472	0.8948	0.8948	0.6005	0.8352	0.8352
4	0.1529	0.9453	0.9453	0.7931	0.8352	0.8352
5	0.0956	0.9652	0.9652	0.9669	0.8206	0.8206
6	0.0689	0.9722	0.9722	1.0665	0.8105	0.8105
7	0.0523	0.9781	0.9781	1.1933	0.8161	0.8161
8	0.0417	0.9784	0.9784	1.2961	0.8217	0.8217
9	0.0380	0.9792	0.9792	1.4842	0.8419	0.8419
1	0.0369	0.9804	0.9804	1.4484	0.8217	0.8217

Tabella 3: Risultati training rete neurale senza dropout

loss	acc	binary accuracy
1.6114	0.8063	0.8063

Tabella 4: Risultati test rete neurale senza dropout

Dai risultati notiamo come il miglioramento sia costante, la rete neurale senza dropout riesce ad avere discreti risultati fino ad arrivare alla decima iterazione del training dove l'accuratezza e l'errore risultato buoni anche se non ottimi. Questo aspetto si nota anche nel set di validazione nella quale il risultato è buona, si sottolinea quindi la presenza di una sproporzione nel dataset che deve essere colmata con metodi correttivi.

6.1.3 Cos'è il Dropout?

Il *Dropout*, è una tecnica che può prevenire il fenomeno dell' overfitting che spesso si verifica allenando le reti neurali.
Il termine dropout fa riferimento proprio al fatto di "far cadere" e quindi escludere, per un breve periodo, una porzione nodi dalla rete neurale nelle fase di training. Questo permette alle unità di non co-adattarsi troppo.
Le unità che vengono temporaneamente nascoste sono scelte in modo casuale. Per maggiori informazioni riguardanti la tecnica del dropout si veda Dropout: A Simple Way to Prevent Neural Networks fromOverfitting

6.2 Logistic regression

	Precision	**Recall**	**f1**	**Accuracy**
Spam	0.989	0.876	0.929	0.979
Ham	0.978	0.998	0.988	0.979

Tabella 5: Risultati fase di test logistic regression

Dai risultati notiamo come l'accuratezza e la precisione arrivino ad un ottimo livello sia per i messaggi di tipo "Spam" che di tipo "Ham", cosa che non si verifica nella metrica di recall per lo "Spam". Questo è dovuto al dataset non bilanciato nella quantità di record delle due categorie.
Si può affermare comunque che il risultato ottenuto dalla logistic regression, sia più che positivo.
Ci teniamo a sottolineare che, volutamente, non sono state prese precauzioni riguardo al dataset e ai suoi problemi di sbilanciamento.

7 Comparazioni

Questa sezione sarà incentrata sul paragonare i risultati ottenuti delle due metodologie applicate, cercando anche di intuire il motivo e i fattori di squilibrio che hanno influito su di esse.

7.1 Tra reti neurali

I risultati in questo caso parlano chiaro, la rete neurale senza dropout ha prestazioni decisamente inferiori nelle prime iterazioni, anche se conclude con un risultato decisamente positivo. D'altro canto, come è stato sottolineato nelle sezioni precedenti, il dropout serve proprio ad evitare *l'overfitting*, cosa che, dato il dataset preso in considerazione, ha aiutato molto. Vediamo infatti come già alla quinta iterazione la rete neurale arrivi ad avere un errore che potremmo dire essere "fisiologico", questo conferma ancora una volta l'importanza di studiare il contesto e prendere gli opportuni accorgimenti.

7.2 Reti neurali e Logistic regression

Per quanto riguarda il confronto tra reti neurali e logistic regression, vi è una netta differenza nel caso dell'utilizzo o meno del dropout. Infatti, dai risultati emerge che la logistic regression è migliore quando non viene utilizzato il metodo correttivo, mentre peggiore altrimenti. Si nota soprattutto nella

metrica di accuracy, nella quale troviamo la logistic regression leggermente al di sotto delle reti neurali con dropout, ma di gran lunga superiore alla rete neurale che non utilizza il dropout.

7.3 Reti neurali perchè migliori?

Abbiamo capito quindi che la discriminante in questo caso è l'utilizzo di azioni preventive e correttive sul dataset e sul rischio di overfitting, detto questo sottolineiamo che il dataset è stato appositamente scelto per far risaltare questa differenza nel confronto finale.

8 Conclusioni

In questo lavoro è stato possibile entrare in contatto con l'utilizzo di tecnologie e framework emergenti negli ultimi anni. Grazie ad essi infatti, il processo di sviluppo di algoritmi di intelligenza artificiale è stato notevolmente semplificato, velocizzato e reso maggiormente "user friendly".
In conclusione, riteniamo che l'astrazione offerta da questi servizi, renda allo stesso tempo necessario approfondire temi maggiormente teorico per capire il loro reale funzionamento e il loro potenziale tecnologico.

Riferimenti

https://it.wikipedia.org/wiki/Spam

https://www.matchilling.com/comparison-of-machine-learning-methods-in-email-spam-detection/

https://en.wikipedia.org/wiki/Artificial_neural_network

https://en.wikipedia.org/wiki/Logistic_regression

https://keras.io/

https://keras.io/preprocessing/text/#Tokenizer

http://faroit.com/keras-docs/2.0.2/preprocessing/text/

https://www.tensorflow.org/

https://en.wikipedia.org/wiki/Cross_entropy

https://arxiv.org/pdf/1412.6980v8.pdf

http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf