UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

RILEVAMENTO DI FAKE NEWS SUL COVID-19 MEDIANTE TECNICHE DI MACHINE LEARNING E DEEP LEARNING

RELATORI

Prof. **Rita Francese**

Prof. **Maria Frasca**

Università degli studi di Salerno

CANDIDATO

Alessandro Aquino

Matricola: 0512106527

Anno Accademico 2021-2022

I problemi iniziano quando non basta spegnere e riaccendere

Sommario

Durante l'emergenza sanitaria è stata fortemente sentita la necessità di un'informazione seria e corretta, sia per capire come comportarsi per tutelare la propria salute e quella altrui, sia per cogliere il senso delle misure eccezionalmente restrittive adottate dal Governo. In particolare, l'emersione del virus ha portato alla ricerca spasmodica di notizie in rete che ha avuto spesso come esito l'imbattersi in informazioni contraddittorie. L'obiettivo che si vuole raggiungere con questa tesi è quello di riuscire ad effettuare una predizione accurata delle notizie su un determinato dataset. Dapprima la scelta di un dataset che contenga il testo di articoli e relative fonti.Per la scelta ho selezionato dei parametri per comprendere quale poteva risultarmi utile per questo progetto, ossia: usabilità(facilità di utilizzo dati e pulizia del dataset),quantità(numero di articoli), tipo(con o senza etichetta, cioè se viene esplicitato che la notizia sia vera o falsa), utilità(pertinenza con idea iniziale e parametri di utilizzo oggetti). Tali parametri sono stati utili a guidarmi nella scelta di unire due dataset che presi singolarmente rispecchiavano solo alcune delle caratteristiche ma accorpati le soddisfano. Una volta scelto il dataset si procede con la fase di data cleaning, qundi: rimozione di tuple vuote, caratteri speciali, lemmatizzazione, rendere al singolare le parole e le stopWords. Al seguito di questa fase si utilizza il tf-idf per capire la rilevanza delle parole. Poi si può procedere ad addestrare gli algoritmi sia di machine learning che di deep learning. Nello specifico gli algoritmi di machine learning adottati sono: Gaussian, Alberi decisionali e Random forest, SVM e Passive-Aggressive; mentre quelli di deep learning sono addestrati sono: MLP, LSTM, LSTM autoencoder e GRU. Una volta addestrati gli algoritmi elencati in precedenza attraverso delle metriche come la matrice di confusione, accuratezza, precisione, recall ho potuto analizzarne i risultati prodotti dall'addestramento.Nello specifico gli algoritmi di machine learning: Gaussian, Random Forest, SVM e Passive-Aggressive hanno ottenuto una precisione $\geq$ 90%; SVM e Passive-Aggressive hanno attenuto un'accuratezza $\geq$ 90%; SVM ha attenuto un recall $\geq$ 90%. Considerando anche le martici di confusione nella sezione 4.3.1, rispecchia quanto detto con le metriche precenti. Il miglior algoritmo di machine learning sul mio dataset è SVM mentre il peggiore è il Decision Tree. Invece nello specifico gli algoritmi di deep learning: Mlp, Gru, Lstm e Lstm autoencoder hanno un'accuratezza, precisione e recall $\geq$ 90%.Ciò nonostante GRU ha riportato valori che superano in tutte e tre le metriche il 95%. Per tanto è il migliore dei quattro ed in generale è il miglior algoritmo sul dataset.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

La disinformazione su COVID-19 è proliferata ampiamente sui social media,spaziando da false "cure", come fare gargarismi con limone o acqua salata e iniettarsi candeggina, alle false teorie del complotto secondo cui il virus è stato bioingegnerizzato in un laboratorio a Wuhan o che la rete cellulare 5G sta causando o esacerbando i sintomi di COVID-19. Sta diventando sempre più chiaro che la disinformazione su COVID-19 è un problema comune. Ad esempio, un sondaggio di Ofcom nel Regno Unito ha rilevato che quasi la metà della popolazione del Regno Unito ha riferito di essere stata esposta a notizie false sul coronavirus. Risultati simili sono stati riportati da Pew negli Stati Uniti. In particolare, tra le persone esposte, quasi due terzi hanno riferito di averlo visto quotidianamente, il che è problematico poiché è noto che l'esposizione ripetuta aumenta la fiducia nelle notizie false. Sebbene l'approvazione di massa delle teorie del complotto sul virus non sia ancora diffusa, sostanziali minoranze nel Regno Unito e negli Stati Uniti riferiscono di ritenere che il virus sia creato dall'uomo o prodotto apposta da potenti organizzazioni. In effetti, un sondaggio YouGov ha rilevato che circa il 28% degli americani e il 50% dei telespettatori di Fox News pensa che Bill Gates stia pianificando di utilizzare il vaccino COVID-19 per implementare microchip nelle persone. Inoltre, una recente analisi dei video YouTube sul coronavirus più visti ha rilevato che oltre il 25% dei migliori video sul virus conteneva informazioni fuorvianti, raggiungendo oltre 62 milioni di visualizzazioni in tutto il mondo.[13] Negli ultimi anni sono stati sviluppati software

basati sul machine learning e deep learning che vanno a fronteggiare tale problematica. Di conseguenza, lo scopo della tesi è effettuare, innanzitutto, un'analisi dello stato dell'arte, la scelta di un dataset che contenga il testo di articoli e relative fonti..Per la scelta ho selezionato dei parametri per comprendere quale poteva risultarmi utile per questo progetto, ossia: usabilità(facilità di utilizzo dati e pulizia del dataset),quantità(numero di articoli), tipo(con o senza etichetta, cioè se viene esplicitato che la notizia sia vera o falsa), utilità(pertinenza con idea iniziale e parametri di utilizzo oggetti). Tali parametri sono stati utili a guidarmi nella scelta di unire due dataset che presi singolarmente rispecchiavano solo alcune delle caratteristiche ma accorpati le soddisfano. Una volta scelto il dataset si procede con la fase di data cleaning, qundi: rimozione di tuple vuote, caratteri speciali, lemmatizzazione(porre all'infinito i verbi), rendere al singolare le parole e le stopWords(parole brevi che si ripetono spesso come: articoli, congiunzioni ...). Al seguito di questa fase si utilizza il tf-idf per capire la rilevanza delle parole. Poi si può procedere ad addestrare gli algoritmi sia di machine learning che di deep learning. Gli algoritmi di machine learning sono: Gaussian, Alberi decisionali e Random forest, Support-vector machine e Passive-Aggressive; mentre quelli di deep learning sono addestrati sono: Multi-layer Perceptron, LSTM, LSTM autoencoder e GRU. Al seguito dell'addestramento, attraverso delle metriche come: la matrice di confusione, accuratezza, precisione, recall ho potuto stabilire,confrontando i vari risultati, che l'algoritmo GRU ha avuto risultati migliori rispetto agli altri, sul mio dataset.

1.2 Risultati ottenuti

I risultati ottenuti dalle due tecniche mostrano come gli algoritmi di machine learning e di deep learning riportano risultati buoni. Inoltre le tecniche di data cleaning utilizzate sono risultate ottimali e gli algoritmi utilizzati hanno un'accuratezza, precisione e recall, in media, maggiore del 90%.
Nello specifico gli algoritmi di machine learning:

- Gaussian, Random Forest, SVM e Passive-Aggressive hanno ottenuto una precisione $\geq$ 90%;
- SVM e Passive-Aggressive hanno attenuto un'accuratezza $\geq$ 90%;
- SVM ha attenuto un recall $\geq$ 90%;

Considerando anche le martici di confusione nella sezione 4.3.1, rispecchia quanto detto con le metriche precenti. Il miglior algoritmo di machine learning sul mio dataset è SVM mentre

il peggiore è il Decision Tree. Invece nello specifico gli algoritmi di deep learning: Mlp, Gru, Lstm e Lstm autoencoder hanno un'accuratezza, precisione e recall $\geq$ 90%.Ciò nonostante GRU ha riportato valori che superano in tutte e tre le metriche il 95%. Per tanto è il migliore dei quattro ed in generale è il miglior algoritmo sul dataset.

1.3 Struttura della tesi

All'interno di questa sezione verrà esposta la struttura della tesi, la quale è composta da cinque capitoli con le rispettive sotto sezioni.

Il primo capitolo fornisce un introduzione al lavoro svolto, descrivendo il contesto applicativo, e quindi alcuni dati per capire la gravità delle fake news relative al covid-19. Infine una breve panoramica dei risultati ottenuti.

Il secondo capitolo oltre a descrivere concetti fondamentali per capire il lavoro svolto, presenta un analisi di alcune tecniche oggi utilizzate per il rilevamento di fake news. Sottolineando i problemi e i vantaggi di tali tecniche.

All'interno del terzo capitolo si discute dell'implementazione di algoritmi di machine learning e deep learning. In particolare, si discute dell'implementazione realizzata su un dataset diverso,da quelli dello stato dell'arte per effettuare poi confronto tra i risultati ottenuti dalle due implementazioni dimachine learning e deep learning.

Il quarto capitolo analizza i risultati ottenuti dagli algoritmi tenendo in considerazione diversi dati come: l'accuratezza, precisione,recall e anche la complessità degli algoritmi.

Nell'ultimo capitolo si riassumono i risultati più importanti e gli sviluppi futuri.

CAPITOLO 2

Background e Stato dell'arte

Per capire a fondo le tecniche e i problemi descritti nei capitoli successivi è indispensabile fornire al lettore alcune definizioni, e descrizioni di alcuni algoritmi. Di conseguenza nel capitolo 2.1 verranno presentati gli elementi necessari per comprendere il lavoro svolto. Successivamente, nel capitolo 2.2, verranno analizzate delle tecniche per il rilevamento di fake news.

2.1 Background

2.1.1 Significato di machine learning e deep learning

Machine learning significa che le prestazioni di un programma per computer migliorano con l'esperienza rispetto ad alcune classi di attività e misurazioni delle prestazioni. In quanto tale, mira ad automatizzare il compito di costruzione di modelli analitici per eseguire compiti cognitivi come il rilevamento di oggetti o la traduzione del linguaggio naturale. Ciò si ottiene applicando algoritmi che apprendono in modo iterativo dai dati di addestramento specifici del problema, che consentono ai computer di trovare intuizioni nascoste e schemi complessi senza essere programmati in modo esplicito. Soprattutto nelle attività relative a dati ad alta dimensione come classificazione, regressione e clustering, ML mostra una buona applicabilità. Imparando dai calcoli precedenti ed estraendo regolarità da enormi database, può aiutare a produrre decisioni affidabili e ripetibili. Per questo motivo, gli algoritmi ML sono stati applicati con successo in molte aree, come il rilevamento delle frodi, il punteggio di credito,

l'analisi dell'offerta migliore, il riconoscimento vocale e delle immagini o l'elaborazione del linguaggio naturale.[5]

Il deep learning è un tipo di machine learning e intelligenza artificiale (AI) che imita il modo in cui gli esseri umani acquisiscono determinati tipi di conoscenza. Il deep learning è un elemento importante della scienza dei dati, che include statistiche e modelli predittivi . È estremamente vantaggioso per i data scientist che hanno il compito di raccogliere, analizzare e interpretare grandi quantità di dati; il deep learning rende questo processo più semplice e veloce. Nella sua forma più semplice, il deep learning può essere considerato un modo per automatizzare l' analisi predittiva . Mentre gli algoritmi di machine learning tradizionali sono lineari, gli algoritmi di deep learning sono impilati in una gerarchia di crescente complessità e astrazione.[10]

2.1.2 Classificazione

La classificazione, come dice la parola stessa, consiste nel predire una variabile categorica con lo scopo di classificare degli elementi analizzando le caratteristiche già note (chiamate anche features). L'algoritmo, anche detto classificatore, viene addestrato sul training set, cioè un insieme di istanze già correttamente classificate ed etichettate. Successivamente viene valutato su un altro insieme di istanze chiamato test set, da questa fase si ottengono diverse metriche che indicano la qualità dell'algoritmo. In generale possiamo dire che un insieme di istanze utilizzate dal modello viene chiamato dataset.
In questo contesto la variabile categorica può assumere due valori: real o fake, e ovviamente indica la veriticità della notizia. Il training set sarà formato da un insieme di righe, dove ogni riga contiene le features di una notizia precedentemente classificata.

2.1.3 TF-IDF

TF-IDF è orientato a capire la rilevanza delle parole. Wikipedia dice infatti che TF-IDF è una statistica numerica che è intesa per riflettere quanto importante una parola è relativamente ad un documento, in una collezione o corpus. TF-IDF considera sia la frequenza di una parola e sia la rarità di quella parola, ovvero quanto spesso è stata incontrata tra tutti i documenti.Per fare un esempio pratico prendiamo in considerazione queste frasi:

- frase 1: good boy
- frase 2: good girl

- frase 3: boy girl good

Si va a performare dunque il conteggio della frequenza di ogni termine tra tutte le frasi/documenti:

- good 3
- boy 2
- girl 2

Adesso la prima cosa da fare per applicare TF-IDF, è applicare il TF ovvero il Term Frequency. La formula per il calcolo del Term Frequency è il:

$$tf_{t,d} = \frac{n_{t,d}}{Numberoftermsinthedocument} \tag{2.1.1}$$

Quindi per l'esempio corrente abbiamo:

	Frase1	Frase2	Frase3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

Tabella 2.1: Calcolo del Term Frequency

Adesso bisogna calcolare la Inverse document frequency, la cui formula è:

$$idf_t = log\frac{numberofdocuments}{numberofdocumentswithterm't'} \tag{2.1.2}$$

IDF misura quanto importante è un termine:

Frase1	Frase2
good	$\log(\frac{3}{3}) = 0$
boy	$\log(\frac{3}{2})$
girl	$\log(\frac{3}{2})$

Tabella 2.2: Calcolo dell'IDF

Adesso per calcolare TF-IDF basta moltiplicare i valori di TF*IDF:

	F1	f2	F3	O/P
	good	boy	girl	
Frase1	$\left(\frac{1}{2} \times \log\left(\frac{3}{3}\right)\right) = 0$	$\left(\frac{1}{2} \times \log\left(\frac{3}{2}\right)\right)$	$0 * \log(\frac{3}{2}) = 0$	
Frase2	0	$\log\left(\frac{3}{2}\right) * 0 = 0$	$\left(\frac{1}{2} \times \log\left(\frac{3}{2}\right)\right)$	
Frase2	$\left(\frac{1}{3} \times \log\left(\frac{3}{3}\right)\right) = 0$	$\left(\frac{1}{3} \times \log\left(\frac{3}{2}\right)\right)$	$\left(\frac{1}{3} \times \log\left(\frac{3}{2}\right)\right)$	

Tabella 2.3: Calcolo del TF*IDF

Se osserviamo la prima frase, l'importanza è stata data alla parola "boy" in quanto ha un valore più alto rispetto alle altre parole nella stessa frase che hanno valore 0. Nella seconda frase, allo stesso modo, l'importanza è stata data alla parola "girl", mentre invece nell'ultima frase "boy" e "girl" hanno la stessa importanza. TF-IDF da un valore maggiore alle parole che sono meno frequenti e questo valore sarà più alto quando i valori IDF e TF sono alti, ovvero quando le parole sono rare in tutti i documenti combinati, ma frequenti in un singolo documento.

2.1.4 Gaussian

È una tecnica di classificazione basata sul teorema di Bayes con un presupposto di indipendenza tra i predittori. In parole povere, un classificatore Naive Bayes presuppone che la presenza di una caratteristica particolare in una classe non sia correlata alla presenza di qualsiasi altra caratteristica.

Ad esempio, un frutto può essere considerato una mela se è rosso, rotondo e di circa 3 pollici di diametro. Anche se queste caratteristiche dipendono l'una dall'altra o dall'esistenza delle altre caratteristiche, tutte queste proprietà contribuiscono indipendentemente alla probabilità che questo frutto sia una mela ed è per questo che è noto come 'Naive'.

Il modello Naive Bayes è facile da costruire e particolarmente utile per set di dati molto grandi. Insieme alla semplicità, Naive Bayes è noto per superare anche i metodi di classificazione altamente sofisticati.

Il teorema di Bayes fornisce un modo per calcolare la probabilità a posteriori P(c | x) da P(c), P(x) e P(x | c). come mostrato:
$P(c \mid x) = \frac{P(x|c)P(c)}{P(x)}$
$P(c \mid \mathrm{X}) = P\left(x_1 \mid c\right) \times P\left(x_2 \mid c\right) \times \cdots \times P\left(x_n \mid c\right) \times P(c)$
dove:

- P (c | x) è la probabilità a posteriori della classe(c, target) dato il predittore (x,attributi).

- P (c) è la probabilità a priori di classe .
- P (x | c) è la probabilità che è la probabilità del predittore data la classe .
- P (x) è la probabilità a priori del predittore .

Di seguito ho un set di dati di allenamento del tempo e la corrispondente variabile target "Play" (suggerendo possibilità di gioco). Ora, dobbiamo classificare se i giocatori giocheranno o meno in base alle condizioni meteorologiche. Seguiamo i passaggi seguenti per eseguirlo.

Step 1: convertire il set di dati in una tabella di frequenza

Step 2: crea una tabella di probabilità trovando le probabilità come Probabilità di coperto = 0,29 e la probabilità di giocare è 0,64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Figura 2.1: Step 2

Step 3: ora, si utilizza l'equazione bayesiana ingenua per calcolare la probabilità a posteriori per ciascuna classe. La classe con la probabilità a posteriori più alta è il risultato della previsione.[11]

2.1.5 Alberi decisionali e Random forest

L'algoritmo di classificazione "Decision tree" mira a creare un albero i cui nodi rappresentano un sotto-insieme di caratteristiche e i cui archi rappresentano delle decisioni.
Nel contesto della classificazione di fake news un albero di esempio potrebbe essere il seguente:

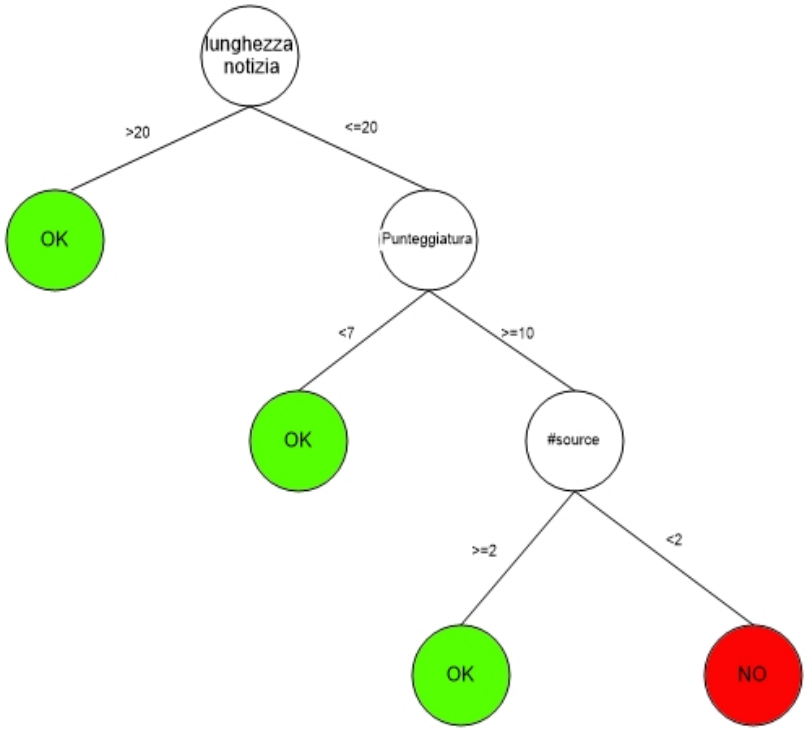

Figura 2.2: Esempio di decision tree

Gli alberi decisionali sono particolarmente utili perché permettono un alto livello di leggibilità, infatti come è facilmente intuibile dalla figura di esempio, una notizia falsa viene classificata falsa solo se la lunghezza della notizia è minore dei venti caratteri, il numero della punteggiatura è maggiore o uguale a sette e il numero di source è minore di due. In tutti gli altri casi viene classificata come vera.

La radice dell'albero sarà la caratteristica che riesce a dividere meglio il dataset, cioè la caratteristica con information gain massima. Il nodo che è figlio della radice avrà un information gain minore rispetto al padre ma maggiore rispetto al nodo figlio. Quindi un nodo al livello n avrà un information gain minore rispetto al nodo $n-1$, ma maggiore rispetto al nodo del livello $n+1$.

L'algoritmo Random forest è un algoritmo di ensemble. In pratica costruisce un certo numero di alberi decisionali e calcola la predizione finale seguendo la tecnica del majority voting. Il che significa che la predizione finale del Random forest è uguale alla predizione fatta dalla maggior parte degli alberi decisionali. Ad esempio se sette alberi su dieci considerano una notizia falsa, allora il Random forest classifica la notizia come falsa.

Una cosa molto importante da notare è che il Random forest crea gli alberi decisionali scegliendo le features randomicamente e non calcolando l'information gain. Se considerasse

l'information gain creerebbe alberi tutti uguali il che non avrebbe senso perché farebbero tutti la stessa predizione.

2.1.6 Support-vector machine

Il Support-vector machine (SVM) è un algoritmo di machine learning capace di apprendere da un insieme di istanze (training set) già classificate ed etichettate.

SVM mappa gli esempi del training set in punti nello spazio con l'obiettivo di massimizzare la distanza tra le due categorie (in questo caso notizie vere e false). Per la separazione delle istanze SVM usa gli iperpiani, intuitivamente una buona separazione è data dall'iperpiano che ha la più grande distanza al punto più vicino di ogni classe. Per chiarire questo concetto si riporta un esempio di uno spazio bidimensionale:

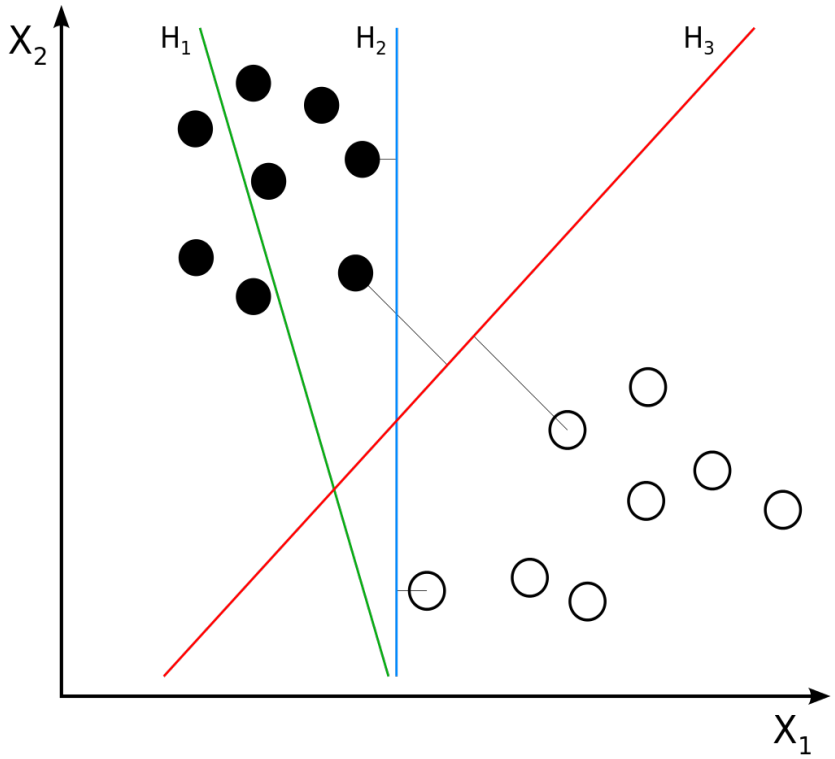

Figura 2.3: Esempio di separazione lineare usando le SVM

L'iperpiano H1 non separa correttamente le istanza, e quindi può essere scartato. L'iperpiano H2 invece separa le istanze ma lo fa con un margine minimo. L'iperpiano H3 è quello ideale siccome separa le istanze e massimizza la distanza tra i due punti più vicini delle due classi. Quando il modello deve classificare una nuova istanza la mappa nello stesso spazio. Se ricade "sotto" l'iperpiano H3 allora la nuova istanza viene classificata come "Bianca", se ricade "sopra" allora viene classificata come "Nera".

Il numero delle dimensioni che formano lo spazio è dato dal numero di features, con tre features si ottiene uno spazio di tre dimensioni e un iperpiano di due dimensioni, quattro features implicano uno spazio di quattro dimensioni e un iperpiano di tre, e cosi via. Le istanze che sono più vicine all'iperpiano vengono chiamate "support vector".

Senza scendere troppo nel dettaglio, il mapping delle istanze nel piano avviene grazie alla funzione di kernel, questo rappresenta un punto di forza dell'SVM perché oltre alle funzioni standard si possono creare funzioni di kernel ad-hoc, e quindi SVM risulta molto versatile. Inoltre lavora bene quando le istanze sul piano sono ben separate e con la minima presenza di accavallamenti o rumore, [4].

2.1.7 Passive-Aggressive

Gli algoritmi passivi-aggressivi sono chiamati così perché[1]:

- Passivo: se la previsione è corretta, conservare il modello e non apportare modifiche. cioè, i dati nell'esempio non sono sufficienti per causare modifiche nel modello.
- Aggressivo: se la previsione non è corretta, apportare modifiche al modello. cioè, alcune modifiche al modello potrebbero correggerlo.

2.1.8 Multi-layer Perceptron(MLP)

Gli algoritmi di deep learning, che ora andrò a mostrare, utilizzano le reti neurali artificiali come struttura principale. Ciò che li distingue dagli algoritmi visti fin'ora è che non richiedono il contributo di esperti durante la fase di progettazione e ingegneria delle funzionalità. Le reti neurali possono apprendere le caratteristiche dei dati. Gli algoritmi di Deep Learning raccolgono il set di dati e ne apprendono i modelli, imparano a rappresentare i dati con le funzionalità che estraggono da soli. Quindi combinano diverse rappresentazioni del set di dati, ciascuna identificando un modello o una caratteristica specifica, in una rappresentazione più astratta e di alto livello del set di dati. Questo approccio pratico, senza molto intervento umano nella progettazione e nell'estrazione delle caratteristiche, consente agli algoritmi di adattarsi molto più velocemente ai dati a portata di mano.

Le reti neurali sono ispirate, ma non necessariamente da un modello esatto, della struttura del cervello. C'è ancora molto che non sappiamo sul cervello e su come funziona, ma ha servito come ispirazione in molte aree scientifiche grazie alla sua capacità di sviluppare l'intelligenza. E sebbene ci siano reti neurali create con l'unico scopo di capire come funzionano i cervelli, il Deep Learning come lo conosciamo oggi non ha lo scopo di replicare come

funziona il cervello; invece si concentra sull'abilitazione di sistemi che apprendono più livelli di composizione del modello. Il primo algoritmo è MLP che ha livelli di input e output e uno o più livelli nascosti con molti neuroni impilati insieme.I neuroni in un MLP possono utilizzare qualsiasi funzione di attivazione arbitraria. Il MLP rientra nella categoria degli algoritmi

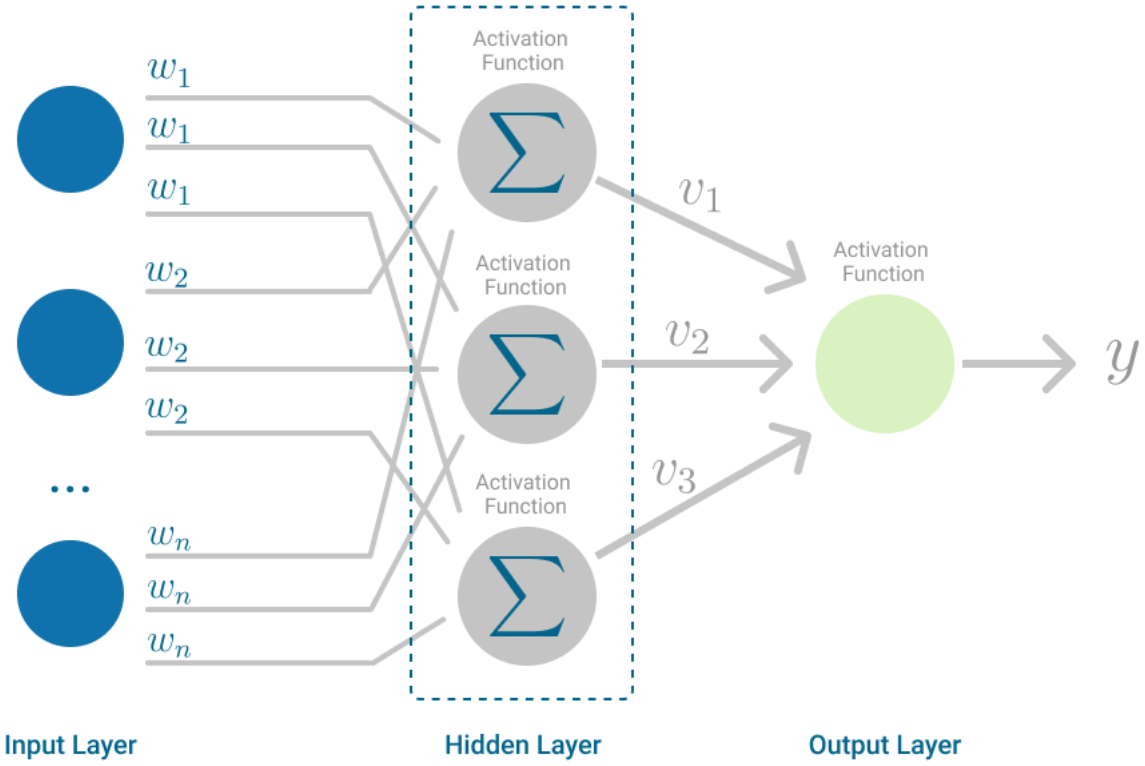

Figura 2.4: Esempio di MLP

feedforward , perché gli input sono combinati con i pesi iniziali in una somma pesata e soggetti alla funzione di attivazione. Ma la differenza è che ogni combinazione lineare viene propagata al livello successivo. Ogni livello sta alimentando il successivo con il risultato del loro calcolo, la loro rappresentazione interna dei dati. Questo passa attraverso i livelli nascosti fino al livello di output. Ma ha di più. Se l'algoritmo calcolasse solo le somme pesate in ciascun neurone, propagasse i risultati al livello di output e si fermasse lì, non sarebbe in grado di apprendere i pesi che riducono al minimo la funzione di costo. Se l'algoritmo calcolasse solo un'iterazione, non ci sarebbe alcun apprendimento effettivo. È qui che entra in gioco la Backpropagation

Backpropagation

La backpropagation è il meccanismo di apprendimento che consente al Multilayer Perceptron di regolare in modo iterativo i pesi nella rete, con l'obiettivo di ridurre al minimo la

funzione di costo. C'è un requisito fondamentale per il corretto funzionamento della backpropagation. La funzione che combina input e pesi in un neurone, ad esempio la somma pesata deve essere differenziabile. Queste funzioni devono avere una derivata limitata , poiché la discesa del gradiente è in genere la funzione di ottimizzazione utilizzata in MultiLayer Perceptron. In ogni iterazione, dopo che le somme pesate sono state inoltrate attraverso tutti i livelli, il gradiente dell'errore quadratico medio viene calcolato su tutte le coppie di input e output. Quindi, per propagarlo indietro, i pesi del primo livello nascosto vengono aggiornati con il valore del gradiente. È così che i pesi vengono propagati al punto di partenza della rete neurale! Questo processo continua fino a quando il gradiente per ciascuna coppia input-output non è convergente, il che significa che il gradiente appena calcolato non è cambiato più di una soglia di convergenza specificata , rispetto all'iterazione precedente[2].

2.1.9 LSTM(Long-Short Term Memory) e LSTM autoencoders

LSTM è un tipo di rete neurale ricorrente ma è migliore delle tradizionali reti neurali ricorrenti in termini di memoria. Avere una buona presa sulla memorizzazione di determinati modelli, gli LSTM funzionano abbastanza meglio. LSTM può avere più livelli nascosti e mentre passa attraverso ogni livello, le informazioni rilevanti vengono conservate e tutte le informazioni irrilevanti vengono scartate in ogni singola cella. LSTM ha 3 cancelli principali.

1. FORGET Gate
2. INPUT Gate
3. OUTPUT Gate

Diamo un'occhiata uno per uno.

Forget Gate

Questa porta è responsabile di decidere quali informazioni vengono conservate per il calcolo dello stato della cella e quali non sono rilevanti e possono essere scartate. h t-1 è l'informazione dallo stato nascosto precedente (cella precedente) e x t è l'informazione dalla cella corrente. Questi sono i 2 ingressi dati al cancello Forget. Vengono passati attraverso una funzione sigmoidea e quelli che tendono a 0 vengono scartati e altri vengono passati ulteriormente per calcolare lo stato della cella.

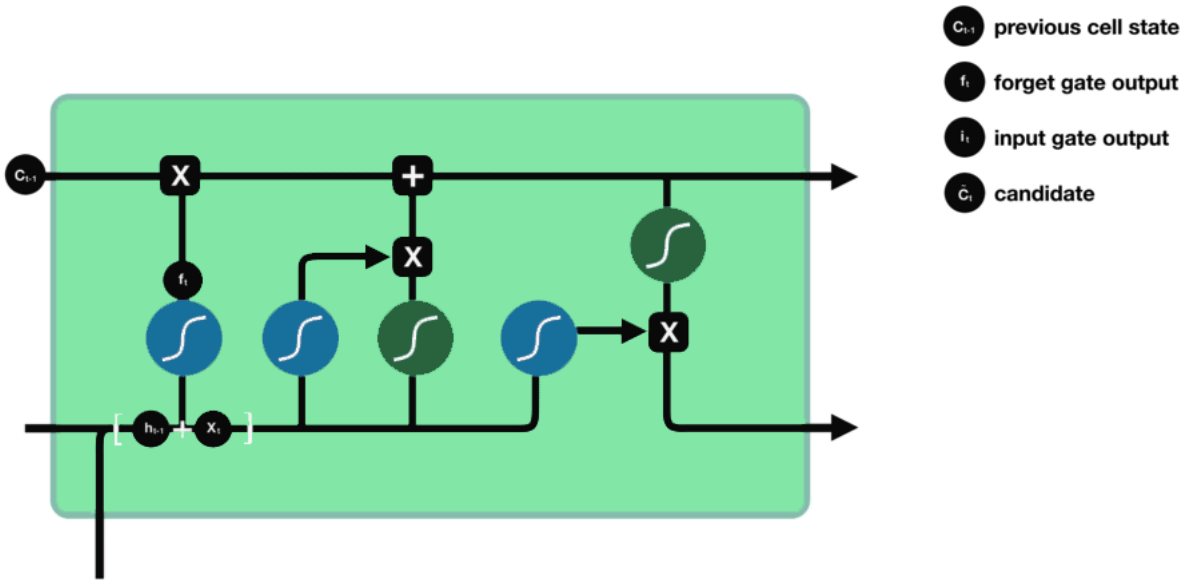

Figura 2.5: Esempio di Forget Gate

Input Gate

Input Gate aggiorna lo stato della cella e decide quali informazioni sono importanti e quali no. Poiché il gate di dimenticanza aiuta a scartare le informazioni, il gate di input aiuta a scoprire informazioni importanti e memorizzare determinati dati nella memoria che sono rilevanti. h t-1 e x t sono gli input che sono entrambi passati rispettivamente attraverso le funzioni sigmoide e tanh. La funzione tanh regola la rete e riduce la distorsione.

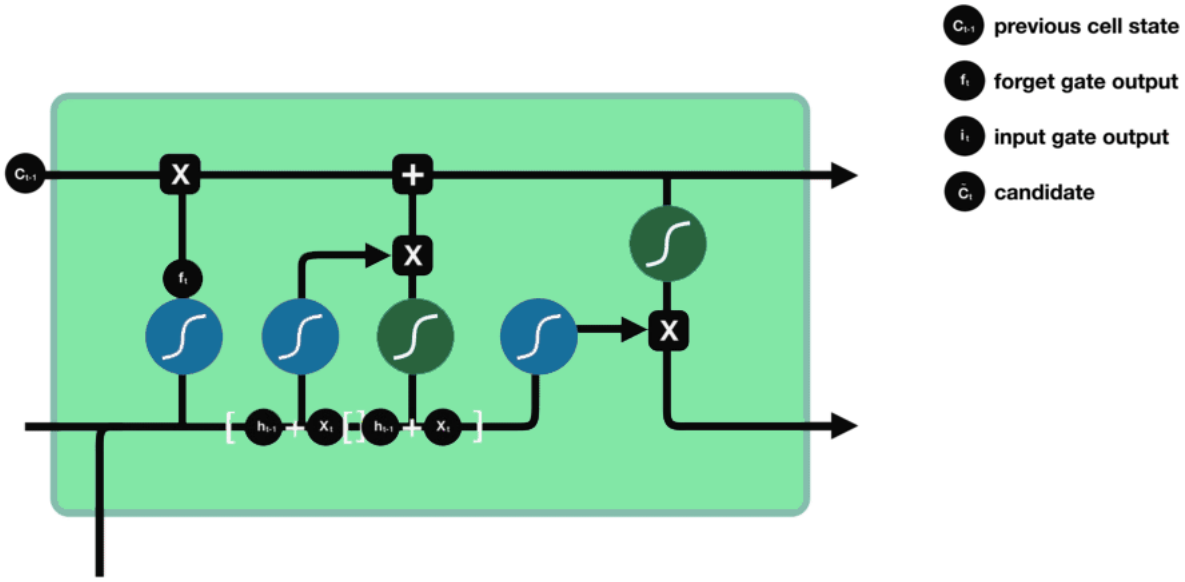

Figura 2.6: Esempio di Input Gate

Cell State

Tutte le informazioni ottenute vengono quindi utilizzate per calcolare il nuovo stato della cella. Lo stato della cella viene prima moltiplicato con l'uscita della porta dimenticata. Ciò ha la possibilità di far cadere i valori nello stato della cella se viene moltiplicato per valori prossimi a 0. Quindi un'aggiunta puntuale con l'output dal gate di input aggiorna lo stato della cella a nuovi valori che la rete neurale trova rilevanti.

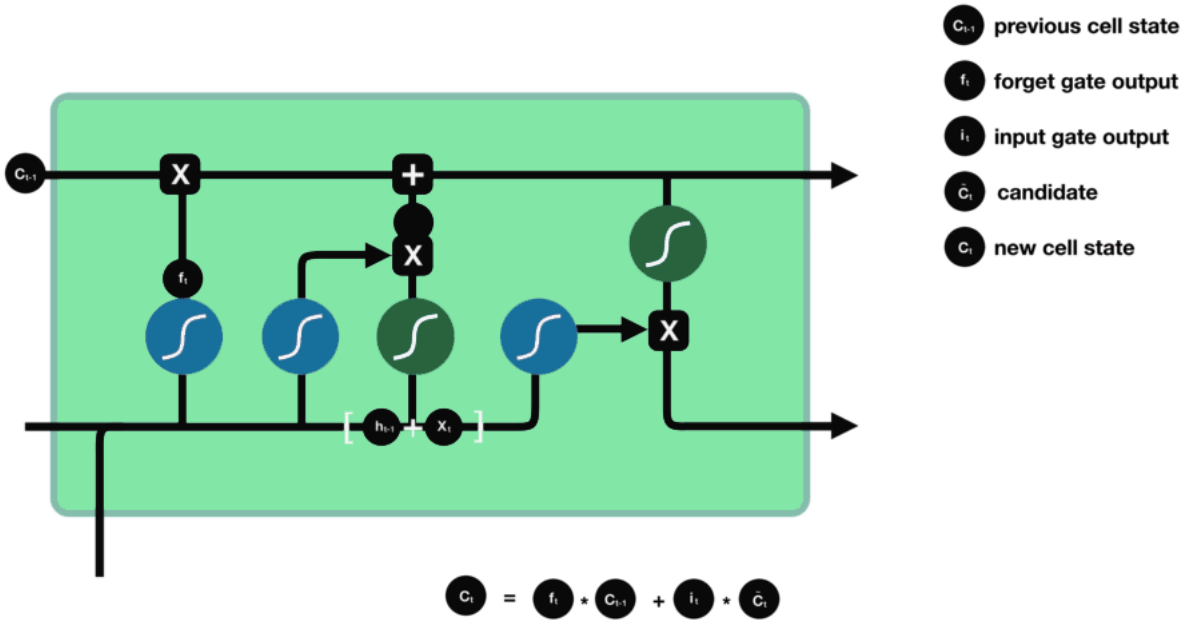

Figura 2.7: Esempio di Cell State

Output gate

L'ultima porta che è la porta di Uscita decide quale dovrebbe essere il prossimo stato nascosto. h t-1 e x t vengono passati a una funzione sigmoidea. Quindi lo stato della cella appena modificato viene passato attraverso la funzione tanh e viene moltiplicato con l'output del sigmoide per decidere quali informazioni dovrebbe trasportare lo stato nascosto.

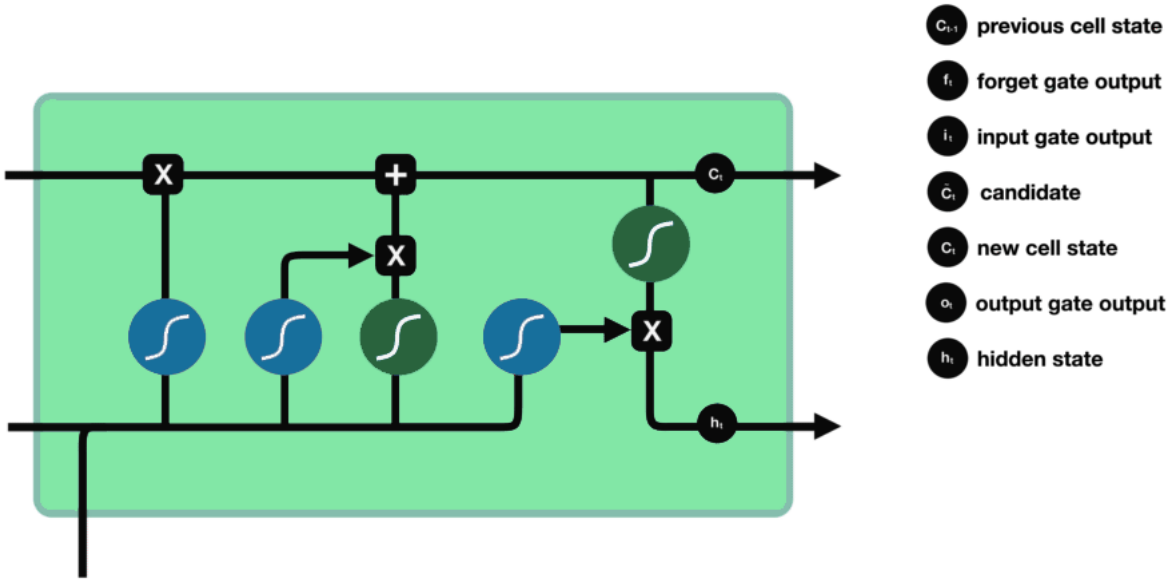

Figura 2.8: Esempio di Output gate

Le reti neurali tradizionali soffrono di memoria a breve termine. Inoltre, un grosso inconveniente è il problema del gradiente che scompare. (Mentre la backpropagation il gradiente diventa così piccolo da tendere a 0 e un tale neurone non è di alcuna utilità in ulteriori elaborazioni.) Gli LSTM migliorano efficacemente le prestazioni memorizzando le informazioni rilevanti che sono importanti e trova il modello.
Una buona ragione per utilizzare LSTM è che è efficace nella memorizzazione di informazioni importanti. Se osserviamo le altre tecniche di classificazione della rete non neurale, vengono addestrate su più parole come input separati che sono solo parole che non hanno alcun significato reale come una frase e mentre predice la classe darà l'output in base alle statistiche e non in base al significato . Ciò significa che ogni singola parola è classificata in una delle categorie. Questo non è lo stesso in LSTM. In LSTM possiamo usare una stringa di parole multiple per scoprire la classe a cui appartiene. Questo è molto utile quando si lavora con l'elaborazione del linguaggio naturale. Se utilizziamo livelli appropriati di incorporamento e codifica in LSTM, il modello sarà in grado di scoprire il significato effettivo nella stringa di input e fornirà la classe di output più accurata. Il codice seguente elaborerà l'idea su come viene eseguita la classificazione del testo utilizzando LSTM[9].

Autoencoders

Un autoencoder è un tipo di rete neurale artificiale utilizzata per apprendere le codifiche dei dati, addestrando la rete a catturare le parti più importanti dell'immagine di input. Gli

autoencoder sono costituiti da 3 parti:

- Encoder: un modulo che comprime i dati di input in una rappresentazione codificata che in genere è più piccola di diversi ordini di grandezza rispetto ai dati di input.
- Bottleneck: un modulo che contiene le rappresentazioni della conoscenza compressa ed è quindi la parte più importante della rete.
- Decoder: un modulo che aiuta la rete a "decomprimere" le rappresentazioni della conoscenza e ricostruisce i dati dalla sua forma codificata. L'output viene quindi confrontato con una verità fondamentale.

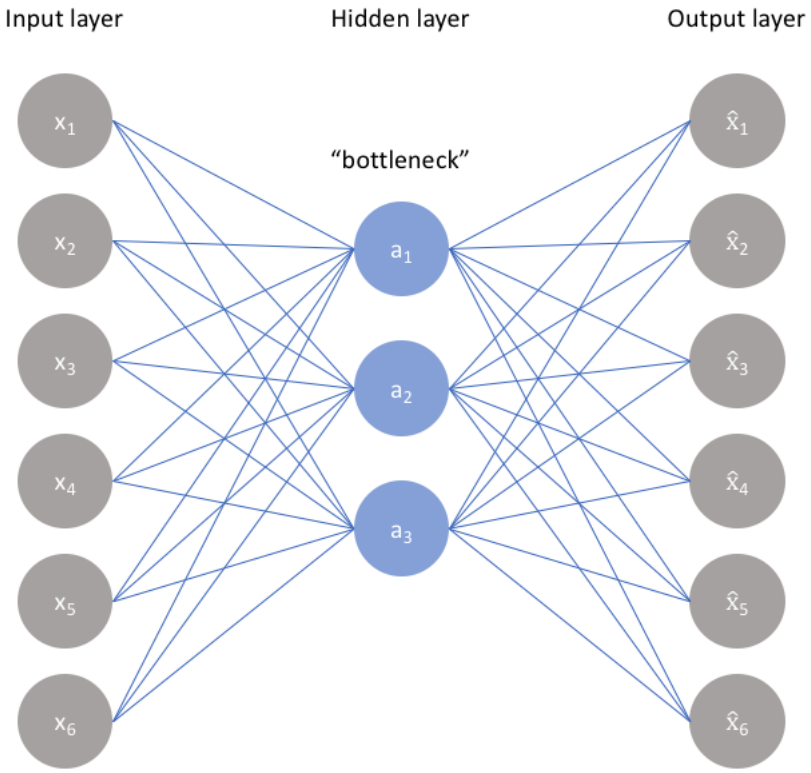

Figura 2.9: Esempio di autoencoder

LSTM Autoencoders

Per un dato set di dati di sequenze, un LSTM Autoencoders è configurato per leggere la sequenza di input, codificarla, decodificarla e ricrearla. Le prestazioni del modello vengono valutate in base alla capacità del modello di ricreare la sequenza di input. Una volta che il modello raggiunge il livello di prestazioni desiderato ricreando la sequenza, la parte del decodificatore del modello può essere rimossa, lasciando solo il modello dell'encoder.

Questo modello può quindi essere utilizzato per codificare le sequenze di input in un vettore di lunghezza fissa. I vettori risultanti possono quindi essere utilizzati in una varietà di applicazioni, non da ultimo come rappresentazione compressa della sequenza come input per un altro modello di apprendimento supervisionato.

2.1.10 Gated Recurrent Unit(GRU)

Sapendo come funziona un LSTM il GRU è simile. Il GRU è la nuova generazione di reti neurali ricorrenti. Il GRU si è sbarazzato del Cell State ed ha utilizzato lo stato nascosto per trasferire le informazioni. Ha anche solo due porte, una reset gate e update gate.

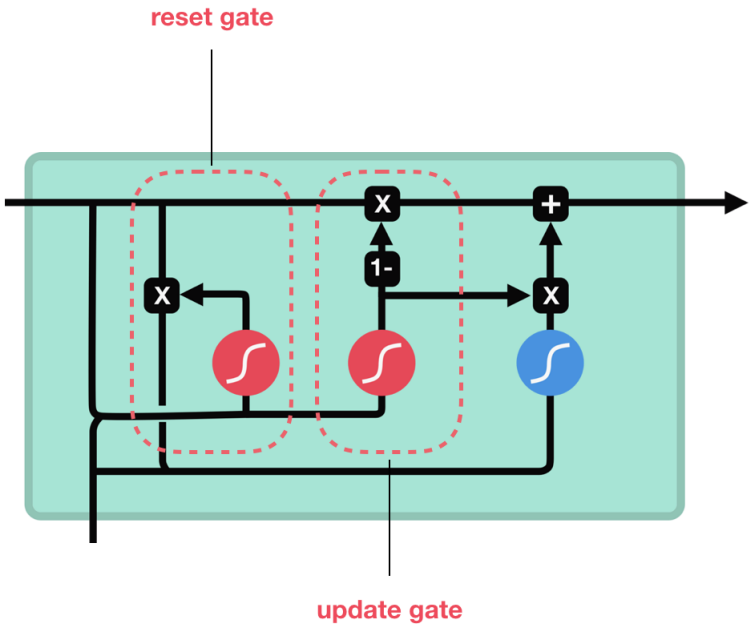

Figura 2.10: Esempio di cella GRU e suoi cancelli

Update gate

L'Update gate agisce in modo simile al forget ed input gate di un LSTM. Decide quali informazioni buttare via e quali nuove informazioni aggiungere.

Reset Gate

Il Reset Gate è un altro cancello utilizzato per decidere quante informazioni passate dimenticare. E questo è un GRU. GRU ha meno operazioni sui tensori; quindi, sono un po'

più veloci da allenare rispetto agli LSTM. Non c'è un vincitore chiaro quale sia il migliore.Ho usato entrambi per determinare quale funziona meglio.

2.1.11 Matrice di confusione

La matrice di confusione (confusion matrix) è uno strumento per analizzare gli errori compiuti da un modello di machine learning.E' utile per valutare la qualità delle previsioni del modello di classificazione. In particolar modo, la matrice mette in evidenza dove sbaglia il modello, in quali istanze risponde peggio e quali meglio.

Costruzione della matrice di confusione

Prendo il semplice caso di un classificatore binario. Le classi sono due: SI o NO.Un esempio di classificatore binario è il filtro antispam sulla posta in arrivo. Il modello deve decidere se l'email in entrata è spam oppure no. Elenco le classi del problema nelle righe e nelle colonne.
Nelle righe indico le classi effettive. Sono le classi delle risposte corrette.
Nelle colonne indico le classi di previsione.Sono le classi delle risposte del modello.
Come mostrato in figura.

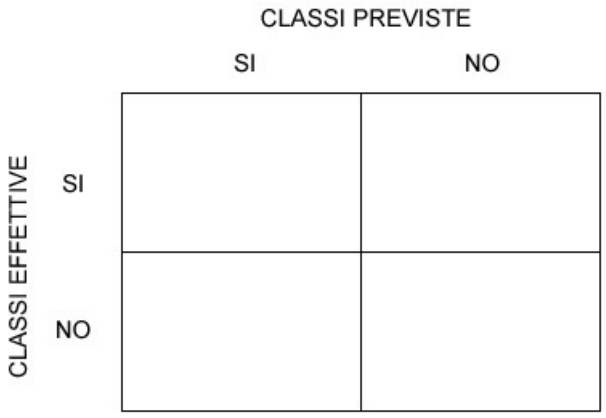

Figura 2.11: Esempio matrice di confusione

Ad esempio, il modello analizza 80 email e le classifica spam/no spam. In 60 casi il modello classifica correttamente mentre in 20 sbaglia.

Gli errori e le risposte corrette però non sono tutte uguali. Possono verificarsi quattro casi:

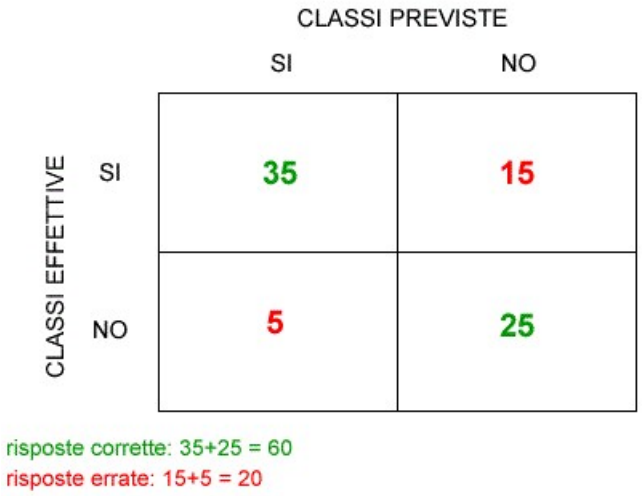

Figura 2.12: Esempio matrice di confusione per spam

- True positive (TP) Se la classe prevista è SI ed è uguale alla classe effettiva, si tratta di un caso di true positive (vero positivo). Il modello ha risposto correttamente SI.
- True negative (TN) Se la classe prevista è NO ed è uguale alla classe effettiva, si tratta di un caso di true negative (vero negativo).
- Il modello ha risposto correttamente NO.
- False positive (FP) Se la classe prevista è SI ma è diversa

Metriche della matrice di confusione

Dalla matrice di confusione posso ottenere facilmente diverse metriche, illustro le principali:

- Il tasso di errore (error rate) misura la percentuale di errore delle previsione sul totale delle istanze. Varia da 0 (peggiore) a 1 (migliore).
  $ERR = \frac{TP+FN}{TP+TN+FP+FN}$
- L'accuratezza (accuracy) misura la percentuale delle previsioni esatte sul totale delle istanze. E' l'inverso del tasso di errore. Varia da 0 (peggiore) a 1 (migliore).
  $ACC = \frac{TP+TN}{TP+TN+FP+PN} = 1 - ERR$
- La precisione (precision) è la percentuale delle previsioni positive corrette (TP) sul totale delle previsioni positive del modello (giuste TP o sbagliate FP).
  $PR = \frac{TP}{TP+FP}$

- Il richiamo (o recall) o sensitività (sensitivity) è la percentuale delle previsioni positive corrette (TP) sul totale delle istanze positive. Varia da 0 (peggiore) a 1 (migliore). $RECALL = \frac{TP}{TP+FN}$

2.2 Stato dell'arte

All'interno di questo capitolo si discute di alcune delle tecniche più caratteristiche e già presenti, sulla rilevazione di fake news. Alcune si basano su algoritmi di Machine Learning altri su algortimi di Deep Learning. Gli approcci analizzati provengono da:

- Approccio basato sul machine learning utilizzando i seguenti algoritmi: SVC, logistic regression, SGD classifier, random forest classifier, bagging classifier, AdaBoost classifier, decision tree classifier, and K-nearest neighbors classifier (KNN). [6]
- Approccio basato sul deep learning utilizzando i seguenti algoritmi:recurrent neural networks (RNN), long short-term memory (LSTM),Multi-layer Perceptron(MLP),Gated Recurrent Unit(GRU) [3]

2.2.1 Problema in esame

Secondo lo studio dell'International Fact Checking Network (IFCN) tra gennaio e aprile 2020, le fake news diffuse sui social media possono essere classificate come segue: contenuto su sintomi, cause e cure, documenti governativi, diffusione del virus, rappresentazione ingannevole di foto, commenti di politici e cospirazioni che incolpano particolari gruppi, paesi o comunità per la diffusione del virus. La fake news diffusa sui social media ha portato alla crisi economica in alcuni paesi. Ad esempio, in alcuni paesi le persone hanno smesso di consumare cibo non vegetariano poiché è stata diffusa la notizia falsa che animali e uccelli potrebbero essere infettati da COVID-19 e il consumo di cibo non vegetariano può diffondere il virus nelle persone.

2.2.2 Approccio Con Machine learning

L'approccio utilizzato dal paper [6] tiene in considerazione dei gravi impatti della diffusione di notizie false che incidono sulla privacy e sulla sicurezza degli utenti, la necessità del momento è progettare un meccanismo di successo per rilevare/prevedere le notizie false. In questo lavoro si tenta di classificare le fake news relative al COVID-19. Il set di dati utilizzato

in questo lavoro è una fusione di notizie false e reali raccolte da diverse piattaforme di social media e siti Web . Questo set di dati viene quindi sottoposto a preelaborazione per rimuovere del testo irrilevante come la punteggiatura degli URL e i dati rumorosi, ecc. Quindi, il testo risultante viene suddiviso in diverse piccole parole utilizzando la tokenizzazione. Quindi, le funzionalità più importanti vengono estratte da questi token. Queste funzionalità vengono quindi addestrate da algoritmi ML all'avanguardia per classificare se le notizie relative al COVID-19 sono false o reali.

Nella fase successiva, le funzionalità estratte vengono addestrate utilizzando diversi algoritmi ML all'avanguardia. Gli algoritmi ML utilizzati in questo lavoro sono SVC lineare, regressione logistica, classificatore SGD, classificatore foresta casuale, classificatore bagging, classificatore AdaBoost, classificatore albero decisionale e classificatore K-nearest neighbors. Di seguito riportati i risultati dell'addestramento degli algoritmi Le metriche ottenute dal

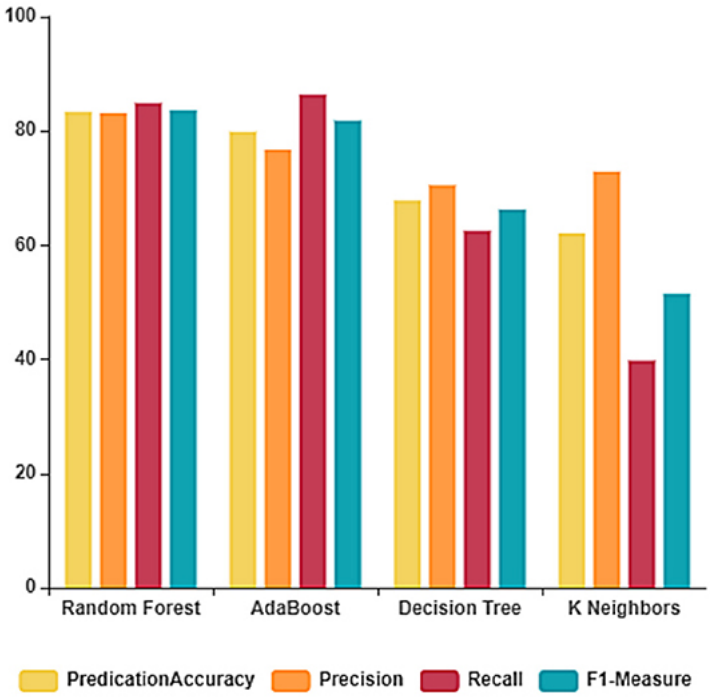

Figura 2.13: Risultati papaer [6]

Random Forest dimostrano che l'approccio, basato su algoritmi di machine learning, è sicuramente una buona soluzione. Però bisogna considerare anche una analisi predittiva con algormtimi di deep learning e non solo lineare con i classici del machine learning in modo tale da avere una maggiore precisione e affidabilità di predizione.

2.2.3 Approccio Con Deep Learning

L'approccio utilizzato dal seguente paper[3], utilizzando un nuovo set di dati di riferimento chiamato LIAR: è un nuovo set di dati pubblicamente disponibile per il rilevamento di notizie false. Ha raccolto da POLITIFACT.COM una breve dichiarazione decennale, 12,8K etichettata manualmente in vari contesti , che fornisce un rapporto analitico dettagliato e un collegamento al suo livello di origine per ciascun caso. Questo set di dati può essere utilizzato anche per la ricerca di verifica dei fatti. Il set di dati LIAR include 12.836 brevi dichiarazioni etichettate per veridicità, soggetto, contesto/luogo, oratore, stato, partito e storia precedente. Con queste dimensioni e un arco di tempo di dieci anni, i casi LIAR vengono raccolti in un contesto più naturale, come dibattiti politici, spot televisivi, post di Facebook, tweet, interviste, comunicati stampa, ecc. In ogni caso, l'etichettatore fornisce un lungo rapporto di analisi a terra ogni giudizio. Hanno valutato diversi metodi di apprendimento popolari basati su questo set di dati. Le linee di base includono la regressione logistica, le macchine vettoriali di supporto, LSTM e il modello CNN. Il nostro lavoro si è chiarito nei seguenti passaggi come segue: **Primo passo**: preparare il set di dati LIAR in quattro livelli:

- Il primo livello è dividere ogni frase da trattare separatamente.
- Il secondo livello sta rimuovendo le parole d'arresto e include l'identificazione delle parole inutili in ogni affermazione come (the, a, an, ecc.).
- Il terzo livello è la radice in cui ogni parola ritorna al suo infinito.

Secondo passaggio : l'output dello stemming sarà l'input per l'incorporamento delle parole che ha svolto un ruolo importante nel deep learning basato sull'analisi dell'inganno che include la rappresentazione di ogni singola parola in ogni frase tramite un vettore dimensionale e ottenere la relazione tra due parole non solo sintattica ma anche stesso (poiché 'vedere' e 'guardare' sono molto diversi nella sintassi, ma il loro significato è in qualche modo correlato). Un altro vantaggio è che l'algoritmo rileva le parole che appaiono per lo più insieme (come 'usura' e 'vestiti') e mostra la loro relazione e quindi questo è in grado di prevedere la parola successiva .

Terzo passo : i risultati del livello di incorporamento delle parole saranno l'input per i modelli RNN (vanilla, GRU) e la tecnica LSTM.

Quarto passaggio : l'output del passaggio quattro otterrà il risultato finale determinando se la notizia è veritiera o ingannevole.

Come è comune nei problemi di data mining, una volta creati i modelli, il processo potrebbe essere ripetuto con nuovi dati e nuove funzionalità. Gli algoritmi hanno prodotto i seguenti risultati:

Model	Test Accurancy
SVMs	0.255
Logistic Regress0ion	0.247
Bi-LSTMs	0.233
CNN	0.270
Vanilla	0.215
GRU	0.217
LSTM	0.2166

Tabella 2.4: Risultati del paper [3]

Abbiamo scoperto che il peggior risultato dei nostri esperimenti ottiene da Vanilla a causa della sua incapacità di risolvere compiti complessi che hanno un'applicazione pratica, Ha anche cambiato il formato delle informazioni originali, il che significava che non era in grado di contenere l'importante contenuto della memoria per più di pochi passi di tempo, e anche la scomparsa del gradiente è uno dei suoi svantaggi.

LSTM ha anche mostrato inefficienza rispetto a GRU e CNN perché i suoi due principali inconvenienti, in primo luogo, è più costoso calcolare l'output della rete e applicare la propagazione indietro. abbiamo semplicemente più matematica da fare a causa della complessa attivazione. Tuttavia questo non è importante quanto il secondo punto, in secondo luogo, la memoria esplicita aggiunge molti altri pesi a ciascun nodo, che devono essere tutti addestrati. Ciò aumenta la dimensionalità del problema e rende potenzialmente più difficile trovare una soluzione ottimale.

Il miglior risultato dei nostri esperimenti è GRU, abbiamo visto un leggero miglioramento nei suoi risultati rispetto a vanilla e LSTM a causa della risoluzione del problema di sfumatura del gradiente che è un problema in vaniglia ed è facile da modificare e non necessita di unità di memoria, quindi, più veloce da allenare rispetto a LSTM e dare secondo le prestazioni.

Confrontando i nostri risultati con il risultato di William Yang, abbiamo scoperto che la CNN è la migliore tra tutti i risultati poiché la CNN tende ad essere molto più veloce (5 volte

più veloce) della RNN e più efficiente dipende dalla nostra implementazione e a causa di Nvidia storicamente si è concentrata molto più sulla CNN che sulla RNN, poiché la visione artificiale utilizza principalmente la CNN.

CAPITOLO 3

Implementazione di algoritmi di machine learning e deep learning

All'interno di questo capitolo verranno sviluppati gli algoritmi analizzati nel backgroud, utilizzando le tecniche trattante nello stato dell'arte. Verrà adottato un altro dataset contenente fake news sul covid-19, con l'obiettivo di confrontare i risultati ottenuti.

3.1 Data Collection e data description

A seguito di una ricerca per trovare il miglior dataset possibile che si è conclusa con 5 poteziali dataset utilizzabili. Di seguito elencati:

1. Harvard [1]
2. Frontiersin [2]
3. Zenodo [3]
4. Kaggle [4]
5. GitHub [5]

[1] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/YOGY5W

[2] https://www.frontiersin.org/articles/10.3389/fpubh.2021.788074/full

[3] https://zenodo.org/record/4441377

[4] https://www.kaggle.com/datasets/jannalipenkova/covid19-public-media-dataset

[5] https://github.com/Programmer-RD-AI/COVID19-Fake-News-Dataset-NLP

Per la scelta ho selezionato dei parametri per comprendere quale poteva risultarmi utile per questo progetto. Di seguito riportati:

- **Usabilità**: facilità di utilizzo dati e pulizia del dataset
- **Quantità** : numero di articoli
- **Tipo**: con o senza etichetta, cioè se viene esplicitato che la notizia sia vera o falsa
- **Utilità**: pertinenza con idea iniziale e parametri di utilizzo oggettivi

Per ognuno di questi parametri ho attribuito una valutazione da 1 a 5.
Di seguito riportati i risultati che ogni dataset ha ricevuto.

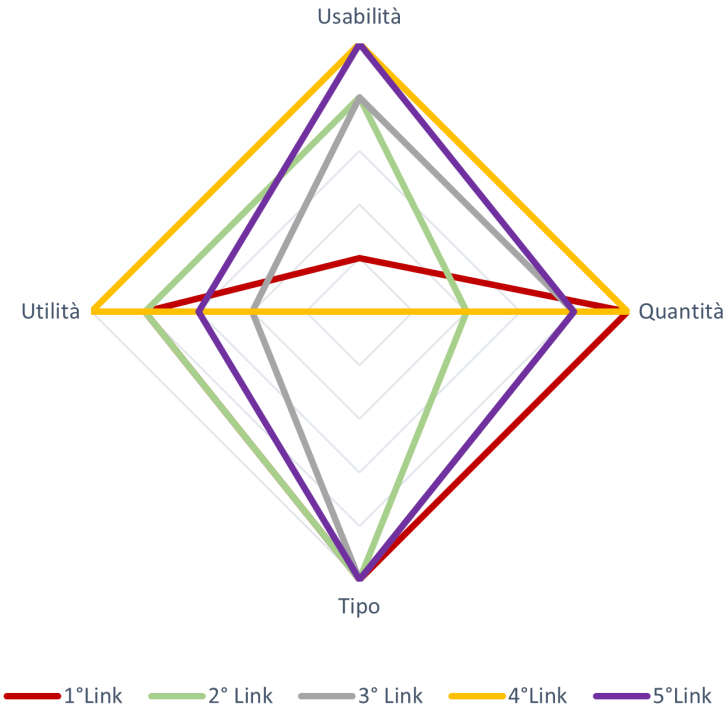

Figura 3.1: Risultati delle valutazioni dei dataset

La mia scelta finale ricade nell'effettuare il merge dei due dataset: Frontiersin [6] e GitHub [7] cosi da compensare la quantità dei file mancanti su Frontiersin con GitHub e diversificare la sorgente delle informazioni di Frontiersin con GitHub. Il dataset risultato è composto da 3 colonne e 7574 righe. Le 3 colonne sono:

- **Title**: titolo dell'articolo
- **Text** : testo dell'articolo

[6] https://www.frontiersin.org/articles/10.3389/fpubh.2021.788074/full
[7] https://github.com/Programmer-RD-AI/COVID19-Fake-News-Dataset-NLP

- **Label**: indica la tipologia della notizia, real se è vera, fake altrimenti.

3.1.1 Data Pre-Processing

La prima parte della lavorazione dei dati si concentra sul merge dei due dataset. I problemi principali riguardano:

- Rimozione della colonna del "Nome" dal dataset di Frontiers [8]
- Rimozione della colonna "Indice" dal dataset di GitHub [9]
- Aggiunta della colonna source nel dataset di Github
- Rimozione di eventuali notizie ripetute
- Uniformare la colonna label usando la stessa nomenclatura
- Rimuove ';' all'interno del text perchè usato come delimitatore di colonne

La decisione del rimuove la colonna "nome" ed "indice" è stata presa perchè non presente rispettivamente nell'altro dataset, inoltre i dati persi non vanno ad inficiare significativamente le prestazione degli algoritmi dato che non contengono informazioni importanti. Invece per la colonna source è stata aggiunta facilmente nel dataset di GitHub perchè le notizie provengono unicamente da Twitter e la source può rilevarsi un campo fondamentale come elemento di veriticità della notizia. Inoltre anche rimuovere eventuali notizie ripetute è fondamentale per evitare che ci siano decisione errate degli algortmi. È stato sufficiente rimuovere gli articoli con source provenienti da Twitter del dataset di Frontiers dato che erano poco meno di 100 e su un dataset di oltre 7000 articoli non influenzavano particolarmente. Dopo un'attenta uniformazione dei dati si possono unire i dataset e iniziare con la fase di Data Cleaning.

3.1.2 Data Cleaning

Prima di iniziare il Data Cleaning procedo con il caricare il dataset su Colab e leggere il .csv e separare le tre colonne con ';'.

```
1    uploaded = files.upload()
2    df = pd.read_csv(r'/content/data.csv',sep=";",quoting=3, on_bad_lines='skip')
```

[8] https://www.frontiersin.org/articles/10.3389/fpubh.2021.788074/full
[9] https://github.com/Programmer-RD-AI/COVID19-Fake-News-Dataset-NLP

Notizie duplicate o campi vuoti

La prima operazione è quella di rimuovere eventuali notizie duplicate,se ci sono. Inoltre si è scelto di rimuovere anche le righe con valori null dato che non aggiungo alcun contributo gli algoritmi.

```
    if(!df.duplicated())
        df.drop_duplicates(inplace = True)
    df.dropna(inplace = True)
```

Punteggiatura

La seconda operazione è quella della rimozione di tutta la punteggiatura. In questo caso ho deciso di lasciare il '!' e '?', con l'idea che notizie con molti punti interrogativi o esclamativi posso essere indice di notizia falsa o quanto meno avere un peso.

```
    df = df.apply(lambda x: x.str.replace(',', ''))
    df = df.apply(lambda x: x.str.replace(':', ''))
    df = df.apply(lambda x: x.str.replace(';', ''))
    df = df.apply(lambda x: x.str.replace('.', ''))
    df = df.apply(lambda x: x.str.replace('"', ''))
    df = df.apply(lambda x: x.str.replace('#', ''))
    df.replace('\'','', regex=True, inplace=True)
    df = df.apply(lambda x: x.str.replace('(', ''))
    df = df.apply(lambda x: x.str.replace(')', ''))
```

Singolarizzare

La terza operazione è quella di trasformare le parole al singolare. In questo modo si evita che una parola che lo stesso significato ma posta al singolare o plurale possa essere considerata più volte e con diverso peso.

```
    df['text'] = df['text'].astype('str')
    df['text'] = df['text'].apply(lambda x: ' '.join([singularize(item) for item
    in x.split()]))
```

Lemmatizzare

La quarta operazione è quella di trasformare tutti i verbi all'infinito in questo modo, come per i singolari, si evita che una parola avente lo stesso significato venga cosiderata più volte.Inoltre la lemmatizzazione di textBlob(libreria dedicata) chiede di indicare se deve

lemmatizzare un verbo o un sostantivo. Dato che non ho previsto una fase di POS, ho deciso di lemmatizzare soltanto i verbi in quanto sembrano essere quelli che hanno più necessità e che racchiudono il significato maggiore.

```
    df['text'] = df['text'].astype('str')
    df['text'] = df['text'].apply(lambda x: ' '.join([Word(item).lemmatize("v")
    for item in x.split()]))
```

Stop Words

La quinta operazione consiste nella rimozione delle StopWord. Le stopword sono le parole che vengono usate molto frequentemente, e sono così frequenti, che perdono un po' il loro significato semantico. Parole come "of, are, the, it, is" sono alcuni esempi di stopword. Di seguito riportato anche un vocabolario di esempio per evitare un eccessiva lunghezza del codice.

```
stopWords = {"the": "", "The": "", "A": "", "a": ""}
print(stopWords.get("As"))
df['text'] = df['text'].astype('str')
column_names = ["text", "source", "lable"]
df1 = pd.DataFrame(columns = column_names)
c=0
for x in df['text']:
  var=x.split(" ")
  i=0
  for v in var:
    if (stopWords.get(v) == ""):
      var[i] = stopWords.get(v)
    i+=1
  if c == 1:
    print(var)

  df.iloc[c] = pd.Series([' '.join(var),df['source'].iloc[c],df['label'].iloc[c
    ]])

  c+=1
```

3.2 Implementazione del TF-IDF

Una volta "pulito" il testo si procede con l'implementazione TF-IDF. Di seguito riportati anche commenti relativi al codice per facilitarne la comprensione.
Il parametro "min_df" imposta il numero minimo di documenti in cui è contenuto qualsiasi termine.Può essere un numero intero che imposta il numero in modo specifico o un decimale compreso tra 0 e 1 che viene interpretato come percentuale di tutti i documenti.Il prossimo è "max_df" che controlla in modo simile il numero massimo di documenti in cui è possibile trovare qualsiasi termine. Se il 90% dei documenti contiene la parola "spork", è così comune che non è molto utile. Infine, abbiamo "ngram_range" che è una tupla contenente l'intervallo di dimensioni di ngram da utilizzare. (1, 3) significa utilizzare 1 grammo, 2 grammi e 3 grammi. Stiamo solo usando 1 e 2 grammi.

```
cvec = CountVectorizer(min_df=.0025, max_df=.1, ngram_range=(1,2))
```

Calcola tutti gli n-grammi trovati in tutti i documenti

```
cvec.fit(df.text)
list(islice(cvec.vocabulary_.items(), 20))
```

Controlla quanti n-grammi totali abbiamo

```
len(cvec.vocabulary_)
```

trasformare il documento in una rappresentazione di "Bag of words" che essenzialmente è solo una colonna separata per ogni termine contenente il conteggio all'interno di ogni documento. Successivamente, daremo un'occhiata alla scarsità di questa rappresentazione che ci consente di sapere quanti valori diversi da zero ci sono nel set di dati.Più i dati sono scarsi, più difficile sarà modellare

```
cvec_counts = cvec.transform(df.text)
print('sparse matrix shape:', cvec_counts.shape)
print('conteggio diverso da zero:', cvec_counts.nnz)
print('scarsit : %.2f%%' % (100.0*cvec_counts.nnz/(cvec_counts.shape[0]*
cvec_counts.shape[1])))
```

ricavare i primi 20 termini più comuni

```
occ = np.asarray(cvec_counts.sum(axis=0)).ravel().tolist()
counts_df = pd.DataFrame({'term': cvec.get_feature_names(), 'occurrences':
occ})
counts_df.sort_values(by='occurrences', ascending=False).head(20)
```

Ora che abbiamo il conteggio dei termini per ogni documento, possiamo usare TfidfTransformer per calcolare i pesi per ogni termine in ogni documento

```
transformer = TfidfTransformer()
transformed_weights = transformer.fit_transform(cvec_counts)
print(transformed_weights.shape)
```

i primi 20 termini in base al peso medio di tf-idf:

```
weights = np.asarray(transformed_weights.mean(axis=0)).ravel().tolist()
weights_df = pd.DataFrame({'term': cvec.get_feature_names(), 'weight':
weights})
weights_df.sort_values(by='weight', ascending=False).head(20)
```

Ripetiamo il tf-idf anche per la colonna source e poi unisco le due tabelle

```
x = hstack([transformed_weights,transformed_weights2])
```

3.3 Implementazione degli algoritmi di machine learning

Un processo di addestramento implica la selezione degli iperparametri migliori/ottimali utilizzati dagli algoritmi di apprendimento per fornire il miglior risultato. Gli iperparametri,quindi, sono esplicitamente definiti dall'utente per controllare il processo di apprendimento". Qui il prefisso "iper" suggerisce che i parametri sono parametri di primo livello utilizzati per controllare il processo di apprendimento. In questa sezione andrò a mostrare l'implementazione e gli iperparametri scelti attraverso un'analisi empirica per ogni algoritmo.Utilizzerò la libreria sklearn apposita per gli algoritmi di machine learning. Oltre alla scelta degli iperparametri per l'addestramento utilizzo le funzioni seguenti:

- **confusion_matrix()**: prende in input due paramentri: la colonna delle label corrette e il secondo parametro le predizioni fatte dell'algoritmo. In output restituisce la matrice di confusione(una rappresentazione dell'accuratezza di classificazione statistica.)
- **cross_val_predict() - cross_val_score()** : il primo metodo realizza la Cross validation è un metodo applicato a un modello e a un set di dati nel tentativo di stimare l'errore fuori campione. È diventato abbastanza popolare grazie alla sua semplicità e utilità.Quando adattiamo un modello a un set di dati, lo facciamo riducendo al minimo una sorta di funzione di perdita; il più delle volte, useremo la funzione di perdita di errore al quadrato per semplicità. È noto, e dovrebbe essere abbastanza ovvio, che la stima dell'errore di previsione risultante utilizzando gli stessi dati che abbiamo utilizzato per adattare il modello produrrà risultati eccessivamente ottimistici. Pertanto, è prassi comune testare il modello su un nuovo set di dati per fornire una migliore stima dell'errore di previsione fuori campione. Tuttavia, quando la raccolta dei dati è proibitiva

in termini di costi, potremmo preferire non "buttare via" una parte significativa dei dati in un set di dati di test. In questo caso potrei passare alla k-fold cross validation, il cui aspetto più popolare è la convalida incrociata 10 volte. Nella convalida incrociata k-fold, il set di dati viene suddiviso casualmente in k partizioni, Quindi adattiamo il nostro modello a un set di dati costituito da k-1 delle k parti originali e utilizziamo la parte rimanente per la convalida. Cioè stimiamo l'errore fuori campione utilizzando la parte di dati lasciata fuori dalla procedura di adattamento. Ripetiamo questo k volte e la nostra stima per l'errore fuori campione è la media sulle k esecuzioni di convalida.[7]
Il secondo metodo è utilizzato per eseguire il leave-one-out (LOOCV) , che utilizza il seguente approccio:
-Suddividere un set di dati in un set di addestramento e un set di test, utilizzando tutte le osservazioni tranne una come parte del set di addestramento;
-Creare un modello utilizzando solo i dati del training set;
-Utilizzare il modello per prevedere il valore di risposta di un'osservazione lasciata fuori dal modello e calcolare l'errore quadratico medio (MSE);
-Ripetere questo processo n volte. Calcolare che l'MSE del test sia la media di tutti gli MSE del test.
Il problema principale di questo approccio è il costo in termini di tempo da utilizzare dato che n è grande e impiega molto tempo per adattarsi ad un dataset ed è computazionalmente costoso.
Provati entrambi ho deciso di utilizzare il cross_val_predict() dato che ottengo buoni risultati dalla matrice di confusione e impiega, esponenzialmente, meno tempo nelle operazioni, inoltre ho aumentato ulteriormente la cross validation portando una maggiore precisione.

- **print_metrics()**: Stampa in modo ordinato la matrice di confusione.

3.3.1 Alberi decisionali

Gli iperparametri principali forniti a questo algoritmo:

- **criterion**: La funzione per misurare la qualità di una divisione. I criteri supportati sono "gini" per l'impurità di Gini e "log_loss" e "entropia" entrambi per il guadagno di informazioni di Shannon.

- **splitter** : La strategia utilizzata per scegliere la divisione in ogni nodo. Le strategie supportate sono "best" per scegliere la divisione migliore e "random" per scegliere la migliore divisione casuale.
- **min_samples_split**: Il numero minimo di campioni necessari per dividere un nodo interno.
- **min_samples_leaf** : Il numero minimo di campioni richiesto per essere in un nodo foglia.
- **min_weight_fraction_leaf** : La frazione pesata minima della somma totale dei pesi (di tutti i campioni di input) richiesta per trovarsi su un nodo foglia.

```
1   x = print_metrics(confusion_matrix(df.label, cross_val_predict(
    DecisionTreeClassifier(), x.toarray(), df.label, cv=20)))
```

3.3.2 Gaussian

Gli iperparametri principali forniti a questo algoritmo:

- **Priors**: quando vengono fornite le priorità (in una matrice) non verranno modificate in base al set di dati.
- **var_smoothing** :il valore float fornito verrà utilizzato per calcolare le varianze maggiori di ciascuna caratteristica e aggiungerlo alla varianza del calcolo della stabilità.

```
1   print_metrics(confusion_matrix(df.label, cross_val_predict(GaussianNB(priors=
    None, var_smoothing=1e-10), x.toarray(), df.label, cv=20)))
```

3.3.3 Random Forest

Gli iperparametri principali forniti a questo algoritmo:

- **n_estimators**: Il numero di alberi nella foresta.
- **criterion** :La funzione per misurare la qualità di una divisione. Ho scelto il criterion="entropy" perchè dava una matrice di confunsione con metrice migliori.
- **warm_start** :Quando è impostato su True, riutilizza la soluzione della chiamata precedente per adattare e aggiungere più stimatori all'insieme, altrimenti, adatta semplicemente una foresta completamente nuova.

```
print_metrics(confusion_matrix(df.label, cross_val_predict(
RandomForestClassifier(n_estimators=100, criterion = "entropy",
min_samples_split = 2, warm_start = True), x.toarray(), df.label, cv=20)))
```

3.3.4 Supportvector machine(SVC)

Gli iperparametri principali forniti a questo algoritmo:

- **kernel**: seleziona il tipo di hyperplane utilizzato per separare i dati. L'uso di 'linear' utilizzerà un iperpiano lineare (una linea nel caso di dati 2D). 'rbf' e 'poly' usano un iperpiano non lineare
- **gamma**: è un parametro per iperpiani non lineari. Più alto è il valore gamma che tenta di adattarsi esattamente al set di dati di addestramento(Possiamo vedere che l'aumento della gamma porta a un overfitting poiché il classificatore cerca di adattare perfettamente i dati di allenamento)

```
print_metrics(confusion_matrix(df.label, cross_val_predict(SVC(kernel = '
linear', gamma = 10), x.toarray(), df.label, cv=20)))
```

3.3.5 Passive-Aggressive

Gli iperparametri principali forniti a questo algoritmo:

- **C**: Questo è il parametro di regolarizzazione e denota la penalizzazione che il modello apporterà su una previsione errata
- **max_iter**: il numero massimo di iterazioni che il modello effettua sui dati di addestramento.

```
print_metrics(confusion_matrix(df.label, cross_val_predict(
PassiveAggressiveClassifier(C = 3.0, max_iter = 3000),x.toarray(), df.label,
cv=20)))
```

3.4 Implementazione degli algoritmi di Deep Learning

Per L'implementazione degli algoritmi di deep learning ho eseguito lo stesso data cleaning degli algortmi di machine learning.

3.4.1 Long-Short Term Memory(LSTM)

Definizione e Formazione del Modello

consiste in diversi tipi di operazioni e livelli sequenziali:

1. Un tokenizer viene utilizzato per trasformare ogni articolo in un vettore di parole indicizzate (token).
2. Un livello di incorporamento di parole che apprende un vettore di incorporamento con m dimensioni per ogni parola univoca e applica questo incorporamento per le prime n parole in ogni articolo, generando una matrice m x n .
3. Strati convoluzionali 1D e max pooling.
4. Layer LSTM seguiti da layer dropout.
5. Un ultimo strato completamente connesso.

Il Tokenizzatore

Un tokenizer viene utilizzato per dividere ogni articolo di notizie in un vettore di parole sequenziali, che viene successivamente convertito in un vettore di numeri interi assegnando un indice intero univoco a ciascuna parola

Livelli di incorporamento di parole

Gli incorporamenti di parole sono rappresentazioni vettoriali apprendibili di parole che rappresentano il significato delle parole in relazione ad altre parole. Gli approcci di deep learning possono apprendere l'incorporamento di parole da raccolte di testo in modo tale che le parole con vettori di incorporamento simili tendano ad avere significati simili o rappresentare concetti simili.

Strati convoluzionali 1D e Max-pooling

Questi componenti sono la parte convoluzionale della rete neurale convoluzionale ricorrente.Per i dati di testo è necessario utilizzare livelli convoluzionali e di pooling 1D. Uno strato convoluzionale 1D ha una serie di kernel, che sono vettori a bassa dimensione che scorrono in modo incrementale attraverso il vettore di input mentre i prodotti punto vengono calcolati per produrre il vettore di output. Come i livelli convoluzionali 1D, anche i livelli

di raggruppamento massimo 1D operano sui vettori ma riducono le dimensioni dell'input selezionando il valore massimo dalle regioni locali nell'input.

Livello completamente connesso

La parte finale di questo modello è semplicemente uno strato completamente connesso. Questo livello riceve l'output dell'ultimo livello LSTM e calcola una somma ponderata dei valori vettoriali, applicando un'attivazione sigmoidea a questa somma per produrre l'output finale, un valore compreso tra 0 e 1 corrispondente alla probabilità che un articolo sia una notizia reale.

Realizzazione

Gli iperparametri principali sono i seguenti:

La classe,di seguito mostrata, rappresenta una pipeline che può essere adattata direttamente ai dati di testo preelaborati senza dover eseguire in anticipo passaggi come la tokenizzazione e l'indicizzazione delle parole. La classe LSTM_Text_Classifier estende le classi BaseEstimator e ClassifierMixin da Scikitlearn, consentendogli di comportarsi come uno stimatore Scikitlearn.

- **Dense**: Uno strato denso è lo strato più utilizzato, che è fondamentalmente uno strato in cui ogni neurone riceve input da tutti i neuroni nello strato precedente, quindi "densamente connesso". I livelli densi migliorano la precisione complessiva e 5-10 unità o nodi per livello sono una buona base. Quindi la forma di output dello strato denso finale sarà influenzata dal numero di neuroni/unità specificati.

- **dropout**: Ogni strato LSTM dovrebbe essere accompagnato da uno strato di dropout. Tale strato aiuta a evitare l'overfitting durante l'allenamento bypassando i neuroni selezionati casualmente, riducendo così la sensibilità ai pesi specifici dei singoli neuroni. Sebbene i livelli di eliminazione possono essere utilizzati con i livelli di input, non dovrebbero essere utilizzati con i livelli di output poiché ciò potrebbe alterare l'output del modello e il calcolo dell'errore. Sebbene l'aggiunta di maggiore complessità possa rischiare un overfitting (aumentando i nodi negli strati densi o aggiungendo un numero maggiore di strati densi e avendo una scarsa precisione di convalida), questo può essere affrontato aggiungendo dropout. Un buon punto di partenza è il 20%, ma il valore di abbandono dovrebbe essere mantenuto basso (fino al 50%). Il valore del 20% è

ampiamente accettato come il miglior compromesso tra la prevenzione dell'overfitting del modello e il mantenimento dell'accuratezza del modello.

- **layering rate**: Questo iperparametro definisce la velocità con cui la rete aggiorna i propri parametri. L'impostazione di un tasso di apprendimento più elevato accelera l'apprendimento ma il modello potrebbe non convergere (uno stato durante l'allenamento in cui la perdita si assesta all'interno di un intervallo di errore attorno al valore finale) o addirittura divergere. Al contrario, un tasso più basso rallenterà drasticamente l'apprendimento poiché i passaggi verso la funzione di minima perdita saranno minimi, ma consentirà al modello di convergere senza intoppi. Di solito si preferisce un tasso di apprendimento in decrescita e questo iperparametro viene utilizzato nella fase di addestramento e ha un piccolo valore positivo, per lo più compreso tra 0,0 e 0,1.

- **weight**: Idealmente, è meglio utilizzare diversi schemi di inizializzazione del peso in base alla funzione di attivazione utilizzata. Tuttavia, più comunemente viene utilizzata una distribuzione uniforme mentre si scelgono i valori di peso iniziali. Non è possibile impostare tutti i pesi a 0,0 poiché l'asimmetria nel gradiente di errore è evidenziata dall'algoritmo di ottimizzazione; per iniziare a cercare in modo efficace. Diversi set di pesi determinano diversi punti di partenza del processo di ottimizzazione, portando potenzialmente a diversi set finali con diverse caratteristiche prestazionali. I pesi dovrebbero infine essere inizializzati casualmente su numeri piccoli (un'aspettativa dell'algoritmo di ottimizzazione stocastica, altrimenti noto come discesa del gradiente stocastico) per sfruttare la casualità nel processo di ricerca.

- **Epochs**: Questo iperparametro imposta quante iterazioni complete del set di dati devono essere eseguite. Sebbene in teoria questo numero possa essere impostato su un valore intero compreso tra uno e infinito, questo dovrebbe essere aumentato fino a quando l'accuratezza della convalida inizia a diminuire anche se aumenta la precisione dell'allenamento (e quindi rischiando un overfitting). Ho provato l'algoritmo LSTM con diverse epoche dalle 10 alle 100 per poi stabilirle a 20 perchè i risultati si stabilizzano lì. Analogamente anche per GRU però ho lasciato le 100 epoche perchè i risultati continuavano a migliorare.

Metodo __init__: crea il tokenizer per il modello e salva tutti i parametri

- **embedding_vector_length**: la lunghezza dei vettori di parole che verranno appresi dal livello di incorporamento

- **max_seq_length**: la sequenza di parole più lunga che verrà presa in considerazione dal classificatore (es. 500 parole)
- **lstm_layers**: un elenco con il numero di LSTM in ogni livello ricorrente
- **batch_size**: la dimensione del batch utilizzata per addestrare il modello
- **num_epochs**: il numero massimo di epoche per cui allenarsi
- **use_hash** : se usare o meno il trucco dell'hashing per l'indicizzazione delle parole
- **dropout**: il tasso di dropout utilizzato nei livelli di dropout del modello
- **conv_params**: un dizionario con parametri per la parte convoluzionale del modello

```
class LSTM_Text_Classifier(BaseEstimator, ClassifierMixin):
    def __init__(self, embedding_vector_length, max_seq_length, lstm_layers,
batch_size=32, num_epochs=3, use_hash=False,
                 dropout=None, conv_params=None):
        self.embedding_vector_length = embedding_vector_length
        self.max_seq_length = max_seq_length
        self.lstm_layer_sizes = lstm_layers
        self.num_epochs = num_epochs
        self.batch_size = batch_size
        self.use_hashing_trick = use_hash
        if not self.use_hashing_trick:
            self.tokenizer = Tokenizer()
        self.dropout = dropout
        self.conv_params = conv_params
```

Questa funzione è analoga alla funzione fit Scikit-learn utilizzata per la sua API di stima.

- X (simile ad un array) - caratteristiche (dati di testo)
- y (simile ad una matrice) - target (etichette di classe)
- validation_data - una tupla con le caratteristiche e gli obiettivi di convalida

```
def fit(self, X, y, validation_data):

    all_X = pd.concat([X, validation_data[0]])
    if self.use_hashing_trick:
        all_words = set()
        for text in all_X:
```

```
                new_words = set(text_to_word_sequence(text))
                all_words = all_words.union(new_words)
            self.max_vocab = len(all_words)*1.3

            for i in range(len(X)):
                X[i] = hashing_trick(X[i], self.max_vocab, hash_function='md5')
            X_pad = sequence.pad_sequences(X, maxlen=self.max_seq_length)

            X_valid = validation_data[0]

            for i in range(len(X_valid)):
                X_valid[i] = hashing_trick(X_valid[i],self.max_vocab,
hash_function='md5')
            X_valid_pad = sequence.pad_sequences(X_valid, maxlen=self.
max_seq_length)

            y_valid = validation_data[1]

        else:
            print('Fitting Tokenizer...')
            self.tokenizer.fit_on_texts(all_X)
            self.max_vocab = len(self.tokenizer.word_index) + 20
            X = X.apply(self._text_to_int_sequence)
            X_pad = sequence.pad_sequences(X, maxlen=self.max_seq_length)

            X_valid = validation_data[0].apply(self._text_to_int_sequence)
            X_valid_pad = sequence.pad_sequences(X_valid, maxlen=self.
max_seq_length)

            y_valid = validation_data[1]

        self.model = Sequential()
        self.model.add(Embedding(self.max_vocab, self.embedding_vector_length,
input_length=self.max_seq_length))

        if self.conv_params is not None:
            use_pooling = False
            if self.conv_params['pool_size'] is not None:
                use_pooling = True

            for i in range(self.conv_params['n_layers']):
                self.model.add(Conv1D(filters=2*(i+1)*self.conv_params['filters'
```

```
    ],
                                        kernel_size=self.conv_params['kernel_size'
    ],
                                        padding='same', activation='relu'))
                if use_pooling:
                    self.model.add(MaxPooling1D(pool_size=self.conv_params['
    pool_size']))


        if len(self.lstm_layer_sizes) > 1:
            for lstm_layer_size in self.lstm_layer_sizes[:-1]:
                self.model.add(LSTM(lstm_layer_size, return_sequences=True))
                self.model.add(Dropout(self.dropout))
            self.model.add(LSTM(self.lstm_layer_sizes[-1]))
        else:
            self.model.add(LSTM(self.lstm_layer_sizes[0]))
        if self.dropout is not None:
            self.model.add(Dropout(self.dropout))
        self.model.add(Dense(1, activation='sigmoid'))
        self.model.compile(loss='binary_crossentropy', optimizer='adam', metrics
    =['accuracy'])
        early_stopping = EarlyStopping(monitor='val_accuracy',
                               min_delta=0,
                               patience=1,
                               verbose=2, mode='max')
        checkpoint = ModelCheckpoint(filepath='best_model',
                                     monitor='val_accuracy',
                                     mode='max',
                                     save_best_only=True)
        callbacks_list = [early_stopping, checkpoint]
        print(self.model.summary())
        print('Fitting model...')
        self.model.fit(X_pad, y, validation_data=(X_valid_pad, y_valid),
                  epochs=self.num_epochs, batch_size=self.batch_size)
```

Questa funzione è analoga alla funzione di previsione Scikit-learn utilizzata per la sua API di stima. Prima preelabora il testo data e lo converte in una sequenza intera.X (simile a una matrice) - inserisci dati di testo

```
    def predict(self, X):
        if type(X) == pd.core.frame.DataFrame or type(X) == pd.core.series.Series
    :
```

```
            X = X.apply(self._text_to_int_sequence)
            X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
            return self.model.predict(X)
        elif type(X) == str:
            X = self._text_to_int_sequence(X)
            X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
            return self.model.predict(X)
        else:
            X = map(X, self._text_to_int_sequence)
            X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
            return self.model.predict(X)
```

Questa funzione è un wrapper della funzione load_model Keras

```
    def load_model(self, file_path):
        self.model = load_model(file_path)
```

Questa funzione è un wrapper della funzione punteggio Scikit-learn per la sua API di stima.

```
    def score(self, X, y):
        pred = self.predict(X)
        return accuracy_score(y, pred)
```

Usando questa classe, ho creato un modello con i seguenti componenti nel codice seguente:

- Un livello di incorporamento di parole che apprende un vettore embedding a 64 dimensioni per ogni parola e aggrega i vettori dalle prime 512 parole di un articolo per generare una matrice di embedding di 512 x 64 per ogni articolo di input.
- Tre strati convoluzionali con 128 filtri convoluzionali e una dimensione del kernel di 5, ciascuno seguito da uno strato di pooling massimo .
- Due strati LSTM con 128 neuroni, ciascuno seguito da uno strato di abbandono con un tasso di dropout del 20%.
- Un livello completamente connesso all'estremità della rete con un'attivazione sigmoidea, che emette un singolo valore compreso tra 0 e 1 e indica la probabilità che un articolo sia una vera notizia .

```
    lstm_classifier = LSTM_Text_Classifier(embedding_vector_length=64,
    max_seq_length=512, dropout=0.2, lstm_layers=[128, 128], batch_size=32,
    num_epochs=20, use_hash=False,
```

```
    conv_params={'filters': 128,
                 'kernel_size': 6,
                 'pool_size': 2,
                 'n_layers': 3})
    y = df["label"].values
    y = np.asarray(y).astype("float64")
```

Per valutare efficacemente le prestazioni di questo modello, è necessario suddividere i dati in set di training, validazione e test separati. In base al codice riportato di seguito, il 30% dei dati viene utilizzato per i test e del restante 70%, il 14% (20% di 70) viene utilizzato per la convalida e il restante 56% viene utilizzato per l'addestramento . Formazione modello Dopo aver definito questo modello complesso, sono stato in grado di addestrarlo sul set di addestramento monitorando le sue prestazioni sul set di convalida. Il modello è stato addestrato per 80 epoche e ha raggiunto le sue massime prestazioni di convalida alla fine della ventesima epoca di addestramento in base al codice e all'output di seguito.

```
    X_train, X_test, y_train, y_test = train_test_split(df.text, y, test_size
    =0.3, random_state=42)
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
    test_size=0.2, random_state=42)
    lstm_classifier.fit(X_train, y_train, validation_data=(X_valid, y_valid))
```

3.4.2 Long-Short Term Memory Autoencoder

In questa sezione, costruiremo una rete LSTM Autoencoder e ne visualizzeremo l'architettura e il flusso di dati.
Un UDF per convertire i dati di input in array 3-D come richiesto per la rete LSTM.

```
def temporalize(X, y, lookback):
    output_X = []
    output_y = []
    for i in range(len(X)-lookback-1):
        t = []
        for j in range(1,lookback+1):
            # Raccoglie i record passati fino al periodo di ricerca
            t.append(X[[(i+j+1)], :])
        output_X.append(t)
        output_y.append(y[i+lookback+1])
    return output_X, output_y

 # definisce le serie temporali di input serie
```

```
timeseries =  cvec_counts.transpose()

timesteps = timeseries.shape[0]
n_features = timeseries.shape[1]
```

Come richiesto per le reti LSTM, è necessario rimodellare un dato di input in n_samples x timesteps x n_features .

```
    timesteps = 1
    X, y = temporalize(X = transformed_weights.toarray(),y = np.zeros(len(
    transformed_weights.toarray())), lookback = timesteps)

    n_features = transformed_weights.shape[1]
    X = np.array(X)
    X = X.reshape(X.shape[0], timesteps, n_features)
```

Definizione del modello autoencoder.

```
    model = Sequential()
    model.add(LSTM(128, activation='relu', input_shape=(timesteps,n_features),
    return_sequences=True))
    model.add(LSTM(64, activation='relu', return_sequences=False))
    model.add(RepeatVector(timesteps))
    model.add(LSTM(64, activation='relu', return_sequences=True))
    model.add(LSTM(128, activation='relu', return_sequences=True))
    model.add(TimeDistributed(Dense(n_features)))
    model.compile(optimizer='adam', loss='mse')
    model.summary()
    from sklearn.model_selection import train_test_split
    X_train , X_test , y_train , y_test = train_test_split ( X , y , test_size =
    0.2 , random_state = 1 )
    model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=0)
```

3.4.3 Gated Recurrent Unit(GRU)

L'implementazione del GRU rimane simile a quella del LSTM, vado a modificare il model del problema e le percentuali del testing.Inoltre aumento sensibilmente il numero di epoche portando dei risultati poco più vantaggiosi.

```
    model = keras.Sequential()
    model.add(layers.GRU(32, input_shape=(28, 28)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(10))
    model.add(layers.Dropout(0.3))
```

```
print(model.summary())
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train/255.0, x_test/255.0
x_validate, y_validate = x_test[:-10], y_test[:-10]
x_test, y_test = x_test[-10:], y_test[-10:]
model.fit(x_train, y_train, validation_data=(x_validate, y_validate),
batch_size=32, epochs=100)
```

3.4.4 Multi-layer Perceptron(MLP)

L'implementazione di MLP è possibile effettuarla anche con la libreria sklearn. Utilizzando i seguenti iperparametri:

- **early_stopping**: l'interruzione anticipata per interrompere la formazione quando il punteggio di convalida non migliora. impostato su true, metterà da parte automaticamente il 10% dei dati di addestramento come convalida e terminerà l'addestramento quando il punteggio di convalida non migliora almeno di tol per n_iter_no_change epoche consecutive.
- **max_iter**: Numero massimo di iterazioni.
- **activation**: la funzione dell'unità lineare rettificata, restituisce f(x) = max(0, x)
- **learning_rate**: tasso di apprendimento per gli aggiornamenti del peso.'constant' è una velocità di apprendimento costante data da 'learning_rate_init'.
- **solver** : il risolutore per l'ottimizzazione del peso.
- **hidden_layer_sizes**: L'i-esimo elemento rappresenta il numero di neuroni nell'i-esimo strato nascosto.

```
print_metrics(confusion_matrix(df.label, cross_val_predict(MLPClassifier(
early_stopping = True, max_iter=100, activation = 'relu',learning_rate = '
constant',solver = 'adam' ,hidden_layer_sizes = (10,),),x.toarray(), df.label
, cv=20)))
```

CAPITOLO 4

Analisi dei Risultati

All'interno di questo capitolo si discute dei risultati ottenuti da: TF-IDF, dagli algoritmi di machine learning e deep learning. Quindi successivamente si prosegue con il discutere le differenze e le similitudini tra i risultati.

4.1 TF-IDF: peso delle parole per le notizie reali e false

In questa sezione mostrerò i risultati ottenuti dal TF-IDF, considerando il rilevanza che hanno delle parole sulle notizie false e quali sulle notizie reali.

4.1.1 Notizie reali

Nel grafico si riportano le 20 parole che hanno un peso maggiore e il peso relativo.

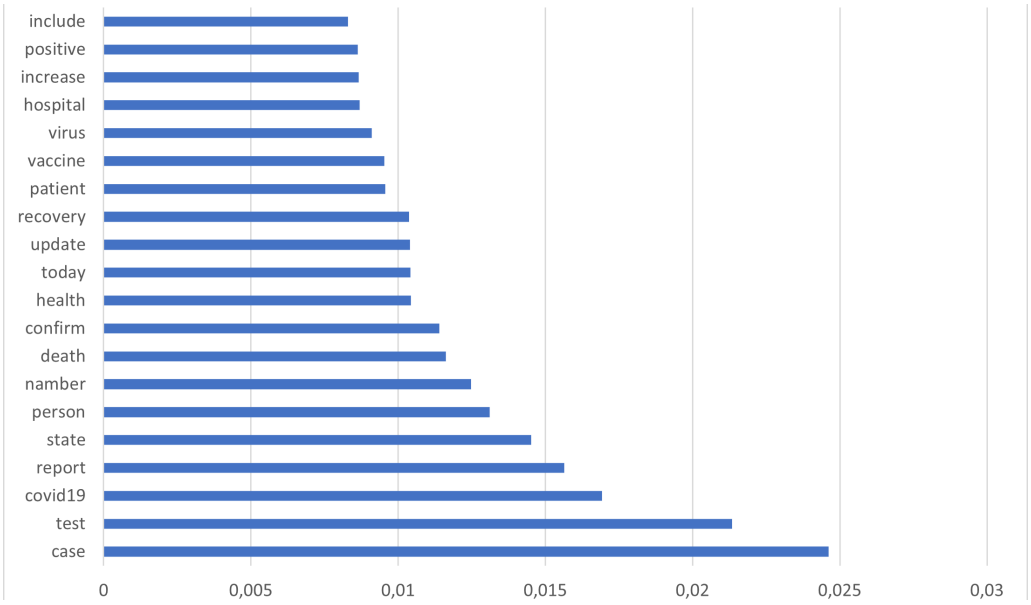

Figura 4.1: Risultati del tf-idf per notizie reali

4.1.2 Notizie false

Nel grafico si riportano le 20 parole che hanno un peso maggiore e il peso relativo.

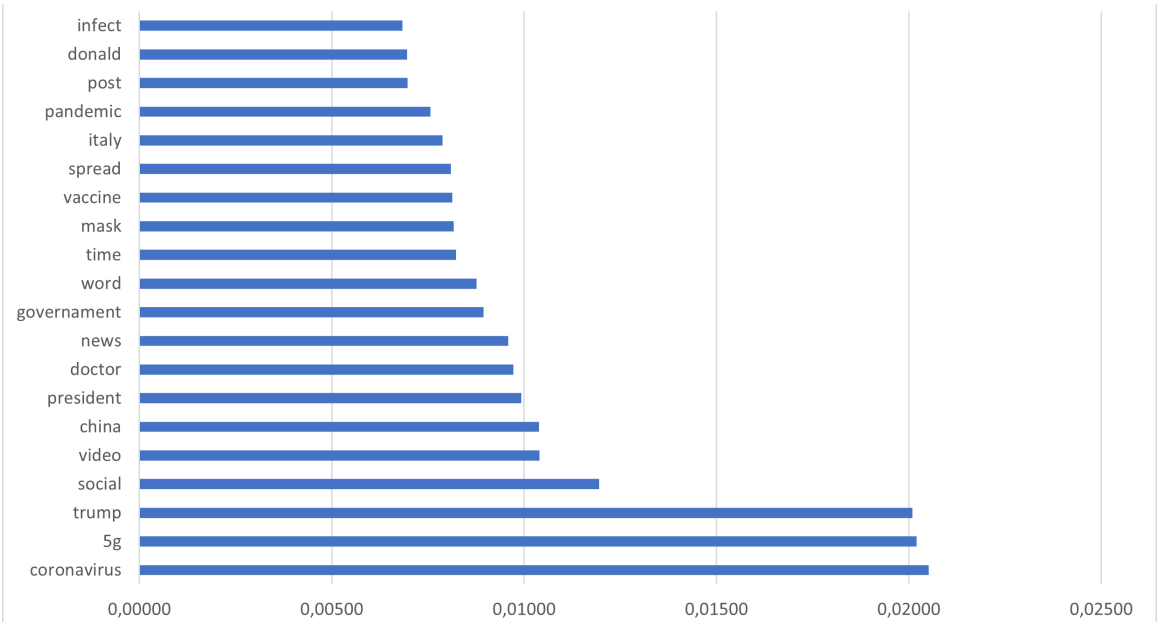

Figura 4.2: Risultati del tf-idf per notizie false

Da quello che si evince dai grafici riportati in precedenza, che le fake news seguono un pattern, ossia quello che le notizie false provengono dai social sfruttando la disinformazione prodotta anche su tematiche scientifiche. Dato che questa caratterista è ben delineata dalla frequenza delle parole del tf-idf per tanto, gli algoritmi descritti in seguito, riportano delle

buone metriche.

4.2 Algoritmi di Machine Learning

Gli algoritmi di Machine learning hanno riportato i seguenti risultati:

	Precision	Accuracy	Recall
Gaussian	0.9031	0.8785	0.8588
Decision Tree	0.8220	0.8243	0.8459
Random Forest	0.9030	0.8926	0.8894
SVM	0.9394	0.9124	0.8957
PassiveAggressive	0.9150	0.9018	0.8947

Tabella 4.1: Metriche degli algoritmi di Machine learning

Di seguito una rappresentazione grafica che illustra i dati precedenti:

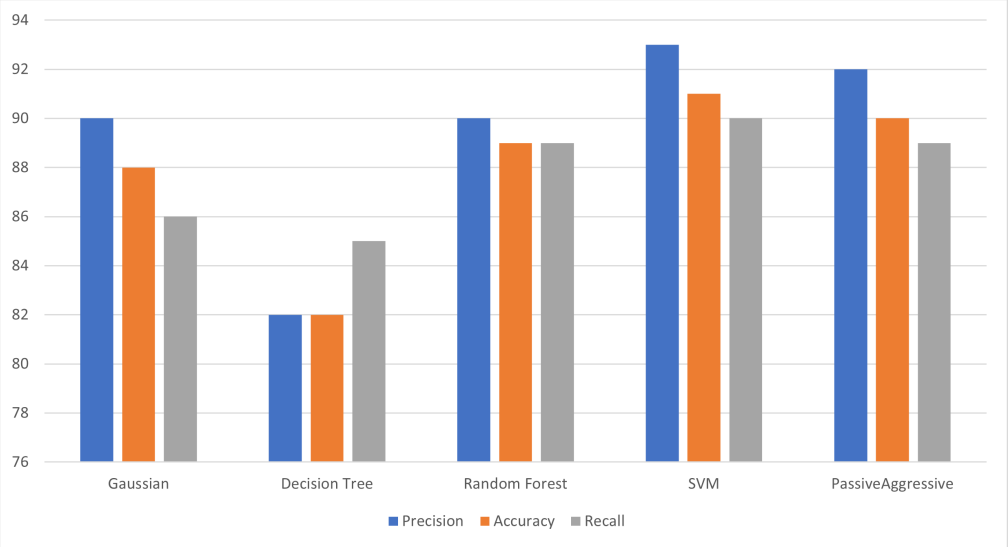

Figura 4.3: Risultati degli algotitmi di Machine Learning

4.2.1 Matrice di confusione

	True positive	False positive	False negative	True negative
Gaussian	3255	349	535	3137
Decision Tree	3206	694	584	2792
Random Forest	3371	362	419	3124
SVM	3395	242	395	3244
PassiveAggressive	3391	315	399	3171

Tabella 4.2: Matrice di confusione degli algortmi di Machine learning

4.2.2 Tempo di Addestramento

Algoritmi	Tempo(s)
Gaussian	9
Decision Tree	480
Random Forest	420
SVM	1084
PassiveAggressive	65

Tabella 4.3: Tempo di addestramento degli algortmi di Machine learning

Da quello che si evince dai risultati, rispecchia quanto detto nello stato dell'Arte [8]. Dato che le percentuali di accuratezza e precisione riportati dall'algoritmo di Decision Tree e Random Forest, restano in proporzione simili. Inoltre Random Forest mostra un numero maggiore di True positive e True negative rispetto al Decision Tree nella matrice di confusione. Ciò comunque non basta per decretare il Random Forest,prestazionalmente, migliore di SVM su questo dataset.È perché in questo dataset i dati sono meno di 10000 e facili da classificare, quindi SVM funziona più velocemente e fornisce risultati migliori. Però c'è da considerare che la complessità computazionale delle Support Vector Machines (SVM) è più alta di quella delle Random Forests (RF). Ciò significa che l'addestramento di una SVM sarà più lungo da addestrare rispetto a una RF quando la dimensione dei dati di addestramento è maggiore(come mostrato nella tabella del tempo di addestramento). Questo deve essere considerato quando si sceglie l'algoritmo. Pertanto, le Random Forest dovrebbero essere preferite quando il set di dati aumenta; però per il nostro dataset va più che bene.

Nella sezione successiva verrà descritto con precisione la complessità computazione dei due algoritmi presi in esame.

4.2.3 Complessità computazionale dei modelli ML

La complessità temporale può essere vista come la misura della velocità o della lentezza con cui un algoritmo si esibirà per la dimensione dell'input. La complessità temporale è sempre data rispetto a una certa dimensione di input (diciamo n). La complessità dello spazio può essere vista come la quantità di memoria aggiuntiva necessaria per eseguire l'algoritmo. Come la complessità temporale, è data anche rispetto ad alcune dimensioni di input (n). La complessità di un algoritmo/modello è spesso espressa utilizzando la Big O Notation, che definisce un limite superiore di un algoritmo, delimita una funzione solo dall'alto.[12] Il grafico sottostante visualizza diversi casi di complessità per gli algoritmi.
Per scrivere la complessità di calcolo assumiamo come:

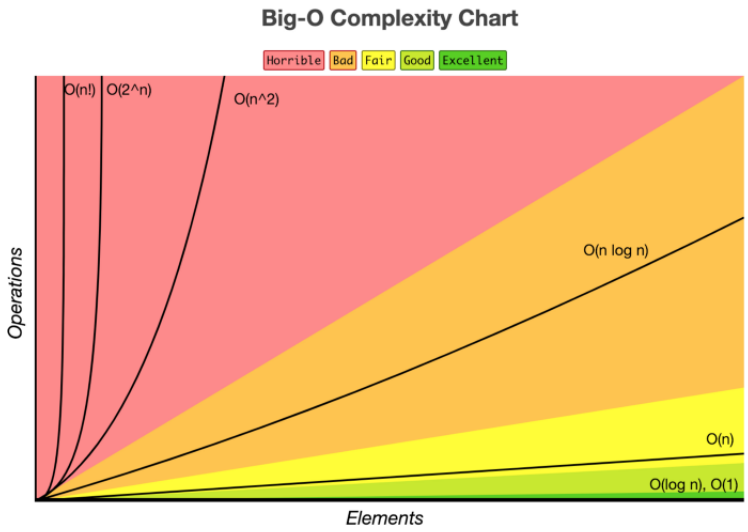

Figura 4.4: Complessità

n= numero di esempi di addestramento, d= numero di dimensioni dei dati,
k= numero di vicini

Complessità SVM

Complessità del tempo di addestramento =$O(n^2)$
Complessità run-time = O(k*d)
K= numero di vettori di supporto,d=dimensionalità dei dati

La complessità di Random Forest

Complessità del tempo di addestramento =$O(n * log(n) * d * k)$

k=numero di alberi decisionali

Complessità di runtime = O(profondità dell'albero* k)

Complessità spaziale = O(profondità dell'albero *k)

4.3 Algoritmi di Deep Learning

Gli algoritmi di Deep learning hanno riportato i seguenti risultati:

	Precision	Accuracy	Recall
MLP	0.9262	0.9128	0.9047
GRU	0.9858	0.9762	0.9701
LSTM	0.9098	0.9114	0.9203
LSTM Autoencoder	0.9139	0.9167	0.9011

Tabella 4.4: Metriche degli algoritmi di Deep learning

Di seguito una rappresentazione grafica che illustra i dati precedenti:

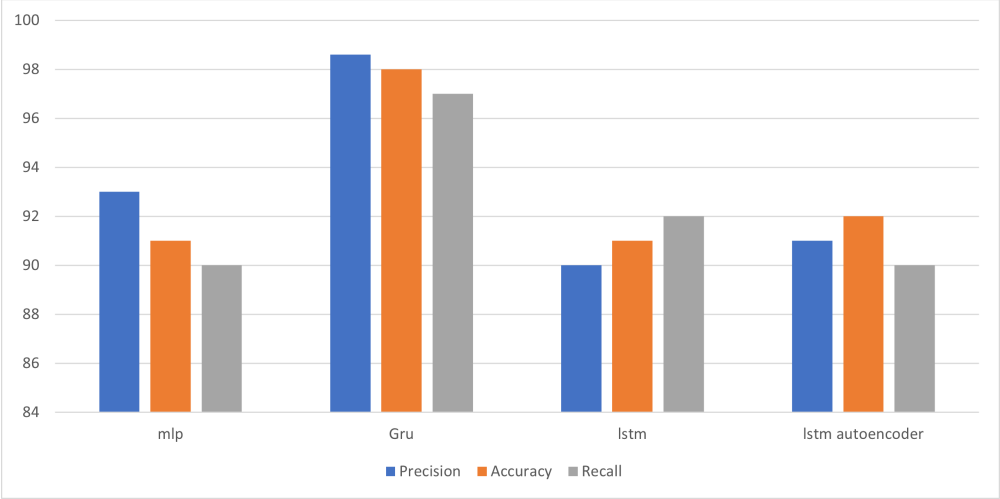

Figura 4.5: Risultati degli algotitmi di Deep Learning

4.3.1 Matrice di confusione

	True positive	False positive	False negative	True negative
MLP	3429	273	361	3213
GRU	3615	98	54	3807
LSTM	3388	265	435	3173
LSTM Autoencoder	3401	254	412	3196

Tabella 4.5: Matrice di confusione degli algortmi di Deep learning

4.3.2 Tempo di Addestramento

Algoritmi	Tempo(s)
MLP	592
GRU	2124
LSTM	16200
LSTM Autoencoder	12134

Tabella 4.6: Tempo di addestramento degli algortmi di Deep learning

Da quello che si evince dai risultati,l'algoritmo GRU è significativamente il migliore tra i quattro. Confrontando invece LSTM e LSTM autoencoder: l'autoencoder riduce la dimensionalità mantenendo quante più informazioni possibili. Quindi il modello di rete neurale estrae quante più informazioni rilevanti possibili. Il risultato è essenzialmente il trasferimento dell'apprendimento. Il problema di LSTM, tuttavia, è che i livelli di codifica cercano di preservare la quantità di informazioni piuttosto che la qualità delle informazioni; ciò denota le prestazioni lievemente inferiori di LSTM. Invece, confrontando LSTM e GRU dato che i metodi di formazione per queste reti sono gli stessi. Però le differenze dovute al numero di porte (GRU ha due porte, i due LSTM hanno tre porte); GRU non possiede alcuna memoria interna, non ha una porta di uscita che invece è presente in LSTM. In LSTM la porta di ingresso e la porta di destinazione sono accoppiate da una porta di aggiornamento e in GRU una porta di ripristino viene applicata direttamente allo stato nascosto precedente. In LSTM la responsabilità del reset del gate è assunta dalle due porte, cioè input e target. GRU utilizza meno parametri di addestramento e quindi utilizza meno memoria ed esegue più velocemente di LSTM mentre LSTM è più accurato su un set di dati più ampio. Si

può scegliere LSTM se si ha a che fare con sequenze di grandi dimensioni e la precisione è preoccupata, GRU viene utilizzato quando si ha un consumo di memoria inferiore e si desiderano risultati più rapidi. Inoltre data la compattezza della struttura dell'algoritmo GRU mi ha permesso di addestrare l'algoritmo su 100 epoche,oltre al fatto che ha mostrato dei miglioramenti in queste epoche e che i tempi di addestramento restano ragionevoli a differenza LSTM. Ciò denota le prestazioni ottimali di GRU anche per LSTM autoencoder che risultano più complessi da utilizzare su dati di tipo testo e computazionale meno efficienti.

CAPITOLO 5

Conclusioni

Dal lavoro svolto e dai risultati analizzati, risulta che l'idea di utilizzare algoritmi di machine learning e deep learning porta sicuramente allo sviluppo di una tecnica in grado di classificare le fake news in modo accurato. Infatti, la presenza dei due dataset da cui estrarre dati, aumenta la probabilità di classificare correttamente anche le notizie false provenienti da fonti diversificate.

Inoltre, le analisi effettuate nel capitolo precedente mi portano a ragionare sul SVM e Gru e su quale dei due sia migliore sul mio dataset. Arrivando alla conclusione che quando si tratta di prestazioni o precisione del modello , le reti neurali sono generalmente l'algoritmo di riferimento. Ciò è dovuto alle sue regolazioni di iperparametri come epoca, tasso di allenamento, funzione di perdita, ecc. Anche se i tempi di addestramento sono molto più lunghi le prestazioni sono migliori. Per tali motivazioni sono proteso all'algoritmo GRU.

Di conseguenza, per gli sviluppi futuri, potrebbe essere interessante sviluppare ulteriori algoritmi su tecniche differenti di data cleaning con lo scopo di verificare o contestare i risultati descritti nei capitoli precedenti.

Ringraziamenti

Dedico questo spazio del mio elaborato alle persone che hanno contribuito, con il loro supporto, alla realizzazione dello stesso. Ringrazio le mie relatrici Rita Francese e Maria Frasca, che mi hanno guidato con grande professionalità e dedizione.Inoltre anche dalle opportunità date dalla conoscenza di nuovi rami del settore che non conoscevo. Un grazie di cuore al mio collega e ormai amico NicolaPio Gagliarde , con cui ho condiviso l'intero percorso universitario. È grazie a lui che ho superato i momenti più difficili. Senza i suoi consigli, non ce l'avrei mai fatta. Ringrazio gli amici dell'Università che sono stati sempre al mio fianco. In particolare Gabriele De Guglielmo, Silvio Venturino, Catello Staiano, Eduardo Scarpa, Umberto Della Monica, Antonio Romano, Carmine Leo, Alfredo Cannavaro. Ringrazio il mio branco di amici: Federica, Marco, Jacopo, Giulia, Alessia, Petrillo, Martina, Minichiello, Aldo, O Svizzero, Tarantino, Fuschillo, Pietro, Reppucci, Christian, Belardone, Lorenzo, Gianluca. Ringrazio Luca che mi è vicino dalle elementari e anche se ora non ci vediamo più così spesso sono fiero di te e di quello che dimostri a tutti, tutti i giorni. Ringrazio la mia dolce metà, Viola, per essermi sempre accanto e sopportare i miei continui sbalzi di umore e le mie insicurezze. Sei il mio sostegno quotidiano e spero lo resterai per sempre. Ringrazio infinitamente mamma e papà che mi hanno sempre sostenuto, appoggiando ogni mia decisione, fin dalla scelta del mio percorso di studi. Ringrazio i miei fratelli Antonio e Matteo per i consigli quotidiani e le motivazioni ricevute. Ringrazio Maria che ormai è una sorella anche se non è 'componente' per le mangiate. Ringrazio nonna che c'è sempre stata, e anche se dovessi essere in una giornata no è sempre in pensiero per me e per cosa ho mangiato. Ringrazio le persone che ormai non ci sono più da tempo che mi hanno aiutato. Ringrazio anche le persone che non hanno creduto in me e mi hanno abbattuto rendendo il mio percorso più complesso ma soprattutto più formativo. Infine, dedico questa tesi a me stesso, ai miei sacrifici e alla mia tenacia che mi hanno permesso di arrivare fin qui.

Bibliografia

[1] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive aggressive algorithms. 2006. (Citato a pagina 11)

[2] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998. (Citato a pagina 13)

[3] Sherry Girgis, Eslam Amer, and Mahmoud Gadallah. Deep learning algorithms for detecting fake news in online text. In *2018 13th international conference on computer engineering and systems (ICCES)*, pages 93–97. IEEE, 2018. (Citato alle pagine vi, 21, 23 e 24)

[4] Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006. (Citato a pagina 11)

[5] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021. (Citato a pagina 5)

[6] Suleman Khan, Saqib Hakak, N Deepa, Kapal Dev, and Silvia Trelova. Detecting covid-19 related fake news using feature extraction. *Frontiers in Public Health*, page 1967, 2022. (Citato alle pagine v, 21 e 22)

[7] Shane McMahon. What does ten times ten-fold cross validation of data set mean and its importance?, 01 2015. (Citato a pagina 33)

[8] Shubha Mishra, Piyush Shukla, and Ratish Agarwal. Analyzing machine learning enabled fake news detection techniques for diversified datasets. *Wireless Communications and Mobile Computing*, 2022, 2022. (Citato a pagina 49)

[9] Verónica Pérez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news. *arXiv preprint arXiv:1708.07104*, 2017. (Citato a pagina 16)

[10] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015. (Citato a pagina 5)

[11] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004. (Citato a pagina 8)

[12] Ivor W Tsang, James T Kwok, Pak-Ming Cheung, and Nello Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(4), 2005. (Citato a pagina 50)

[13] Sander van Der Linden, Jon Roozenbeek, and Josh Compton. Inoculating against fake news about covid-19. *Frontiers in psychology*, 11:566790, 2020. (Citato a pagina 1)