Federated learning

Progetto di Machine Learning

Elisa Verza
0311317

***Abstract*—Il progetto ha lo scopo di analizzare le performance di tre reti neurali, addestrate su task di machine learning diversi (classificazione binaria, classificazione multiclasse, regressione). Lo stesso modello viene addestrato prima in un ambiente centralizzato e successivamente simulando un ambiente di machine learning federato poi vengono effettuati degli esperimenti per analizzare la variazione di prestazioni al variare di parametri quali numero di client coinvolti, strategia di aggregazione sul server, distribuzione dei dati, numero di round di addestramento.**

I. Introduzione

Sono stati scelti tre task di tipologia diversa:

- Spaceship Titanic: task di classificazione binaria. La navicella spaziale Titanic ha subito un incidente che non ha causato danni, ma alcuni passeggeri sono stati trasportati in un'altra dimensione, si tratta di capire quali sono questi passeggeri per aiutare nelle operazioni di soccorso;
- Calcolo BMI: task di regressione il cui obbiettivo è quello di predire il valore dell'indice di massa corporea;
- Problematic use of internet: task di classificazione multiclasse il cui obbiettivo è quello di predire l'indice di gravità sull'uso problematico di internet da parte di bambini ed adolescenti.

Per ogni dataset è stata effettuata una prima fase di preprocessing per effettuare: l'imputazione di eventuali valori nulli, feature engeneering, normalizzazione di dati numerici ed encoding di dati categorici. Il preprocessing è stato effettuato prima sul training set e successivamente su validation e test set utilizzando per l'imputazione dei valori nulli la media o la moda dei valori del training set e per l'encoding sono stati utilizzati gli encoder di cui era già stato fatto il fitting in fase di train.
Si passa quindi alla definizione delle reti neurali per la risoluzione del problema, in tutte e tre le reti è stata inserita una callback per l'early stopping che monitora la loss con un valore di patience di 20 ed un delta considerato di incremento di 0.0001. Il learning rate segue un exponential decay da un learning rate iniziale di 0.01 fino ad un minimo di 0.0001.
Una volta eseguito l'addestramento la rete viene addestrata nuovamente con tutti i dati di train, rieffettuando il preprocessing separato tra test e training set, il numero di epoche viene fissato al numero individuato dall'early stopping.
Infine, una volta eseguita l'evaluation, il modello, i pesi, il training ed il test set vengono salvati su file.
Per quanto riguarda l'addestramento federato vengono presi i dataset di train e test già preprocessati nella risoluzione del problema centralizzato, vengono quindi suddivisi i dati in client e convertiti in un formato adatto all'apprendimento federato. A questo punto si definisce la rete che sarà uguale a quella del caso centralizzato. Si possono quindi inizializzare client e server specificando la strategia di aggregazione.
L'addestramento viene eseguito per un numero di round in ognuno dei quali ogni client esegue un numero di epoche di addestramento prima di restituire i pesi al server che provvederà all'aggregazione. I parametri del numero di round, epoche e numero di client è customizzabile.

II. Tecnologie e Limitazioni

Le reti neurali centralizzate vengono implementate utilizzando le API Sequential di Keras. Per quanto riguarda l'addestramento federato viene utilizzato il framework di simulazione `tensorflow-federated` che mette a disposizione due set di API:

- `Federated-Learning`: sono API di più alto livello e forniscono alcune implementazioni di ML federato con strategie di aggregazione dei pesi nel server prestabilite.
- `Federated-Core`: API di più basso livello con cui poter definire il comportamento del client e del server e quindi la possibilità di definire strategie di aggregazione personalizzate

Entrambe hanno delle limitazioni, in particolare il progetto è implementato utilizzando il primo set di API che risulta più versatile per la definizione del modello. Per entrambe le tipologie di API bisogna prima implementare un normale `tf.keras.Model` che poi viene trasformato il un modello adatto all'addestramento federato. Però nel caso di `Federated-Core` è possibile ottenere un'istanza di modello federato (`tff.learning.models.FunctionalModel`) tramite una funzione, `functional_model_from_keras`, che supporta solo modelli dove il grafo computazionale generato in fase di training e testing non varia, escludendo quindi la possibilità di utilizzare livelli di batch normalization e dropout.
La funzione corrispondente delle API `Federated-Learning`, `from_keras_model` invece non presenta questa limitazione.
Per questo secondo set però non esiste un modo esplicito per definire il numero di epoche che ogni client deve effettuare ogni round, ma viene passato ad ognuno un dataset contenente i propri dati replicati un numero di volte pari alle epoche

desiderate.
In entrambi i casi è necessario passare i dati per i client in un oggetto `ClientData` ottenuto utilizzando l'API `from_client_and_tf_fn` che prende in input una lista di client id e un `tf.data.Dataset` con label e feature per ogni client. Per questo motivo i dati devono essere preprocessati prima di distribuirli tra i client.

III. Spaceship Titanic

Si tratta di un task di classificazione binaria. Il dataset è composto dai dati dei passeggeri del titanic con le seguenti features:

- PassengerId: Un ID unico che contiene il gruppo di appartenenza della persona ed il numero incrementale della persona in quel gruppo.
- HomePlanet: Il pianeta di partenza (che corrisponde al pianeta di residenza della persona)
- CryoSleep: Indica se il passeggero viaggerà o meno in stato incosciente all'interno della sua cabina
- Cabin: Indica il braccio, il numero e il lato della cabina dove alloggia il passeggero
- Destination, Name, Age: La destinazione, il nome e l'età del passeggero
- VIP: Indica se il passeggero ha pagato per servizi speciali
- RoomService, FoodCourt, ShoppingMall, Spa, VRDeck: Il totale di spesa di ogni passeggero in questo tipo di servizi.

Infine la variabile target 'Transported' specifica se il passeggero è stato trasportato in un'altra dimensione.

A. Preprocessing

Per prima cosa viene preso solo il numero del gruppo dall'ID e vengono aggiunte due colonne: Group_size per indicare il numero di appartenenti al gruppo e Solo per specificare se il passeggero viaggia in un gruppo composto da una sola persona. Dalla colonna Name viene preso solo il cognome ed utilizzato per aggiungere una colonna, Family_size che conta il numero di persone con lo stesso cognome, che quindi si suppone appartengano alla stessa famiglia. Dalle colonne RoomService, FoodCourt, ShoppingMall, Spa, VRDeck vengono estratte altre due colonne: Expenditure che contiene la somma delle spese e No_spending che indica se la persona ha effettuato o meno spese in almeno una delle categorie. Infine viene creata una colonna per inserire i passeggeri in una fascia d'età (13-17 anni calsse 1, 18-25 anni classe 2, 26-30 anni classe 3, 31-50 anni classe 4).
Vengono poi imputati i valori nulli per colonna:

- Home Planet e Destination: supponendo possano essere feature dipendenti dal gruppo di appartenenza (estratto dall'ID) ed essendo categoriche viene eseguito il test di indipendenza del chi quadro (`scipy.stats.chi2_contingency`) ottenendo dei valori di p-value inferiori a 0.05 permettendo quindi di rigettare l'ipotesi nulla della non dipendenza delle feature. Quindi i valori nulli delle colonne vengono sostituiti con la moda del gruppo di appartenenza, resta il problema di gruppi composti da una sola persona, in questo caso il valore nullo viene sostitutito con 'Unknown'
- Age: per l'età vengono distinte tre categorie di passeggeri di cui ne viene presa la medi,i passeggeri che viaggiano da soli (è più probabile che siano adulti), i passeggeri che viaggiano in famiglia e non spendono (è meno probabile che siano adulti) ed infine chi viaggia in famiglia e spende.
- Surname: sempre utilizzando il test del chi quadro si osserva la dipendenza tra gruppi e cognomi, quindi le persone della stesso gruppo con cognome nullo gli verrà assegnata la moda del gruppo, di nuovo per le persone che viaggiano sole viene assegnato cognome 'Unknown'.
- Room Service, Food Court, Shopping Mall, Spa, VR Deck: essendo tutte categorie di spese, per ogni categoria con valore nullo viene assegnata la media delle altre categorie della persona stessa.
- Cabin Deck, Number, Side: di nuovo si nota la correlazione con l'appartenenza al gruppo e come nei casi precedenti si sostituiscono i valori nulli con la moda del gruppo e con 'Unknown' nel caso di passeggeri soli.
- VIP e Cryo Sleep: non si notano particolari dipendenze per cui si imputano i valori nulli con la moda della colonna.

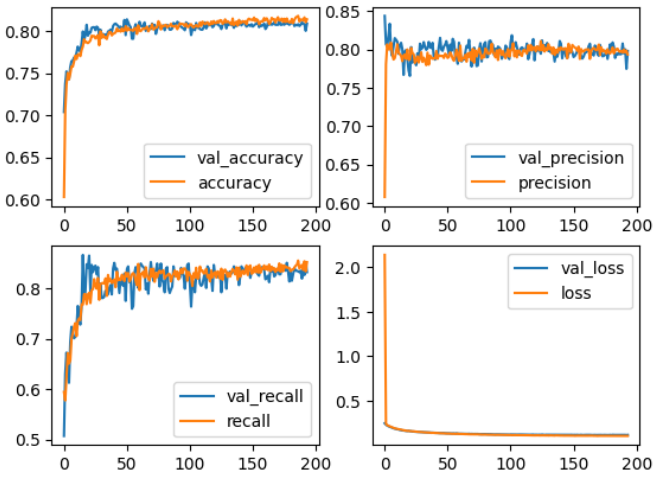
A questo punto viene effettuata la feature selection andando a vedere la correlazione delle colonne con la label dopo aver effettuato l'encoding delle colonne numeriche con lo standard scaler e delle colonne categoriche con l'ordinal encoder. Vengono eliminate tutte le colonne che hanno correlazione in modulo inferiore al 10%.
Infine viene fatto il one hot encoding delle colonne che in precedenza erano categoriche ad eccezione della colonna 'Cabin_number' che avendo molti valori diversi viene lasciata con l'encoding ordinale.

B. Modello

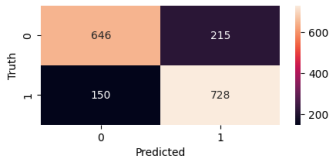
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	272
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1024)	17408
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 256)	262400
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

Total params: 313105 (1.19 MB)
Trainable params: 313105 (1.19 MB)
Non-trainable params: 0 (0.00 Byte)

Il modello inizialmente definito solo con i livelli densi tendeva ad andare in overfitting così sono stati aggiunti i livelli di dropout con probabilità 10% ed è stato aggiunto anche un kernel regularizer di tipo L2 di parametro 30e-6. Tutti i livelli hanno funzione di attivazione ReLU e kernel initializer HeUniform ad eccezione del livello di output che ha come attivazione Sigmoid e kernel initializer GlorotUniform. Infine per la loss si usa binary focal crossentropy.



Essendo il dataset bilanciato i valori di accuracy, precision e recall non presentano variazioni, notando comunque una precision sempre leggermente più bassa rispetto alle altre due indicando una tendenza a predire falsi positivi.



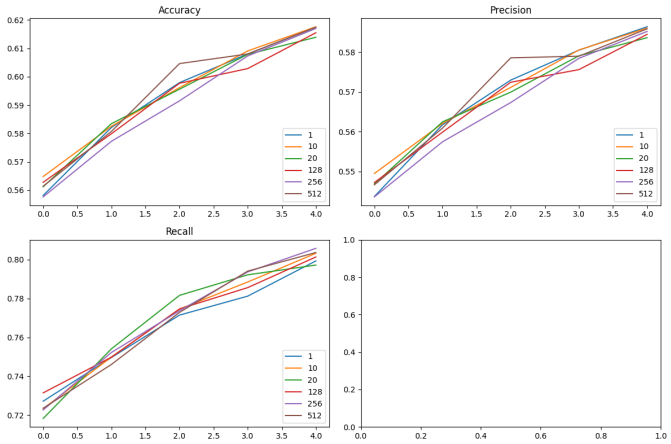
Infine le due evaluation fatte sul test avendo addestrato il modello con e senza validation presentano poche variazioni, ma comunque positive:

	Loss	Accuracy	Precision	Recall
val_ds	0.1161	0.7849	0.7571	0.8451
no val_ds	0.1110	0.7901	0.7720	0.8429

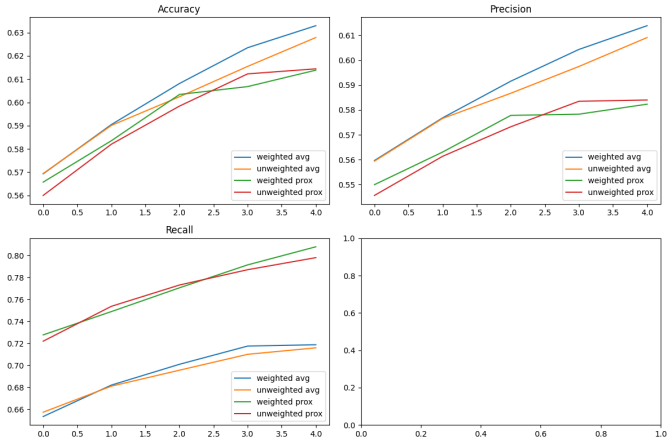
C. *Esperimenti*

Gli esperimenti eseguiti per il modello federato si possono categorizzare nel seguente modo:

1) Algoritmi di aggregazione: Il primo set di esperimenti riguarda la strategia di aggregazione dei pesi dei client da parte dei server. Sono state valutate 4 strategie, due sulla media dei pesi (pesata e non pesata sul numero di elementi del dataset per client) e due con l'algoritmo di prossimità che esegue l'aggregazione dei pesi come per la media, ma con un parametro che aggiunge un termine di regolarizzazione per evitare che i pesi si allontanino troppo dai pesi del server. Per il tuning del termine di prossimità sono state eseguite più prove con diversi valori del parametro ovvero 1, 10, 20, 128, 256, 512. Le prove sono state eseguite prendendo in considerazione 10 client per 5 round con 10 epoche per round. I client vengono selezionati tutti, quindi l'addestramento avviene su tutti i dati.



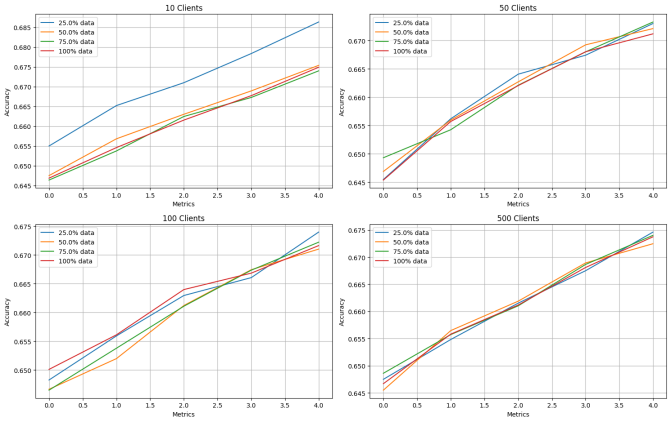
Dai grafici si vede come la variazione sia minima e soprattutto si compensa tra le metriche. Viene scelto il parametro 10 che sembra essere più stabile all'aumentare dei round ed al variare delle metriche. Viene quindi creato il dataset federato assegnando ad ogni client una riga del dataset con una probabilità che segue una distribuzione di pareto in modo da renderne i client più sbilanciati e poter testare l'importanza o meno di considerare la media pesata. Viene fissato il numero di client, epoche per client per round e round totali come in precedenza.



Da notare come il comportamento dei due tipi di algoritmo segua andamenti speculari, per quanto riguarda la media pesata e non pesata si nota un andamento migliore per la precision rispetto alla recall, viceversa per gli algoritmi di prossimità. Per la media, introdurre il bilanciamento sembra migliorare

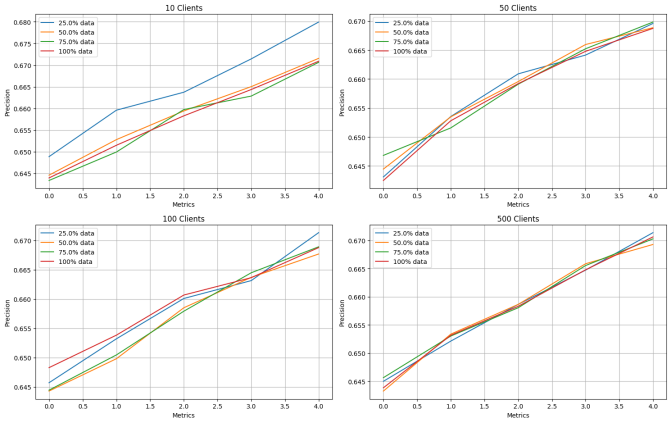
le performance, mentre per l'algoritno di prossimità sembra essere quasi uguale.

2) Numero e percentuale client: Vengono fatti una serie di esperimenti al variare del numero di client, con 10, 50, 100 e 500 client, individuando con 500 il limite massimo di client supportati a livello di risorse. Per ogni blocco di client ne viene perso il 25% 50% 75% 100% in modo tale da eseguire l'addestramento con porzioni crescenti di dati. Viene fissato il numero di epoche per client per round e numero di round come nel caso precedente. Questa volta i client vengono presi in modo bilanciato, con circa lo stesso numero di entry ciascuno e le label divise 50% di positive e 50% negative. Le epoche sono in totale 200, questo numero è stato scelto per andare quanto più vicino possibile al numero di epoche del caso centralizzato, si nota come le prestazioni restano più basse. Seguono i grafici con l'andamento delle metriche al variare dei round e delle percentuali considerate di client, seguite dalla tabella con riportati i valori delle metriche sul test set.



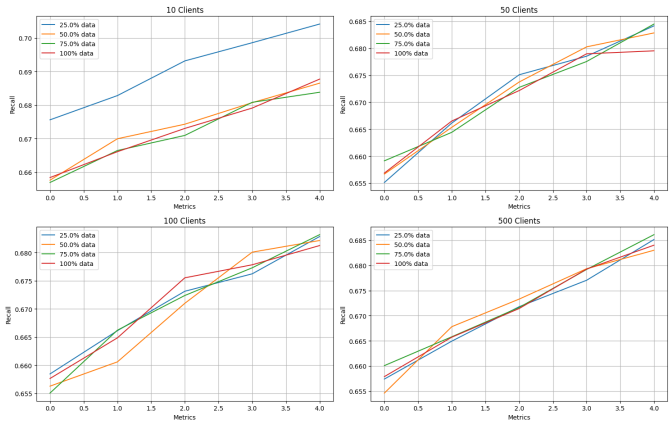
Client	0.25	0.50	0.75	1
10	0.718229	0.722829	0.721104	0.719954
50	0.719379	0.720529	0.718804	0.719379
100	0.720529	0.719954	0.718804	0.719379
500	0.718804	0.719379	0.719379	0.719954

Accuracy



Client	0.25	0.50	0.75	1
10	0.759358	0.770492	0.768076	0.765265
50	0.764228	0.762735	0.763193	0.764228
100	0.764865	0.765265	0.763908	0.764228
500	0.762483	0.764228	0.762803	0.765265

Precision

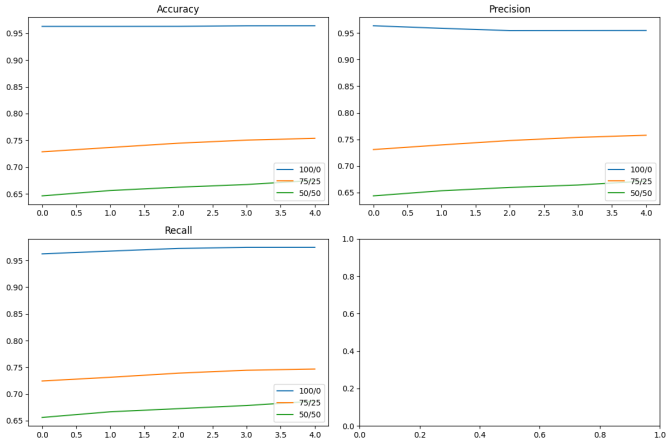


Client	0.25	0.50	0.75	1
10	0.646925	0.642369	0.641230	0.642369
50	0.642369	0.648064	0.642369	0.642369
100	0.644647	0.642369	0.641230	0.642369
500	0.643508	0.642369	0.644647	0.642369

Recall

Per quanto riguarda i risultati dell'addestramento per tutte e tre le metriche si nota un andamento migliore per il caso con 10 client di cui ne viene preso il 25%, anche se per quanto riguarda i risultati sul test set non si nota molta differenza tra le varie configurazioni. In generale per l'addestramento la metrica più alta risulta essere la recall, mentre al contrario sul test set sono più alte accuracy e precision.

3) Distribuzione label: Vengono fatti una serie di esperimenti per valutare le prestazioni al variare della percentuale che ogni client osserva delle due label. I valori degli altri parametri vengono lasciati invariati (10 client presi tutti quanti, 20 epoche per 5 round, media pesata come strategia di aggregazione).
Il primo esperimento prevede che tutti i client abbiano il 50% di label positive ed il 50% negative. Nel secondo metà dei client vedono il 75% di esempi positivi ed il 25% negativi e l'altra metà il viceversa. Nel terzo esperimento metà dei client vedono solo label positive e metà solo label negative.
Anche in questo caso si possono vedere i grafici delle tre metriche al variare dei round per le tre distribuzioni dei dati, seguiti dall tabella con la valutazione sul test set.



Client	Accuracy	Precision	Recall
100/0	0.481311	0.488785	0.595672
75/25	0.720529	0.767759	0.640091
50/50	0.719954	0.763832	0.644646

Si nota subito come l'addestramento in cui le classi sono del tutto separate tra client vada in overfitting presentando tutte e tre le metriche tra il 95% ed il 100%, ma con dei valori inferiori al 50% sul test. Sul train risulta migliore la configurazione 75%-25%, ma sul test la differenza è minima. Si nota di nuovo come sul test la recall risulti essere la metrica più bassa.

IV. Indice BMI

Il secondo task è di regressione. Anche questo task è preso da una challenge conclusa di Kaggle il cui obiettivo era quello di collocare le persone in una categoria di peso: sottopeso, normopeso, sovrappeso e tre gradi di obesità. Le categorie sono individuate per range di BMI (indice di massa corporea) per questo motivo è stato trasformato in un task di regressione con l'obiettivo di predire il valore del BMI.
Il dataset è composto da 16 colonne che riportano:

- Attributi fisici della persona: Gender, Age, Height and Weight
- Abitudini della persona: familiarità con l'obesità (family_history_with_overweight), consumo frequente di alimenti ad alto contenuto calorico (FAVC), frequenza di consumo di verdure (FCVC), Numero di pasti principali (NCP), consumo di cibo tra i pasti (CAEC), fumo (SMOKE) consumo di acqua giornaliero (CH20), consumo di alcol (CALC), monitoraggio del consumo di calorie (SCC), frequenza dell'attività fisica (FAF), tempo di utilizzo di dispositivi tecnologici (TUE), trasporto utilizzato (MTRANS)

A. Preprocessing

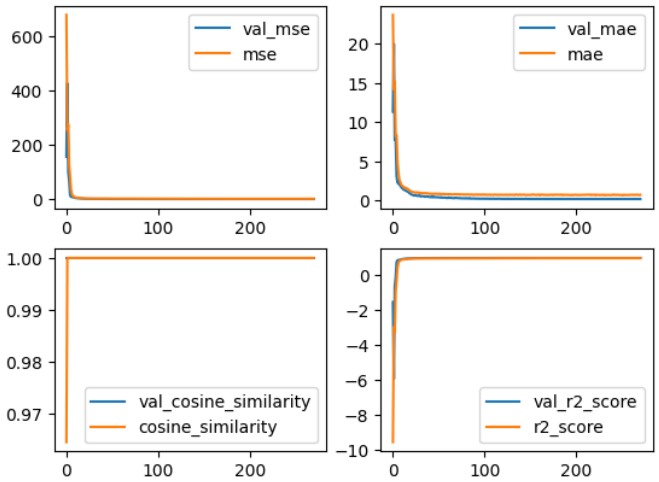
Come prima cosa viene creata la nuova colonna target con i valori del BMI al posto delle categorie, calcolata come $\frac{\text{Weight}}{\text{Height}^2}$. Il dataset non presenta valori nulli, quindi si passa direttamente al calcolo della correlazione delle feature con la label per eliminare le feature che hanno meno del 10% in modulo di correlazione. Per fare ciò si applica alle colonne numeriche lo standard scaler e alle colonne categoriche l'ordinal encoder. Infine viene fatto il one hot encoding sulle colonne che hanno al più due valori diversi.

B. Modello

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 128)	1920
dense_11 (Dense)	(None, 256)	33024
dropout_1 (Dropout)	(None, 256)	0
dense_12 (Dense)	(None, 128)	32896
dense_13 (Dense)	(None, 64)	8256
dense_14 (Dense)	(None, 1)	65

Total params: 76161 (297.50 KB)
Trainable params: 76161 (297.50 KB)
Non-trainable params: 0 (0.00 Byte)

La rete non presentava molto overfitting, è stato comunque aggiunto un livello di dropout al 10% che ha migliorato le performance. Tutti i livelli hanno kernel initializer la distribuzione normale e funzione di attivazione ReLU ad eccezione del livello di output che ha funzione di attivazione lineare. Infine come loss viene utilizzato il mean squared error. L'addestramento viene interrotto dall'early stopping tra la 250 e la 300 esima epoca.
Le metriche osservate sono il mean squared error, mean absolute error, cosine similarity e r2 score. In modo da poter osservare eventuali errori grandi e outlayer tramite MSE e MAE, mentre cosine similarity osserva la somiglianza tra output e predizione, infine R2score indica quanto il modello ha colto la varianza dei dati.



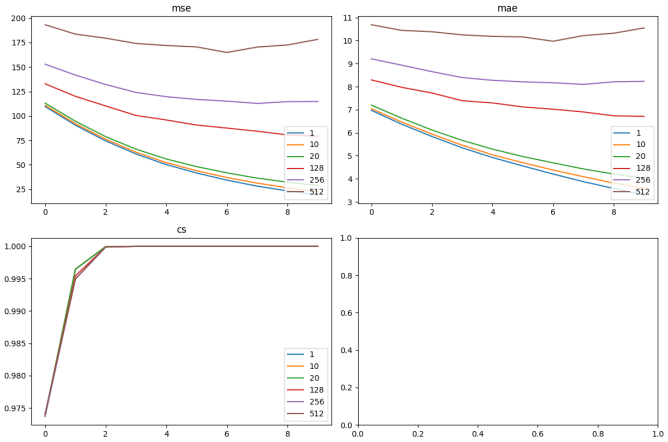
In questo caso le evaluation mostrano come riaddestrare il modello senza validation non porti miglioramenti significativi.

	MSE	MAE	Cosine Similarity	R2 score
val_ds	0.0893	0.2054	1	0.9986
no val_ds	0.1760	0.2809	1	0.9973

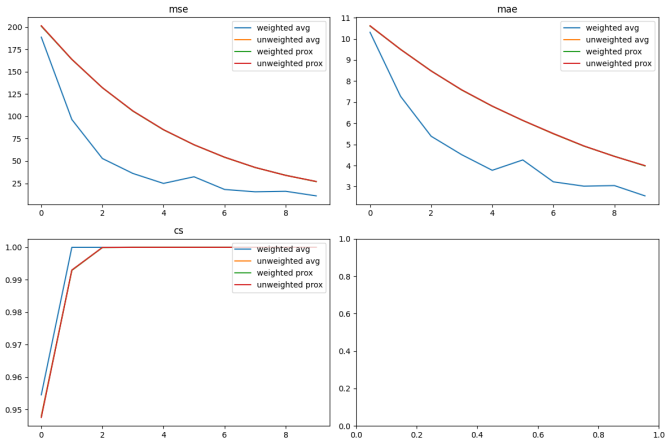
C. *Esperimenti*

Nel caso federato sono stati riscontrati problemi di compatibilità con la metrica R2Score, seguono i grafici dei risultati degli esperimenti per le metriche MSE, MAE e CosineSimilarity:

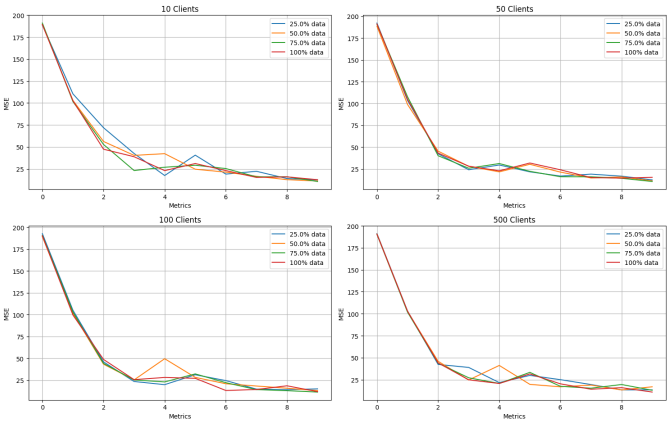
1) Algoritmi di aggregazione: Solo nel caso dell'algoritmo di aggrgazione della media pesata è stato possibile inserire exponential decay per il learning rate, le API degli altri algoritmi non ne prevedono la possibilità, si è anche notato come avere un learning rate decrescente in questa rete risulta particolarmente importante in termini di convergenza, quindi ci si aspettano anche per questo motivo migliori performance dall'algoritmo con la media pesata. In questo caso risulta evidente dai grafici come il parametro di proximal strenght migliore risulti essere 1, da notare il miglioramento al decrescere del parametro. Gli esperiementi in questione sono stati eseguiti fissando a 10 il numero di round, epoche per round e client.



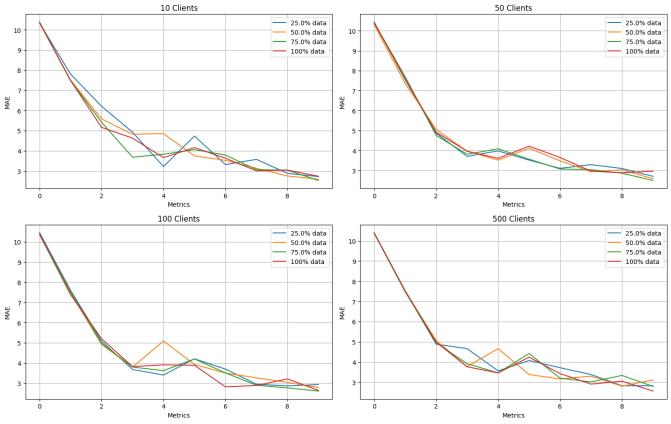
A questo punto sono stati eseguiti gli addestramenti per ogni tipo di alagoritmo. L'unico algoritmo che si discosta, raggiungendo prestazioni migliori è anche in questo caso la media pesata.



2) Numero e percentuale client: Analogamente a quanto fatto per il task 1 per questo esperimento vengono fissati solo il numero di epoche per round ed il numero di round, anche in questo caso a 10.
Dai grafici è stata esclusa la CosinSimilarity avendo un andamento uguale per tutti i casi e fisso ad 1 già a partire dal secondo round.



Client	**0.25**	**0.50**	**0.75**	**1.00**
10	1.151367	1.131969	1.027498	1.236657
50	1.105877	1.462863	0.888762	1.120466
100	1.095871	2.249199	1.088252	2.563741
500	1.278896	1.169425	2.475834	1.683926



Clienti	0.25	0.50	0.75	1.00
10	0.774175	0.728853	0.665204	0.808346
50	0.803339	0.973575	0.706844	0.765709
100	0.676860	1.276716	0.775814	1.414995
500	0.856241	0.808085	1.371117	1.079685

In generale ad eccezione del caso con 500 client sembra andare meglio il caso con il 75% di client considerati, andamento confermato anche dai risultati sul test set. Si nota anche come i valori delle metriche sul test set migliorino notevolmente rispetto a quelle rilevate alla fine del train.

V. Internet and Mental Health

Il terzo task è di classificazione multiclasse. Il dataset è stato preso da una challenge di Kaggle riguardante uno studio per la diagnosi ed il trattamento dei disturbi mentali a partire da aspetti comportamentali (dati presi da accellerometri da polso, moitoraggio di attività fisica e questionari) e da aspetti legati al comportamento sul web. Si tratta di predire il Severity Impairment Index, ovvero un indice sull'uso problematico di internet, ci sono 4 livelli di gravità.

A. Preprocessing

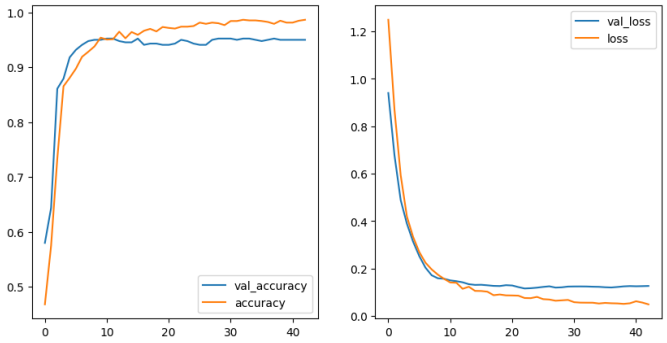
Il dataset ha un numero elevato di colonne con valori nulli presenti anche nella colonna delle label, quindi come prima cosa vengono rimossse tutte le righe con label nulla e successivamente imputati i valori nulli con la media per i valori numerici e con la moda i dati categorici.
Viene poi fatto l'encoding con standard scaler e ordinal encoder, in questo caso non viene effettuato one hot encoding di nessuna colonna.
Si effettua poi la feature selection calcolando la correlazione con la colonna delle label, anche in questo caso vengono scartate le colonne che hanno in modulo correlazione minore del 10% arrivando così ad avere 44 colonne.
Infine trattandosi di classificazione multiclasse viene datto one hot encoding delle label da passare alla rete neurale.

B. Modello

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 112)	5040
dropout (Dropout)	(None, 112)	0
dense_1 (Dense)	(None, 272)	30736
dropout_1 (Dropout)	(None, 272)	0
dense_2 (Dense)	(None, 4)	1092

Total params: 36868 (144.02 KB)
Trainable params: 36868 (144.02 KB)
Non-trainable params: 0 (0.00 Byte)

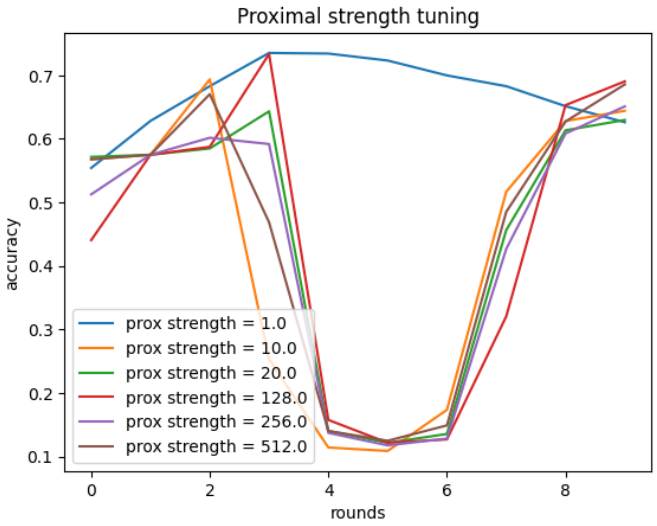
E' stato aggiunto un livello di dropout per ogni livello denso con probabilità 20%. La rete è composta da due livelli con funzione di attivazione softmax e tangente iperbolica, il livello di output ha funzione di attivazione softmax ed ha kernel initializer RandomNormal.
Come funzione di loss viene usata la categorical crossentropy, si osserva come metrica l'accuracy. Si è valutato inizialmente di monitorare anche la top k categorical accuracy, ma avendo 4 categorie il task risultava meno significativa.



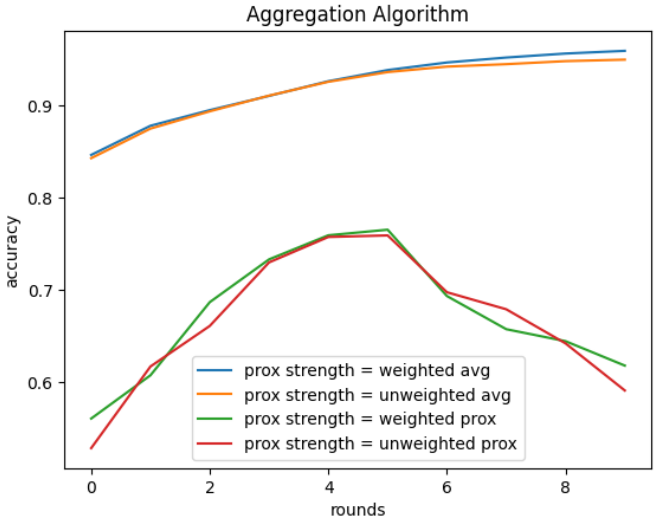
Riaddestrando la rete senza il validation set si passa da un'accuracy del 96.35% al 97.26% quindi un leggero miglioramento.

C. Esperimenti

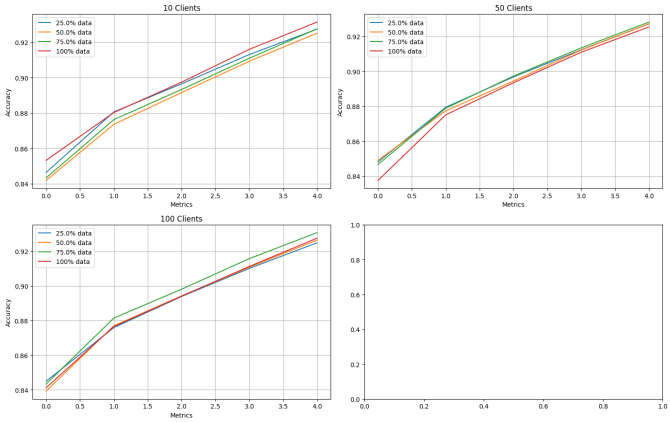
1) Algoritmi di aggregazione: Anche in questo caso è stato eseguito prima il tuning del parametro di proximal strenght, fissando a 10 il numero di round, epoche per round e client. Tutti i valori testati ad eccezione di 1 presentano una forte instabilità soprattutto tra la quarta e sesta epoca motivo per cui si è scelto di utilizzare il parametro 1 anche se al round finale aveva prestazioni inferiori.



Per quanto riguarda l'algoritmo di aggregazione si nota subito un forte divario tra le due categorie di algoritmi, mentre la differenza è inferiore nel caso si consideri o meno l'algoritmo pesato mostrando comunque un andamento migliore per questi ultimi.

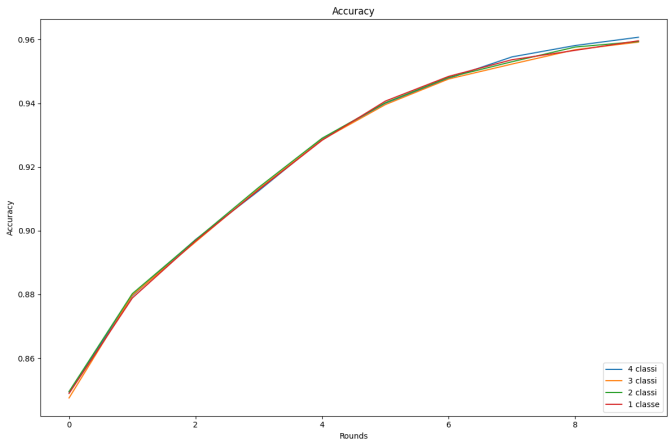


2) Numero e percentuale client: Effettuando questi esperimenti si è notato come con dataset con molte colonne le prestazioni in termini di risorse degenerino, infatti per questo task non è stato possibile effettuare l'esperimento con 500 client, infatti arrivati all'addestramento al 75% di client il sistema manda in crash il kernel.
Le performance con le varie configurazioni ricadono tutte in un intervallo molto piccolo. Nel caso di 10 client sembrerebbe essere migliore la configurazione che considera tutti i client e quindi tutti i dati, ma sul test, va meglio il caso al 50%, con 50 e 100 client va meglio il caso con il 75% dei client, ma solo con 100 client questo risultato viene confermato.



Clienti	0.25	0.50	0.75	1.00
10	0.832116	0.841241	0.808394	0.819343
50	0.833941	0.748175	0.793795	0.828467
100	0.755474	0.797445	0.819343	0.806569

3) Distribuzione label: Come ultimo esperimento sono state effettuati tre addestramenti con 4 client fissando il numero di round e di epoche per round a 10, ogni volta ogni client vede una classe in meno fino ad arrivare ad avere solo una classe per client. La suddivisione è stata fatta in modo da dividere in modo circa uguale le classi viste da più client.



Anche se i valori non si discostano molto tra loro, l'addestramento con tutte le classi risulta essere il più alto.