Progettazione ed addestramento di una Rete Neurale per Time Series Classification

Machine Learning 2021

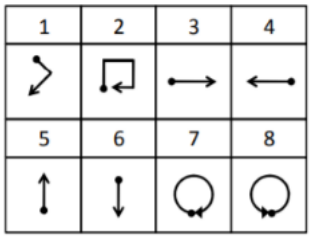
Pasquale Caporaso
studente di ingegneria informatica
Università degli Studi di Roma "Tor Vergata"
Roma, Italia
caporasopasquale97@gmail.com

Ezio Emanuele Ditella
studente di ingegneria informatica
Università degli Studi di Roma "Tor Vergata"
Roma, Italia
ezioemanuele.ditella@gmail.com

I. Introduzione

In questo elaborato si andrà a discutere di come è stato risolto il problema di **Hand Gesture Classification**, usando come input i dati storici generati dall'accelerometro a tre assi di un Wii Remote.

In particolare, lo **scopo** del progetto sarà quello di costruire un predittore che sia in grado di identificare il tipo di movimento che è stato fatto con un Wii Remote. I possibili movimenti che il classificatore deve essere in grado di riconoscere sono i seguenti



Vale la pena osservare che rispetto ad un generico problema di Machine Learning per Time Series, il nostro permette di sganciarci dalle seguenti difficoltà:

- Definizione del **lag**: non è necessario stabilire il numero di campioni che si pensa possa influenzare una predizione futura
- Definizione dell'**orizzonte di predizione**: dato che si vuole fare classificazione, non si usano tecniche di Multi-step ahead forecast e dunque non sono affrontate le problematiche da esse introdotte
- Dipendenza temporale: i campioni del dataset sono temporalmente indipendenti tra di loro, quindi non sono necessari complessi metodi di validazione come il Walk Forward

Per lo sviluppo di tale progetto si è deciso di adoperare le metodologie definite dal modello di processo **CRISP-DM** (Cross-Industry Standard Process for Data Mining), e dunque il ciclo di vita di tale progetto è stato suddiviso nelle seguenti fasi:

- *Business Understanding*: identificazione di requisiti e obiettivi di progetto
- *Data Understanding*: collezione e familiarizzazione iniziale dei dati
- *Data Preparation*: preparazione e raffinamento delle informazioni (dataset) raccolte
- *Modeling & Evaluation*: esecuzione, comparazione e valutazione degli algoritmi di machine learning

II. Business understanding

Gli obiettivi che ci siamo posti per questo studio erano principalmente due. Il primo, che ci è stato imposto come traccia per il progetto, era quello di raggiungere un'accuratezza superiore al 97,5%, il secondo, derivato dalle nostre risorse limitate, è stato quello di creare un modello che non richiedesse troppa potenza computazionale per l'addestramento. Questi obiettivi, come descritto nelle prossime sezioni, sono stati entrambi raggiunti.

III. Data understanding

La fase di Data Understanding è stata particolarmente importante per questo progetto, infatti, ci siamo accorti che all'interno del dataset, alla sua fine, erano presenti dei dati anomali.

Questi dati, come mostrato nelle figure 1 e 2, presentano un alto livello di rumore, questa interferenza è presente su tutti e tre gli assi e, ovviamente, causa un forte calo di prestazioni nella fase di addestramento.

Tuttavia, non conoscendo l'origine di questo problema, non possiamo eliminarli dal dataset in quanto potrebbero essere stati causati da una situazione che dobbiamo comunque essere in grado di identificare. Questa decisione ha sicuramente avuto un impatto sulle prestazioni finali del modello ma ci ha anche dato la possibilità di classificare dei sample più rumorosi.

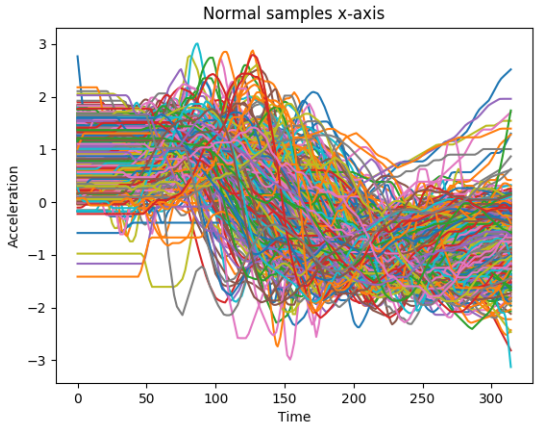


Fig. 1. Sample normali

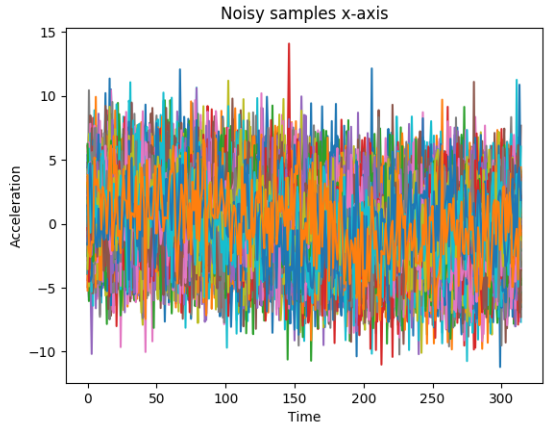


Fig. 2. Sample con rumore

IV. Data Preparation

Una delle fasi più cruciali del machine learning è partizionare e raffinare il dataset in modo da ottimizzare il processo di addestramento dei classificatori.

Il dataset fornito, come discusso precedentemente, contiene dei sample 'normali' al suo inizio, e sample 'rumorosi' verso la fine. Al fine di **distribuire** quanto più possibile tali sample tra di loro, il primissimo passo è stato quello di fare uno shuffle sul dataset iniziale.

Un altra operazione di raffinamento dei campioni è consistita nel portare tutte le (315) osservazioni temporali di tutti gli assi alla **stessa precisione** (6 cifre dopo la virgola).

Inoltre il dataset dispone di un numero molto basso di campioni (5 mila), dai quali si è dovuto estrarre una porzione da riservare per il test set (20%) su cui effettuare l'evaluation finale dei classificatori. Ciò ha ulteriormente ridotto il numero di campioni rimanenti per la fase di training, preannunciando cattive performance dei classificatori.

A. Augmentation

Per rimediare a tale problematica si è deciso di utilizzare la tecnica dell'**augmentation** (sulla porzione di dataset rimanente per il training set). Nella fattispecie sono state provate le seguenti 3 tattiche:

- **Simple**: Aggiunta/sottratta una stessa quantità random (tra 0 e 1 con probabilità 0.85 e tra 2 e 3 con probabilità 0.15) a tutte le 315 osservazioni di tutti e 3 gli assi
- **Burst**: Ad una porzione contigua delle 315 osservazioni temporali di ogni asse è stata aggiunta/sottratta una stessa quantità random
- **Gradual**: Come la tecnica **Burst**, ma l'aggiunta/sotttrazione avviene in modo graduale (eg. selezionato intervallo 100-200; aggiunta random pari a 2; aggiunta graduale da 100 a 150; da 150 a 200 ripristino graduale)

Ognuna di queste tattiche è stata associata ad un *fattore di augmentation*, D_a, in modo che per ogni istanza del dataset venissero generati **altri** D_a istanze mediante la corrispondente tecnica di augmentation. Rispetto il classificatore migliore la tattica che ha ottenuto performance migliori è stata la **Simple** con *fattore di augmentation* pari a 50. Ciò ha permesso di portare il training set da 3500 elementi a 175000.

V. Modeling and Evaluation

A. Classificatori

Durante questo progetto abbiamo testato diversi modelli, siamo partiti da quelli studiati da Fawaz e Forestier [1], ottimizzandoli sia manualemnte, sia tramite cross-validation al nostro problema. I classificatori che abbiamo testato sono i seguenti:

```
# MLP comlessa
Dense(units=1800, activation='relu')
BatchNormalization()
Dense(units=900, activation='relu')
BatchNormalization()
Dense(units=450, activation='relu')
BatchNormalization()
Dense(units=8, activation='softmax')
opt = tf.keras.optimizers.Adam()
loss='categorical_crossentropy'
metrics=['accuracy']
```

```
# MLP semplice
Dense(units=300, activation='relu')
Dense(units=150, activation='relu')
Dense(units=75, activation='relu')
Dense(units=8, activation='softmax')
opt = tf.keras.optimizers.Adam()
loss='categorical_crossentropy'
metrics=['accuracy']
```

```
# MLP super semplice
Dense(units=315*3, activation='relu')
Dense(units=8, activation='softmax')
opt = tf.keras.optimizers.Adam()
loss='categorical_crossentropy'
metrics=['accuracy']
```

```
# CNN Shallow
Conv1D(16, kernel_size=105,
activation='relu',padding="same")
MaxPooling1D(pool_size=3)
BatchNormalization()
Flatten()
Dense(units=8, activation='softmax')
opt = tf.keras.optimizers.Adam()
loss='categorical_crossentropy'
metrics=['accuracy']
```

```
# CNN
Conv1D(16, kernel_size=40,
activation='sigmoid',padding="same")
MaxPooling1D(pool_size=2)
Conv1D(32, kernel_size=40,
activation='sigmoid',padding="same")
MaxPooling1D(pool_size=2)
Flatten()
Dense(units=8, activation='softmax')
opt = tf.keras.optimizers.Adam()
loss='categorical_crossentropy'
metrics=['accuracy']
```

B. Addestramento & Validazione

Al fine di avere una valutazione quanto più possibile priva di varianza dipendente dal validation set, è stato applicato **K-Fold Cross-Validation** (con K=5 Fold). L'uso di tale tecnica di validazione ha permesso di determinare i migliori iperparametri di ogni classificatore addestrato.

In figura 3, 4 è mostrato l'andamento della funzione di loss sul validation e training set dei migliori classificatori trovati.

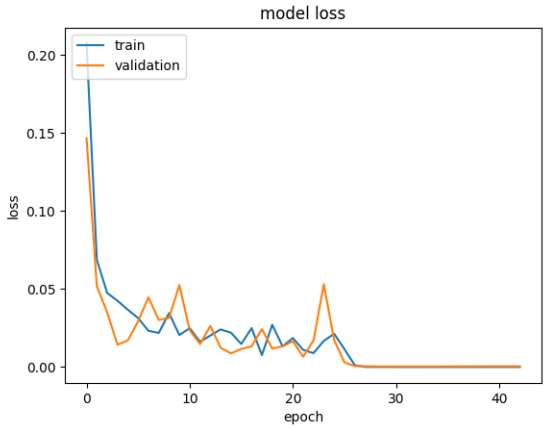


Fig. 3. Simple MLP

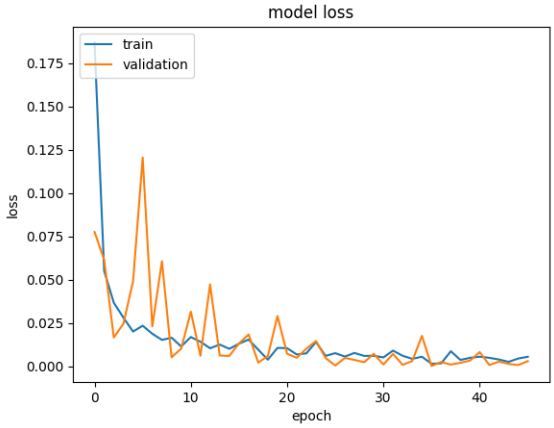


Fig. 4. Simple DNN

C. Stratification & Sampling

Per far sì che nel Training Set ci fosse una distribuzione uniforme di campioni appartenenti alla stessa classe (**dopo** aver estratto il test set e **prima** di fare data agumentation) si è deciso di applicare SMOTE (*Synthetic Minority Oversampling TEchnique*) come tecnica di *sampling*.
In particolare è stata usata la tecnica *Simple* di data agumentation per ricreare delle istanze appartenenti ad una certa classe.

Dopo ogni evaluation sul test set, si è riscontrato che un particolare sottoinsieme di movimenti è risultato *particolarmente difficile* da predirre (in particolar modo i movimenti 4, 5, 6).
Per tali classi si è aumentato il numero di istanze, usando la tecnica precedentemente descritta. **L'aumento** è stato fatto in modo **proporzionale** al numero delle classi predette in modo scorretto (eg. una classe di movimento predetta sempre correttamente non ha portato all'aumento delle istanze nel training; una classe predetta scorrettamente 10 volte ci ha spinto ad aumentare le istanze per quella classe di 100 elementi).

D. Bagging ensamble

Dopo aver definito i nostri modelli ed averli applicati al dataset augmentato ci siamo accorti che essi avevano un'alta varianza nelle prestazioni. Nel tentativo di risolvere questo problema abbiamo deciso di utilizzare una tecnica di ensamble, il bagging, questa tecnica consiste nella divisione del dataset in N parti e l'addestramento di N modelli, ognuno che studia una di queste sezioni, alla fine, per ottenere una predizione, si sommano le probabilità di tutti i modelli e viene scelta la classe con la somma più alta. Un'altra possibilità per la combinazione delle predizione è di affidare la decisione, per ogni sample, al modello che ha il più alto valore di probabilità, quest'ultima tecnica ha rivelato, in genere, delle prestazioni migliori, anche se il miglior modello prodotto funziona meglio con l'altra modalità.

Questa strategia, combinata al data augmentation e alla stratification, ha portato ad un aumento significativo dell'accuratezza.

E. Evaluation

Per valutare i notrsi modelli abbiamo deciso di utilizzare un *test set* preso casualmente (con seed costante) all'inizio dell'addestramento, ovvero prima di applicare Augmentation, K-Fold o divisioni per l'ensamble. Questo, anche se ha ridotto i dati su cui lavorare nella fase di addestramento, ci ha permesso di avere una valutazione indipendente dall'addestramento effettuato che rispecchia più accuratamente le prestazioni del modello in situazioni nuove.

VI. Conclusioni

Le prestazioni dei modelli che abbiamo testato sono le seguenti:

	Accuratezza(%)			
Modello	Normale	Augment	Ensamble	Everything
MLP molto semplice	91	96,4	96,1	97,73
MLP semplice	93	97,6	97,9	98,33
MLP complessa	96,1	95,8	96,6	97,46
CNN	95,4	97	97,3	97,46
CNN Shallow	95,6	96,7	na	na

La maggior parte dei modelli studiato si sono avvicinati al threshold di consegna ma la MLP semplice, oltre ad avere prestazioni migliori, ha anche un tempo di addestramento relativamente basso, quindi abbiamo deciso di scegliere lei come modello di consegna.

VII. Valutazione del modello con Test Set

Per valutare la bontà del miglior predittore selezionato basta seguire i seguneti passi (dalla cartella principale del progetto):

1) Inserire i file del test set nella cartella "dataset"
2) Modificare i nomi del test set **se** diversi da quelli della consegna (test_gesture_x/y/x.csv e test_label.csv) nel file "libraries/constants.py"
3) Eseguire con python3 il file "WiiMotion-TSC.py"
4) Usando il menu, selezionare in ordine le opzioni **8, 10, 5/7)**

   (*Solo nel caso in cui l'ensemble non fosse permesso*)
5) Usando il menu, selezionare in ordine le opzioni **8, 9, 6**

References

[1] Ismail Fawaz, H., Forestier, G., Weber, J. et al. *Deep learning for time series classification: a review.*. Data Min Knowl Disc 33, 917–963 (2019)