
Università degli Studi di Bari Aldo Moro

# Relazione tecnica Sustainability of RecSys

Emanuele Fontana

Tirocinio tesi triennale in Informatica
Anno accademico 2023/2024

# Indice

# 1 Introduzione

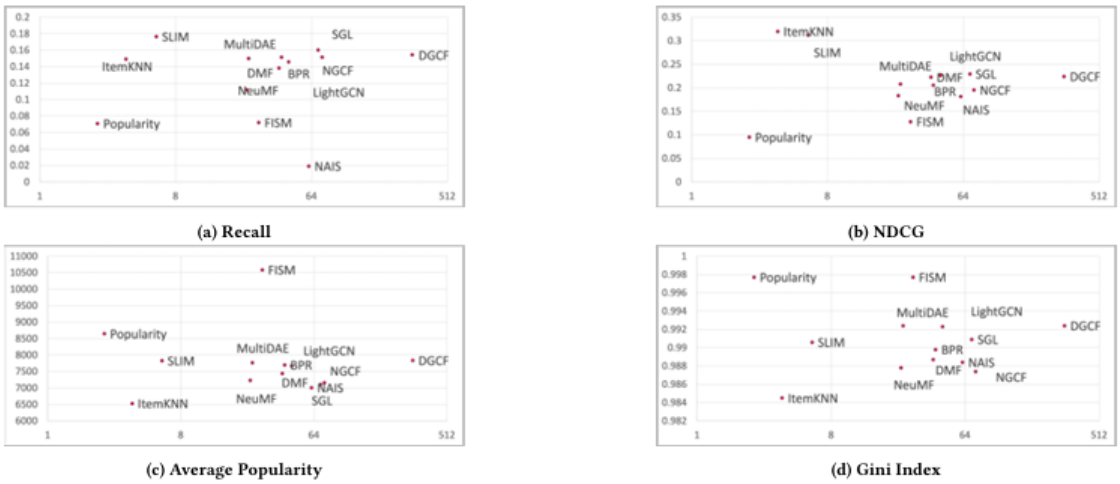
Tracciare le emissioni degli algoritmi di raccomandazione e cercare di prevederle è molto importante quando si parla di sviluppo sostenibile in campo RecSys. Ancora oggi si tende a trascurare l'impatto ambientale di un'attività e, in questo ambito, si è molto propensi nell'utilizzare dei modelli molti complessi e pesanti che richiedono molte risorse per essere addestrati ed eseguiti per ottenere delle buone performance. Spesso, però, modelli molto più leggeri e semplici riescono a ottenere delle performance molto simili (se non superiori) a modelli più complessi e il tutto con un impatto ambientale decisamente minore. Ad oggi il carbon dioxide equivalent ($CO_2$eq) è il principale indicatore utilizzato da governi e enti per misurare l'impatto ambientale di un'attività. Il $CO_2$eq è un'unità di misura che esprime l'equivalente in $CO_2$ di tutti i gas serra emessi da un'attività, in modo da poter confrontare l'impatto ambientale di attività diverse. Una strategia comune per calcolare il $CO_2$eq è quella di moltiplicare tra loro il **carbon intensity(CI)** e l'**energia consumata(PC)** dall'attività (nel nostro caso l'esecuzione di algoritmi).

$$emission = CI \cdot PC$$

In particolare i valori di CI dipendono dalle diverse fonti di energia utilizzate durante la computazione (es. energia solare, energia eolica, etc.). Se $s$ è la fonte di energia, $e_s$ sono le emissioni per KW/h di energia e $p_s$ è la percentuale di energia prodotta dalla fonte s, allora il CI è dato da:

$$CI = \sum_{s \in S} e_s \cdot p_s$$

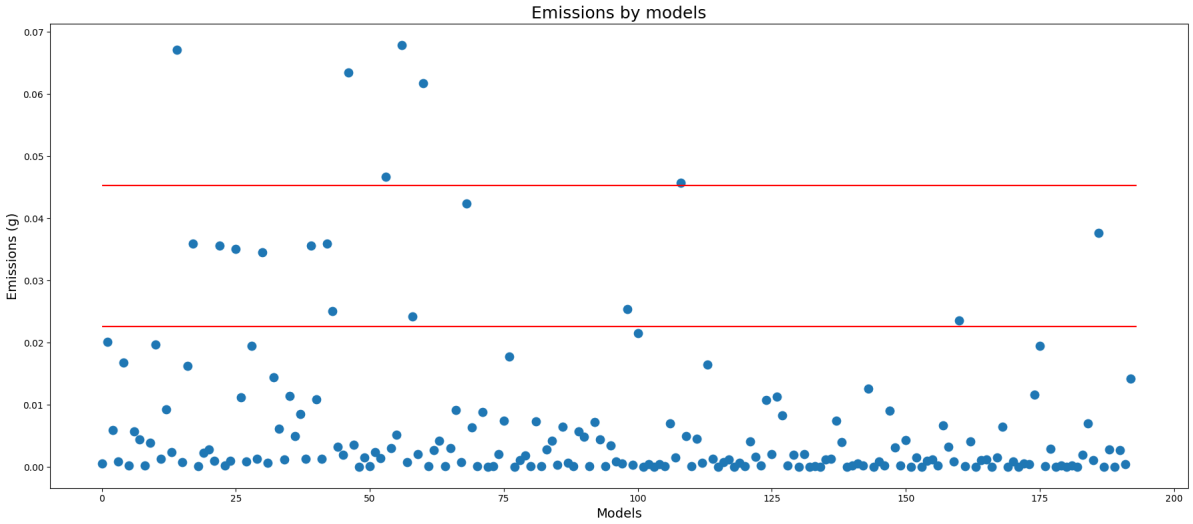
Lo scopo di questo lavoro è quello di valutare e prevedere l'impatto ambientale di un sistema di raccomandazione (RecSys) in base alla sua sostenibilità. Per quanto riguarda la valutazione, mediante la libreria CodeCarbon[1] è stato possibile misurare le emissioni prodotte dalla macchina durante l'addestramento con parametri di default per un dato modello dato un dataset. In questo ambito Spillo et al.[19] mostrano come spesso algoritmi più semplici riescono ad avere delle performance molto simili a modelli più complessi, ma con un impatto ambientale decisamente minore.




Trade-off tra emissioni e performance con dataset Mind

Per quanto riguarda la previsione questo documento si propone di presentare lo stato attuale del lavoro svolto in questo ambito.

[1] CodeCarbon




Emissioni prodotte dai vari modelli

L'esperimento è stato condotto su dataset presenti all'interno della libreria python RecBole[2], una libreria open-source che offre un'implementazione di modelli di raccomandazione. I dataset utilizzati per l'addestramento dei modelli sono:

- **MovieLens-1M**[3]
- **Amazon_Book_60core_kg** [4]
- **Mind**[5]

[2]RecBole
[3]Dataset MovieLens
[4]Dataset Amazon_Book_60core_kg
[5]Dataset MIND

# 2 Dataset del regressore

In questo capitolo si analizza il dataset utilizzato e come questo è stato trattato per l'addestramento dei modelli. Inoltre, si descrivono le feature di input e output del modello.
Il dataset nella sua totalità è composto da 13 feature di input e una feature di output. Le feature di input possiamo suddividerle in 4 categorie:

- **Feature relative al dataset**, quali *n_users*, *n_items*, *n_inter*, *sparsity*
- **Feature relative al knowledge graph**, quali *kg_entities*, *kg_relations*, *kg_triples*, *kg_items*
- **Feature relative all'hardware utilizzato per l'addestramento**, quali *cpu_cores*, *ram_size*, *is_gpu*
- **Feature relative al modello**, quali *model_name*, *model_type*

Nel dataset sono presenti 201 righe (dunque 201 esperimenti distinti).

## 2.1 Descrizione delle feature di output

La feature di output *emissions* rappresenta le emissioni di $CO_2$eq prodotte dalla macchina durante l'addestramento del modello.

## 2.2 Descrizione delle feature di input

| Feature | Descrizione |
|---|---|
| n_users | Numero di utenti presenti nel dataset |
| n_items | Numero di items presenti nel dataset |
| n_inter | Numero di interazioni nel dataset. Per interazione si intendono le varie interazioni (valutazioni) tra gli utenti nel dataset e gli item nel dataset |
| sparsity | Sparsità del dataset. La sparsità indica la percentuale di valori mancanti nel dataset (quindi mancanza di interazione tra utenti e item) |
| kg_entities | Numero di entità nel knowledge graph. Un'entità è un oggetto distintivo o un concetto unico all'interno del Knowledge Graph |
| kg_relations | Numero di relazioni nel knowledge graph. Le relazioni rappresentano i legami o collegamenti tra le entità all'interno del Knowledge Graph. Sono spesso definite dai predicati nelle triple |
| kg_triples | Numero di triple nel knowledge graph. Una triple è una struttura dati fondamentale nel Knowledge Graph che consiste in tre parti: soggetto, predicato e oggetto. Queste triple rappresentano le relazioni tra le entità |
| kg_items | Numero di items nel knowledge graph. Gli "Items" nel contesto del Knowledge Graph sono gli oggetti specifici o le entità che sono inclusi nel grafo |
| cpu_cores | Numero di core della CPU |
| ram_size | Dimensione della RAM |
| is_gpu | Booleano che indica se la macchina ha usato una GPU per l'addestramento |
| model_name | Nome del modello |
| model_type | Tipo del modello |

Descrizione delle feature di input

Per quanto riguarda la feature *model_type* abbiamo i seguenti valori:

- **General**: Modelli che si basano su tecniche tradizionali
- **Knowledge**: Modelli che incorporano conoscenza esterna (knowledge graph) per migliorare le raccomandazioni

Per quanto riguarda la feature *model_name* abbiamo i seguenti valori:

- **BPR** [17]: Basato su Bayesian Personalized Ranking (General)
- **CDAE** [35]: Utilizza Autoencoder per la raccomandazione (General)
- **CFKG** [1]: Incorpora conoscenza esterna (knowledge graph) per migliorare le raccomandazioni (Knowledge)
- **CKE** [37]: Combina Collaborative Filtering e Knowledge Graph Embedding per la rappresentazione della semantica (Knowledge)
- **DGCF** [32]: Incorpora due grafi: uno basato sulle interazioni utente-item e uno basato su relazioni semantiche (Knowledge)
- **DMF** [36]: Utilizza tecniche di Deep Learning per la raccomandazione (General)
- **DiffRec** [28]: Utilizza il concetto di "diffusione" (propagazione graduale di informazioni attraverso il grafo delle interazioni), concentrandosi sulla generazione di raccomandazioni personalizzate (General)
- **ENMF** [5]: Effettua l'addestramento senza campionamento (General)
- **FISM** [10]: Metodo item-based che usa matrici latenti per imparare la similarità tra gli items (General)
- **GCMC** [23]: Sfrutta tecniche di graph auto-encoder (General)
- **ItemKNN** [2]: Classico algoritmo KNN basato sugli items (General)
- **KGCN** [27]: Cattura le relazioni tra items attraverso il knowledge graph (Knowledge)
- **KGIN**: [31] Utilizza conoscenze aggiuntive per esplore le relazioni tra utenti e items (Knowledge)
- **KGNNLS** [25]: Utilizza tecniche di graph neural network per incorporare conoscenza esterna (Knowledge)
- **KTUP** [4]: Invece di trasferire la conoscenza del knowledge graph, trasferisce la conoscenza nel knowledge-graph (Knowledge)
- **LDiffRec** [29]: Rispetto a DiffRec, introduce clustering degli items per ridurre la dimensionalità (General)
- **LINE** [22]: Usa un innovativo metodo di embedding per gestire in modo efficiente grandi quantità di informazioni (General)
- **LightGCN** [6]: Semplifica il design delle Graph Convolutional Network per raccomandazioni (General)

- **MKR** [26]: Sfrutta le "crosscompress units" per apprendere e raccomandare simultaneamente (Knowledge)
- **MacridVAE** [14]: Mira a rendere le rappresentazioni apprese più interpretabili (General)
- **MultiDAE** [11]: Estende gli autoencoder al Collaborative Filtering (General)
- **MultiVAE** [12]: Estende i varaitional autoencoder al Collaborative Filtering (General)
- **NCEPLRec** [33]: Modello One-Class Collaborative Filtering che cerca di risolvere il popularity bias (General)
- **NCL** [13]: Modello Collaborative Filtering che incorpora esplicitamente i vicini sfruttando la struttura del grafo (General)
- **NGCF** [30]: Sfrutta la struttara user-item del grafo propagando gli embeddings attraverso esso (General)
- **NeuMF** [9]: Applica reti neurali al Collaborative Filtering (General)
- **Pop**: Raccomanda gli item più popolari (General)
- **Random**: Effettua raccomandazioni casuali (General)
- **RecVAE** [18]: Migliora MultiVAE modellando le distribuzioni di probabilità in modo più accurato (General)
- **RippleNet** [24]:Utilizza i knowledge-graph per migliorare le raccomandazioni (Knowledge)
- **SGL** [34]: Mediante dei task auto-supervisionati cerca di migliorare modelli come LightGCN (General)
- **SLIMElastic** [16]: Genera raccomandazioni top-N aggregando informazioni dai profili di acquisto/valutazione degli utenti (General)
- **SimpleX** [15]: Modello Collaborative Filtering che si concentra nel migliorare le funzioni di perdita (General)
- **SpectralCF** [38]: Modello Collaborative Filtering incentrato sulla scoperta di relazioni complesse (General)
- **EASE** [20]: Modello basato su elementi semplici descritti in letteratura (General)
- **NAIS** [8]: Rete neurale per Collaborative Filtering item-based (General)
- **ADMMSLIM** [21]: Migliora le tecniche SLIM (General)
- **ConvNCF** [7]: Usa reti neurali per Collaborative Filtering (General)
- **NNCF** [3]: Integra le informazioni sul vicinato nelle reti neurali (General)

Altri valori unici presenti per le varie feature di input sono:

- **n_users**: [22155, 23679, 6040]
- **n_items**: [54458, 4414, 3706]

- **n_inter**: [1465871, 1048575, 1000209]
- **sparsity**: [0.99878504, 0.98996762, 0.95531637]
- **kg_entities**: [26315, 0, 79347]
- **kg_relations**: [16, 0, 49]
- **kg_triples**: [96476, 0, 385923]
- **kg_items**: [11446, 0, 3655]
- **cpu_cores**: [12, 4]
- **ram_size**: [64, 16, 27.40581512]
- **is_gpu**: [1, 0]

## 2.3 Pre-Processing

Per poter sfruttare le feature di input per l'addestramento del modello, è stato necessario effettuare un pre-processing. Le feature *model_name* e *model_type* sono state trasformate in variabili numeriche. In particolare il valore *general* è stato trasformato in 0 e il valore *knowledge* è stato trasformato in 1. Per quanto riguarda la feature *model_name*, ogni valore è stato trasformato in un numero intero univoco. In questo modo, il modello può sfruttare queste feature per l'addestramento. Prima di cominciare con l'addestramento dei modelli la feature di output è stata separata dalle feature di input. I dati sono poi stati suddivisi rispettivamente in training set e test set. In particolare il 70% dei dati è stato usato per l'addestramento, mentre il 30% è stato usato per la valutazione. Inoltre, mediante il *random_state*=2, è garantita la riproducibilità dell'esperimento.

# 3 Modelli di regressione

## 3.1 Introduzione ai regressori

I regressori sono dei modelli di machine learning il cui obiettivo è quello di prevedere un valore numerico continuo (in questo caso il valore delle emissioni). I regressori vengono valutati sulla base di metriche quali, per esempio, l'errore quadratico medio (MSE), l'errore assoluto medio (MAE), e l'errore quadrato logaritmico medio (MSLE).

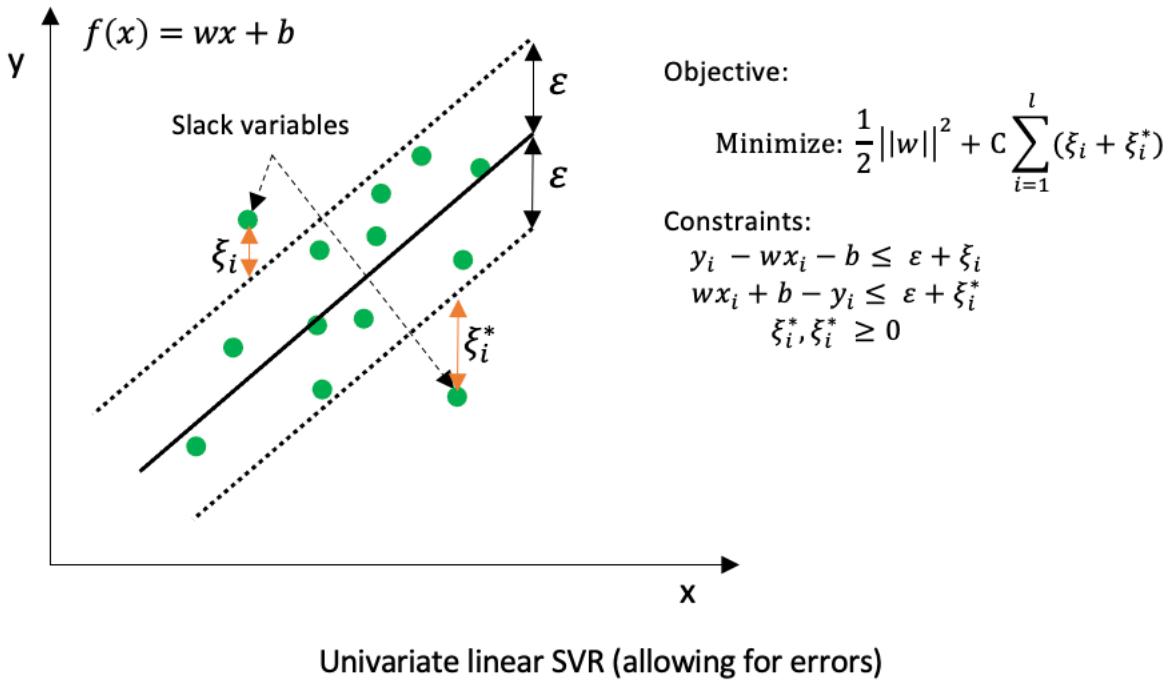
## 3.2 Regressori utilizzati

In questo lavoro sono stati utilizzati i seguenti regressori:

- **Support Vector Regression (SVR)**[6]
- **Decision Tree Regressor** [7]
- **Random Forest Regressor** [8]
- **AdaBoost Regressor** [9]

### 3.2.1 Support Vector Regression (SVR)

La SVR è un algoritmo che estende il concetto di Support Vector Machine (SVM)[10] al caso della regressione.




Esempio di SVR

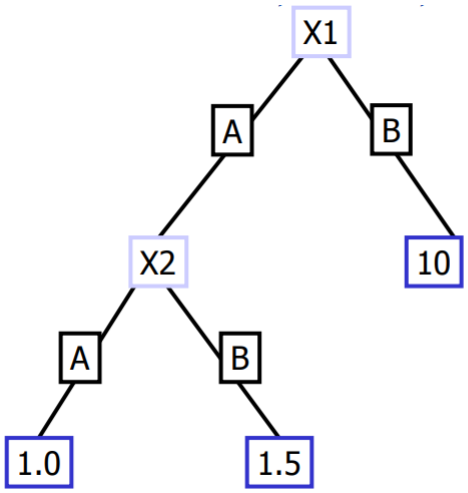
[6] SVR
[7] Decision Tree Regressor
[8] Random Forest Regressor
[9] AdaBoost Regressor
[10] Algoritmi di classificazione che cercano di trovare un iperpiano ottimale per ottenere separabilità lineare

In questo lavoro il modello è stato costruito usando gli iperparametri di default. Questi ultimi sono:

- **kernel=rbf**: E' il tipo di Kernel[11] utilizzato. Questo kernel valuta quanto due punti siano simili sulla base della loro distanza
- **degree=3**: grado del polinomio utilizzato per la trasformazione dei dati nello spazio
- **gamma=scale**: coefficiente del kernel
- **coef0=0.0**: termine indipendente nel kernel
- **tol=0.001**: tolleranza per il criterio di arresto
- **C=1.0**: parametro di regolarizzazione
- **epsilon=0.1**: controlla la tolleranza del modello nei confronti degli errori nella predizione
- **shrinking=True**: se abilitare o meno la riduzione del set di supporto
- **cache_size=200**: dimensione della cache in MB
- **max_iter=-1**: numero massimo di iterazioni. -1 indica nessun limite

### 3.2.2 Decision Tree Regressor

I Decision Tree Regressors sono modelli di regressione basati su alberi decisionali. Questi algoritmi suddividono ripetutamente il set di dati in base alle caratteristiche (features) per creare una struttura ad albero che rappresenta la relazione di decisione. Ogni nodo interno dell'albero rappresenta una decisione basata su una caratteristica, e le foglie dell'albero contengono i valori di output previsti.




Esempio di albero decisionale di regressione

In questo lavoro due iperparametri sono stati decisi all'atto della costruzione del modello:

- **max_depth=5**: profondità massima dell'albero

[11] Funzione matematica usata per mappare i dati originali in uno spazio di dimensione superiore

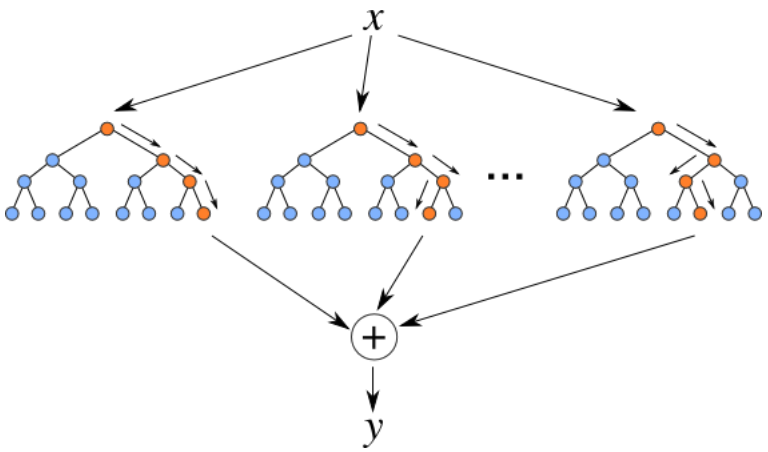
- **random_state=3**: controlla la casualità nell'addestramento del modello. Quando viene fissato a un numero intero specifico l'addestramento del modello sarà deterministico, cioè produrrà gli stessi risultati in ogni esecuzione

mentre gli altri iperparametri sono stati lasciati ai valori di default:

- **criterion=squared_error**: Criterio per stabilire la qualità di una suddivisione
- **splitter=best**: Strategia per scegliere la suddivisione in ogni nodo
- **min_samples_split=2**: Numero minimo di campioni richiesti per suddividere un nodo interno
- **min_samples_leaf=1**: Numero minimo di campioni richiesti per essere in una foglia
- **min_weight_fraction_leaf=0.0**: La frazione minima dei campioni totali (pesati) necessaria affinché si verifichi un nodo foglia
- **max_features=None**: Numero di features da considerare quando si cerca la migliore suddivisione. None indica tutte
- **max_leaf_nodes=None**: Numero massimo di foglie. None indica nessun limite
- **min_impurity_decrease=0.0**: Un nodo verrà suddiviso se questa suddivisione induce una diminuzione dell'impurità maggiore o uguale a questo valore
- **ccp_alpha=0.0**: Parametro di complessità usato per la potatura
- **monotonic_constraints=None**: Vincoli monotoni sui valori delle features

### 3.2.3 Random Forest Regressor

I Random Forest Regressors sono modelli di regressione basati su alberi decisionali. Vengono costruiti su più alberi decisionali che combinano le loro previsioni per ottenere una previsione più accurata e stabile.




Esempio di Random Forest Regresor

In questo lavoro tre iperparametri sono stati decisi all'atto della costruzione del modello:

- **n_estimators=500**: numero di alberi nella foresta
- **max_depth=5**: profondità massima dell'albero

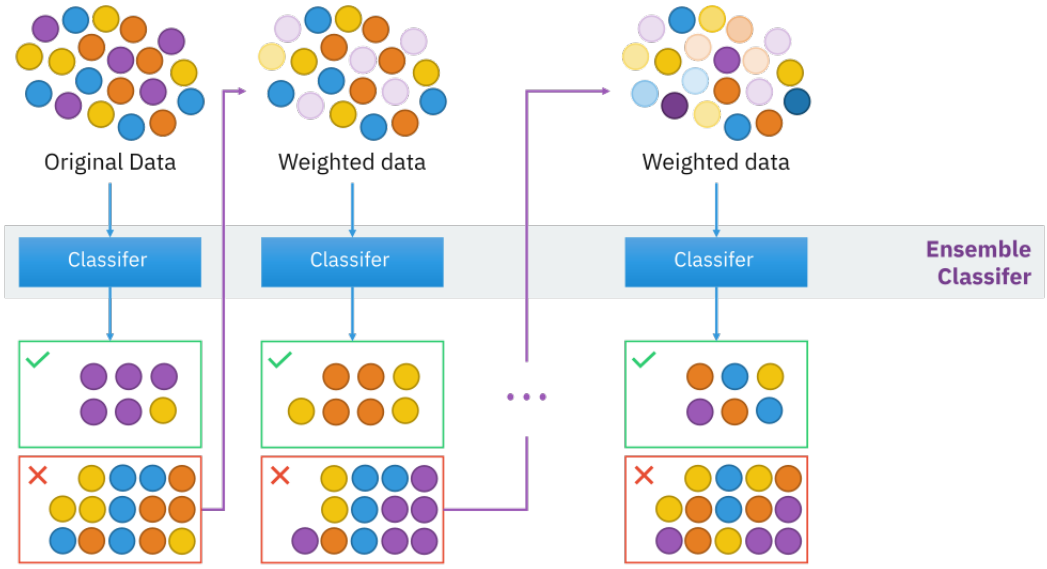
- **random_state=3**: controlla la casualità nell'addestramento del modello. Quando viene fissato a un numero intero specifico l'addestramento del modello sarà deterministico, cioè produrrà gli stessi risultati in ogni esecuzione

mentre gli altri iperparametri sono stati lasciati ai valori di default:

- **criterion=squared_error**: Criterio per stabilire la qualità di una suddivisione
- **min_samples_split=2**: Numero minimo di campioni richiesti per suddividere un nodo interno
- **min_samples_leaf=1**: Numero minimo di campioni richiesti per essere in una foglia
- **min_weight_fraction_leaf=0.0**: La frazione minima dei campioni totali (pesati) necessaria affinché si verifichi un nodo foglia
- **max_features=1**: Numero di features da considerare quando si cerca la migliore suddivisione
- **max_leaf_nodes=None**: Numero massimo di foglie. None indica nessun limite
- **min_impurity_decrease=0.0**: Un nodo verrà suddiviso se questa suddivisione induce una diminuzione dell'impurità maggiore o uguale a questo valore
- **ccp_alpha=0.0**: Parametro di complessità usato per la potatura
- **bootstrap=True**: Se utilizzare il bootstrap per la costruzione degli alberi
- **oob_score=False**: Se calcolare l'errore out-of-bag
- **n_jobs=None**: Numero di lavori da eseguire in parallelo. None indica 1
- **warm_start=False**: Se utilizzare la soluzione precedente come inizializzazione
- **max_samples=None**: Numero massimo di campioni da utilizzare per la costruzione di ciascun albero. None indica tutti
- **monotonic_constraints=None**: Vincoli monotoni sui valori delle features

### 3.2.4 AdaBoost Regressor

L'AdaBoost Regressor è un modello basato sull'algoritmo di boosting AdaBoost[12]. Questo algoritmo costruisce un modello di previsione combinando più modelli di previsione più deboli.




Struttura di modello basato su AdaBoost

In questo lavoro due iperparametri sono stati decisi all'atto della costruzione del modello:

- **n_estimators=500**: numero di stimatori
- **random_state=3**: controlla la casualità nell'addestramento del modello. Quando viene fissato a un numero intero specifico l'addestramento del modello sarà deterministico, cioè produrrà gli stessi risultati in ogni esecuzione

mentre gli altri iperparametri sono stati lasciati ai valori di default:

- **estimator=None**: stimatore base. Se None, viene utilizzato DecisionTreeRegressor
- **learning_rate=1.0**: tasso di apprendimento
- **loss=linear**: funzione di perdita

[12] Crea un modello forte combinando modelli più deboli, ognuno allenato su diverse parti del dataset

# 4 Risultati

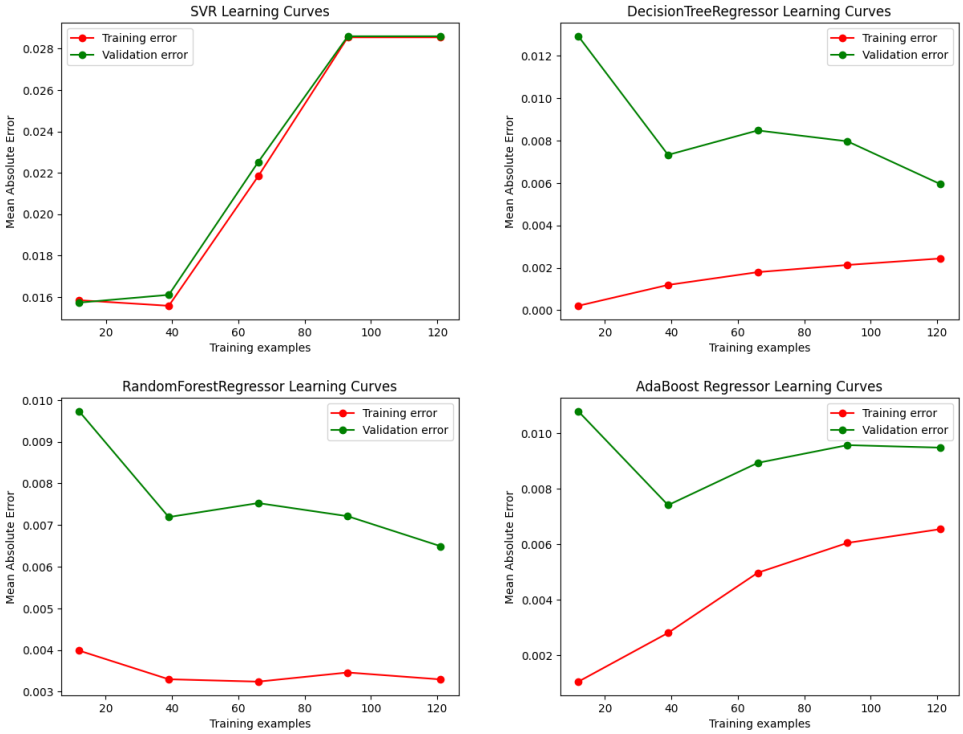
In questo capitolo si analizzano i risultati degli esperimenti che sono stati condotti. I regressori utilizzati sono stati valutati utilizzando le seguenti metriche per la regressione:

- **Mean Absolute Error (MAE)**: è la media della differenza assoluta tra le previsioni e i valori reali.
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
- **Root Mean Squared Error (RMSE)**: è la radice quadrata della media della differenza tra le previsioni e i valori reali al quadrato.
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$
- **Mean Logarithmic Squared Error (MSLE)**: è la media del logaritmo dei quadrati degli errori.
$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

## 4.1 Risultati ottenuti

| **Regressor** | **MAE** | **RMSE** | **MSLE** |
|---|---|---|---|
| SVR | 0.0288215 | 0.0008862 | 0.0008537 |
| Decision Tree | 0.0048531 | 0.0000969 | 0.0000918 |
| Random Forest | 0.0054369 | 0.0001088 | 0.0001026 |
| AdaBoost | 0.0071778 | 0.0001113 | 0.0001059 |

Risultati ottenuti




Learning curve dei regressori

## 4.2 Analisi dei risultati

**SVR.** Dal punto di vista delle metriche il regressore SVR ha ottenuto i peggiori risultati in termini di MAE, RMSE e MSLE. La learning curve mostra che il modello è in overfitting, infatti gli errori aumentano all'aumentare del numero di istanze.

**Decision Tree Regressor.** Dal punto di vista delle metriche, il Decision Tree Regressor ha ottenuto i migliori risultati. La learning curve mostra una curva di training che aumenta all'aumentare del numero di istanze, mentre la curva di validation diminuisce. Le due curve si avvicinano, ma non si sovrappongono. Sembra esserci un leggero underfitting.

**Random Forest Regressor.** Dal punto di vista delle metriche il Random Forest Regressor risulta essere il secondo miglior modello. La learning curves di train e di validation set diminuiscono entrambe all'aumentare del numero di istanze, ma non si sovrappongono. Anche qui sembra esserci underfitting.

**AdaBoost Regressor.** Dal punto di vista delle metriche il AdaBoost Regressor ha ottenuto risultati peggiori rispetto al Random Forest Regressor ma migliori rispetto al SVR. La learning curve mostrano che il modello non è riuscito a generalizzare bene

## 4.3 Conclusioni

Il dataset sembra essere troppo piccolo per poter generalizzare bene i modelli. Inoltre, il dataset è molto sbilanciato, con pochi valori per ogni feature. Questo potrebbe essere un motivo per cui i modelli non generalizzano bene. Inoltre, i modelli non sono stati ottimizzati, quindi potrebbero essere migliorati mediante ricerca di iperparametri. Per i risultati attuali, il Decision Tree Regressor è il modello migliore.

# Riferimenti bibliografici

[1] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms*, 11(9), 2018.

[2] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, page 273–280, New York, NY, USA, 2013. Association for Computing Machinery.

[3] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. A neural collaborative filtering model with interaction-based neighborhood. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1979–1982, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *The World Wide Web Conference*, WWW '19, page 151–161, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient neural matrix factorization without sampling for recommendation. *ACM Trans. Inf. Syst.*, 38(2), jan 2020.

[6] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 639–648, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. *arXiv preprint arXiv:1808.03912*, 2018.

[8] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, December 2018.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[10] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 659–667, New York, NY, USA, 2013. Association for Computing Machinery.

[11] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 689–698, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[12] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 689–698, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[13] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *Proceedings of the ACM Web Conference 2022*, WWW '22. ACM, April 2022.

[14] Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. *Learning disentangled representations for recommendation*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[15] Kelong Mao, Jieming Zhu, Jinpeng Wang, Quanyu Dai, Zhenhua Dong, Xi Xiao, and Xiuqiang He. Simplex: A simple and strong baseline for collaborative filtering. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 1243–1252, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506, 2011.

[17] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 452–461, Arlington, Virginia, USA, 2009. AUAI Press.

[18] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I. Nikolenko. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 528–536, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Giuseppe Spillo, Allegra De Filippo, Cataldo Musto, Michela Milano, and Giovanni Semeraro. Towards sustainability-aware recommender systems: analyzing the trade-off between algorithms performance and carbon footprint. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 856–862, 2023.

[20] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, WWW '19, page 3251–3257, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Harald Steck, Maria Dimakopoulou, Nickolai Riabov, and Tony Jebara. Admm slim: Sparse recommendations for many users. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 555–563, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[23] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion, 2017.

[24] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information*

*and Knowledge Management*, CIKM '18, page 417–426, New York, NY, USA, 2018. Association for Computing Machinery.

[25] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems, 2019.

[26] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Multi-task feature learning for knowledge graph enhanced recommendation, 2019.

[27] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The World Wide Web Conference*, WWW '19, page 3307–3313, New York, NY, USA, 2019. Association for Computing Machinery.

[28] Wenjie Wang, Yiyan Xu, Fuli Feng, Xinyu Lin, Xiangnan He, and Tat-Seng Chua. Diffusion recommender model. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 832–841, New York, NY, USA, 2023. Association for Computing Machinery.

[29] Wenjie Wang, Yiyan Xu, Fuli Feng, Xinyu Lin, Xiangnan He, and Tat-Seng Chua. Diffusion recommender model. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 832–841, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 165–174, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the Web Conference 2021*, WWW '21, page 878–887, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. Disentangled graph collaborative filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1001–1010, New York, NY, USA, 2020. Association for Computing Machinery.

[33] Ga Wu, Maksims Volkovs, Chee Loong Soon, Scott Sanner, and Himanshu Rai. Noise contrastive estimation for one-class collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 135–144, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21. ACM, July 2021.

[35] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, page 153–162, New York, NY, USA, 2016. Association for Computing Machinery.

[36] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, volume 17, pages 3203–3209. Melbourne, Australia, 2017.

[37] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 353–362, New York, NY, USA, 2016. Association for Computing Machinery.

[38] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. Spectral collaborative filtering. In *Proceedings of the 12th ACM conference on recommender systems*, pages 311–319, 2018.