ROCKET vs Transformer

Time series Classification/Regression

Alessandro Mini Andrea Neri

Università degli Studi di Firenze

Anno accademico 2021-2022

Table of Contents

Introduzione

In questo progetto per il corso di Machine Learning si analizzano **ROCKET** e **TRANSFORMER**, due modelli per eseguire classificazione e regressione su serie temporali.

Il lavoro è organizzato come segue:

1. Introduzione al problema della TSA (Time series Analysis).
2. Introduzione e analisi di TST.
3. Introduzione e analisi di ROCKET.
4. Confronto (sperimentale) tra i due e altre due tecniche "black box": XGboost e RandomForest.

Definizione serie temporale

Una **serie storica** [10] (o temporale) è definita come un insieme di variabili casuali ordinate rispetto al tempo, ed esprime la dinamica di un certo fenomeno.

Ad una serie storica si associa un *dataset* $\mathcal{D}$ del tipo :

$$\mathcal{D} = \{x^{(i)}, y^{(i)}\}$$

con:

$$x^{(i)} = \{x_0^{(i)}(t), x_1^{(i)}(t), \ldots, x_k^{(i)}(t)\}$$

$$y^{(i)} = \{y_0^{(i)}\}$$

in cui:

1. k è il numero dei campi
2. $x^{(i)}$ è una riga della serie in ingresso
3. $y^{(i)}$ è l'output da prevedere, una classe o un valore
4. t è il tempo in cui sono stati campionati i valori

Le serie temporali possono essere:

1. **Monovariate**; hanno solo una variabile dipendente.
2. **Multivariate**: hanno più di una variabile dipendente.

Analizzare le serie temporali è un task complesso perché:

1. I metodi sono complessi.
2. Costo computazionale elevato.
3. I progetti non sfruttano efficentemente le GPU.
4. Il *labelling* è un problema complesso.
5. La maggior parte delle tecniche nascono per serie univariate.

Esempio di serie temporale

Time	Temperature	cloud cover	dew point	humidity	wind
5:00 am	59 °F	97%	51 °F	74%	8 mph SSE
6:00 am	59 °F	89%	51 °F	75%	8 mph SSE
7:00 am	58 °F	79%	51 °F	76%	7 mph SSE
8:00 am	58 °F	74%	51 °F	77%	7 mph S
9:00 am	60 °F	74%	51 °F	74%	7 mph S
10:00 am	62 °F	74%	52 °F	70%	8 mph S
11:00 am	64 °F	76%	52 °F	65%	8 mph SSW

Figura: Esempio di serie temporale multivariata, in questo esempio la temperatura dipende (chiaramente) dalle condizioni atmosferiche, ed allo stesso modo l'umidità.

Nell'approcciarsi all'analisi delle serie temporali si possono distinguere:

- **Approccio classico**
  - ARIMA
  - ARMA
  - ARCH
  - ...
- **Approccio moderno**
  - LSTM
  - RestNet
  - ROCKET
  - XGBoost
  - TST
  - ...

Sia Transformer che ROCKET rientrano nella seconda categoria, segue l'analisi del primo modello (Transformer).

Introduzione

TST, *Time Series Transformer* è un modello proposto da *George Zerveas et al.*[12] , basato sul modello *Transformer* (a sua volta proposto da Vaswani nell'articolo "Attention is all you need [11], per eseguire:

- Regressione
- Classificazione

Su serie temporali, con l'intento quindi di modellare i cambiamenti che con applicazioni che possono essere:

- Analisi dell'andamento degli strumenti finanziari
- Spese energetiche
- Previsioni su pandemie

TST è appunto basato su Transformer, che è a sua volta basato sulla tecnica degli "**Autoencoder**" di cui si darà un accenno.

Autoencoder (1)

Un autoencoder [1] *(Rumelhart et. al) - 1986* è un metodo di apprendimento (non supervisionato) basato su una rete neurale che prende l'input, lo comprime in un context vector (Attraverso un processo di encoding) e poi lo decomprime per generare un output. Formalmente:

1. **Encoder**: la parte della rete che comprime l'input in un vettore contestuale e che può essere rappresentato dalla funzione di transizione $\phi : X \rightarrow F$.
2. **Decoder**: la parte che si occupa di ricostruire l'input sulla base delle informazioni precedentemente raccolte. È rappresentato dalla funzione di transizione $\psi : F \rightarrow X$.

Autoencoder (2)

ϕ, ψ costituiscono poi il problema di *minimizzazione* per l'apprendimento dell'autoencoder:

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2$$

Nel caso semplice in cui è presente un solo hidden layer, dato x e h *iperparametro* e σ funzione di attivazione, l'encoder realizza una funzione:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

con:

1. W matrice di pesi inizializzati *random*.
2. b vettore bias.

W, b vengono determinati da un processo di training.
Il decoder realizza:

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

Autoencoder (3)

Il training dell'Autoencoder viene eseguito cercando di minimizzare l'errore di ricostruzione (square error) con una loss del tipo:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

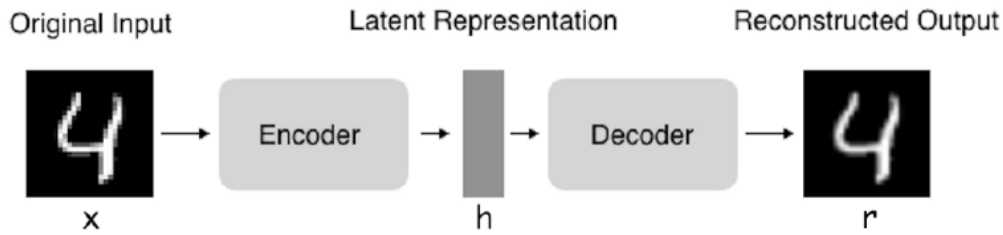


Figura: Struttura di un generico Autoencoder

Un tipo particolare di autoencoders sono i *Denoising Autoencoder (Vincent et. al - 2008)*, in cui il processo di training consiste nell'imparare a rimuovere il rumore (aggiunto) all'input.

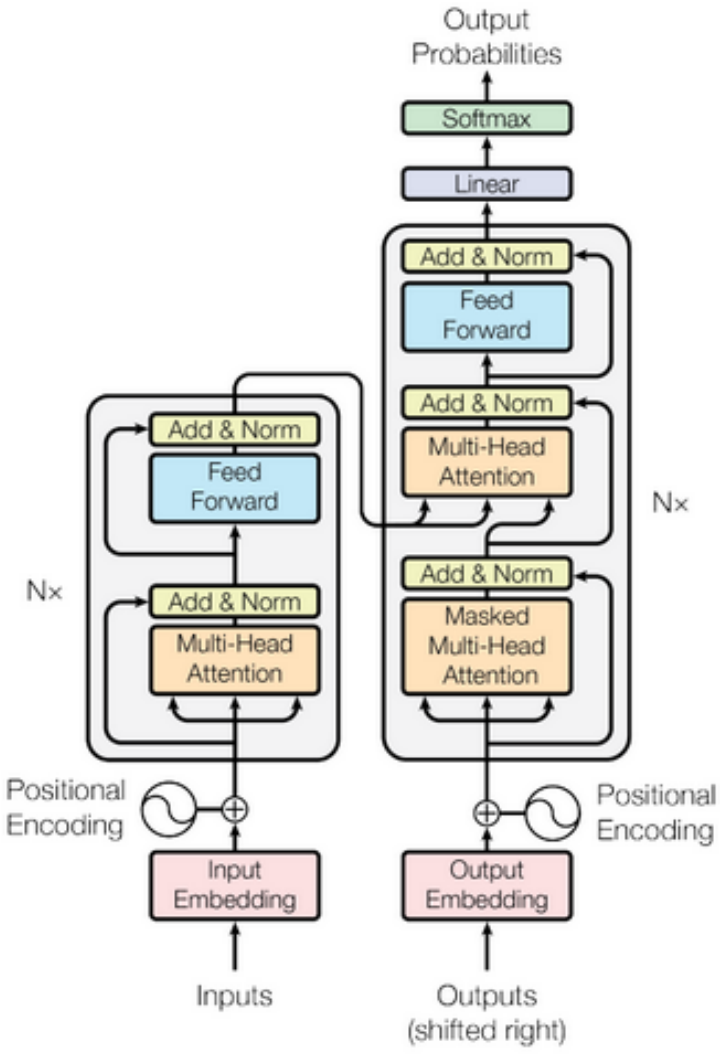
Autoencoder (4)

Il funzionamento di un denoising autoencoder può essere riassunto come segue:

1. l'input x viene corrotto con un rumore stocastico $\tilde{\boldsymbol{x}} \sim q_D(\tilde{\boldsymbol{x}}|\boldsymbol{x})$.
2. L'input corrotto viene mappato dall'autoencoder con funzione di encoding:
$$\boldsymbol{h} = f_\theta(\tilde{\boldsymbol{x}}) = s(\boldsymbol{W}\tilde{\boldsymbol{x}} + \boldsymbol{b}).$$
3. Il modello infine ricostruisce: $\boldsymbol{z} = g_{\theta'}(\boldsymbol{h})$ attraverso la funzione di *decoding*

In generale l'obiettivo è quello di costruire z quanto più possibile vicino ad x.
Si introduce quindi il modello TST basato su Transformers.

L'architettura di Transformer



Transformer è un modello (la cui architettura è riportata a sx.) di apprendimento multi-layer basato su autoencoder in cui si evidenziano i seguenti componenti principali:

- Segmento Encoder.
- Segmento Decoder.
- Ingresso (inputs e targets)
- Segmento in uscita (regressore o classificatore)

Transformer presenta più *meccanismi di attenzione*, si cerca cioè di imitare l'idea dell'attenzione umana.

L'architettura di Transformer (2) - Funzionamento Generale

Il funzionamento è il seguente:

1. L'input ed il target vengono immessi nella struttura.
2. Si esegue l'*encoding posizionale* delle sequenze di input e di output (sinusoidal positional encoding), questo è importante perché si parla di sequenze in cui c'è correlazione:

$$x_i \textit{ correlato } x_{i-1} \ldots$$

3. L'input viene quindi passato attraverso l'*encoder* che implementa il (primo) *meccanismo di attenzione* del transformer.
   Contemporaneamente anche il target viene passato attraverso il relativo encoder e il (secondo) meccanismo di attenzione, quello che prende il nome di *masked multi-head attention.*

L'architettura di Transformer (3) - Funzionamento Generale

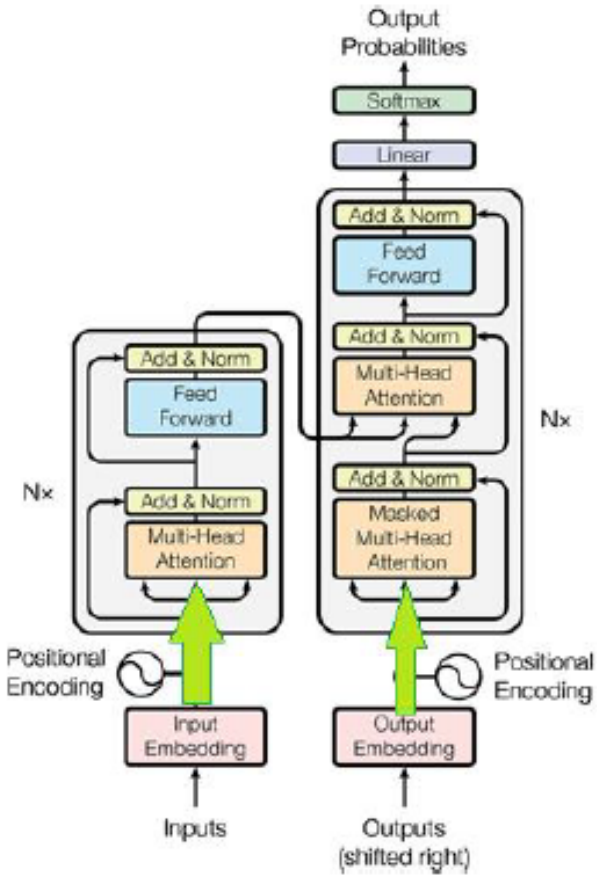


Figura: Processing dell'input del transformer (punti 1,2,3).

L'architettura di Transformer (4) - Funzionamento Generale

4. L'encoder dell'input estrae K, V, cioè un set di chiavi e valori, l'encoder del'output estrate un set Q di queries.
5. Si alimentano K, V, Q all'ultimo layer del transformer, in cui si incontra l'ultimo meccanismo di attenzione.

L'architettura di Transformer (5) - Funzionamento Generale

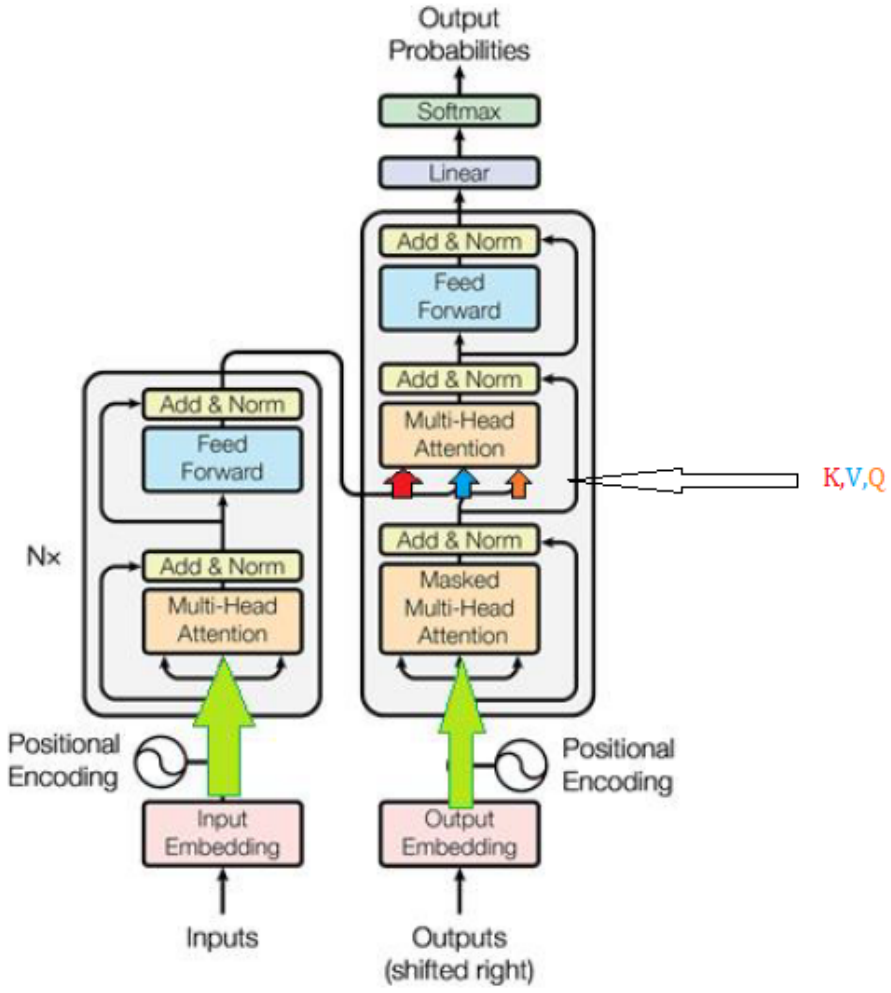
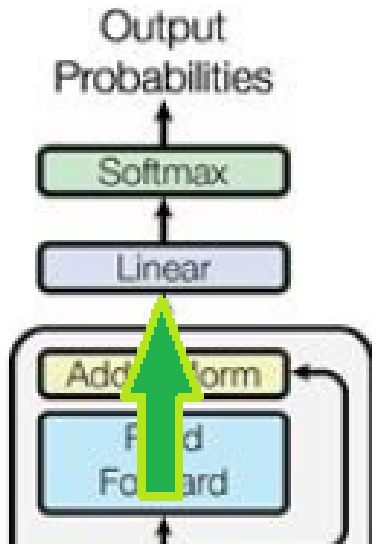


Figura: Processing dell'input del transformer (punti 4,5).

L'architettura di Transformer (6) - Funzionamento Generale

6 Il risultato di (5) viene poi alimentato in un layer feed forward, infine fatto passare attraverso un layer lineare.



7 Nel caso in cui si debba eseguire un task di regressione il processo si ferma, altrimenti si utilizza un layer *softmax*.

L'architettura di Transformer (7) - Attenzione

L'autore Vaswani [11] definisce l'**attenzione** (che è l'elemento contraddistintivo del transformer) come segue:

> *"Una funzione di attenzione può essere descritta come mappare una query e un insieme di coppie chiave-valore a un output, dove query, chiavi, valori e output sono tutti vettori. L'output viene calcolato come somma ponderata dei valori, dove il peso assegnato a ciascun valore viene calcolato da una funzione di compatibilità della query con la chiave corrispondente."*

L'*attenzione* è quindi a tutti gli effetti modellata come una funzione.

L'architettura di Transformer (8) - Attenzione

Dati K, Q, V Si distinguono quindi due tipi di [6]:

- **Attenzione "scaled dot product":**

$$A(Q) = \frac{softMax(Q \cdot K^T)}{\sqrt{d \cdot k}}$$

- **Multi-Head attention:** si esegue l'attenzione dot product h volte

$$MultiHead(Q, K, V) = [h_1 ... h_n] \cdot W_0$$

in cui

$$h_i = A(QW_i^Q, KW_i^K, VW_i^V)$$

Un meccanismo di attention prende il nome di *self-Attention* se le tre matrici vengono dallo stesso layer.

Funzionamento di TST

TST è il framework basato su transformer e proposto dagli autori (2020) [12] per agire su serie temporali, come nel caso di Google BERT si utilizza soltanto l'**encoder**, quindi si ha soltanto attention di tipo scaled dot product.
Ogni esempio di training è un vettore X che è una serie temporale di lunghezza w e m variabili diverse, che va a creare w vettori di features.

$$X = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,m} \\ X_{2,1} & X_{2,2} & \dots & X_{2,m} \\ \dots & \dots & \dots & \dots \\ X_{w,1} & X_{w,2} & \dots & X_{w,m} \end{bmatrix}$$

Ogni campione viene poi normalizzato secondo

$$x_i = \frac{x_i - \mu(X*)}{\sigma(X*)}$$

Con X* = tutto il traning set.

Funzionamento di TST (2)

La funzione di *transizione* ϕ scelta dagli autori, è:

$$\phi(x_t) = u_t = W_p \cdot x_t + b_p$$

in cui:

- W_p è una matrice di d righe e m colonne.
- b_p è un vettore di d elementi

Sono entrambi parametri che devono essere trovati attraverso training. Per l'encoding posizionale gli autori usano un metodo basato su sinusoidi

$$D_{it} = \begin{cases} \sin(\omega_{k \cdot t)} \text{ se } i = 2k \\ \cos(\omega_{k \cdot t)} \text{ se } i = 2k+1 \end{cases} \quad (1)$$

Con

$$w_k = \frac{1}{10000^{\frac{2k}{d}}}$$

I risultati vengono dati in ingresso all'encoder

Funzionamento di TST (3)

I risultati $u_1, \ldots, u_n$ sono poi dati in ingresso alla struttura del transformer e passano attraverso il primo layer, si ricavano gli insiemi K, V, Q per il primo layer di self-attention e così via fino a proseguire come flusso tutta la struttura.

Il training viene eseguito come quello dei Denoising Autoencoder, ma il rumore non è Gaussiano, si crea una matrice di rumore binaria che viene poi moltiplicata per la matrice di ingresso

$$X^* = M \cdot X$$

In cui il $\cdot$ è il prodotto element-wise, le sequenze di $0, 1 \approx Geom(k)$ o comunque non Bernoulliane.

Funzionamento di TST (4)

Gli autori in un *training epoch* eseguono l'encoder e ricavano $z_1, \ldots, z_n$ finali e quindi attraverso un metodo lineare provano a recuperare $x_1 \ldots n_n$ originale senza mascheramento.
La formula del classificatore è:

$$X_{t_{originale}} = W_0(z_1, \ldots, z_n) + b_0$$

dove W_0 e b_0 sono i parametri appresi dal modello.
Per il training si usa una *MSE-loss* in cui si calcola l'MSE solo sugli elementi mascherati. La formula della loss è la seguente:

$$\mathcal{L}_{MSE} = \frac{1}{|M|} \cdot \sum \sum_{(t,i) \in M} (\hat{x}(t,i) - x(t,i))^2$$

Funzionamento di TST (4)

In output si possono avere due comportamenti a seconda se si deve fare regressione o classificazione:

- **Classificazione**: l'output dell'encoder viene passato in un layer softmax
- **Regressione**: l'output dell'encoder viene passato attraverso in regressore lineare, gli autori propongo un regressore di tipo: $y^* = w_0 z + b_0$, con loss quadratica $\mathcal{L} = \|y_{pred} - y_{orig}\|^2$

Segue quindi l'analisi del modello ROCKET.

ROCKET

ROCKET (2019)[3] è un classificatore per serie temporali che:

- Utilizza kernel convoluzionali *completamente casuali* per estrarre featrues.
- E' più rapido di quasi tutti i competitors, fornendo risultati ugualmente attendibili.

Le features che utilizza ROCKET per la classificazione sono due:

1. **P.P.V** (Proportion of positive values) definita come segue:

$$ppv = \frac{|positivi|}{|positivi| + |falsi\_positivi|}$$

2. **max** (massimo di un kernel) definito come $\max_{k_{(i,j)}} \mathcal{K}$ con $\mathcal{K}$ matrice del kernel.

Convoluzione

Prima di analizzare il funzionamento di rocket è importante richiamare la teoria della convoluzione.

Convoluzione

$(f * g)(t)$ (convoluzione di f e g definisce un operazione che prende in ingresso una funzione (o matrice) in ingresso (f), una funzione (o una matrice) *kernel* (g) e restituisce un output (feature map)[7].

1. **Caso Continuo** (f,g continue in $\mathcal{R}$ e Lebesgue-Integrabili):

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

2. **Caso Discreto** $(f, g) : \mathcal{Z} \to \mathcal{Z}$:

$$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]\, g[m]$$

Convoluzione (2)

Una convoluzione si dice avere un *coefficente di dilatazione* (o banalmente una dilatazione) di l se è del tipo:

$$(f *_l g)[n] = \sum_{\tau=-\infty}^{\infty} f(\tau) g(t - l \cdot \tau)$$

Nel contesto di ROCKET un **kernel** [5] è una matrice, e la convoluzione è *discreta*.
Un kernel viene definito da:

1. La sua lunghezza (scelta come $|k| \approx Uniform$ con $|k| << |X|$.
2. Pesi estratti $w_i \approx Norm(0, 1)$.
3. Bias estratti $b_i \approx Uniform(-1, 1)$
4. Dilatazione $d = \lfloor 2^x \rfloor$ con $x \approx Uniform(0, A)$ con $A = log_2 \frac{l_{input}-1}{l_{kernel}-1}$

Convoluzione (3) - Kernel Dilation

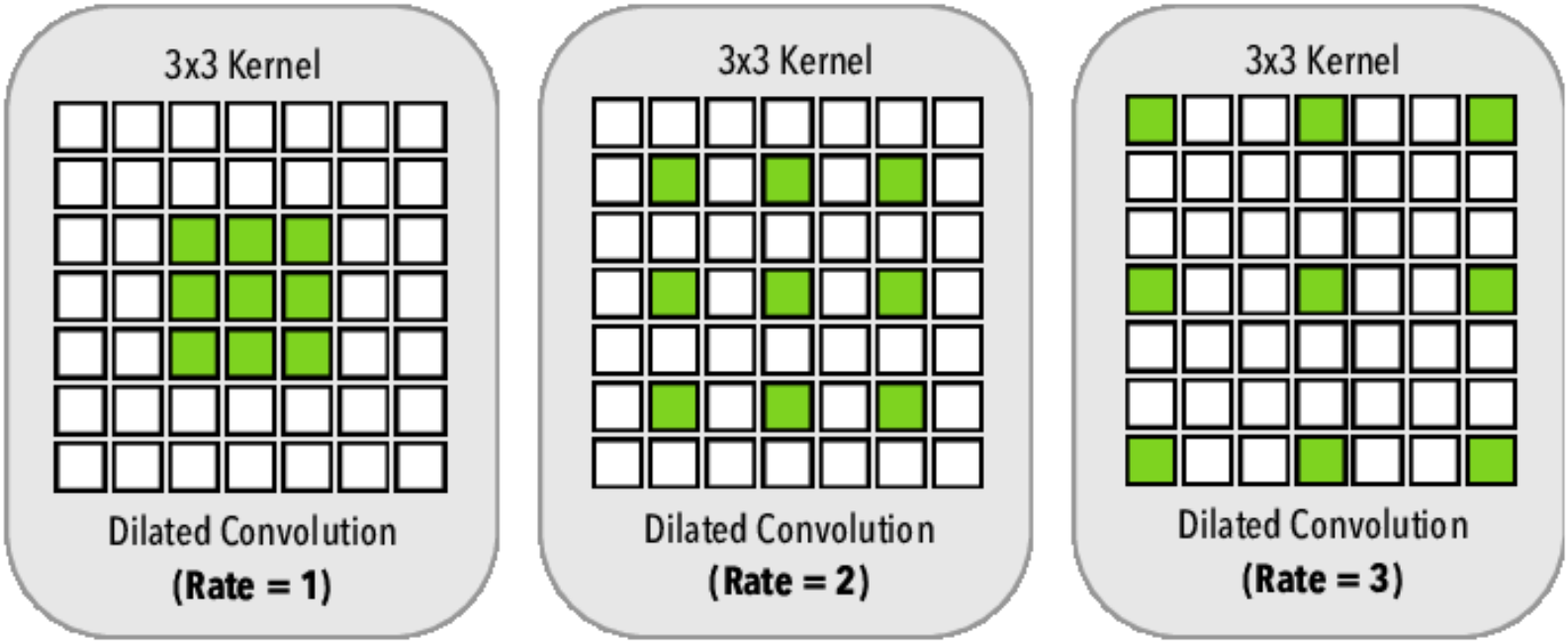


Figura: Convoluzione (discreta) con diversi *coefficenti di dilatazione* su kernel 3x3. Nell'immagine si nota l'"ombra" del kernel dilatato sulla matrice target.

ROCKET - Funzionamento

Dal momento che i parametri (1,2,3,4) sono casuali si parla di *kernel completamente casuali*.

Considerando una serie temporale come una matrice in ingresso $X \in \mathcal{R}^{(w,m)}$, Il funzionamento di ROCKET può essere schematizzato come segue:

1. Si prende in ingresso la serie X.
2. Ci si applicano *tutti* (o un subset) dei kernel casuali attraverso un operazione di *convoluzione dilatata*:

$$X_i \cdot k_i = \sum_{j=0}^{|kernel|-1} X_{i+(j \cdot d)} \cdot \omega_j$$

3. Dalla matrice convoluta si estraggono le features (max,ppv).
4. Si applica un classificatore lineare.

ROCKET - Funzionamento (2) (Classificazione)

Per eseguire la classificazione (dopo aver estratto ppv, max) gli autori propongono due strategie:

- **Logit regression** (Con sdg).
- **Ridge regression** (Con $L_2 Regularization$).

Si osserva che per dataset piccoli La $L_2 Ridge$ funziona meglio sia in velocità di esecuzione, fornendo un *accuracy* indistinguibile dalla Logit.

ROCKET - Funzionamento (3) (Costi)

Gli autori alla fine del paper propongono una breve **analisi dei costi** in cui:

1. **Trasformazione**: applicare i kernel costa $\mathcal{O}(k \cdot |input| \cdot n$ con n num. esempi.
2. **Classificazione**:
   - 2.1 Nel caso di logit con sgd il costo è lineare rispetto al numero di esempi.
   - 2.2 Nel caso Ridge il costo è $\mathcal{O}(n \cdot f^2)$ dove n è il numero di esempi di training e f è il numero di features.

La velocità di esecuzione è una caratteristica forndamentale di ROCKET, lo stesso Autore propone Minirocket (2020) [4], in cui si ottimizza la velocità (fino a $75x$ rispetto a rocket) e si cerca di renderlo "completamente deterministico".

Risultati e Confronto

In questa sezione verranno confrontati **Transformer** e **ROCKET** nei task di **classificazione** e **regressione**.

- **Rocket** - *Vaswani et. al.*
  1. epochs: 10
  2. n.kernel : 10.000
- **Transformer** - *Zerveas et. al.*
  1. Epochs : 100
  2. Learning Rate: 0.01
  3. d-model 128
  4. batch-size 32
  5. ottimizzatore: rAdam
- **XGBoost** - *Mini, Neri.* [8]
  1. Estimators: 10.000
- **RandomForest** - *Mini, Neri.* [8]
  1. Estimators: 2000
  2. Depth: 35
  3. n_jobs: 12

Cosa ci si aspetta

Date le due tecniche principali che si sono analizzate ci si aspetta che:

1. TST produca risultati più accurati di ROCKET, sia in classificazione che in regressione in quanto è una tecnica più complessa e basata su deep-learning.
2. ROCKET impieghi un tempo di training inferiore rispetto a TST.
3. ROCKET,TST performino meglio dei metodi ensemble (RandomForest, XGBoost).

Risultati e Confronto (2) - Datasets

Prima di eseguire il vero e proprio confronto è importante dare una panoramica sui dataset che vengono utilizzati.
Si è scelto di riprodurre gli esperimenti (Su alcuni) dei dataset utilizzati dagli autori, ma aggiornati.

- Per il task di **classificazione** si utilizza UCR[2] (Hoang Anh Dau et al) che contiene sotto-dataset con i relativi TRAIN (set) e TEST.
- Per il task di **regressione** si utilizza "http://tseregression.org/" [9] (Chang Wei Tan et al), in cui nuovamente per ogni sotto-dataset si ha il train set ed il test set.

Si elencheranno quindi i risultati sia in classificazione che in regressione.

Risultati e Confronto (Classificazione)

In questa sezione si mostrano i risultati nel task di classificazione dei 4 algoritmi visti precedentemente.

	TST(supervised)	Time(s)	TST(pretrained)	Time(s)	ROCKET	Time(s)	XGBoost	Time(s)
FaceAll	0,7580	1696	0,7420	1551	0,9468	123,612	0,8065	107
UWaveGestureLibraryAll	0,9185	61320	0,9107	69800	0,9753	47,638	0,9223	299
Adiac	0,6522	1578	0,6343	1312	0,7854	1,749	0,61	54
Yoga	0,7883	89908	0,7977	8903	0,9111	28,364	0,8083	27
ACSF1	0,67	57120	0,69	82800	0,8820	33,296	0,65	155
ShapesAll	0,7017	13377	0,6967	12922	0,9067	68,129	0,6416	1077
Strawberry	0,9514	3641	0,9595	3167	0,9816	19,918	0,9729	26
Wine	0,6667	326	0,6667	292	0,8037	0,378	0,6296	15
Symbols	0,8844	1606	0,8774	1391	0,9737	5,502	0,6301	49
InsectWingbeatSound	0,6414	2346	0,6490	2313	0,6565	4,897	0,6257	94
AVG Accuracy	0,7632		0,7624		0,8823		0,7297	
AVG Time		23291,8		18445,1		33,3483		190,3

Tabella: Confronto delle tecniche di classificazione su dataset *UCR*.

Risultati e Confronto (Classificazione) (2)

A titolo di esempio si riporta il grafico loss/accuracy del dataset "Strawberry" per mostrare l'andamento del processo di training.

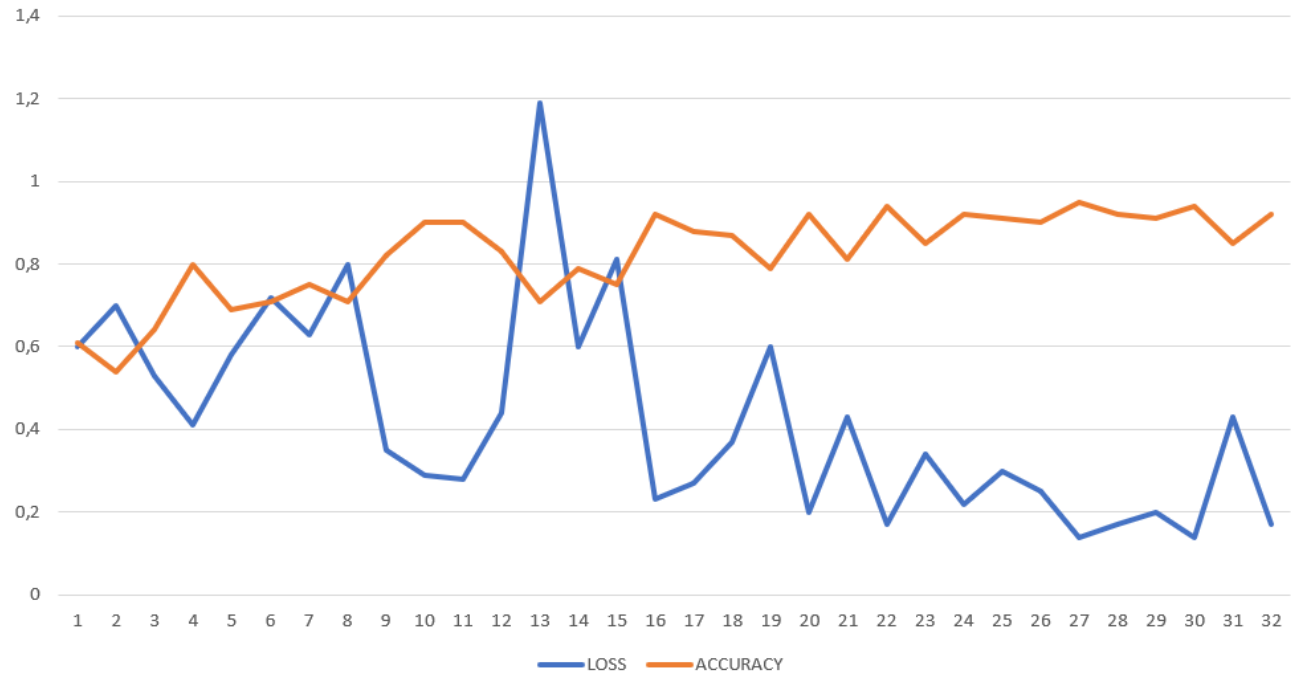


Figura: Grafico accuracy-loss, si nota come i valori di loss e accuracy siano inversamente proporzionali e che, al crescere del numero di epoche diminuisce la loss ed aumenta l'accuracy

Risultati e Confronto (Classificazione) (3)

Come visto a lezione, si riportano i risultati attraverso *TensorBoard*.

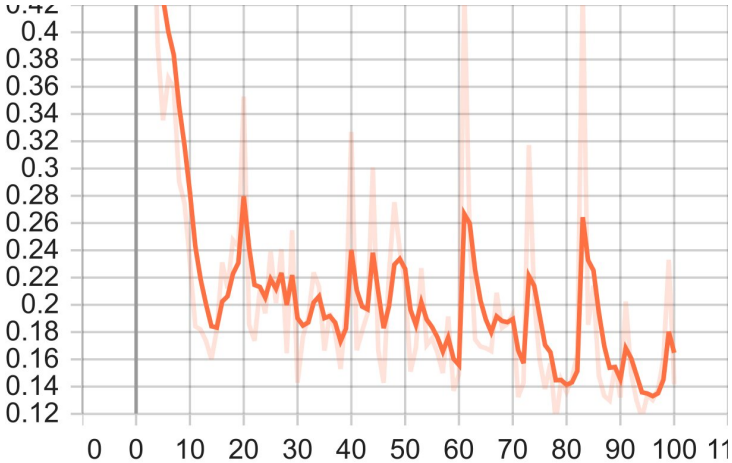


Figura: Loss sul train set di Strawberry.

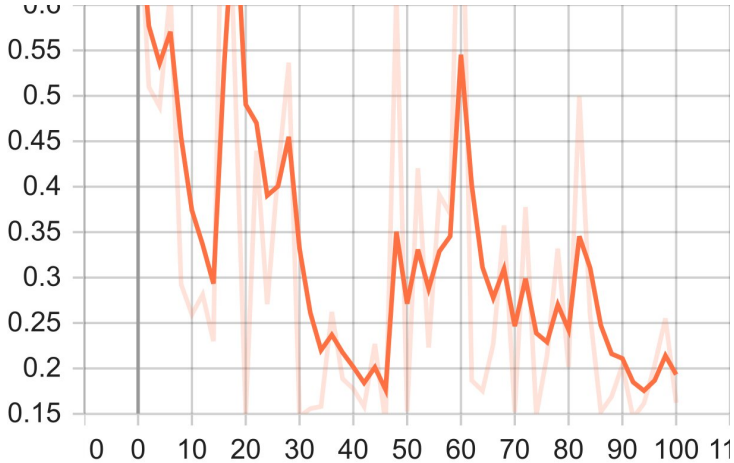


Figura: Loss sul validation set di Strawberry.

Risultati e Confronto (Classificazione) (4)

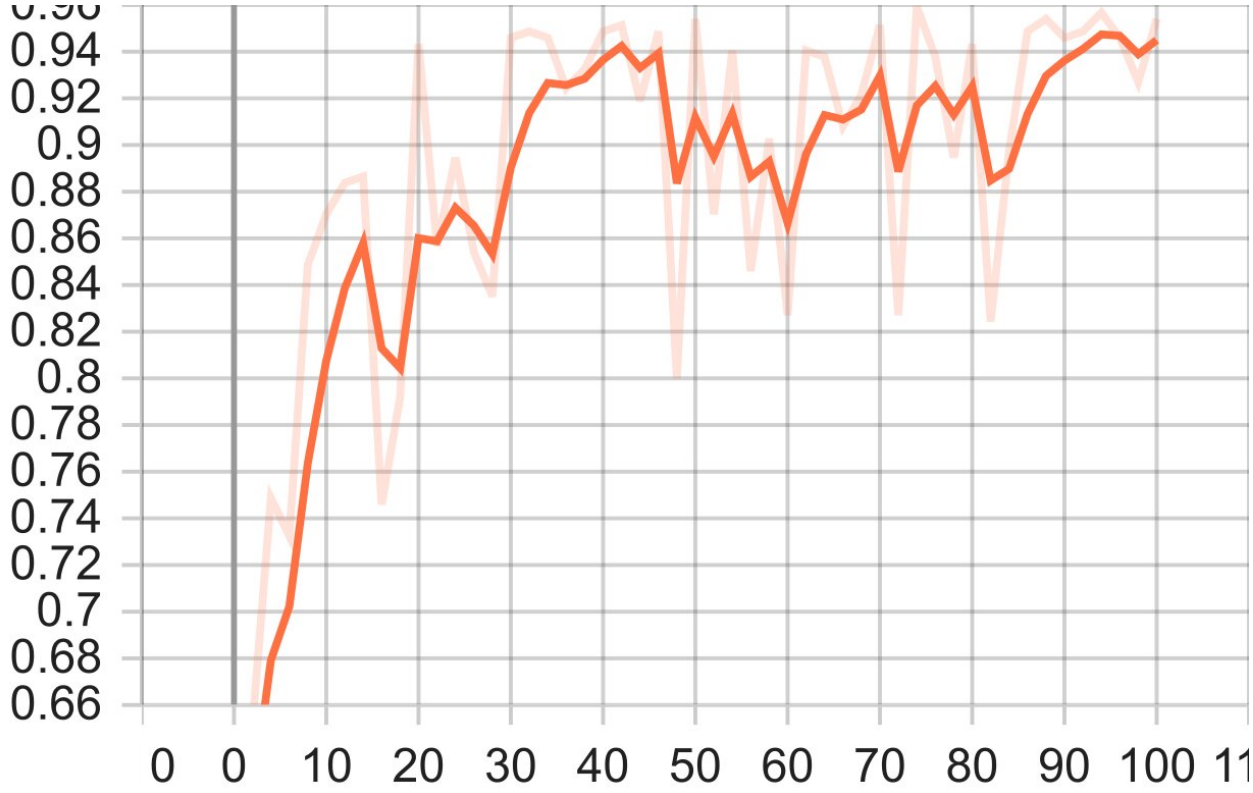


Figura: Grafico dell'accuracy sul dataset Strawberry

Risultati e Confronto (Regressione)

In questa sezione si mostrano i risultati nel task di regressione dei 4 algoritmi visti precedentemente.

	TST(supervised)	**Time(s)**	**TST(pretrained)**	**Time(s)**	**Random Forest**	**Time(s)**	**XGBoost**	**Time(s)**
BeijingPM10Quality	97,2512	3411	92,4407	3529	97,2168	136	96,7656	274
BeijingPM25Quality	64,0453	3069	59,4698	3555	64,3514	141	63,4187	272
AppliancesEnergy	3,1703	226	3,1676	210	3,3996	21	3,9637	158
LiveFuelMoisture	44,1029	27300	43,8310	27956	43,9939	966	47,5618	1347
IEEEPPG	32,2257	93180	29,4403	94875	31,9537	797	31,7114	1232
Covid3Month	0,0547	187	0,0480	185	0,0424	2	0,0512	9
AVG RMSE	40,1417		38,0662		40,1596		40,5787	
AVG Time		21228,83		21718,33		343,83		548,67

Tabella: Confronto delle tecniche di regressione su dataset Monash.

Risultati e Confronto (Regressione) (2)

A titolo di esempio si riporta il grafico della loss del dataset "LiveFuelMoisture" per mostrare l'andamento del processo di training.

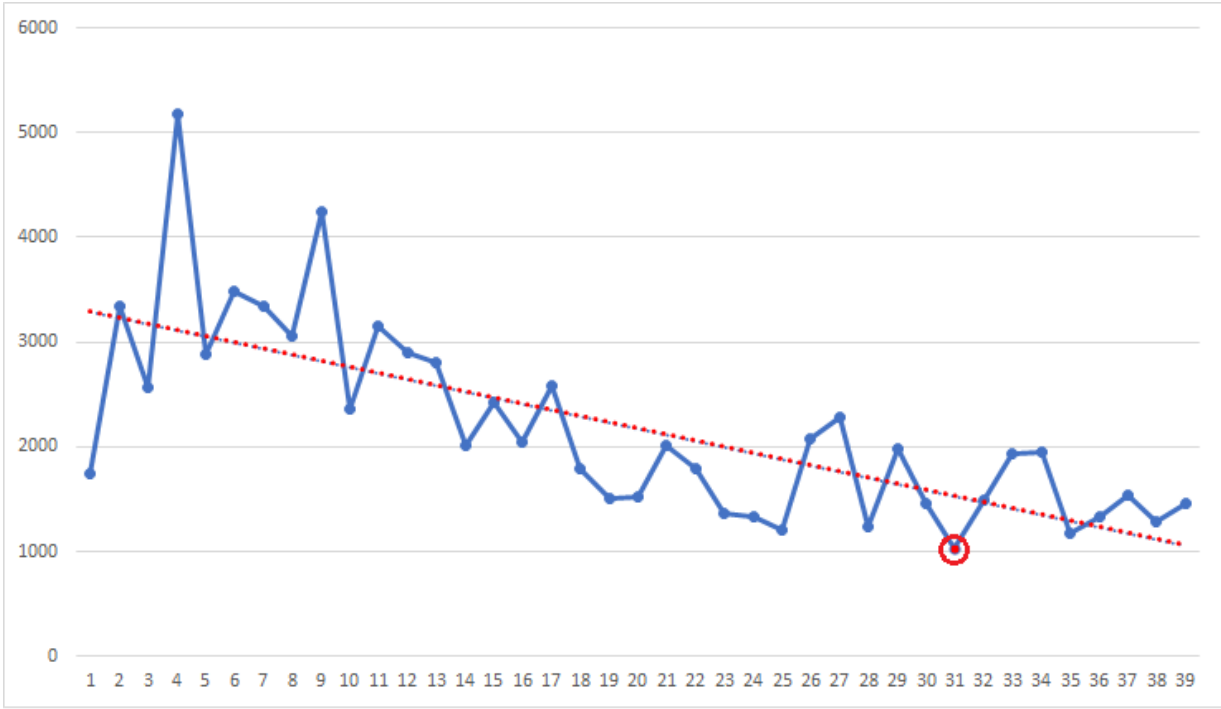


Figura: Si nota l'andamento del Loss durante il processo di training e la best epoch evidenziata in rosso (epoca 54).

Risultati e Confronto (Regressione) (3)

Come visto a lezione, si riportano i risultati attraverso *TensorBoard*.

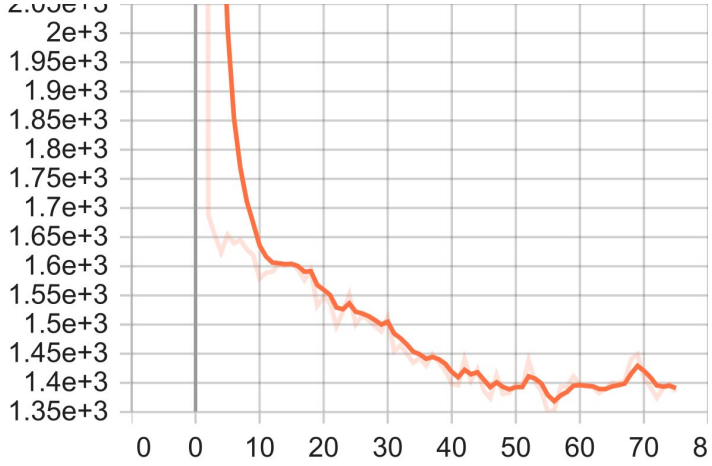


Figura: Loss sul train set di LiveFuelMoisture.

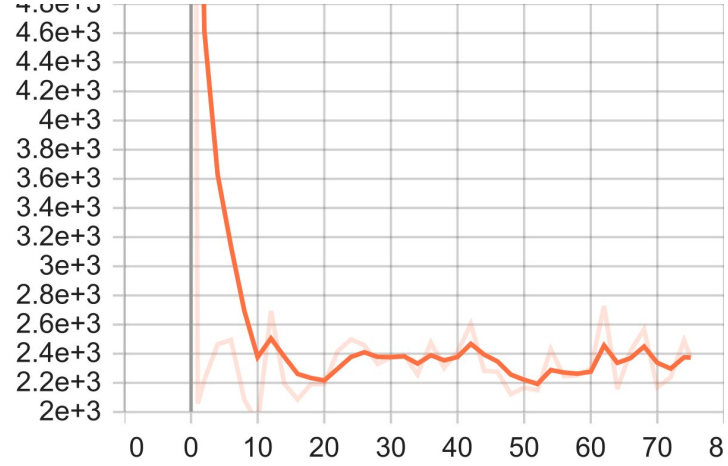


Figura: Loss sul validation set di LiveFuelMoisture.

Ulteriori test - Regressione

In quest'ultima sezione si confrontano alcune configurazioni ulteriori per la regressione e la classificazione, prendendo come target rispettivamente Covid3Month e Wine.

Default	**Normalizzazione L2**	**Batch Norm**	**L2+Batch Norm**
0,0547	**0,0455**	0,0493	0,0492

Tabella: Confronto tra diverse configurazioni su Covid3Month (regressione).

Default	**Geometric**	**Bernoulli**
0,048	0,0519	**0,0476**

Tabella: Confronto tra mascheramento (pretraining) geometrico e bernoulliano per regressione con normalizzazione L2.

Ulteriori test (2) - Classificazione

Default	Normalizzazione L2	Batch Norm	L2+Batch Norm
0,0547	**0,0455**	0,0493	0,0492

Tabella: Confronto tra diverse configurazioni su Wine (Classificazione).

Default	Geometric	Bernoulli
0,6667	**0,6827**	0,6791

Tabella: Confronto tra mascheramento (pretraining) geometrico e bernoulliano per classificazione con normalizzazione Batch.

Grazie per l'attenzione

Riferimenti I

[1] *Autoencoder*. en. Page Version ID: 1055131436. Nov. 2021. URL: https://en.wikipedia.org/w/index.php?title=Autoencoder&oldid=1055131436 (visitato il 24/11/2021).

[2] Hoang Anh Dau et al. "The UCR Time Series Archive". In: *CoRR* abs/1810.07758 (2018). arXiv: 1810.07758. URL: http://arxiv.org/abs/1810.07758.

[3] Angus Dempster, François Petitjean e Geoffrey I. Webb. "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels". In: *CoRR* abs/1910.13051 (2019). arXiv: 1910.13051. URL: http://arxiv.org/abs/1910.13051.

[4] Angus Dempster, Daniel F. Schmidt e Geoffrey I. Webb. "MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification". In: *CoRR* abs/2012.08791 (2020). arXiv: 2012.08791. URL: https://arxiv.org/abs/2012.08791.

Riferimenti II

[5] Prakhar Ganesh. *Types of Convolution Kernels : Simplified*. Medium, ott. 2019. URL: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37.

[6] Benyamin Ghojogh e Ali Ghodsi. "Attention Mechanism, Transformers, BERT, and GPT: Tutorial and Survey". In: (dic. 2020). DOI: 10.31219/osf.io/m6gcn.

[7] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[8] Andrea Neri e Alessandro Mini. *ML Project*. URL: https://github.com/Raydar32/MLProject.

[9] Chang Wei Tan et al. "Time Series Extrinsic Regression". In: *Data Mining and Knowledge Discovery* (2021), pp. 1–29. DOI: https://doi.org/10.1007/s10618-021-00745-9.

Riferimenti III

[10] *Time series*. en. Page Version ID: 1055662092. Nov. 2021. URL: https://en.wikipedia.org/w/index.php?title=Time_series&oldid=1055662092 (visitato il 24/11/2021).

[11] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[12] George Zerveas et al. "A Transformer-based Framework for Multivariate Time Series Representation Learning". In: *CoRR* abs/2010.02803 (2020). arXiv: 2010.02803. URL: https://arxiv.org/abs/2010.02803.