

Progetto Architetture Dati

Architetture Dati - Progetto di Giugno 2025

Realizzato da:

Bishara Giovanni - 869532
Singh Probjot - 869434

Lo Scopo del Progetto	4
Introduzione al Dataset - ML	4
La Scelta del Dataset	4
Descrizione delle Features	4
Dataset Pulito	5
Analisi Esplorativa	5
I Modelli - ML	13
Suddivisione del Dataset in Training e Test	13
Considerazioni Iniziali	13
Support Vector Machines	14
Alberi di Decisione	15
Misure di Performance	15
Le Dimensioni di Qualità	17
Completezza - Introduzione al Codice	18
Consistenza - Introduzione al Codice	19
Unicità - Introduzione al Codice	22
Accuratezza - Introduzione al Codice	23
Il Nostro Approccio	25
Background Pipeline Luigi	25
Struttura del Progetto	27
Pipeline Machine Learning	29
Architettura Prima Pipeline	29
Tasks Prima Pipeline	29
Data Explainability	38
Support Vector Machines	44
Decision Tree Classifier	48
Pipeline Esperimenti	52
Architettura Seconda Pipeline	52
Tasks Seconda Pipeline	53
Esperimenti su Data Quality	64
Plots Globali	65
Drop Features	70
Missing Values	71
Outliers	76
Out of Domain Values	79
Flip Labels	82
Duplicate Rows	84
Add Rows	88
Esperimenti Custom	92
Conclusioni	96

Appendice A	
Principal Component Analysis	98
Appendice B	
SHapley Additive exPlanations (SHAP)	99
Appendice C	
Possibili Metodi di Imputazione	100
Appendice D	
Expectation Maximization	101

Lo Scopo del Progetto

Il progetto consiste nello svolgimento di un'analisi approfondita dei concetti di Data Quality, andando a verificare concretamente come la differenza tra un dataset considerato pulito contro un dataset sporco (il concetto di pulizia e sporcizia del dataset sarà approfondito nei paragrafi successivi) possa andare ad influenzare le prestazioni di apprendimento e predizione dei modelli di Machine Learning.

Inoltre, una parte del progetto è stata dedicata alla trasformazione dei dati tramite Principal Component Analysis (PCA) e di come essa possa essere spiegata (Explainability) in modo più formale all'essere umano, attraverso correlazioni tra il target e importanza di ogni componente.

L'approccio utilizzato è stato quello di partire da modelli già addestrati con un dataset pulito, con relativa analisi esplorativa e prestazioni iniziali per poi utilizzarli per gli esperimenti su Data Quality e Explainability. In seguito verrà introdotto in modo riassuntivo il lavoro svolto nel corso di Machine Learning (contrassegnato con ML), presentando il dataset e i modelli utilizzati, per poi passare al cuore vero e proprio del progetto.

Introduzione al Dataset - ML

La Scelta del Dataset

Il dataset è stato selezionato con lo scopo di garantire coerenza e rilevanza nelle successive analisi condotte, utilizzando dati sensati e non fittizi.

In particolare, la scelta è stata orientata verso un insieme di dati che si prestasse ad un'analisi con Principal Component Analysis (PCA), la quale richiede una struttura dati adatta, preferibilmente con variabili numeriche continue e non nulle.

Perciò si è scelto un dataset con categorie maggiormente di tipo numerico continuo.

Il dataset è stato ottenuto su Kaggle attraverso il seguente indirizzo: [Link Dataset](#)

Descrizione delle Features

Il dataset riguarda dunque la classificazione binaria di tipi di vini date le seguenti features, le quali descrivono la composizione chimica di un vino:

- **Fixed acidity (acido tartarico):** Misura della quantità di acido tartarico presente nel vino, espressa in grammi per decimetro cubo (g/dm^3).

- **Volatile acidity (acido acetico)**: Misura della quantità di acido acetico presente nel vino, espressa in grammi per decimetro cubo (g/dm³).
- **Citric acid (acido citrico)**: Quantità di acido citrico presente nel vino, espressa in grammi per decimetro cubo (g/dm³).
- **Residual sugar (zucchero residuo)**: Quantità di zucchero residuo nel vino, espressa in grammi per decimetro cubo (g/dm³).
- **Chlorides (cloruri)**: Concentrazione di cloruri nel vino, espressa in grammi di cloruro di sodio per decimetro cubo (g/dm³).
- **Free sulfur dioxide (anidride solforosa libera)**: Quantità di anidride solforosa libera nel vino, espressa in milligrammi per decimetro cubo (mg/dm³).
- **Total sulfur dioxide (anidride solforosa totale)**: Quantità totale di anidride solforosa presente nel vino, espressa in milligrammi per decimetro cubo (mg/dm³).
- **Density (densità)**: Densità del vino, espressa in grammi per decimetro cubo (g/dm³).
- **pH**: Misura dell'acidità o basicità del vino su una scala da 0 a 14.
- **Sulphates (solfati)**: Concentrazione di solfati nel vino, espressa in grammi di sulfato di potassio per decimetro cubo (g/dm³).
- **Alcohol (alcol)**: Percentuale di alcol nel vino per volume (% vol).
- **Quality**: Qualità di un vino espressa con una valutazione da 0 a 10.

La qualità di un vino si esprime con un valore di valutazione da 0 a 10 dunque potrebbe essere considerata categorica, mentre le altre features, che sono proprietà chimiche, sono esprimibili attraverso valori continui.

Dataset Pulito

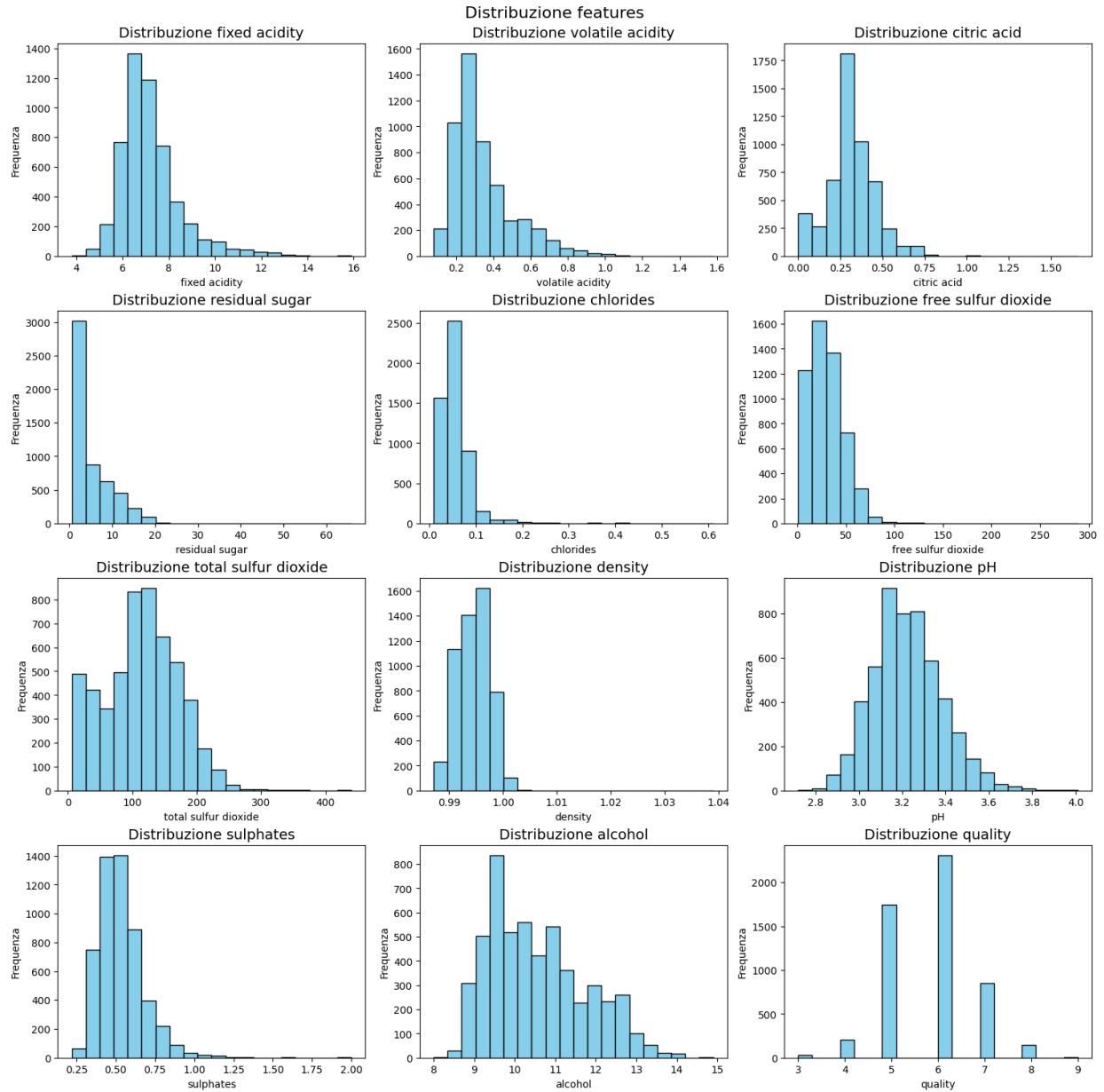
Il Dataset è stato pulito nel seguente modo:

- **Operazione di Casting**: sono state effettuate delle operazioni di casting sulle seguenti feature:
 - **Type**: ovvero il target del progetto il quale è binario, dove prima di tutto è stato effettuato il **label encoding**, al fine di convertire red nel valore 0 e white nel valore 1, per poi essere convertite a loro volta in **bool**, ovvero false e true.
 - **Quality**: la qualità del vino che va da 0 a 10 è possibile associarla ad una variabile **category**, in quanto limitata a solamente questi valori.
- **Pulizia del Dataset**: è stata effettuata una pulizia dei dati **duplicati** e dei dati con features **nulle**.

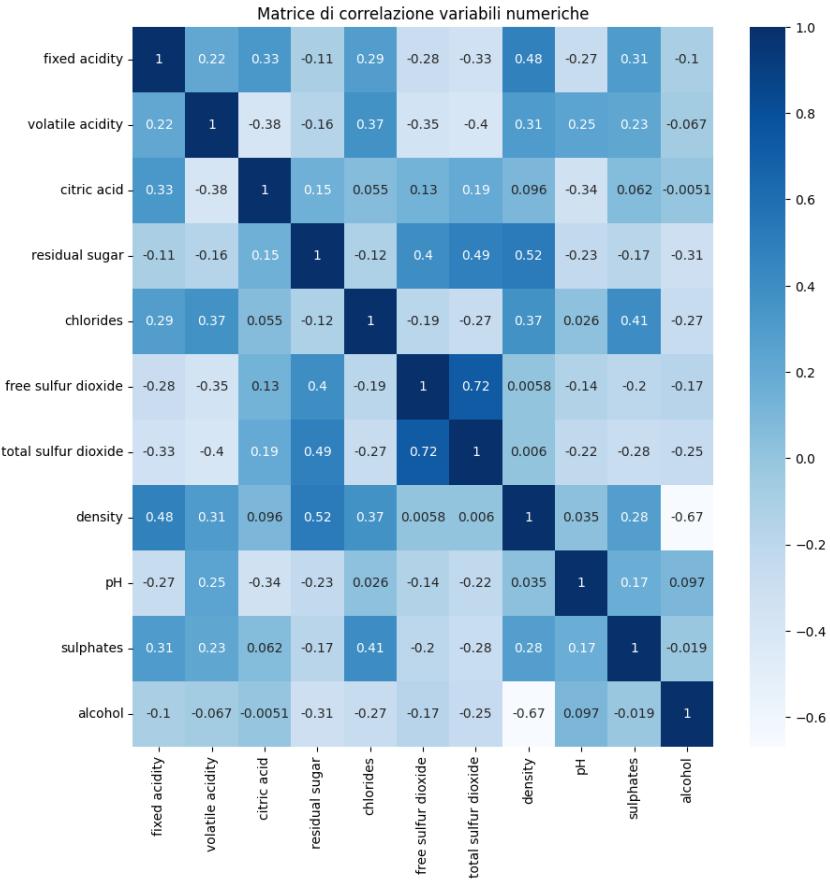
Analisi Esplorativa

Analisi delle Covariate e del Target

Si può innanzitutto osservare il tipo di distribuzione di ciascuna feature, trattando poi in seguito la distribuzione del target.



Per ciascuna coppia di features è stata calcolata la correlazione tramite la matrice di correlazioni:



Come ci si aspetta, la coppia di features più correlata è total sulfur dioxide, free sulfur dioxide; in generale non si notano features particolarmente correlate tra di loro.

Per concludere l'analisi delle features continue, è possibile visualizzare le statistiche relative a ciascuna di esse.

Di seguito vengono presentate le statistiche relative a tutto il dataset, per poi separare in base al valore assunto dal target (sotto-dataset contenente solo vini rossi e sotto-dataset con solo vini bianchi):

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000	5295.000000
mean	7.218008	0.344021	0.318782	5.051029	0.056690	30.046837	114.118225	0.994536	3.224385	0.533199	10.550154
std	1.320690	0.168237	0.147112	4.500641	0.036901	17.827151	56.787187	0.002969	0.160155	0.149851	1.186533
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000
25%	6.400000	0.230000	0.240000	1.800000	0.038000	16.000000	74.000000	0.992200	3.110000	0.430000	9.500000
50%	7.000000	0.300000	0.310000	2.700000	0.047000	28.000000	116.000000	0.994670	3.210000	0.510000	10.400000
75%	7.700000	0.410000	0.400000	7.500000	0.066000	41.000000	154.000000	0.996780	3.330000	0.600000	11.400000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000

red_df.describe()											
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000	1353.000000
mean	8.318477	0.529294	0.273016	2.522986	0.088163	15.854398	46.822986	0.996715	3.309165	0.658374	10.428394
std	1.736520	0.183323	0.195585	1.354667	0.049463	10.418830	33.432546	0.001870	0.154938	0.170917	1.081636
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.100000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996700	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000	63.000000	0.997830	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

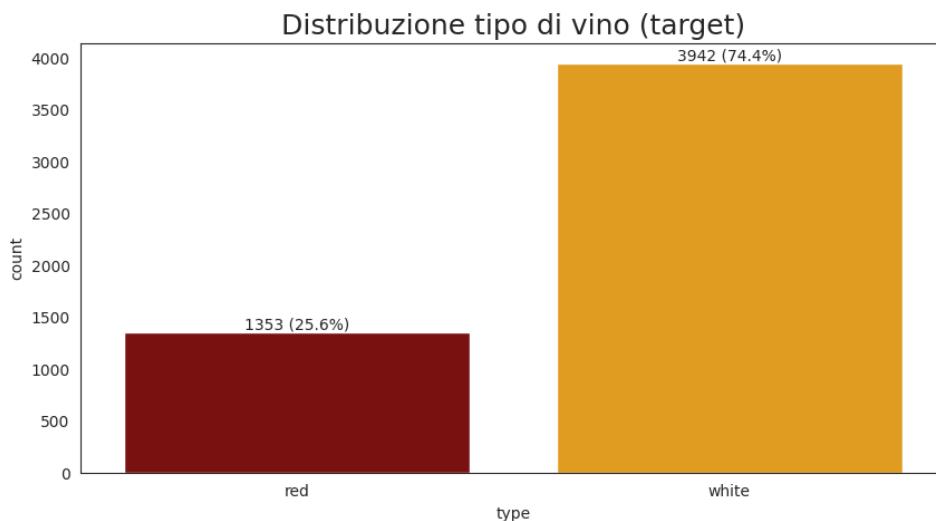
white_df.describe()												
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
count	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	3942.000000	
mean	6.840297	0.280430	0.334490	5.918721	0.045887	34.918062	137.215753	0.993788	3.195287	0.490236	10.591945	
std	0.866067	0.103256	0.122404	4.861389	0.023088	17.227540	43.128509	0.002907	0.151345	0.113653	1.217787	
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	
25%	6.300000	0.210000	0.270000	1.600000	0.035000	23.000000	106.000000	0.991600	3.090000	0.410000	9.500000	
50%	6.800000	0.260000	0.320000	4.700000	0.042000	33.000000	133.000000	0.993500	3.180000	0.480000	10.400000	
75%	7.300000	0.328750	0.390000	8.875000	0.050000	45.000000	166.000000	0.995710	3.290000	0.550000	11.400000	
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	

Analizzando le statistiche, è possibile fare diverse osservazioni, alcuni esempi:

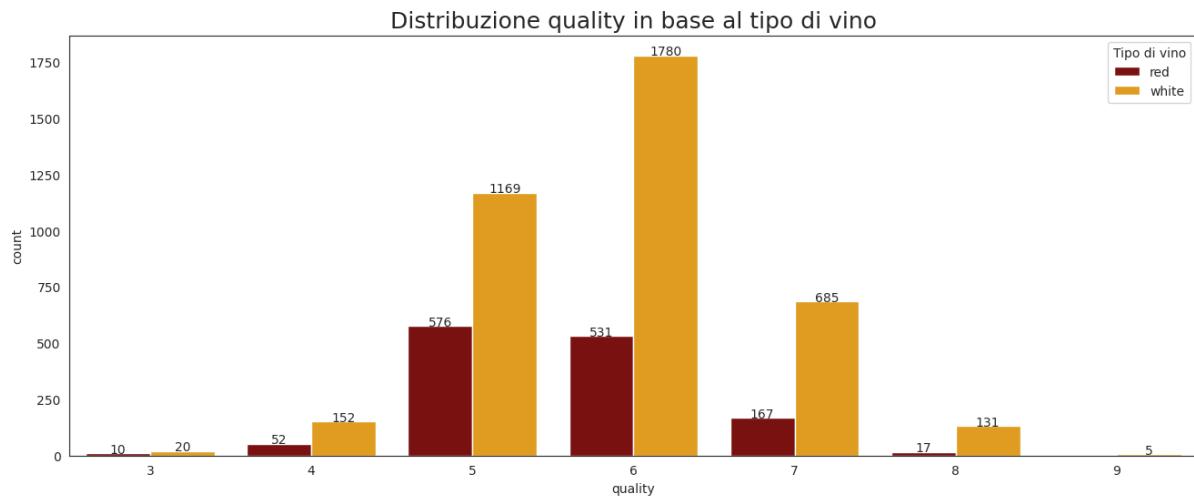
- si può notare che i vini bianchi sono molto più frequenti rispetto ai vini rossi (3942/5295 vs 1353/5295)
- la fixed acidity media nei vini rossi è maggiore rispetto a quella nei vini bianchi
- residual sugar, free sulfur dioxide, total sulfur dioxide presentano medie più alte nei vini bianchi
- residual sugar è più variabile nei vini bianchi
- nessun vino presenta un pH superiore a 4.01.

A questo punto, è possibile trattare la feature **quality** insieme al target **type**.

Prima di tutto, si può evidenziare come sono distribuiti i valori del target.

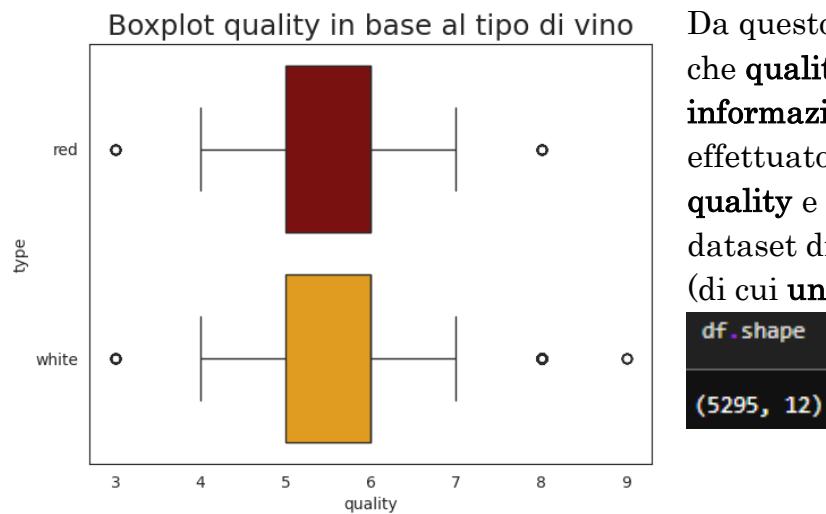


Si ha la conferma di quanto detto in precedenza, cioè il dataset non è bilanciato, avendo 3942 vini bianchi e 1353 vini rossi. Inoltre, si possono contare i vini rossi e i vini bianchi per ogni valore assunto da quality (si ricorda che la quality di un vino è un intero da 0 a 10):



Da questo grafico si può notare che, nonostante la quality possa assumere valori da 0 a 10, nel dataset non ci sono vini con quality inferiore a 3 o superiore a 9. Inoltre, si può notare sia per i vini rossi sia per i vini bianchi si hanno distribuzioni gaussiane: ci si può chiedere se è fondamentale tenere in considerazione la quality.

Per concludere l'analisi di covariate e target, è importante osservare un ultimo grafico che mette in relazione quality e type in una maniera differente, tramite un boxplot.



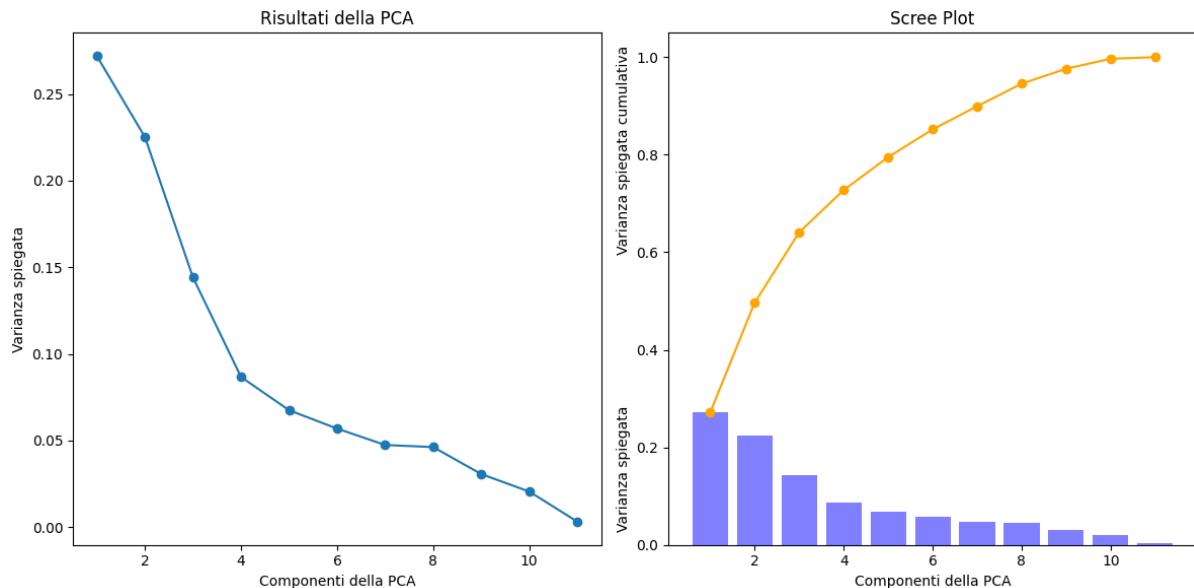
Da questo boxplot emerge il fatto che **quality non aggiunge informazione** sul tipo di vino: viene effettuato il **drop della colonna quality** e dunque si passa ad un dataset di 5295 righe e **12 colonne** (di cui **una è il target**).

```
df.shape
(5295, 12)
```

Principal Component Analysis - Preliminari

Il dataset contiene solo features continue, pertanto è stata provata l'applicazione della [Principal Component Analysis](#).

Dopo aver effettuato la standardizzazione delle variabili per poterle confrontare senza essere affetti da scale diverse di valori, vengono innanzitutto analizzati due grafici.



Questi grafici riguardano il rapporto tra la varianza spiegata e il numero di componenti della PCA: permettono di capire di quante componenti tenere il nuovo spazio, ottenuto tramite feature extraction, se si decide di mantenere un'alta percentuale di varianza spiegata cumulativa come metrica.

La parte inferiore del secondo grafico è chiamata scree plot, e visualizza tramite barre di un istogramma il rapporto appena trattato, mentre la parte superiore è una curva che mostra come cambia la varianza spiegata cumulativa in base al numero di componenti mantenute.

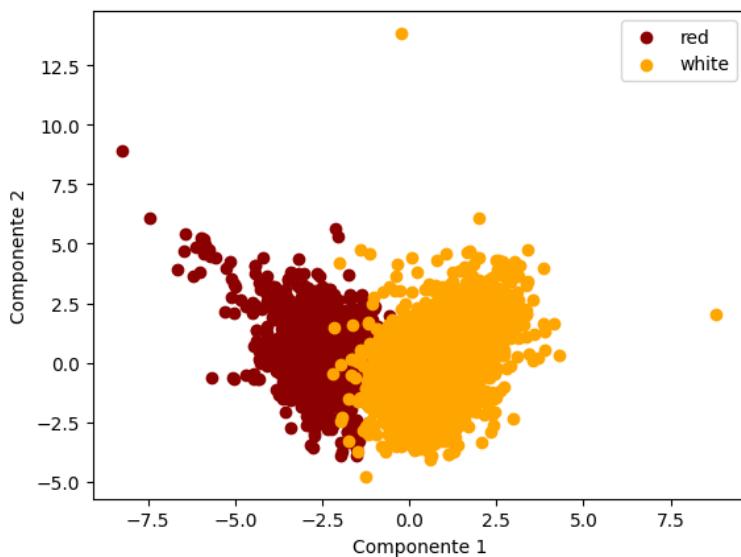
Utilizzando come criterio quello di mantenere circa l'80% della varianza spiegata cumulativa, si può vedere dai grafici che è possibile mantenere 5 componenti.

Come conferma di quanto appena visto, e come ulteriore metodo di visualizzazione dei dati per ogni componente, di seguito si produce una tabella in cui sulle righe si mettono le componenti, mentre sulle colonne autovalore, varianza spiegata e varianza spiegata cumulativa.

	Eigenvalue	Variance Percent	Cumulative Variance Percent
Comp 1	2.991077	27.186472	27.186472
Comp 2	2.476404	22.508515	49.694986
Comp 3	1.585096	14.407246	64.102232
Comp 4	0.953458	8.666165	72.768397
Comp 5	0.742378	6.747617	79.516014

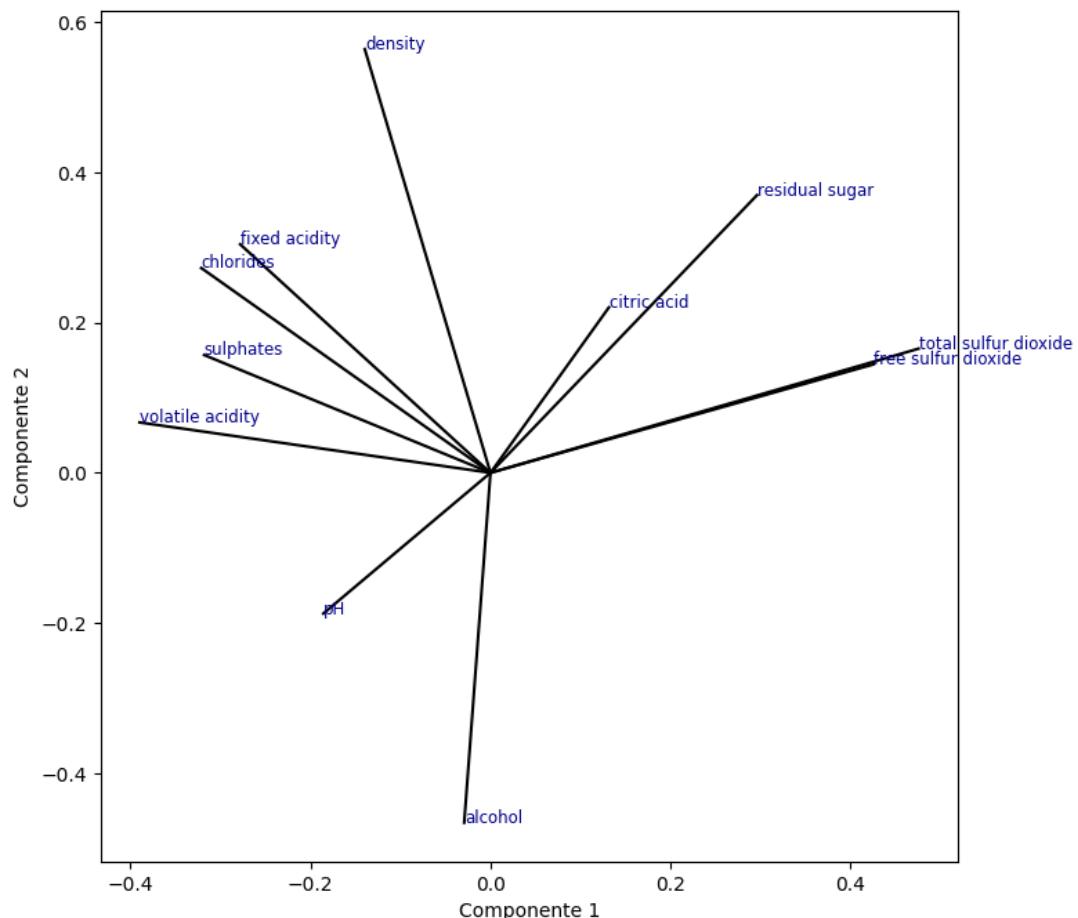
Può essere utile come alternativa per decidere quante componenti tenere, siccome un altro criterio di scelta consiste nel considerare solo componenti con autovalore > 1 .

Nel seguente grafico si verifica se vi è una correlazione lineare tra le prime due componenti di PCA (quelle con più alta varianza spiegata e quindi anche autovalori maggiori): se la risposta dovesse essere affermativa, allora ci sarebbe ridondanza tra i dati e non sarebbe significativo considerare il risultato della PCA.



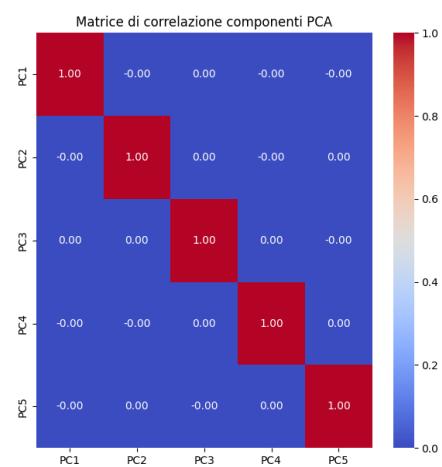
Si può notare che non vi è ridondanza e inoltre può essere uno spunto per la scelta di un modello di Machine Learning, in quanto intuitivamente si può pensare di provare a separare linearmente le due classi.

Per concludere l'analisi con PCA, si può visualizzare la correlazione tra le variabili e le prime due componenti di PCA, quest'ultime corrispondenti rispettivamente all'asse x e y.



Ciascun vettore consente di ricavare diverse informazioni: due vettori correlati positivamente sono raggruppati insieme, mentre in caso di correlazione negativa si trovano in quadranti opposti; infine, è possibile quantificare la qualità di una singola feature in base alla distanza dall'origine.

Un grafico aggiuntivo, utilizzato solo per confermare che la feature extraction di PCA è stata efficace, consiste in una matrice di correlazioni per evidenziare che le cinque componenti di PCA risultanti presentano correlazione pressoché nulla.



I Modelli - ML

I modelli di Machine Learning utilizzano i **dati prodotti dalla PCA** considerando **cinque componenti**.

Suddivisione del Dataset in Training e Test

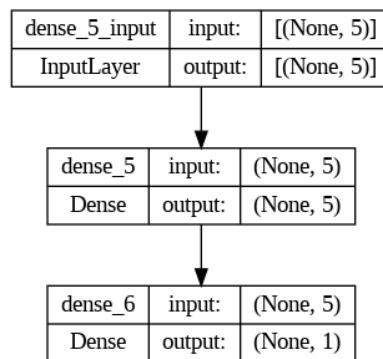
Prima della costruzione effettiva dei modelli di apprendimento si va a definire nello specifico la struttura dei dati da utilizzare per un corretto apprendimento. Si va quindi a definire l'insieme di dati da utilizzare per l'addestramento, test e il target da classificare (nel nostro caso il type).

L'insieme di dati utilizzato per l'addestramento e test è quello ottenuto dalla PCA e diviso nel seguente modo:

- 80% di dati per l'addestramento del modello.
- 20% di dati per il test per effettuare predizioni sul modello addestrato.

Considerazioni Iniziali

Data la struttura del dataset analizzato e il target selezionato, abbiamo da risolvere un problema di classificazione **binaria**. Avere un target binario ci consente di utilizzare diversi modelli per un apprendimento efficace. Inoltre, è stato implementato un modello di baseline con **accuracy** di circa il 75%.

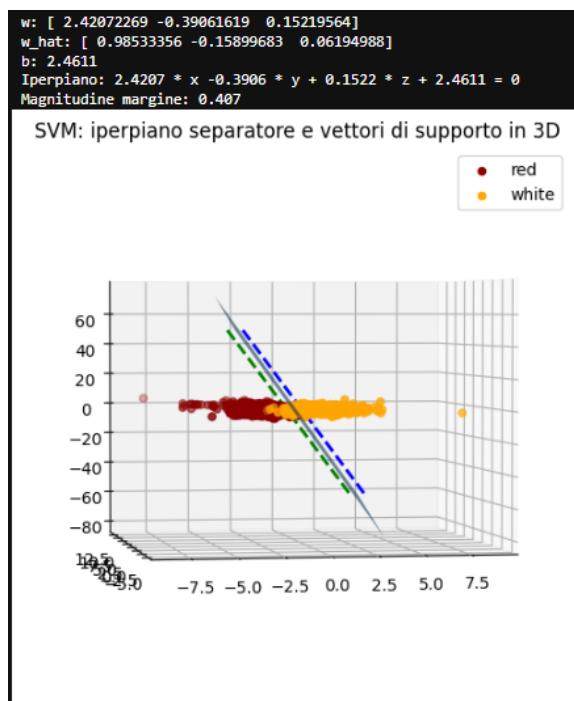


Support Vector Machines

Il secondo modello preso in considerazione consiste in Support Vector Machines. L'utilizzo del modello SVM è stato considerato facendo riferimento a un'intuizione ottenuta durante l'analisi con PCA: osservando lo scatterplot delle prime due componenti (quelle che spiegano la maggior parte della varianza)

mostrato in precedenza, si potrebbe pensare di separare le due classi di vino red e white attraverso un iperpiano separatore. Si sottolinea il fatto che si tratta di un'intuizione, in quanto il modello viene in realtà allenato su cinque dimensioni (cinque componenti ottenuti da feature extraction tramite PCA), ma si può pensare di ottenere buoni risultati essendo che già in due dimensioni vi è una sorta di separazione **lineare** tra le classi. L'approccio utilizzato è sempre quello naive.

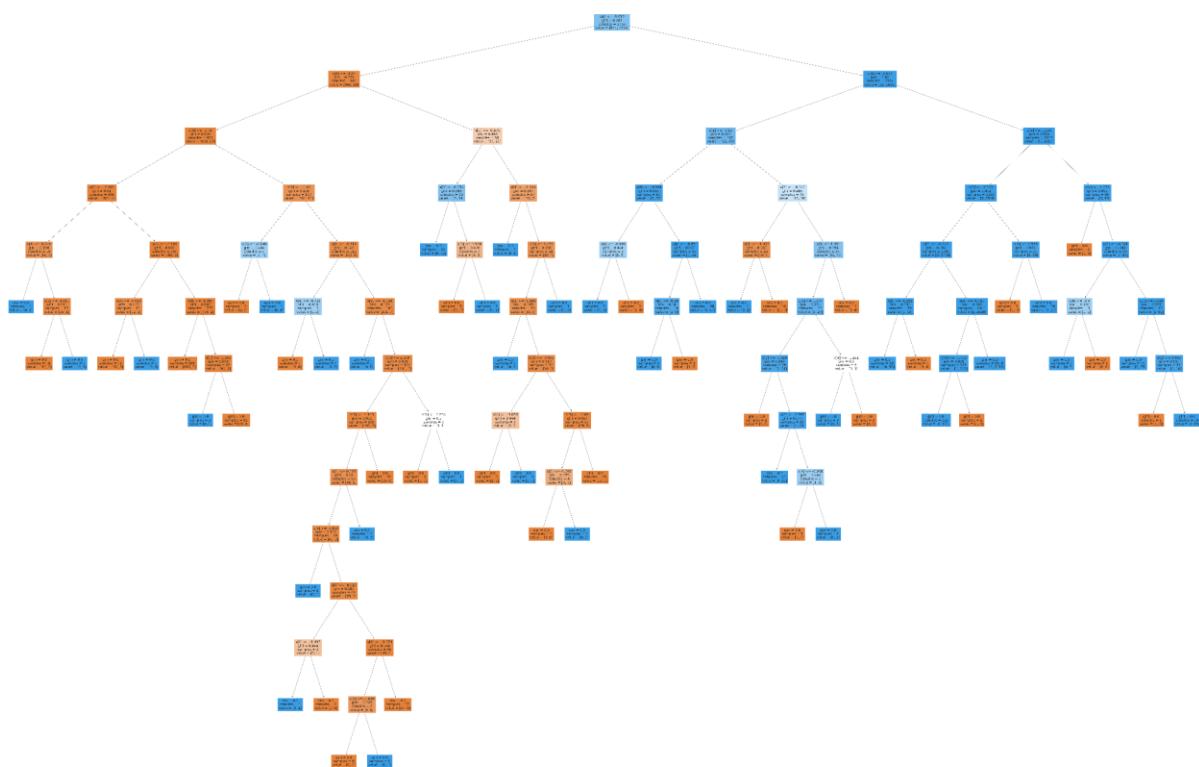
Per l'implementazione è stata utilizzata la libreria **scikit-learn**, nello specifico si crea un Support Vector Classifier per affrontare il task di classificazione binaria. Non vi è la necessità di ricorrere a strategie più avanzate di SVM (come one-vs-all), in quanto il nostro problema non riguarda una classificazione multi-classe, ma binaria.



Alberi di Decisione

Come terzo e ultimo modello è stato scelto l'Albero Decisionale perché è un modello che si presta bene per classificazioni in cui il target assume valori discreti (in particolare qui assume valori binari, concept learning) e per la semplicità e il ridotto tempo di addestramento, rispetto ad altri modelli.

L'approccio utilizzato è stato quello naive, dunque non ponendo alcun limite rispetto alle dimensioni dell'albero in termini di nodi e profondità. L'albero è stato costruito utilizzando la libreria **sklearn** nel modo più semplice possibile, ovvero lasciando gli iperparametri al valore di default.



Misure di Performance

Sono state calcolate misure di performance per ogni modelli:

- **Misure di performance globali:** misure che forniscono una valutazione complessiva delle prestazioni del modello sull'intero insieme di dati di test.
 - **Accuracy:** misura la proporzione di istanze classificate correttamente rispetto al totale delle istanze. È calcolata come il

- rappporto tra il numero di previsioni corrette e il numero totale di previsioni.
- **Precision:** misura la proporzione di tutte le istanze identificate correttamente come positive (veri positivi) rispetto alla somma di tutti i veri positivi e falsi positivi.
 - **Recall:** misura la proporzione di tutte le istanze positive che sono state identificate correttamente dal modello rispetto al numero totale di istanze effettivamente positive.
 - **F1-score:** è la media armonica di precision e recall su tutte le classi.
 - **Stratified 10-Fold Cross Validation:** metodo standard di valutazione, generalmente efficace, per ridurre un possibile overfitting ed aumentare la robustezza del modello. Si sono calcolate le misure descritte in precedenza e gli intervalli di confidenza, dopo averne calcolato la media per ciascuna misura di ciascun fold (divisione dei dati in sottoinsiemi che vengono utilizzati alternativamente come set di addestramento e set di test durante il training). Ogni fold mantiene la stessa distribuzione del target. Inoltre, per migliorare ulteriormente i risultati, si effettua uno shuffling e dunque il risultato può variare in base all'esecuzione.

In seguito riportate le **performance globali** per ogni modello:

Model Name	Accuracy	Precision	Recall	F1 Score
SVM	0.9858	0.9886	0.9923	0.9905
Decision Tree	0.9783	0.9810	0.9898	0.9854

e i relativi **intervalli di confidenza del 95%:**

Model name	Accuracy interval lower	Accuracy interval upper	Precision interval lower	Precision interval upper	Recall interval lower	Recall interval upper	F1 score interval lower	F1 score interval upper
SVM	0.9836	0.9866	0.9772	0.9840	0.9777	0.9829	0.9785	0.9823
Decision Tree	0.9751	0.9826	0.9676	0.9756	0.9660	0.9799	0.9672	0.9773

da notare che i valori sono stati approssimati a quattro cifre dopo la virgola.

Le Dimensioni di Qualità

Le dimensioni di qualità sono uno dei requisiti più importanti da rispettare quando si tratta di un insieme di dati. Rispettare queste dimensioni garantisce che i dati scelti siano attendibili, corretti e precisi, sostanzialmente puliti da eventuali rumori e imprecisioni. Questo processo è fondamentale perché la qualità dei dati influisce direttamente sulla performance e sull'affidabilità dei modelli di Machine Learning.

Per valutare la qualità del dataset di partenza, cioè quello utilizzato per l'addestramento dei modelli nel progetto di Machine Learning, sono state implementate quattro dimensioni di qualità specifiche. Queste dimensioni sono state definite per verificare lo stato del dataset, assicurando che esso sia pulito e privo di anomalie. Le quattro dimensioni di qualità considerate sono nel dettaglio:

- **Completezza:** misura di qualità dei dati che verifica che il dataset sia privo di valori mancanti per ogni feature. Questo significa che ogni campo del dataset contiene dati per tutti i record senza alcuna omissione. Garantire la completezza è fondamentale in quanto i valori mancanti possono compromettere le analisi, portare a conclusioni errate e ridurre la precisione e l'affidabilità dei modelli di Machine Learning. La completezza contribuisce a mantenere l'integrità del dataset, assicurando che tutte le informazioni necessarie siano presenti e utilizzabili.
- **Consistenza:** misura di qualità dei dati che garantisce che il dataset mantenga la coerenza interna e rispetti regole o vincoli predefiniti. Questo implica il controllo di eventuali incoerenze (come un outlier, ovvero un punto che devia notevolmente dal resto dei dati, o per esempio valori non ammissibili) o contraddizioni all'interno del dataset. La consistenza assicura che i dati siano uniformi e coerenti tra loro, ad esempio, rispettando formati standardizzati e relazioni logiche tra i campi. Garantire la consistenza è fondamentale perché le incoerenze nei dati possono portare a errori nelle analisi, risultati fuorvianti e riduzione della credibilità del dataset.
- **Unicità:** misura di qualità dei dati che assicura che ciascun elemento all'interno di un dataset sia unico e non ripetuto. Questo significa che non vi sono duplicati nel dataset e ogni record è esclusivo. Garantire l'unicità è importante perché i dati duplicati possono distorcere le analisi, portare a risultati fuorvianti e compromettere l'integrità complessiva del dataset.
- **Accuratezza:** misura di qualità dei dati che verifica se i dati sono utilizzati correttamente e in modo significativo. Questo implica che i tipi di dati assegnati a ciascuna caratteristica o variabile del dataset corrispondano al

loro significato nel mondo reale. Ad esempio, una variabile che rappresenta una data dovrebbe essere effettivamente di tipo data, e non di tipo stringa o numerico. Garantire la validità è cruciale perché l'uso scorretto dei tipi di dati può portare a errori nelle analisi, interpretazioni sbagliate e risultati imprecisi.

Completezza - Introduzione al Codice

```
#     COMPLETENESS

#     Quality Measure that ensures that the dataset that will be
#     used does not contain any missing values for every feature
#     recorded in the set.
def completeness_test(dataset):
    # Checks if there are any missing values
    missing_values = utils.completeness_verify(dataset)
    # Gets the ratio for each feature of missing values
    completeness_ratio = utils.completeness_ratio(dataset)
    # Return the missing_values and ratio
    return missing_values, completeness_ratio
```

L'implementazione della **Completezza** verifica la presenza dei **missing values** nel dataset, ritornando un'associazione tra la feature e il numero di missing values presenti per quella specifica feature (un conteggio), utilizzando `utils.completeness_verify(dataset)`.

Viene inoltre calcolata anche la percentuale relativa per ogni features con la funzione

`utils.completeness_ratio(dataset)`.

Viene utilizzata la libreria `utils`, che implementa separatamente la logica di ogni controllo, permettendo un codice più ordinato.

I valori ritornati dal test di completezza, come per i test successivi, verranno utilizzati nella pipeline per verificare se la dimensione di qualità è effettivamente rispettata.

In seguito un esempio di output prodotto dal test, dove non vi è nessun valore mancante, a sinistra il conteggio e a destra la percentuale:

```
type          0 type          0.0
fixed acidity 0 fixed acidity 0.0
volatile acidity 0 volatile acidity 0.0
citric acid    0 citric acid   0.0
residual sugar 0 residual sugar 0.0
chlorides      0 chlorides     0.0
free sulfur dioxide 0 free sulfur dioxide 0.0
total sulfur dioxide 0 total sulfur dioxide 0.0
density        0 density       0.0
pH             0 pH           0.0
sulphates      0 sulphates    0.0
alcohol         0 alcohol      0.0
dtype: int64                           dtype: float64
```

Consistenza - Introduzione al Codice

```
#     CONSISTENCY

#     Quality measure that ensures that the dataset used maintains internal coherence
#     and adheres to predefined rules or constraints. This involves checking for any
#     inconsistencies or contradictions within the dataset itself.

def consistency_test(dataset, ranges = feature_ranges):
    # Implement consistency checks based on domain knowledge or specific rules
    inconsistent_values_default = utils.consistency_verify(dataset, ranges)
    # Finds more accurate ranges for another check
    std_bounds = utils.consistency_calculate_bounds_std(dataset, ranges, threshold_std)
    iqr_bounds = utils.consistency_calculate_bounds_iqr(dataset, ranges, threshold_iqr)
    # Checks with calculated bounds
    inconsistent_values_bound_std = utils.consistency_verify(dataset, std_bounds)
    inconsistent_values_bound_iqr = utils.consistency_verify(dataset, iqr_bounds)
    # Finds the outliers for each feature with ranges obtained by mean and std) and iqr
    outliers_std = utils.consistency_outliers(dataset, std_bounds)
    outliers_iqr = utils.consistency_outliers(dataset, iqr_bounds)
    # Return the inconsistent values and outliers
    return inconsistent_values_default, inconsistent_values_bound_std,
inconsistent_values_bound_iqr, outliers_std, outliers_iqr, std_bounds, iqr_bounds
```

L'implementazione della **Consistenza** è la più complessa rispetto alle altre tre dimensioni, e si occupa di:

- Verificare se ogni valore di una feature sia compreso dentro un **range di dominio default** con la funzione `utils.consistency_verify(dataset, ranges)`
- Calcolare i nuovi range stimati, utilizzando due approcci:
 - **Media ± deviazione standard * threshold** (5 di default), utilizzato nella funzione `utils.consistency_calculate_bounds_std(dataset, ranges, threshold_std)`.
 - **Interquartile Range** (calcolato come il terzo quartile $q3$ - primo quartile $q1$) moltiplicato per il threshold (4 di default) e sommato al $q3$ nel caso di upper bound, sottratto $q1$ nel caso di lower bound. La funzione utilizzata per il calcolo è `utils.consistency_calculate_bounds_iqr(dataset, ranges, threshold_iqr)`.
Es. Upper Bound: $q3 + \text{threshold} * \text{IQR}$
 - Ogni lower bound è limitato fino a 0, garantendo solo valori positivi.
- Verificare se ogni valore di una feature sia compreso dentro i due nuovi range calcolati in precedenza sempre utilizzando `utils.consistency_verify(dataset, ranges)`.
- Trovare gli outliers, ovvero i valori al di fuori dei due range calcolati in precedenza ma comunque dentro il **dominio definito dal range di default** con `utils.consistency_outliers(dataset, std / iqr_bounds)`.

Il range di default è definito come segue:

```
# Feature Ranges of the expected values. Any value that it outside the expected range is considered inconsistent
feature_ranges = {
    'fixed acidity': (0, None), # Check for non-negative values
    'volatile acidity': (0, None), # Check for non-negative values
    'citric acid': (0, None), # Check for non-negative values
    'residual sugar': (0, None), # Check for non-negative values
    'chlorides': (0, None), # Check for non-negative values
    'free sulfur dioxide': (0, None), # Check for non-negative values
    'total sulfur dioxide': (0, None), # Check for non-negative values
    'density': (0, None), # Check for non-negative values
    'pH': (0, 14), # Default Range for pH
    'sulphates': (0, None), # Check for non-negative values
    'alcohol': (0, 100), # % Alcohol goes necessarily from 0 to 100%
}
```

mentre i range calcolati possono essere visualizzati in output:

```
=====
CONSISTENCY - STD BOUNDS RESULTS:
=====
fixed acidity: (0.6145579802193915, 13.821457128373623)
volatile acidity: (0, 1.1852054165644146)
citric acid: (0, 1.0543419012093667)
residual sugar: (0, 27.554235702957847)
chlorides: (0, 0.2411949011056968)
free sulfur dioxide: (0, 119.18259086148798)
total sulfur dioxide: (0, 398.0541620677567)
density: (0.9796917405510556, 1.0093805049635807)
pH: (2.4236118482077655, 4.02515869003586)
sulphates: (0, 1.2824521302170306)
alcohol: (4.617486789639095, 16.482820418411897)

=====
CONSISTENCY - IQR BOUNDS RESULTS:
=====
fixed acidity: (1.2000000000000001, 12.899999999999999)
volatile acidity: (0, 1.13)
citric acid: (0, 1.04)
residual sugar: (0, 30.3)
chlorides: (0, 0.1780000000000002)
free sulfur dioxide: (0, 141.0)
total sulfur dioxide: (0, 474.0)
density: (0.9738799999999999, 1.0151000000000001)
pH: (2.229999999999999, 4.210000000000001)
sulphates: (0, 1.279999999999998)
alcohol: (1.8999999999999986, 19.0)
```

I threshold scelti portano un compromesso con range meno stringenti ma complessivamente più accurati ai valori attesi, con i valori di IQR leggermente più flessibili.

In output i relativi esiti per il range di default e due calcolati:

```

=====
CONSISTENCY - INCONSISTENT VALUES DEFAULT RESULTS:
=====
fixed acidity: PASSED
volatile acidity: PASSED
citric acid: PASSED
residual sugar: PASSED
chlorides: PASSED
free sulfur dioxide: PASSED
total sulfur dioxide: PASSED
density: PASSED
pH: PASSED
sulphates: PASSED
alcohol: PASSED

=====
CONSISTENCY - INCONSISTENT VALUES STD BOUNDS RESULTS:
=====
fixed acidity: 8
volatile acidity: 4
citric acid: 2
residual sugar: 2
chlorides: 32
free sulfur dioxide: 7
total sulfur dioxide: 1
density: 2
pH: PASSED
sulphates: 13
alcohol: PASSED

=====
CONSISTENCY - INCONSISTENT VALUES IQR BOUNDS RESULTS:
=====
fixed acidity: 21
volatile acidity: 6
citric acid: 2
residual sugar: 2
chlorides: 67
free sulfur dioxide: 2
total sulfur dioxide: PASSED
density: 1
pH: PASSED
sulphates: 15
alcohol: PASSED

```

Si può notare come la maggior parte delle features abbia dei valori fuori i range stabiliti da IQR e Media Deviazione Standard (outliers).

Nel processo di pipeline è stata introdotta una tolleranza massima del 3% per considerare il test riuscito.

Gli **Outliers** vengono analizzati più attentamente nel seguente output (troncato in quanto molto lungo), dove vengono mostrate le righe e il conteggio preciso di outliers per ogni feature. Ad ogni riga va sommata + 2 quando vi è la necessità di verifica nel file .csv del dataset.

```
=====
CONSISTENCY - OUTLIERS STD INFO RESULTS:
=====
Feature: fixed acidity
Count of outliers: 8
Rows with outliers: [1273, 4156, 4270, 4327, 4415, 4425, 4427, 4507]

Feature: volatile acidity
Count of outliers: 4
Rows with outliers: [4056, 4057, 4524, 5048]

Feature: citric acid
Count of outliers: 2
Rows with outliers: [615, 2568]

Feature: residual sugar
Count of outliers: 2
Rows with outliers: [1378, 2278]
```

Unicità - Introduzione al Codice

```
#      UNIQUENESS

#      Quality measure that guarantees the dataset's elements are distinct,singular,
#      devoid of any redundant or repeated entries. It involves confirming that each
#      data point within the dataset stands out on its own, contributing uniquely to
#      the dataset's richness and integrity.

def uniqueness_test(dataset):
    # Counts unique values for each feature
    unique_counts = utils.uniqueness_verify_unique(dataset)
    # Finds any duplicate data with their row and sum
    duplicate_counts, duplicate_sum = utils.uniqueness_verify_duplicates(dataset)
    # Gets the ratio of unique values
    unique_ratio = utils.uniqueness_ratio(dataset)
    return unique_counts, duplicate_counts, duplicate_sum, unique_ratio
```

L'implementazione della **Unicità** verifica la distribuzione dei valori unici per ogni feature, con la relativa percentuale, attraverso l'utilizzo delle funzioni `utils.uniqueness_verify_unique(dataset)` e `utils.uniqueness_ratio(dataset)`.

Viene inoltre controllata la presenza di dati duplicati per ogni record del dataset, contati e segnalati in una lista con `utils.uniqueness_verify_duplicates(dataset)`

In seguito l'output della distribuzione dei valori unici e la relativa percentuale:

type	2	type	0.037771
fixed acidity	106	fixed acidity	2.001889
volatile acidity	187	volatile acidity	3.531634
citric acid	89	citric acid	1.680831
residual sugar	315	residual sugar	5.949008
chlorides	214	chlorides	4.041549
free sulfur dioxide	135	free sulfur dioxide	2.549575
total sulfur dioxide	276	total sulfur dioxide	5.212465
density	996	density	18.810198
pH	108	pH	2.039660
sulphates	111	sulphates	2.096317
alcohol	111	alcohol	2.096317
dtype: int64		dtype: float64	

e anche i test sui valori duplicati, i quali non risultano presenti in quanto il dataset è pulito:

Duplicated Records: 0

Si può notare come la feature che differenzia di più nella composizione di un vino è la **density** per il dataset trattato.

Accuratezza - Introduzione al Codice

```
# ACCURACY

# Quality measure that verifies if the data is actually correctly
# used and considered. In this case, it's important that the types
# assigned to each feature reflects its meaning in the real world

def accuracy_test(dataset, types = expected_types):
    # Verifies the correct type for each feature
    accuracy_results = utils.accuracy_verify(dataset, types)
    return accuracy_results
```

L'implementazione dell'**Accuratezza** verifica se i dati trattati sono riconosciuti con il tipo corretto. La funzione `utils.accuracy_verify(dataset,types)` verifica, data una lista di tipi corretti, se le features sono trattate accuratamente.

I tipi contenuti in “types” sono i seguenti:

```
# Feature: type that has to be respected
expected_types = {
    'fixed acidity': 'float64',
    'volatile acidity': 'float64',
    'citric acid': 'float64',
    'residual sugar': 'float64',
    'chlorides': 'float64',
    'free sulfur dioxide': 'float64',
    'total sulfur dioxide': 'float64',
    'density': 'float64',
    'pH': 'float64',
```

```
'sulphates': 'float64',
'alcohol': 'float64',
'type': 'bool' # Target is true or false
}
```

Il dataset contiene solamente dati continui, escludendo il target binario, quindi basta che siano considerati come dei *float64*, mentre il target come valore booleano.

In seguito un output del test di accuratezza, dove la dicitura “PASSED” indica se il tipo trattato per la corrispettiva feature è effettivamente corretto:

```
fixed acidity: PASSED
volatile acidity: PASSED
citric acid: PASSED
residual sugar: PASSED
chlorides: PASSED
free sulfur dioxide: PASSED
total sulfur dioxide: PASSED
density: PASSED
pH: PASSED
sulphates: PASSED
alcohol: PASSED
type: PASSED
```

Il Nostro Approccio

Dal punto di vista implementativo, inizialmente si pensava di ricorrere a un unico Jupyter notebook, analogamente al progetto di Machine Learning.

Tuttavia, questo progetto ha bisogno di un grado più alto di scalabilità e di **automatizzazione**, dunque abbiamo deciso di **non utilizzare Google Colab**.

Tra le motivazioni più importanti abbiamo:

- minimizzazione del rischio di bug dovuti a una cattiva organizzazione del progetto (i.e. copy-paste)
- esecuzione automatica di un **numero elevato di esperimenti**
 - con possibilità di dividersi il carico di esecuzione all'interno del team
- aumento della produttività in team, avendo un progetto non monolitico
 - si possono utilizzare branch diversi su GitHub senza creare conflitti in fase di merge

Il nostro approccio non esclude l'utilizzo dei Jupyter notebooks; tuttavia, essi vengono utilizzati in maniera più consapevole rispetto alle potenzialità di altri tool.

I due tool utilizzati principalmente per lo sviluppo del progetto sono

- **Pipeline Python** (libreria Luigi)
 - di importanza vitale per l'esecuzione degli esperimenti
- **Jupyter notebooks**
 - necessari per la visualizzazione dei dati in modo organizzato

Background Pipeline Luigi

Prima di trattare ciascuna componente del progetto, è bene dare brevemente un'introduzione sulle pipeline Luigi.

Come già accennato, al fine di **automatizzare** sia la prima parte del progetto contrassegnata con **ML** (i.e. dal dataset scaricato da Kaggle alla valutazione dei modelli naive di Machine Learning, includendo anche il check delle dimensioni di qualità) sia la parte fondamentale del progetto in cui si effettuano diversi **esperimenti** (ciascun esperimento a sua volta presenta diverse istanze, i.e. parametri diversi) per valutare le variazioni delle performance dei modelli in relazione alla qualità del training set (Data Quality), è stato scelto di sfruttare al massimo la potenzialità della libreria Luigi.

Luigi è una libreria Python costruita ed utilizzata internamente da Spotify per eseguire migliaia di task ogni giorno (ma anche da centinaia di aziende), che permette di creare **pipeline** complesse.

Tra le diverse funzionalità offerte da Luigi si hanno

- la gestione delle dipendenze tra task (i.e. dependency graphs)
- la gestione del workflow

Il tutto è semplificato da un'integrazione command line che richiede codice di alto livello: si evita di perdere tempo sugli aspetti di basso livello e ci si concentra sui task e le loro dipendenze.

Ci sono due “mattoni” fondamentali da tenere in considerazione nel momento in cui si crea una pipeline Luigi:

- **Target**
 - tipicamente si fa riferimento ad un file su disco (LocalTarget)
 - si hanno diverse implementazioni disponibili
 - inoltre, è possibile implementare un nuovo tipo di Target in Python, usando la propria logica
- **Task**
 - concettualmente più interessante, è dove viene svolta la computazione
 - richiede l'implementazione di alcune funzioni per definire il comportamento di un task, tra i più importanti:
 - `requires`: usato per definire le dipendenze
 - `run`: contiene la logica eseguita dal task
 - `output`: consente di definire gli output

I task dipendono l'uno dall'altro e dai target di output: se un task ha dipendenze e i target di output delle dipendenze non esistono, allora vengono eseguite prima le dipendenze, per poi eseguire il task in sé.

Di solito l'esecuzione di un task ha bisogno del passaggio di parametri da riga di comando. Luigi supporta in maniera intuitiva la parametrizzazione, tramite **Parameter** e le diverse implementazioni (e.g. `FloatParameter`, `ListParameter`, ...), dunque non vi è bisogno di effettuare il parsing per supportare tipi diversi di parametri.

Naturalmente, nel caso in cui non sia richiesto obbligatoriamente di specificare il valore di un parametro, viene utilizzato il suo valore di default.

Si può raggiungere un livello ancora più alto di personalizzazione della pipeline combinando la flessibilità dei parametri di Luigi con un file di configurazione, in

cui ad esempio si possono indicare i valori di default senza modificare il codice sorgente.

Struttura del Progetto

Prima di trattare le parti cruciali da cui è composto il progetto (i.e. le pipeline e i notebooks), di seguito se ne presenta la struttura, in termini di filesystem.

Il progetto presenta la seguente **struttura dettagliata**:

Progetto-Archi/

```
└── experiments_pipeline/: directory che contiene la parte di esperimenti.  
    ├── datasets/: contiene i file iniziali per la pipeline degli esperimenti.  
    │   ├── winetype_pca_test.csv: test set (20%) creato dopo l'utilizzo della PCA.  
    │   └── winetype_pca_train.csv: training set creato dopo l'utilizzo della PCA.  
    ├── experiments/: directory utilizzata dalla pipeline degli esperimenti, contiene a  
    │   sua volta una directory per ogni esperimento (i.e. per ogni  
    │   esecuzione della pipeline), con i relativi output.  
    ├── utils/: package Python, i moduli all'interno contengono funzioni utilizzate  
    │   dalla pipeline degli esperimenti, per manipolare il training set.  
    │   ├── __init__.py: file vuoto che indica che la directory attuale è un package.  
    │   ├── features_utils.py: modulo con funzioni che manipolano le features.  
    │   ├── label_utils.py: modulo con funzioni che manipolano il target.  
    │   ├── more_rows_utils.py: modulo con funzioni che aggiungono righe.  
    │   └── aggregated_metrics.csv: performance dei modelli in 659 esperimenti.  
    ├── experiments_config.json: contiene i parametri di ciascun esperimento,  
    │   fondamentale per eseguirli tutti in automatico.  
    ├── generate_json.py: script che genera il JSON appena trattato.  
    ├── luigi.cfg: file di configurazione associato alla pipeline degli esperimenti,  
    │   contiene diversi valori di default dei parametri.  
    ├── luigi.log: file di log creato dalla pipeline degli esperimenti.  
    ├── pipeline.py: codice sorgente della pipeline degli esperimenti  
    │   (chiamata anche “seconda pipeline”).  
    └── run_experiments.py: script che esegue tutti gli esperimenti con i parametri  
        definiti nel JSON.  
    └── imgs/: directory che contiene le immagini delle pipeline.  
        ├── Prima_pipeline_architetture_dati.png: architettura della prima pipeline.  
        └── Seconda_pipeline_architetture_dati.png: architettura della seconda pipeline.  
    └── ml_pipeline/: directory che contiene la prima parte  
        (i.e. ML e dimensioni di qualità).  
        ├── datasets/: contiene il dataset originale e alcuni output della prima pipeline.  
        │   ├── winetype.csv: dataset scaricato da Kaggle.  
        │   ├── winetype_cleaned.csv: dataset “intermedio” generato dalla prima pipeline,  
        │   │   dopo il preprocessing.  
        │   ├── winetype_transformed.csv: dataset “intermedio” generato dalla prima  
        │   │   pipeline, dopo aver trasformato le features.  
        │   ├── winetype_pca.csv: dataset generato dalla prima pipeline, dopo la PCA.  
        │   └── winetype_pca_test.csv: test set (20%) creato dopo l'utilizzo della PCA.
```

```

    └── winetype_pca_train.csv: training set creato dopo l'utilizzo della PCA.
    └── models/: contiene i modelli di Machine Learning salvati su file, fanno
        parte degli output della prima pipeline.
        ├── svm_model.pkl: file Pickle della SVM addestrata.
        └── dtc_model.pkl: file Pickle del Decision Tree addestrato.
    └── performance/: contiene le metriche dei 3 modelli addestrati.
        └── metrics.csv: metriche globali (accuracy, precision, recall, f1 score) e
            relativi intervalli di confidenza 95% ottenuti dalla
            Stratified 10-Fold Cross Validation.
    └── utils/: package Python, i moduli all'interno contengono funzioni utilizzate
        dalla prima pipeline.
        ├── __init__.py: file vuoto che indica che la directory attuale è un package.
        ├── cross_validation.py: modulo con funzioni per Stratified 10-Fold CV.
        ├── data_quality.py: modulo con funzioni che implementano controlli per le
            dimensioni di Data Quality sul dataset trasformato.
        ├── data_quality_utils.py: modulo con funzioni per il modulo appena trattato.
        ├── evaluation.py: modulo con funzioni per la performance evaluation.
        └── predictions.py: modulo con funzioni per ottenere predizioni binarie.
    └── luigi.cfg: file di configurazione associato alla prima pipeline,
        contiene diversi valori di default dei parametri.
    └── luigi.log: file di log creato dalla prima pipeline.
    └── pipeline.py: codice sorgente della pipeline di Machine Learning che include
        le dimensioni di qualità (chiamata anche "prima pipeline").
    └── notebooks/: directory che contiene i Jupyter notebooks.
        ├── experiments.ipynb: Jupyter notebook che visualizza i risultati ottenuti dagli
            esperimenti.
        └── explainability.ipynb: Jupyter notebook che visualizza sia la parte di Machine
            Learning sia una parte aggiuntiva sull'Explainability.
    └── venv/: directory generata con la creazione del Virtual Environment.
    └── README.md: contiene istruzioni su come eseguire il progetto ("How to Run").
    └── requirements.txt: dipendenze Python, librerie necessarie per il venv.

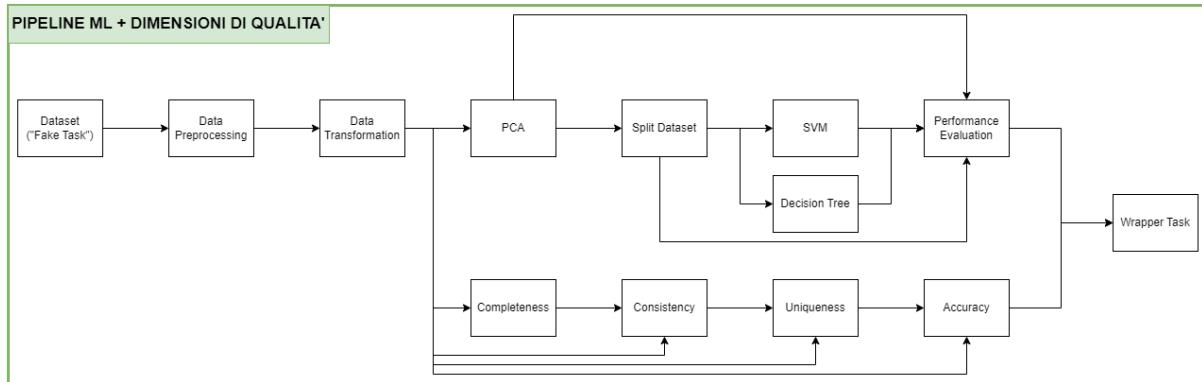
```

Pipeline Machine Learning

In questa sezione si mostrano i dettagli della prima pipeline, ossia la pipeline contenente la parte di Machine Learning e i controlli di Data Quality.

Architettura Prima Pipeline

Innanzitutto viene presentata l'architettura della pipeline, che permette di capire quali sono i task e le dipendenze tra di loro



Gli **archi entranti** di un task sono le **dipendenze** di quel task.

Si può notare che i task "Completeness", "Consistency", "Uniqueness" e "Accuracy" sono indipendenti da "Performance Evaluation" (non esiste un cammino che li collega), motivo per cui esiste un "Wrapper Task" che dipende da entrambi e dunque richiede l'esecuzione di tutta la pipeline.

Riguardo alle dipendenze, si può fare un'ulteriore osservazione: a differenza degli altri task Luigi, i task "Completeness", "Consistency", "Uniqueness" e "Accuracy" non producono alcun file in output, dunque a livello implementativo non si utilizza LocalTarget ma un Target custom ("InMemoryTarget") definito nel codice, basato su un flag in memoria.

Questo mostra ancora una volta quanto sia utile la flessibilità della libreria Luigi, in grado di adattarsi a qualsiasi situazione.

Tasks Prima Pipeline

Viene trattato ciascun task singolarmente, per capire a fondo il funzionamento dell'intera pipeline tramite l'analisi dei singoli task da cui è composta.

“Fake Task”

Il task “Fake Task” è chiamato in questo modo perché in realtà non ha una logica (funzione run): esiste solo per inserire nel task successivo la dipendenza dall'**esistenza del file csv** contenente il dataset originale da Kaggle.

Viene riportato lo snippet di codice per il “Fake Task”:

```
class FakeTask(luigi.Task):
    def output(_):
        return luigi.LocalTarget(self.input_csv)
```

Data Preprocessing

Si tratta del primo task vero e proprio: come accennato, la **dipendenza** (funzione `requires`) è associata al target di output di “**Fake Task**”, dunque è richiesta l’esistenza del dataset originale per poter eseguire questo task.

Il task presenta i seguenti parametri:

- `input-csv`: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
- `cleaned-csv`: percorso al csv di output con i dati preprocessati
 - di default si ha “datasets/winetype_cleaned.csv”

Nella logica del task (funzione `run`) si effettuano le seguenti operazioni:

- si recupera il dataset di Kaggle
- si **eliminano** le righe con **valori mancanti**
- si **eliminano** le **righe duplicate**
- si memorizza il nuovo dataset (preprocessato) nel file di output del task

L’output del task (funzione `output`) di default è il file **winetype_cleaned.csv**, dunque qui è possibile utilizzare `LocalTarget` (così come negli altri task che producono file).

Data Transformation

Questo task ha la **dipendenza** dal task precedente, cioè “**Data Preprocessing**”, poiché si parte dal dataset preprocessato e si applicano ulteriori operazioni.

Il task presenta i seguenti parametri:

- `input-csv`: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- `transformed-csv`: percorso al csv di output con i dati trasformati
 - di default si ha “datasets/winetype_transformed.csv”

Si effettuano le seguenti operazioni:

- si recupera il dataset preprocessato
- si effettua **Label Encoding** sul target (`red` → `False`, `white` → `True`)
- si effettua il cast della feature “`quality`” a categorica
- si effettua il **drop della feature “quality”** (il motivo si trova alla fine di [questo](#))
- si memorizza il nuovo dataset (trasformato) nel file di output del task

L’output del task di default è il file **winetype_transformed.csv**.

PCA

Questo task ha la **dipendenza** dal task precedente, cioè “**Data Transformation**”, poiché si parte dal dataset trasformato e si applicano ulteriori operazioni.

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- pca-csv: percorso al csv di output con i dati dopo aver applicato PCA
 - di default si ha “datasets/winetype_pca.csv”

Si effettuano le seguenti operazioni:

- si recupera il dataset trasformato
- si effettua la **standardizzazione** delle features numeriche (escludendo il target)
- si applica **PCA** mantenendo **5 componenti**
- si memorizza il nuovo dataset (PCA) nel file di output del task

L’output del task di default è il file **winetype_pca.csv**.

Split Dataset

Questo task ha la **dipendenza** dal task precedente, cioè “**PCA**”, poiché si parte dal dataset della PCA e si applicano ulteriori operazioni.

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- train-csv: percorso al primo csv di output, con il training set (dopo PCA)
 - di default si ha “datasets/winetype_pca_train.csv”
- test-csv: percorso al secondo csv di output, con il test set (dopo PCA)
 - di default si ha “datasets/winetype_pca_test.csv”

Si effettuano le seguenti operazioni:

- si recupera il dataset su cui è stata applicata la PCA
- si effettua la **divisione in training set (80%) e test set (20%)**
- si memorizzano training set e test set nei rispettivi file di output del task

L’output del task di default si compone dai file **winetype_pca_train.csv** e **winetype_pca_test.csv**.

SVM

Questo task ha la **dipendenza** da “Split Dataset”, poiché si parte dal training set e si addestra una SVM.

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- svm-model-file: percorso al file di output, con la SVM addestrata
 - di default si ha “models/svm_model.pkl”

Si effettuano le seguenti operazioni:

- si recupera il training set
- si costruisce una **SVM naive**
- si **addestra** la SVM sul training set
- si memorizza il modello nel file di output del task

L’output del task di default è il file **svm_model.pkl**.

Decision Tree

Questo task ha la **dipendenza** da “Split Dataset”, poiché si parte dal training set e si addestra un Decision Tree Classifier.

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- dtc-model-file: percorso al file di output, con l’Albero Decisionale addestrato
 - di default si ha “models/dtc_model.pkl”

Si effettuano le seguenti operazioni:

- si recupera il training set
- si costruisce un **Albero Decisionale naive**
- si **addestra** l’Albero Decisionale sul training set
- si memorizza il modello nel file di output del task

L’output del task di default è il file **dtc_model.pkl**.

Performance Evaluation

Questo task ha la **dipendenza** da

- “PCA” perché in Cross Validation è necessario l’intero dataset
- “SVM” e “Decision Tree” per effettuare performance evaluation sui modelli addestrati precedentemente
- “Split Dataset” perché per le metriche globali è necessario avere il test set

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente
- metrics-csv: percorso al file di output, con le metriche misurate sui modelli
 - di default si ha “performance/metrics.csv”

Si effettuano le seguenti operazioni:

- si recupera il dataset su cui è stata applicata la PCA
- si recupera il test set
- si recuperano i modelli addestrati precedentemente
- per ciascun modello (SVM e Albero Decisionale)
 - si calcolano le **metriche globali**
 - si calcolano gli **intervalli di confidenza 95%** in **Stratified 10-Fold CV**
- si memorizzano tutte le metriche nel file di output del task

L’output del task di default è il file **metrics.csv**.

Completeness

Questo task ha la **dipendenza** da “**Data Transformation**”, poiché si parte dai dati trasformati e si verifica la presenza di valori mancanti.

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente

Si effettuano le seguenti operazioni:

- si recupera il dataset trasformato
- si effettua il controllo sulla [completezza](#)
 - se il **numero di valori mancanti** è **minore** di un threshold (settato di default al **3%** della lunghezza del dataset, cioè **158**) allora il controllo passa e il flag del target (“InMemoryTarget”) viene settato a True, dunque il **task risulta completato**
 - altrimenti il task non risulta completato e perciò la pipeline fallisce

Come accennato, questo task **non produce alcun file**, dunque viene sfruttato un target custom **InMemoryTarget**, in modo tale che il task risulta completato se e solo se il flag del target è settato a True.

Consistency

Questo task ha la **dipendenza** da

- “**Data Transformation**” poiché si parte dai dati trasformati e si verifica la presenza di outlier e valori inconsistenti
- “**Completeness**” per avere un’esecuzione sequenziale dei controlli, e dunque richiedere il completamento dei controlli precedenti

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente

Si effettuano le seguenti operazioni:

- si recupera il dataset trasformato
- si effettua il controllo sulla consistenza
 - se il **numero di valori fuori dal range di default** è **minore** di **158**
 - e il **numero di valori fuori dal range calcolato** col metodo **Mean & Std** è **minore** di **158**
 - e il **numero di valori fuori dal range calcolato** col metodo **IQR** è **minore** di **158**, allora complessivamente il controllo passa e il flag del target (“InMemoryTarget”) viene settato a True, dunque il **task risulta completato**
 - in tutti gli altri casi (i.e. una condizione è falsa) il task non risulta completato e perciò la pipeline fallisce

Come accennato, nemmeno questo task produce alcun file, dunque anche qui viene sfruttato un target custom **InMemoryTarget**.

Uniqueness

Questo task ha la **dipendenza** da

- “**Data Transformation**” poiché si parte dai dati trasformati e si verifica la presenza di valori duplicati
- “**Consistency**” per avere un’esecuzione sequenziale dei controlli, e dunque richiedere il completamento dei controlli precedenti

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente

Si effettuano le seguenti operazioni:

- si recupera il dataset trasformato
- si effettua il controllo sulla unicità
 - se **ogni colonna assume almeno due valori** (i.e. aggiunge informazione)
 - e il **numero di righe duplicate è minore di 158**, allora complessivamente il controllo passa e il flag del target (“InMemoryTarget”) viene settato a True, dunque il **task risulta completato**
 - in tutti gli altri casi (i.e. una condizione è falsa) il task non risulta completato e perciò la pipeline fallisce

Come accennato, nemmeno questo task produce alcun file, dunque anche qui viene sfruttato un target custom **InMemoryTarget**.

Accuracy

Questo task ha la dipendenza da

- **“Data Transformation”** poiché si parte dai dati trasformati e si verifica che ogni colonna del dataset abbia il tipo corretto
- **“Uniqueness”** per avere un’esecuzione sequenziale dei controlli, e dunque richiedere il completamento dei controlli precedenti

Il task presenta i seguenti parametri:

- input-csv: percorso al csv contenente i dati originali
 - di default si ha “datasets/winetype.csv”
 - necessario per gestire le dipendenze correttamente

Si effettuano le seguenti operazioni:

- si recupera il dataset trasformato
- si effettua il controllo sulla accuratezza
 - se **ogni colonna ha il tipo corretto**, allora il controllo passa e il flag del target (“InMemoryTarget”) viene settato a True, dunque il **task risulta completato**
 - altrimenti il task non risulta completato e perciò la pipeline fallisce

Come accennato, nemmeno questo task produce alcun file, dunque anche qui viene sfruttato un target custom **InMemoryTarget**.

Wrapper Task

Si è giunti all'ultimo task della prima pipeline.

Come già accennato, questo task esiste perché per eseguire tutta la pipeline occorre sottolineare la prima osservazione fatta: i task “Completeness”, “Consistency”, “Uniqueness” e “Accuracy” sono indipendenti da “Performance Evaluation”, dunque non è possibile eseguire tutta la pipeline lanciando Luigi una volta sola (i.e. un solo comando).

Perciò, per aggiungere questa possibilità, è necessario avere un task che “avvolge” (wrap) l'ultimo task di entrambi i cammini nel dependency graph, ossia serve richiedere la dipendenza da “Accuracy” e “Performance Evaluation”.

In questo modo, eseguendo il “Wrapper Task” viene controllato che ogni task risulti completato, essendo che non esistono più task indipendenti.

Viene riportato lo snippet di codice per il “Wrapper Task”:

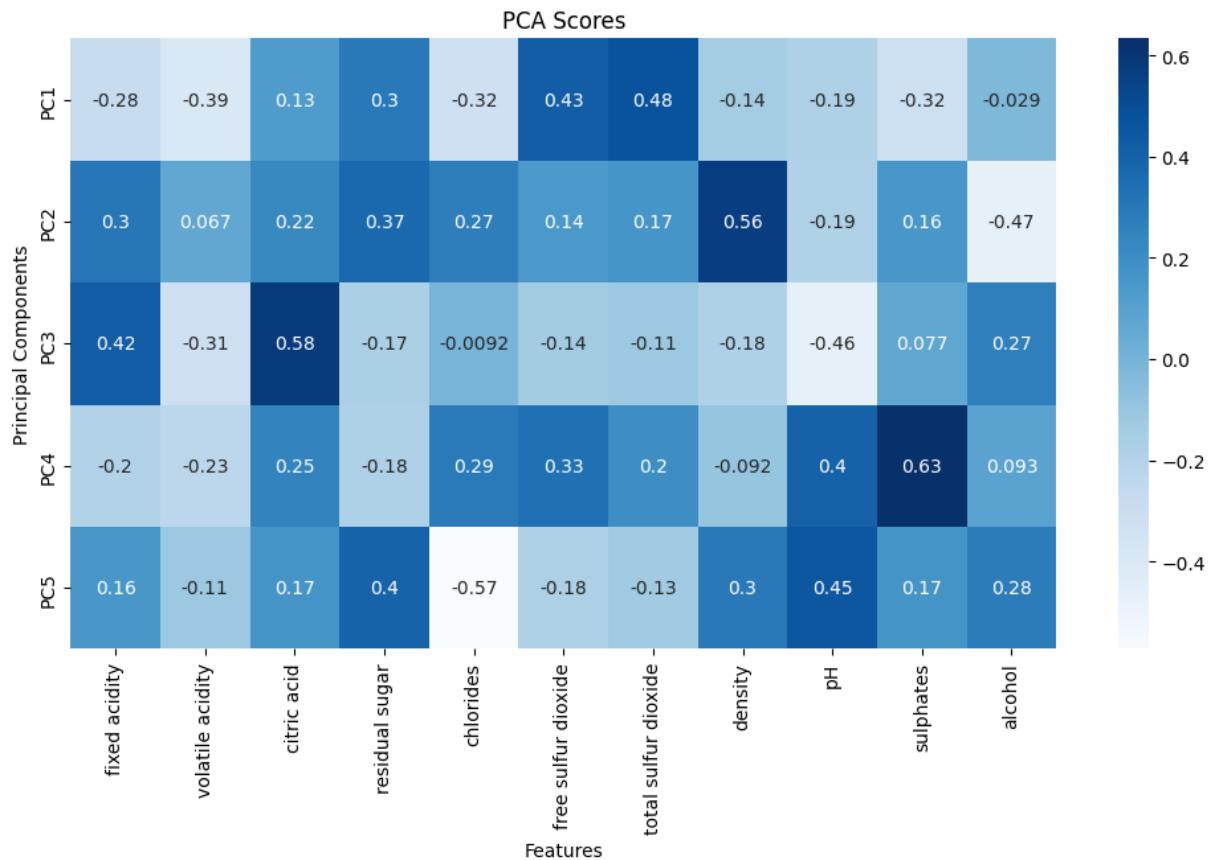
```
class FullPipeline(luigi.WrapperTask):
    def requires(self):
        return [Accuracy(), PerformanceEval()]
```

Data Explainability

Dopo aver terminato di spiegare come la pipeline è strutturata e quali task svolge possiamo incominciare ad introdurre la parte di Explainability. Qui cercheremo di spiegare nella maniera più esaustiva possibile quali componenti della PCA vanno ad influire maggiormente di altre per la classificazione di vini nei modelli presi in esame e per ogni modello verranno fornite spiegazioni sul comportamento di classificazione mediante l'utilizzo di grafici. Osservando la tabella sottostante si può notare - come è normale che sia - che le cinque componenti della PCA non spiegano la stessa percentuale di varianza cumulativa, quindi è ragionevole supporre che le prime componenti avranno un peso maggiore nel determinare l'output del modello rispetto le ultime, non è detto che per tutti i modelli le features avranno la stessa importanza.

	Eigenvalue	Variance Percent	Cumulative Variance Percent
Comp 1	2.991077	27.186472	27.186472
Comp 2	2.476404	22.508515	49.694986
Comp 3	1.585096	14.407246	64.102232
Comp 4	0.953458	8.666165	72.768397
Comp 5	0.742378	6.747617	79.516014

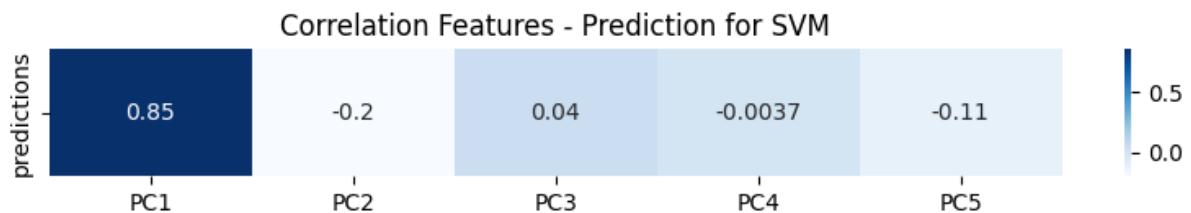
Una prima intuizione che abbiamo avuto per giustificare le scelte fatte dai nostri modelli è capire quali features del dataset originale vanno ad influire maggiormente nella composizione di ogni singola componente della PCA. I risultati sono osservabili nel grafico sottostante.



- PC1 è maggiormente influenzata positivamente dalle features “free sulfur dioxide”, “total sulfur dioxide” e “residual sugar”, per quanto riguarda l’influenza negativa osserviamo “fixed acidity”, “volatile acidity” e “residual sugar”.
- PC2 è maggiormente influenzata positivamente dalle features "density", “fixed acidity” e “residual sugar”, per quanto riguarda l’influenza negativa osserviamo solamente la feature “alcohol”.
- PC3 è maggiormente influenzata positivamente dalle features “citric acid”, “fixed acidity” e “residual sugar”, per quanto riguarda l’influenza negativa osserviamo “volatile acidity” e “pH”.
- PC4 è maggiormente influenzata positivamente dalle features “pH” e “sulphates”, per quanto riguarda l’influenza negativa osserviamo che un minimo contributo lo portano “volatile acidity” e “fixed acidity”.
- PC5 è maggiormente influenzata positivamente dalle features “pH” e “residual sugar”, per quanto riguarda l’influenza negativa osserviamo “chlorides”.

Analizzate le influenze delle varie features sulle componenti della PCA possiamo ora analizzare separatamente per ogni modello quali features (della PCA) hanno un peso maggiore per la classificazione del vino preso in esame. Verrà effettuato inoltre anche uno studio sull'Explainability di ogni modello.

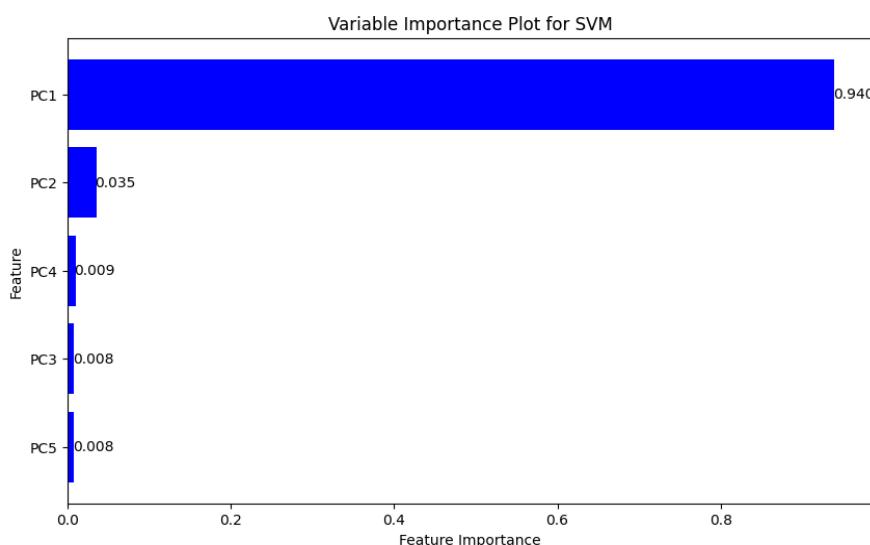
Support Vector Machines



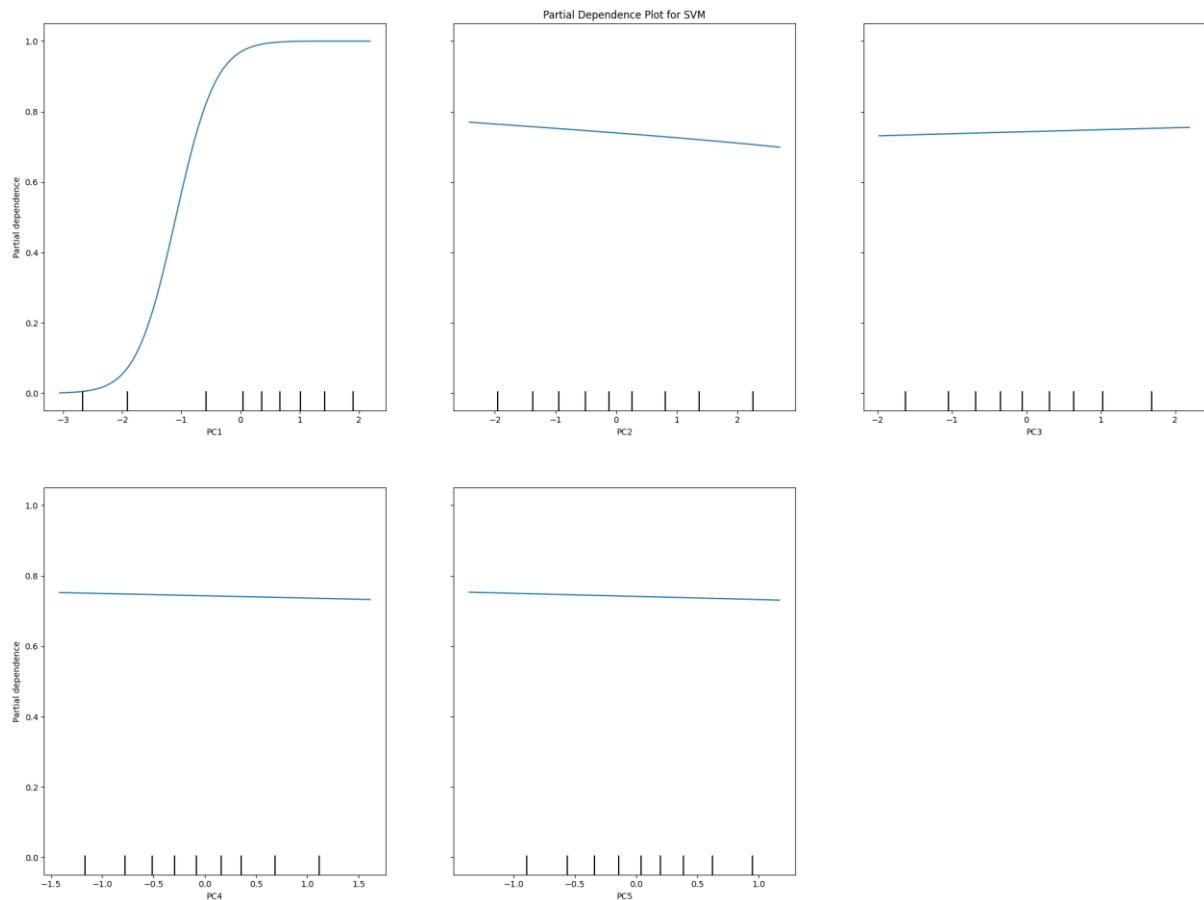
PC1 ha un'altissima correlazione positiva con la predizione del target. Osservando la tabella concludiamo che PC1 influenza in maniera importante il modello nella determinazione del target, PC2 e PC5 hanno una bassa correlazione negativa e contribuiscono in maniera minima al processo di classificazione, le restanti colonne sono invece trascurabili.

Confrontando i valori di correlazione delle tre features più importanti dei modelli finora presentati notiamo che sono del tutto simili senza differenze statisticamente significative, pertanto ci aspettiamo che le metriche di performance presentino risultati simili.

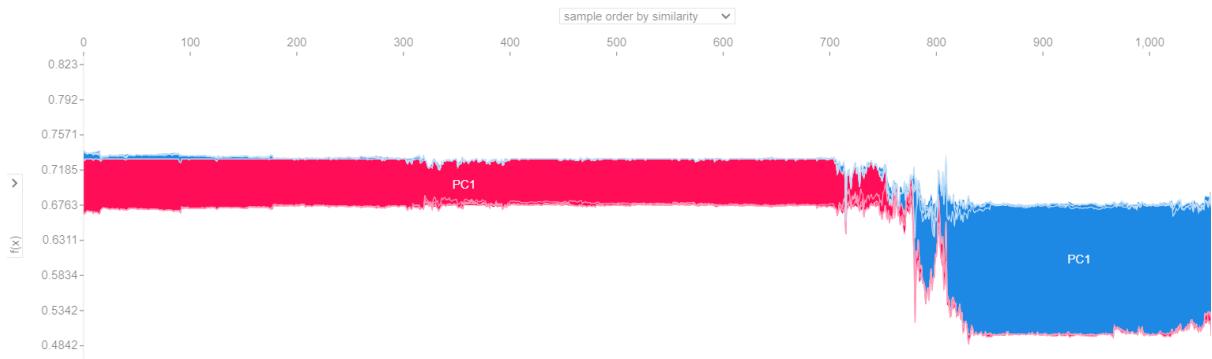
Guardando il grafico di variable importance troviamo coerenza con i risultati ottenuti dal grafico di correlation features, PC1 ha un peso altissimo nel processo di classificazione dell'istanza, PC2 contribuisce marginalmente, PC3, PC4 e PC5 invece forniscono un contributo trascurabile. È utile notare come sebbene PC5 abbia una correlazione maggiore in senso assoluto di PC4 in definitiva influisce meno nella determinazione del tipo di vino.



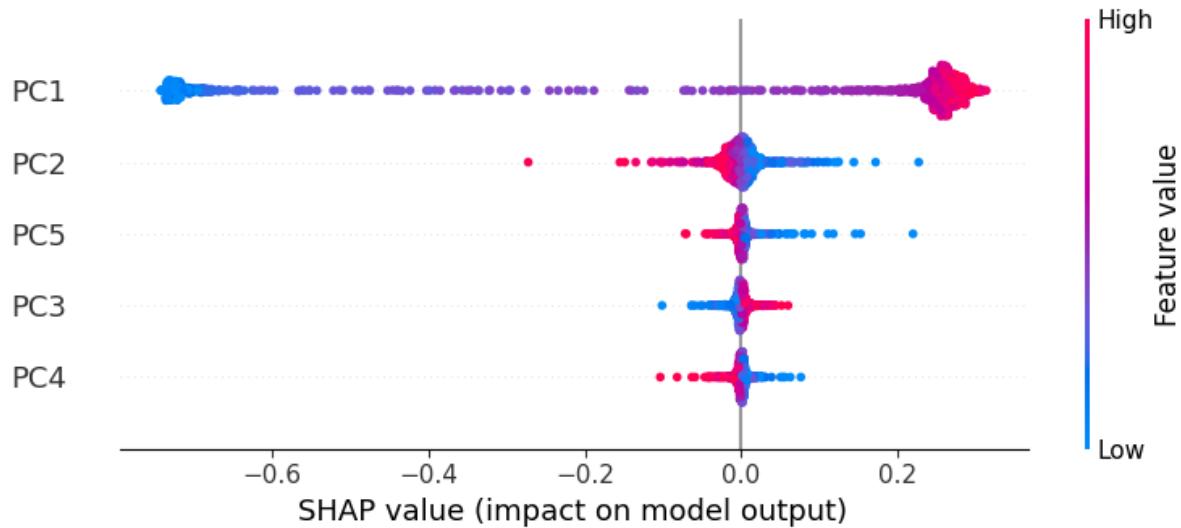
Osservando il grafico sottostante confermiamo che quanto affermato prima nei due grafici precedenti trova riscontro. PC1 è la componente che quasi esclusivamente influisce sul target, per come abbiamo progettato la SVM utilizzata vengono ritornate le probabilità di una determinata istanza di essere un vino bianco oppure rosso, assumendo di arrotondare i valori ottenuti con una threshold fissata a 0.5 quando il valore sarà minore o uguale a -1 il modello classificherà l'istanza corrente in 0 ossia vino rosso, quando, invece, il valore sarà maggiore di 0.5 (il valore effettivo di arrotondamento è vicino a 0.5, è stata usata la funzione round di numpy) l'istanza verrà classificata in 1 ossia vino bianco. PC2 e PC3 influiscono in maniera marginale, PC4 e PC5 invece, non influiscono in maniera significativa, si noti che le linee sono molto simili, praticamente identiche ad una linea parallela all'asse ascisse.



In rosso sono mostrate le istanze di vini bianchi, mentre in blu troviamo le istanze di vini rossi. Passando con il mouse sulle aree rosse (oppure blu) viene mostrato lo SHAP value di ogni colonna della PCA, scorrendo da sinistra verso destra noteremo che i valori di ogni colonna sono in accordo con quanto affermato nel grafico precedente.

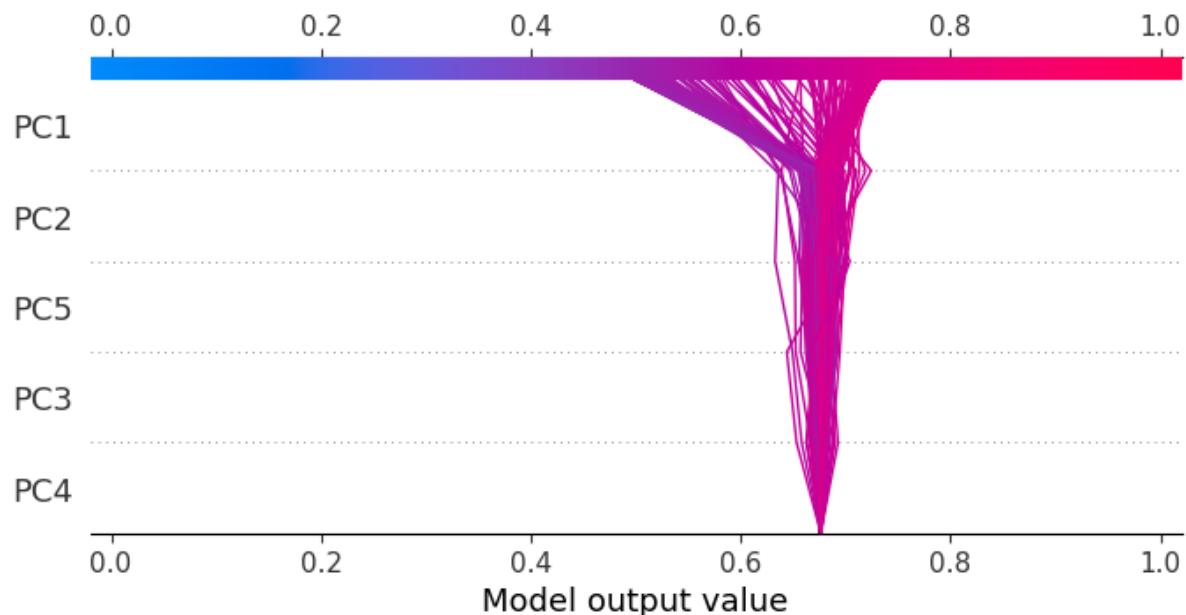


Prestando attenzione al grafico sottostante notiamo che nella feature PC1 più il suo valore è basso più lo SHAP value sarà basso, un trend simile è osservabile anche in PC3 seppur in maniera più ridotta. Per PC5, PC4 e PC2 troviamo invece un trend opposto e ridotto rispetto a quello di PC1.

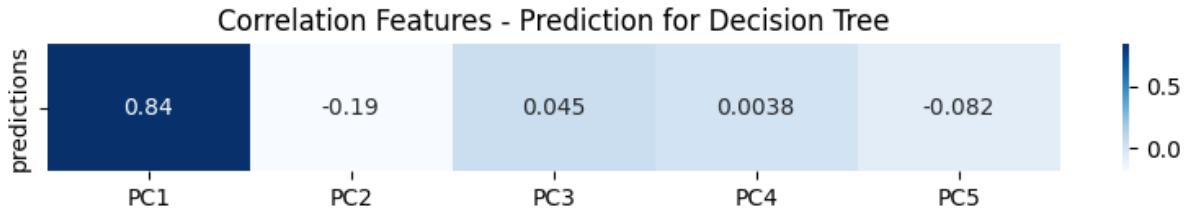


Osservando il grafico sottostante notiamo che tutte le istanze convergono ad un valore di output di circa 0.67, partendo da un range compreso tra (0.5, 0.7) circa. Dopo aver processato PC2 l'intervallo originale si restringe a (0.64, 0.73) circa,

man mano che ogni feature viene data in input possiamo notare un restringimento continuo dell'intervallo ma più lento fino a terminare ogni feature della PCA e ad ottenere come valore di output 0.67.

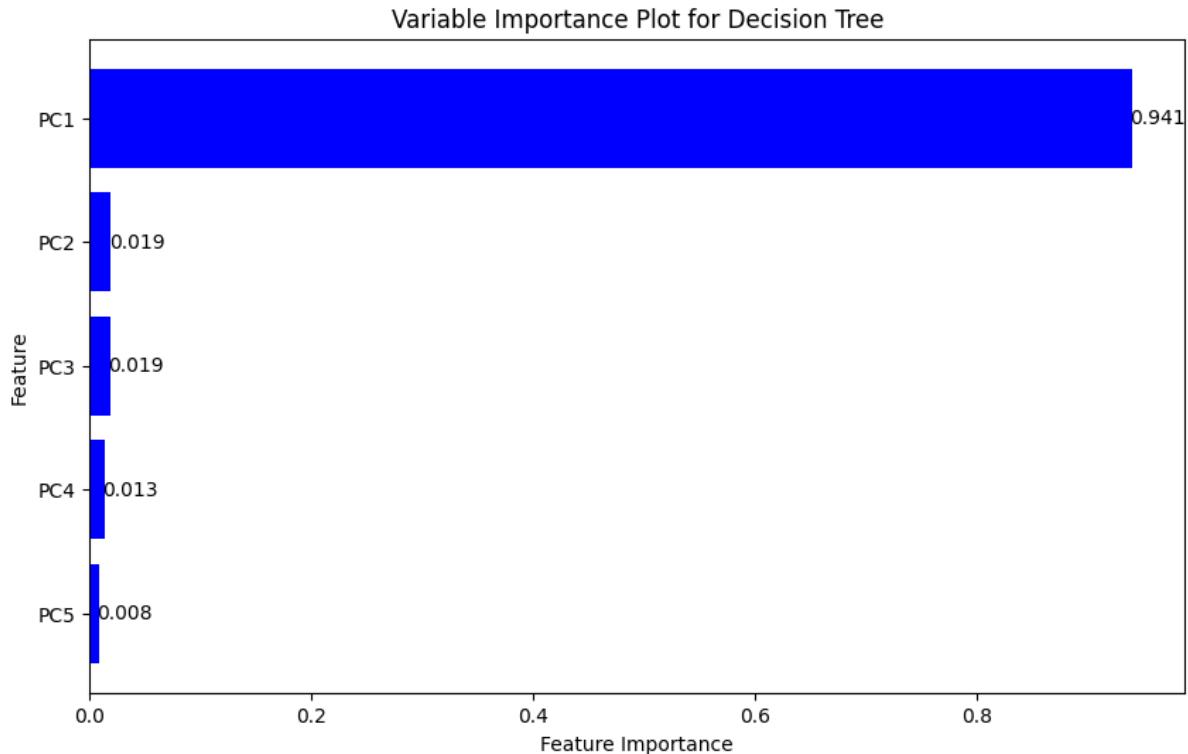


Decision Tree Classifier



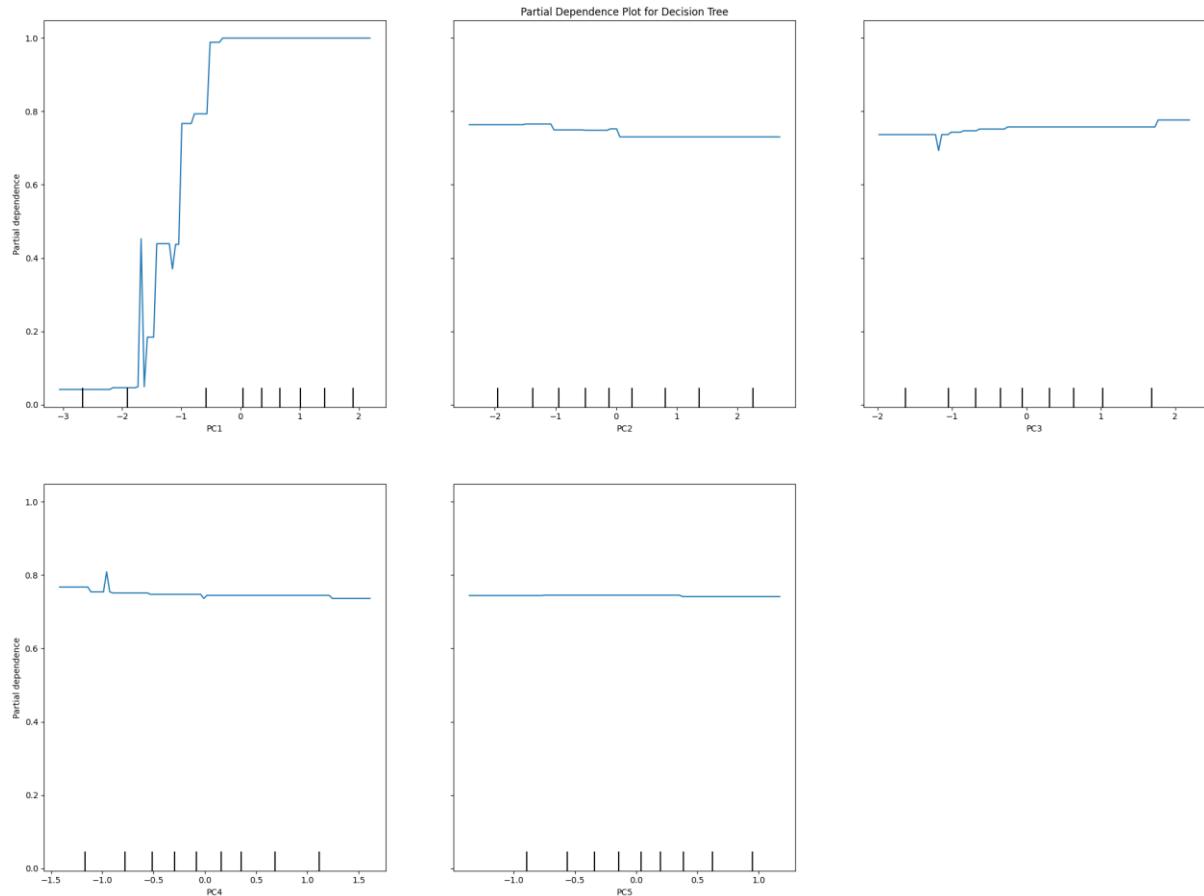
PC1 ha un'altissima correlazione positiva con la predizione del target.

Osservando la tabella concludiamo che PC1 influenza in maniera importante il modello nella determinazione del target, PC2 ha una bassa correlazione negativa e contribuisce in maniera minima al processo di classificazione, le restanti colonne sono invece trascurabili. Andando a confrontare i valori ottenuti dei due modelli già presi in esame notiamo che non vi sono differenze significative, pertanto attendiamo metriche molto simili per tutti e tre i modelli. Guardando il grafico di variable importance troviamo coerenza con i risultati ottenuti dal grafico di correlation features, PC1 ha un peso altissimo nel processo di classificazione dell'istanza, PC2, PC3 e PC4 contribuiscono marginalmente, PC5 invece fornisce un contributo trascurabile. Rapportando i risultati ottenuti con quelli ottenuti in precedenza non vi sono cambiamenti significativi.



Osservando il grafico sottostante confermiamo che quanto affermato prima nei due grafici precedenti trova riscontro. PC1 è la componente che quasi

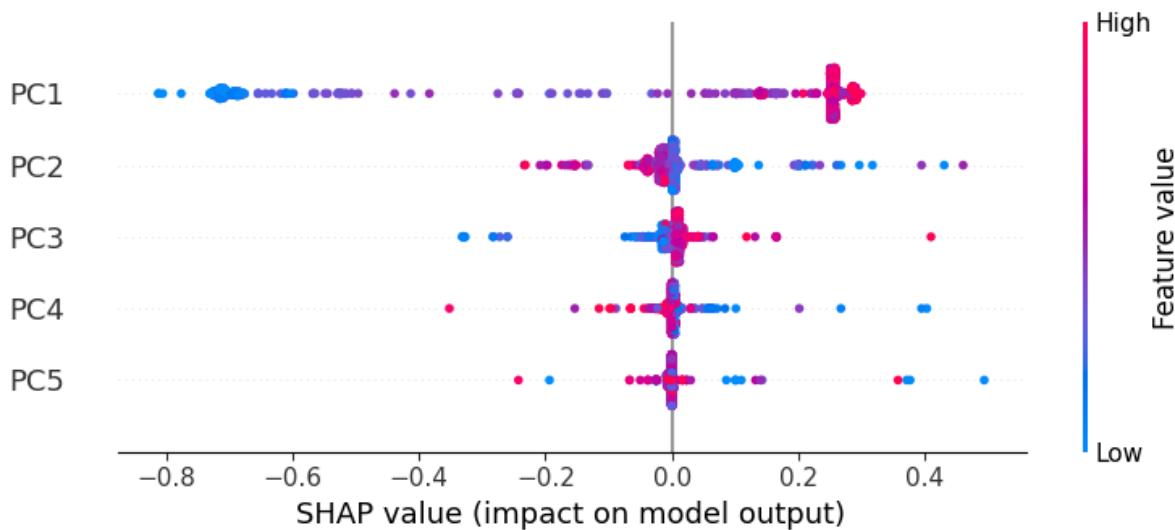
esclusivamente influisce sul target. Osservando il grafico relativo a PC1 notiamo che quando assume valori minori di -1 le istanze del dataset verranno classificate come vini rossi, altrimenti come vini bianchi. PC2, PC3 e PC4 influiscono in maniera marginale, PC5 invece, non influisce in maniera significativa, si noti che la linea è molto simile, praticamente identica una linea parallela all'asse ascisse.



In rosso sono mostrate le istanze di vini bianchi, mentre in blu troviamo le istanze di vini rossi. Passando con il mouse sulle aree rosse (oppure blu) viene mostrato lo SHAP value di ogni colonna della PCA, scorrendo da sinistra verso destra noteremo che i valori di ogni colonna sono in accordo con quanto affermato nel grafico precedente. Comparando il grafico preso ora in esame con quelli precedenti notiamo una separazione molto più netta (si ricorda di visionare questo grafico sul notebook in quanto riesce a fornire maggiori informazioni).

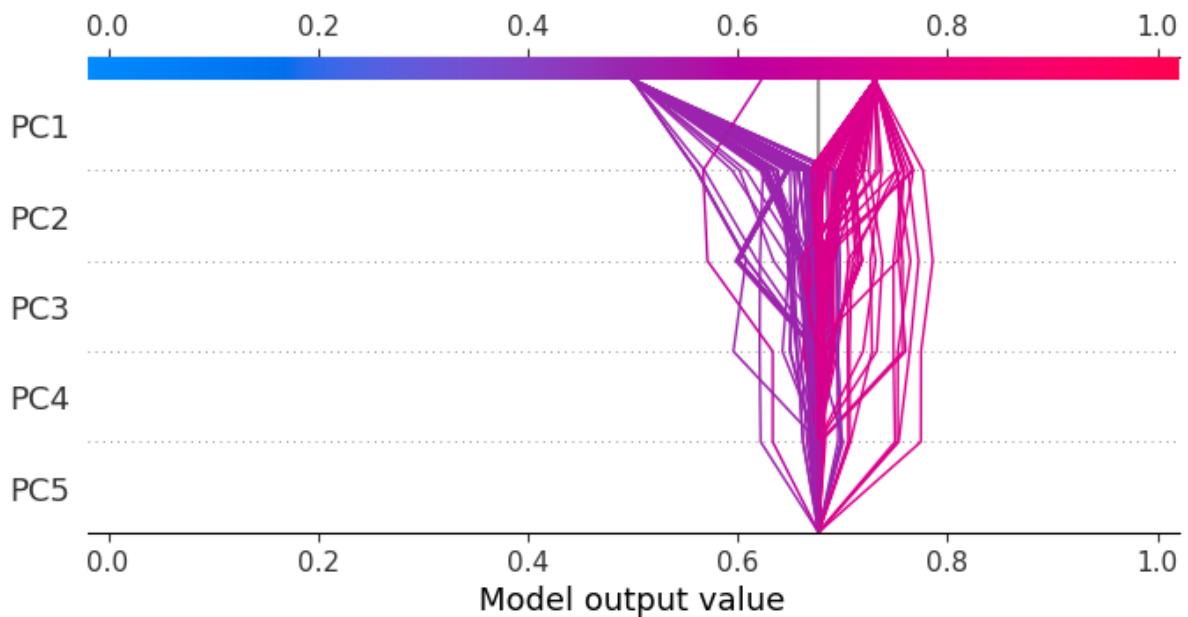


Prestando attenzione al grafico sottostante notiamo che nella feature PC1 più il suo valore è basso più lo SHAP value sarà basso, un trend simile è osservabile anche in PC3 seppur in maniera più ridotta. Per PC5, PC4 e PC2 troviamo invece un trend opposto a quello di PC1.



Il grafico sottostante mostra come ogni feature accumulata alla successiva determina il valore di output. Le features sono ordinate per importanza in maniera decrescente. Osservando il grafico notiamo che tutte le istanze convergono ad un valore di output di circa 0.67, partendo da un range compreso tra (0.5, 0.7) circa. Dopo aver processato PC2 l'intervallo originale si restringe, dopo aver processato PC3 abbiamo un leggerissimo ampliamento, dopo aver processato PC4 e PC5 l'intervallo si è ridotto stabilmente in maniera molto limitata.Terminate le features in input notiamo che ogni istanza va ad ottenere un valore di output di 0.67. Se analizziamo contemporaneamente questo grafico per tutti e due i modelli notiamo che quello del Decision Tree tende a presentare

intervalli più ampi mano mano che le features vengono prese in input, in particolare se compariamo l'intervallo prima di prendere in input PC5 notiamo che quelli di SVM sono decisamente più ridotti di quello del Decision Tree.



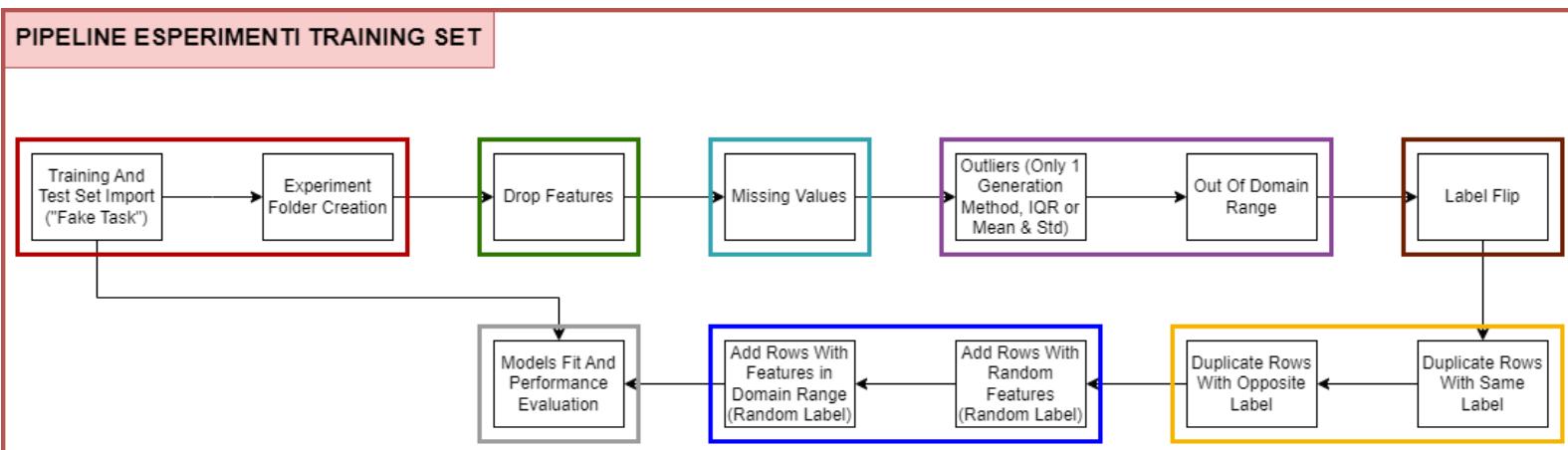
Pipeline Esperimenti

A questo punto, è possibile passare alla seconda parte del progetto (i.e. esperimenti).

In questa sezione si mostrano i dettagli della seconda pipeline, ossia la pipeline contenente la parte di esperimenti e valutazione delle performance dei modelli di ML addestrati con un training set di qualità inferiore, rispetto a quello utilizzato precedentemente nella prima pipeline.

Architettura Seconda Pipeline

Analogamente alla prima pipeline, di seguito viene presentata l'architettura della seconda pipeline, per poter avere una visione globale degli esperimenti



applicati:

Facendo riferimento alla struttura, si possono fare diverse osservazioni.

Innanzitutto, anche per questa pipeline è presente un “Fake Task”, per la stessa motivazione trattata nella prima pipeline: viene utilizzato per richiedere l'esistenza di file richiesti, in questo caso si parte dal training set e dal test set [prodotti dalla prima pipeline](#) nello spazio della PCA.

Nell'immagine si possono notare colori che circondano alcuni task: si tratta semplicemente di una suddivisione a livello concettuale dei diversi esperimenti, mettendo sotto lo stesso colore i task molto simili tra di loro (e.g. i task preliminari di setup in rosso, i task sull'aggiunta di righe nel dataset in blu, ...).

Un'ultima osservazione, ma non meno importante delle altre, è la seguente: la struttura di questa pipeline è interamente sequenziale perché ciascun esperimento produce sempre un output intermedio (che può essere uguale all'input del task se dai parametri si specifica di voler evitare l'esperimento in

questione) passato in input all'esperimento successivo, che a sua volta produce un risultato intermedio e lo passa al task seguente e così via.

N.B: "task" ed "esperimento" possono essere usati come sinonimi da "Drop Features" a "Add Rows With Features In Domain Range (Random Label)", ossia per semplicità ci si può riferire ai task relativi agli esperimenti con il termine "esperimenti".

Inoltre, il termine "esperimento" può riferirsi anche ad un'esecuzione della pipeline (i.e. combinazione di parametri identificata da un nome univoco) nel contesto della subdirectory dedicata a una specifica esecuzione.

Tasks Seconda Pipeline

Si può passare ora all'analisi approfondita dei singoli task.

"Fake Task"

Come già detto, il task "Fake Task" esiste solo per inserire nel task successivo la dipendenza dall'**esistenza di due file csv**, ossia `winetype_pca_train.csv` e `winetype_pca_test.csv`, contenenti rispettivamente il **training set** e il **test set** creati precedentemente (dopo l'applicazione della PCA).

Experiment Folder

Si tratta della prima novità rispetto alla prima pipeline: nel momento in cui si esegue la pipeline degli esperimenti, è **obbligatorio** inserire un **nome univoco** (non è possibile utilizzare lo stesso nome per due esecuzioni della pipeline) per identificare l'**esperimento**; questo nome viene utilizzato per organizzare tutto l'output, della specifica esecuzione, in una subdirectory omonima.

Come ci si aspetta, la **dipendenza** è associata a "**Fake Task**": per far partire questo primo task vero e proprio, si richiede l'esistenza del training set e del test set.

Il task presenta i seguenti parametri:

- `experiment-name`: nome che identifica l'esecuzione della pipeline
 - obbligatorio
- `train-csv`: percorso al csv contenente il training set
 - di default si ha "datasets/winetype_pca_train.csv"
- `test-csv`: percorso al csv contenente il test set
 - di default si ha "datasets/winetype_pca_test.csv"

L'unica operazione effettuata è quella di creare la **subdirectory** con il **nome dell'esperimento**, all'interno della directory experiments.

Come già detto, questo task non produce un file ma una directory.

Siccome Luigi non supporta nativamente target corrispondenti a directory, viene sfruttato un target custom **DirectoryTarget**, in modo tale che il task risulta completato se e solo esiste la directory con il nome dell'esperimento (creata dalla logica del task).

Drop Features

Questo task ha la **dipendenza** dal task precedente, cioè “**Experiment Folder**”, poiché è necessario che esista la directory associata con l'esperimento.

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: lista di features da droppare
 - di default è vuota
- train-csv: usato per le dipendenze
- drop-features-csv-name: nome del file di output del task
 - di default si ha “train_after_drop_features.csv”

Si effettuano le seguenti operazioni:

- si effettua l'**eventuale drop di features** (a seconda dei parametri passati al task) partendo dal file csv in input
 - **importante**: non è possibile droppare il target
- si memorizza il nuovo training set (intermedio) nel file di output del task

L'output del task di default è il file **train_after_drop_features.csv**.

Missing Values

Questo task ha la **dipendenza** dal task precedente, cioè “**Drop Features**”, poiché si parte dal training set intermedio dopo aver eventualmente droppato features.

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: usato per le dipendenze
- features-to-dirty-mv: lista di features da considerare
 - di default è vuota
- missing-values-percentage: percentuale di valori mancanti da introdurre
 - di default si ha 0.0
- wine-types-to-consider-missing-values: lista di tipi di vino da considerare
 - di default si hanno entrambe le classi
- train-csv: usato per le dipendenze
- missing-values-csv-name: nome del file di output del task

- di default si ha “train_after_missing_values.csv”

Si effettuano le seguenti operazioni:

- si effettua l’eventuale **introduzione di valori mancanti** (a seconda dei parametri passati al task) partendo dal file csv in input
 - **importante**: non è possibile utilizzare il target per i valori mancanti
- si memorizza il nuovo training set (intermedio) nel file di output del task

L’output del task di default è il file **train_after_missing_values.csv**.

Outliers

Questo task ha la **dipendenza** dal task precedente, cioè **“Missing Values”**, poiché si parte dal training set intermedio dopo aver introdotto eventuali valori mancanti.

Gli **outliers** si possono introdurre utilizzando il metodo ‘**std**’ (partendo dal range **Mean ± 3 * Std**, si sceglie randomicamente se generare un outlier **al di sopra** dell’intervallo, ma comunque **entro il valore massimo assunto, oppure un outlier al di sotto** dell’intervallo, ma comunque **entro il valore minimo assunto**) oppure il metodo ‘**iqr**’ (il range da cui si parte è **[Q1 - 2 * IQR, Q3 + 2 * IQR]**, dove Q1, Q3 e IQR sono rispettivamente il primo quartile, il terzo quartile e lo scarto interquartile).

Il task presenta i seguenti parametri:

- **experiment-name**: usato per le dipendenze
- **features-to-drop**: usato per le dipendenze
- **features-to-dirty-mv**: usato per le dipendenze
- **wine-types-to-consider-missing-values**: usato per le dipendenze
- **missing-values-percentage**: usato per le dipendenze
- **features-to-dirty-outliers**: lista di features da considerare
 - di default è vuota
- **outliers-percentage**: percentuale di outliers da introdurre
 - si default si ha 0.0
- **wine-types-to-consider-outliers**: lista di tipi di vino da considerare
 - di default si hanno entrambe le classi
- **range-type**: metodo utilizzato per generare gli outliers, tra ‘**std**’ ed ‘**iqr**’
 - di default si ha ‘**std**’
- **train-csv**: usato per le dipendenze
- **outliers-csv-name**: nome del file di output del task
 - di default si ha “**train_after_outliers.csv**”

Si effettuano le seguenti operazioni:

- si effettua l'eventuale **introduzione di outliers** (a seconda dei parametri passati al task) partendo dal file csv in input
 - **importante:** naturalmente non è possibile utilizzare il target (bool)
- si memorizza il nuovo training set (intermedio) nel file di output del task

L'output del task di default è il file **train_after_outliers.csv**.

Out Of Domain Range

Questo task ha la **dipendenza** dal task precedente, cioè “Outliers”, poiché si parte dal training set intermedio dopo aver introdotto eventuali outliers.

I valori fuori dal dominio vengono generati **scommmando un numero random** nell'intervallo **[-0.01, 0.01]** con **Mean + 10 * Std oppure Mean - 10 * Std** (altra **scelta random**, non si tratta di un intervallo, bensì di due possibilità).

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: usato per le dipendenze
- features-to-dirty-mv: usato per le dipendenze
- features-to-dirty-outliers: usato per le dipendenze
- wine-types-to-consider-missing-values: usato per le dipendenze
- wine-types-to-consider-outliers: usato per le dipendenze
- missing-values-percentage: usato per le dipendenze
- outliers-percentage: usato per le dipendenze
- range-type: usato per le dipendenze
- features-to-dirty-oodv: lista di features da considerare
 - di default è vuota
- oodv-percentage: percentuale di valori fuori dal dominio da introdurre
 - si default si ha 0.0
- wine-types-to-consider-oodv: lista di tipi di vino da considerare
 - di default si hanno entrambe le classi
- train-csv: usato per le dipendenze
- oodv-csv-name: nome del file di output del task
 - di default si ha “train_after_oodv.csv”

Si effettuano le seguenti operazioni:

- si effettua l'eventuale **introduzione di valori fuori dal dominio** (a seconda dei parametri passati al task) partendo dal file csv in input
 - **importante:** naturalmente non è possibile utilizzare il target (bool)
- si memorizza il nuovo training set (intermedio) nel file di output del task

L'output del task di default è il file **train_after_oodv.csv**.

Flip Labels

Questo task ha la **dipendenza** dal task precedente, cioè “Out Of Domain Range”, poiché si parte dal training set intermedio dopo aver introdotto eventuali valori fuori dal dominio.

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: usato per le dipendenze
- features-to-dirty-mv: usato per le dipendenze
- features-to-dirty-outliers: usato per le dipendenze
- missing-values-percentage: usato per le dipendenze
- outliers-percentage: usato per le dipendenze
- range-type: usato per le dipendenze
- features-to-dirty-oodv: usato per le dipendenze
- oodv-percentage: usato per le dipendenze
- train-csv: usato per le dipendenze
- wine-types-to-consider-missing-values: usato per le dipendenze
- wine-types-to-consider-outliers: usato per le dipendenze
- wine-types-to-consider-oodv: usato per le dipendenze
- flip-percentage-red: percentuale di vini rossi da considerare
 - si default si ha 0.0
- flip-percentage-white: percentuale di vini bianchi da considerare
 - si default si ha 0.0
- flip-labels-csv-name: nome del file di output del task
 - di default si ha “train_after_flip_features.csv”

Si effettuano le seguenti operazioni:

- si effettua l’eventuale **flip delle etichette dei vini rossi e dei vini bianchi** (a seconda dei parametri passati al task) partendo dal file csv in input
- si memorizza il nuovo training set (intermedio) nel file di output del task

L’output del task di default è il file **train_after_flip_features.csv**.

Duplicate Rows With Same Label

Questo task ha la **dipendenza** dal task precedente, cioè “**Flip Labels**”, poiché si parte dal training set intermedio dopo aver eventualmente flippato etichette di vini rossi e di vini bianchi.

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: usato per le dipendenze

- `features-to-dirty-mv`: usato per le dipendenze
- `features-to-dirty-outliers`: usato per le dipendenze
- `missing-values-percentage`: usato per le dipendenze
- `outliers-percentage`: usato per le dipendenze
- `range-type`: usato per le dipendenze
- `features-to-dirty-oodv`: usato per le dipendenze
- `oodv-percentage`: usato per le dipendenze
- `train-csv`: usato per le dipendenze
- `flip-percentage-red`: usato per le dipendenze
- `flip-percentage-white`: usato per le dipendenze
- `wine-types-to-consider-missing-values`: usato per le dipendenze
- `wine-types-to-consider-outliers`: usato per le dipendenze
- `wine-types-to-consider-oodv`: usato per le dipendenze
- `wine-types-to-consider-same-label`: lista di tipi di vino da considerare
 - di default si hanno entrambe le classi
- `duplicate-rows-same-label-percentage`: percentuale di righe da duplicare, mantenendo la stessa etichetta
 - si default si ha 0.0
- `duplicate-rows-same-label-csv-name`: nome del file di output del task
 - di default si ha “`train_after_duplicate_rows_same_label.csv`”

Si effettuano le seguenti operazioni:

- si effettua l’eventuale **duplicazione di righe mantenendo la stessa etichetta** (a seconda dei parametri passati al task) partendo dal file csv in input
- si memorizza il nuovo training set (intermedio) nel file di output del task

L’output del task di default è il file `train_after_duplicate_rows_same_label.csv`.

Duplicate Rows With Opposite Label

Questo task ha la **dipendenza** dal task precedente, cioè **“Duplicate Rows With Same Label”**, poiché si parte dal training set intermedio dopo aver eventualmente duplicato delle righe mantenendo la stessa etichetta.

Il task presenta i seguenti parametri:

- `experiment-name`: usato per le dipendenze
- `features-to-drop`: usato per le dipendenze
- `features-to-dirty-mv`: usato per le dipendenze
- `features-to-dirty-outliers`: usato per le dipendenze
- `missing-values-percentage`: usato per le dipendenze
- `outliers-percentage`: usato per le dipendenze
- `range-type`: usato per le dipendenze

- `features-to-dirty-oodv`: usato per le dipendenze
- `oodv-percentage`: usato per le dipendenze
- `train-csv`: usato per le dipendenze
- `flip-percentage-red`: usato per le dipendenze
- `flip-percentage-white`: usato per le dipendenze
- `wine-types-to-consider-same-label`: usato per le dipendenze
- `wine-types-to-consider-missing-values`: usato per le dipendenze
- `wine-types-to-consider-outliers`: usato per le dipendenze
- `wine-types-to-consider-oodv`: usato per le dipendenze
- `duplicate-rows-same-label-percentage`: usato per le dipendenze
- `wine-types-to-consider-opposite-label`: lista di tipi di vino da considerare
 - di default si hanno entrambe le classi
- `duplicate-rows-opposite-label-percentage`: percentuale di righe da duplicare,翻转ing l'etichetta
 - di default si ha 0.0
- `duplicate-rows-opposite-label-csv-name`: nome del file di output del task
 - di default si ha “`train_after_duplicate_rows_opposite_label.csv`”

Si effettuano le seguenti operazioni:

- si effettua l’eventuale **duplicazione di righe翻转ando l’etichetta** (a seconda dei parametri passati al task) partendo dal file csv in input
- si memorizza il nuovo training set (intermedio) nel file di output del task

L’output del task di default è il file
`train_after_duplicate_rows_opposite_label.csv`.

Add Rows With Random Features

Questo task ha la **dipendenza** dal task precedente, cioè “**Duplicate Rows With Opposite Label**”, poiché si parte dal training set intermedio dopo aver eventualmente duplicato delle righe翻转ando l’etichetta.

I valori random delle **features** vengono generati nell’intervallo [-100, 100), mentre l’etichetta è un boolean random.

Il task presenta i seguenti parametri:

- `experiment-name`: usato per le dipendenze
- `features-to-drop`: usato per le dipendenze
- `features-to-dirty-mv`: usato per le dipendenze
- `features-to-dirty-outliers`: usato per le dipendenze
- `missing-values-percentage`: usato per le dipendenze
- `outliers-percentage`: usato per le dipendenze
- `range-type`: usato per le dipendenze

- `features-to-dirty-oodv`: usato per le dipendenze
- `oodv-percentage`: usato per le dipendenze
- `train-csv`: usato per le dipendenze
- `flip-percentage-red`: usato per le dipendenze
- `flip-percentage-white`: usato per le dipendenze
- `wine-types-to-consider-same-label`: usato per le dipendenze
- `duplicate-rows-same-label-percentage`: usato per le dipendenze
- `wine-types-to-consider-opposite-label`: usato per le dipendenze
- `duplicate-rows-opposite-label-percentage`: usato per le dipendenze
- `wine-types-to-consider-missing-values`: usato per le dipendenze
- `wine-types-to-consider-outliers`: usato per le dipendenze
- `wine-types-to-consider-oodv`: usato per le dipendenze
- `add-rows-random-percentage`: percentuale di righe da aggiungere, con valori random delle features
 - si default si ha 0.0
- `add-rows-random-csv-name`: nome del file di output del task
 - di default si ha “`train_after_add_rows_random.csv`”

Si effettuano le seguenti operazioni:

- si effettua l’eventuale **aggiunta di righe con valori random delle features** (a seconda dei parametri passati al task) partendo dal file csv in input
- si memorizza il nuovo training set (intermedio) nel file di output del task

L’output del task di default è il file `train_after_add_rows_random.csv`.

Add Rows With Features In Domain Range

Questo task ha la **dipendenza** dal task precedente, cioè “**Add Rows With Random Features**”, poiché si parte dal training set intermedio dopo aver eventualmente aggiunto delle righe con valori random delle features.

Per ottenere features nei **range di dominio** si utilizza **Mean ± 3 * Std**, mentre l’etichetta è un boolean random.

Il task presenta i seguenti parametri:

- `experiment-name`: usato per le dipendenze
- `features-to-drop`: usato per le dipendenze
- `features-to-dirty-mv`: usato per le dipendenze
- `features-to-dirty-outliers`: usato per le dipendenze
- `missing-values-percentage`: usato per le dipendenze
- `outliers-percentage`: usato per le dipendenze
- `range-type`: usato per le dipendenze
- `features-to-dirty-oodv`: usato per le dipendenze

- oodv-percentage: usato per le dipendenze
- train-csv: usato per le dipendenze
- flip-percentage-red: usato per le dipendenze
- flip-percentage-white: usato per le dipendenze
- wine-types-to-consider-same-label: usato per le dipendenze
- duplicate-rows-same-label-percentage: usato per le dipendenze
- wine-types-to-consider-opposite-label: usato per le dipendenze
- duplicate-rows-opposite-label-percentage: usato per le dipendenze
- add-rows-random-percentage: usato per le dipendenze
- wine-types-to-consider-missing-values: usato per le dipendenze
- wine-types-to-consider-outliers: usato per le dipendenze
- wine-types-to-consider-oodv: usato per le dipendenze
- add-rows-domain-percentage: percentuale di righe da aggiungere, con valori delle features nei range di dominio (mentre l'etichetta è un bool random)
 - si default si ha 0.0
- add-rows-domain-csv-name: nome del file di output del task
 - di default si ha “train_after_add_rows_domain.csv”

Si effettuano le seguenti operazioni:

- si effettua l'eventuale **aggiunta di righe con valori delle features nei range di dominio** (a seconda dei parametri passati al task) partendo dal file csv in input
- si memorizza il nuovo training set finale nel file di output del task

L'output del task di default è il file **train_after_add_rows_domain.csv**.

Models Fit And Performance Evaluation

Questo task ha la **dipendenza** da

- **“Add Rows With Features In Domain Range”** poiché si parte dal training set finale, dopo aver applicato tutti i possibili esperimenti
- **“Fake Task”** perché per le metriche globali è necessario avere il test set

Il task presenta i seguenti parametri:

- experiment-name: usato per le dipendenze
- features-to-drop: usato per le dipendenze
- features-to-dirty-mv: usato per le dipendenze
- features-to-dirty-outliers: usato per le dipendenze
- missing-values-percentage: usato per le dipendenze
- outliers-percentage: usato per le dipendenze
- range-type: usato per le dipendenze

- features-to-dirty-oodv: usato per le dipendenze
- oodv-percentage: usato per le dipendenze
- train-csv: usato per le dipendenze
- test-csv: usato per le dipendenze
- flip-percentage-red: usato per le dipendenze
- flip-percentage-white: usato per le dipendenze
- wine-types-to-consider-same-label: usato per le dipendenze
- duplicate-rows-same-label-percentage: usato per le dipendenze
- wine-types-to-consider-opposite-label: usato per le dipendenze
- duplicate-rows-opposite-label-percentage: usato per le dipendenze
- add-rows-random-percentage: usato per le dipendenze
- add-rows-domain-percentage: usato per le dipendenze
- wine-types-to-consider-missing-values: usato per le dipendenze
- wine-types-to-consider-outliers: usato per le dipendenze
- wine-types-to-consider-oodv: usato per le dipendenze
- metrics-csv-name: nome del file di output del task
 - di default si ha “metrics.csv”

Si effettuano le seguenti operazioni:

- si recupera il training set finale dopo aver applicato tutti gli esperimenti
- si utilizzano due [strategie di imputing](#) per **non addestrare con valori mancanti i modelli che non li supportano**
 - l'unico classificatore di scikit-learn che supporta nativamente i valori mancanti (dalla versione 1.4.2) è **DecisionTreeClassifier**
 - la Support Vector Machine (SVC) dà errore in fase di fit se vi sono valori mancanti nel training set
 - il primo approccio utilizzato è uno dei più comuni e consiste nel sostituire i valori mancanti con la **media** calcolata per ogni feature
 - si utilizza poi anche un altro approccio basato sull'[algoritmo EM](#), perché [secondo ricerche](#) performa meglio rispetto alla media, dunque si è scelto di utilizzare due approcci per poi confrontare i risultati ottenuti
- si recupera il test set (droppando eventuali feature dropgate nel training set)
- si ricreano i modelli naive, con la differenza che la **SVM** utilizza un **soft-margin** con l'iperparametro **C = 0.001**
 - vengono ricreati i modelli per diverse motivazioni
 - la SVM richiede un soft-margin perché deve essere robusta a valori estremi che prima non potevamo avere, altrimenti il tempo di training tende ad aumentare in maniera significativa

- il tempo di training non è significativo e quindi ciò non impatta
- per ciascun modello (**SVM (mean)**, **SVM (EM)** e **Albero Decisionale**)
 - si calcolano le **metriche globali**
 - si calcolano gli **intervalli di confidenza 95%** in **Stratified 10-Fold CV**
- si memorizzano tutte le metriche nel file di output del task

L'output del task di default è il file **metrics.csv**.

Esperimenti su Data Quality

Sono stati quindi effettuati, attraverso la pipeline e scripts, degli esperimenti al fine di valutare concretamente la variazione dei valori delle metriche in base alla sporcizia del dataset, ovvero un dataset che non rispetta le dimensioni di qualità introdotte in precedenza.

Nel dettaglio sono stati implementati **659** esperimenti, ovvero tutte le combinazioni possibili riguardanti:

- **Drop Features** (5 Esperimenti) dove vengono droppate le features.
- **Missing Values** (108 Esperimenti) dove, in base ad una percentuale da 10% a 90%, le features da sporcare e il valore del target vengono introdotti dei valori nulli.
- **Outliers** (240 Esperimenti) dove, in base ad una percentuale da 10% a 100%, le features da sporcare, il valore del target e il tipo di range (iqr o std) vengono introdotti degli outliers.
- **Out Of Domain Values** (120 Esperimenti) dove, in base ad una percentuale da 10% a 100%, le features da sporcare e il valore del target vengono introdotti valori estremamente fuori dalla distribuzione dei valori della PCA, fuori dal dominio.
- **Flip Labels** (100 Esperimenti) dove, in base alla percentuale di vini “red” ed una percentuale di vini “white”, da 10% a 100% vengono invertiti i valori del target corrispondente.
- **Duplicate Rows Same Label** (30 Esperimenti) dove, in base ad una percentuale da 10% a 100% e il valore del target, vengono introdotte righe duplicate con la label corretta.
- **Duplicate Rows Opposite Label** (30 Esperimenti) dove, in base ad una percentuale da 10% a 100% e il valore del target, vengono introdotte righe duplicate con la label invertita.
- **Add Rows Random** (10 Esperimenti) dove, in base ad una percentuale da 10% a 100%, vengono introdotte delle righe con dei valori random, anche fuori dal dominio.
- **Add Rows Domain** (10 Esperimenti) dove, in base ad una percentuale da 10% a 100%, vengono introdotte delle righe con dei valori all'interno del dominio.

Va notato che nelle successive sezioni saranno trattati i grafici più significativi in quanto sono molteplici, data la grande quantità di esperimenti. I restanti sono presenti nel notebook “**experiments.ipynb**”

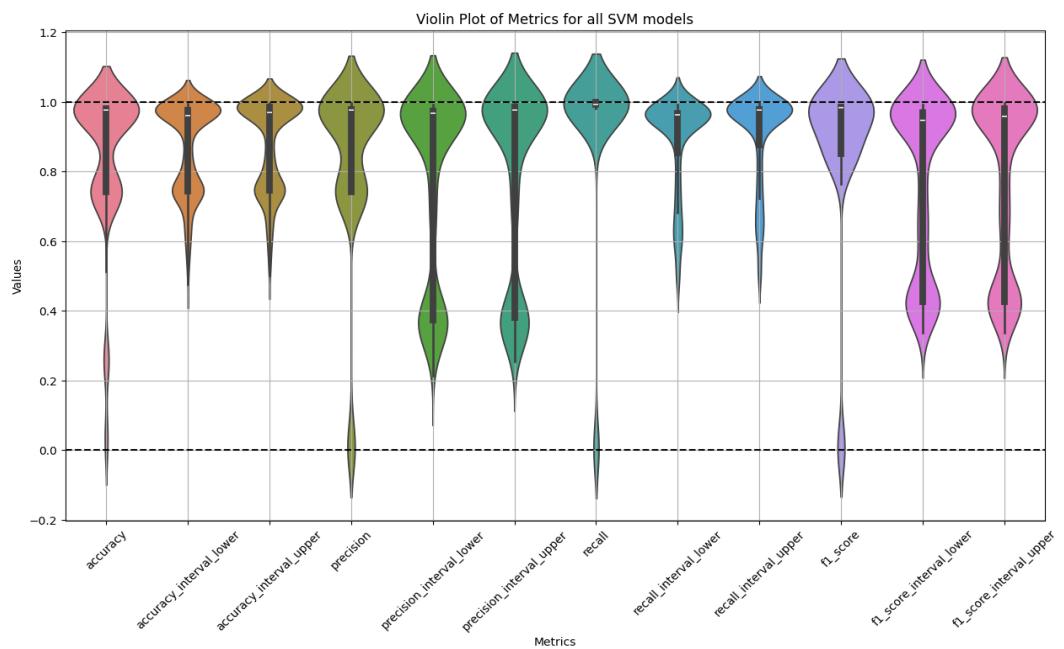
Plots Globali

Per i 659 esperimenti effettuati sono stati studiati gli effetti complessivi e specifici sulle prestazioni dei modelli di Machine Learning. In questa sezione viene fornita un'analisi a livello globale dell'influenza degli esperimenti sulle quattro metriche più importanti considerate, e nel caso dei violin plots anche sugli intervalli di confidenza.

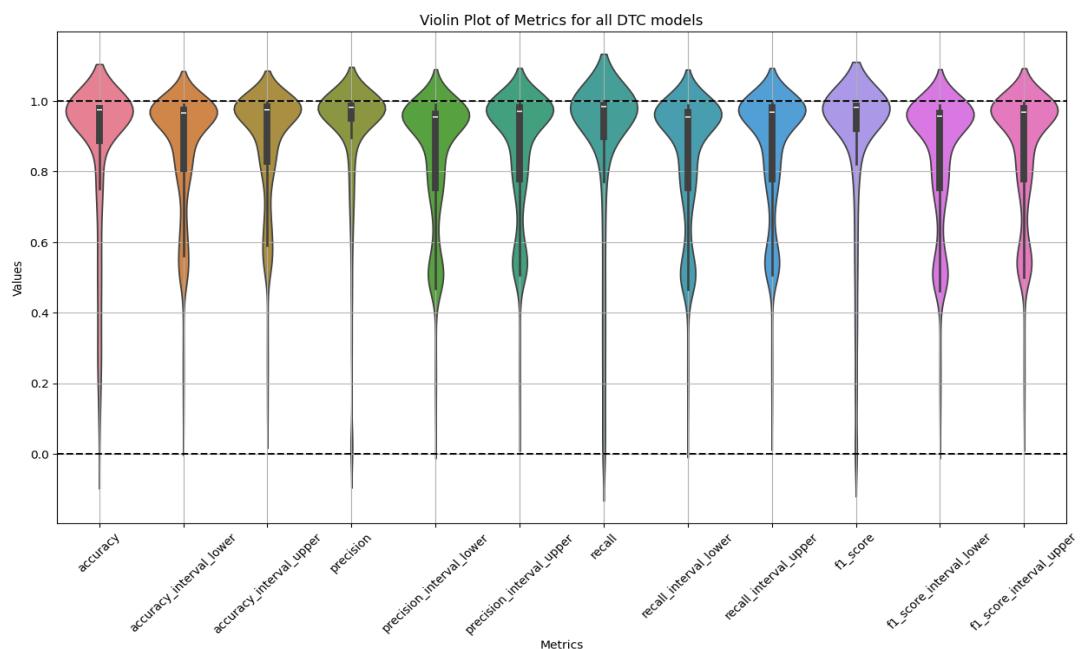
Violin Plot

Prima di passare all'analisi del Violin Plot è necessario specificare che il modello SVM è stato analizzato includendo le varianti con mean ed EM, al fine di poterlo comparare con il Decision Tree.

SVM



Decision Tree



Si può notare come i tre grafici risultano molto simili, con la maggior parte della distribuzione delle metriche concentrate verso 1.

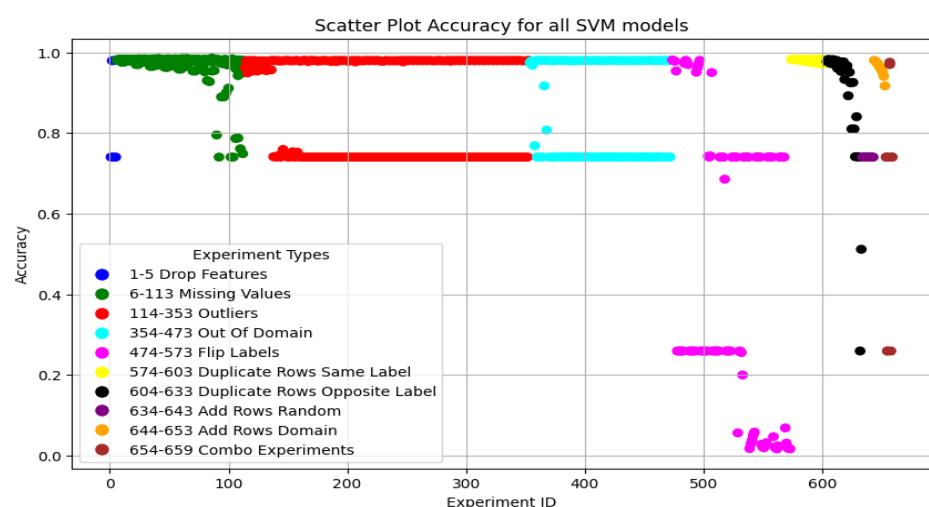
Nel caso specifico del SVM e Decision Tree possiamo notare un allungamento maggiore di ogni box plot negli intervalli di precisione e f1-score, deducendo che sono state infatti le metriche più variate durante gli esperimenti.

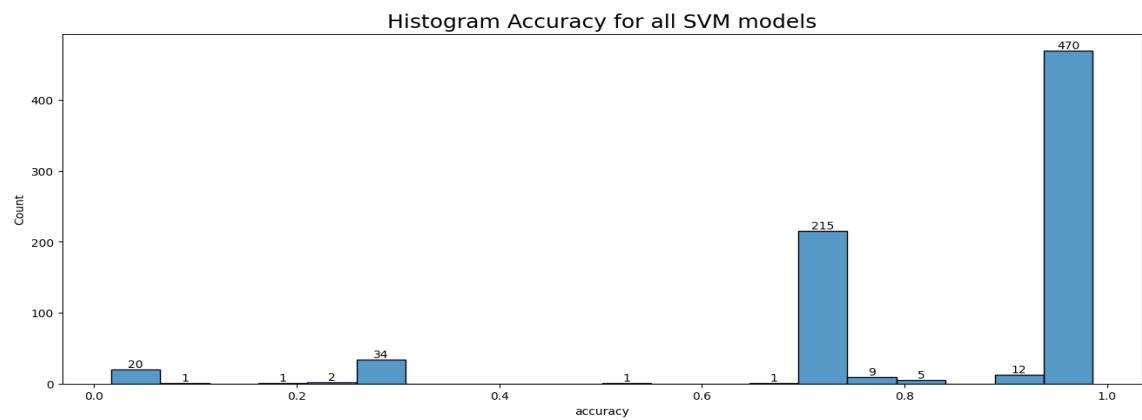
La forma attorno ad ogni box plot invece si occupa di mostrare la probabilità che i valori corrispondenti sull'asse y siano effettivamente assunti, dove una larghezza maggiore indica una maggior probabilità.

Scatter Plot e Bar Plot

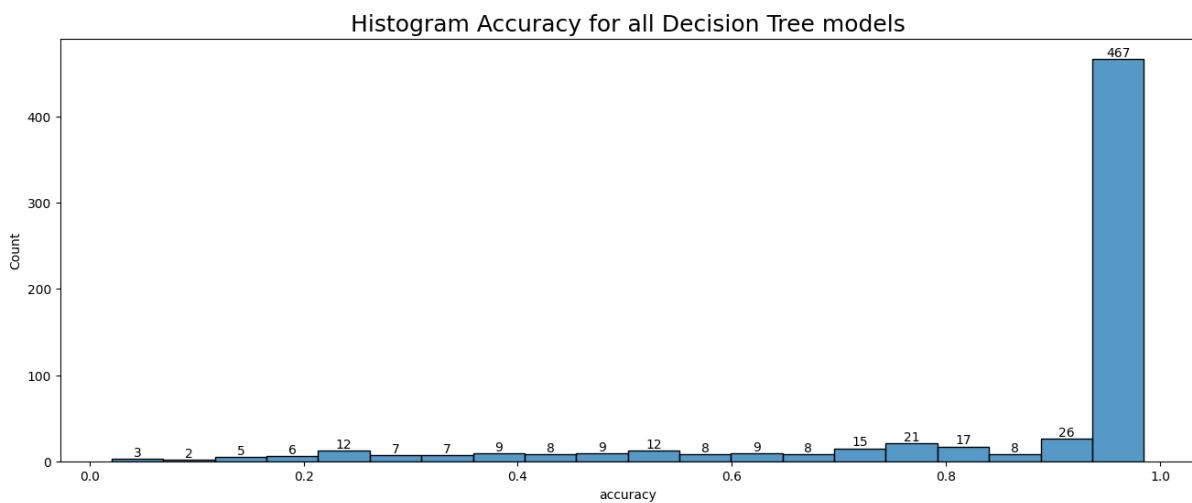
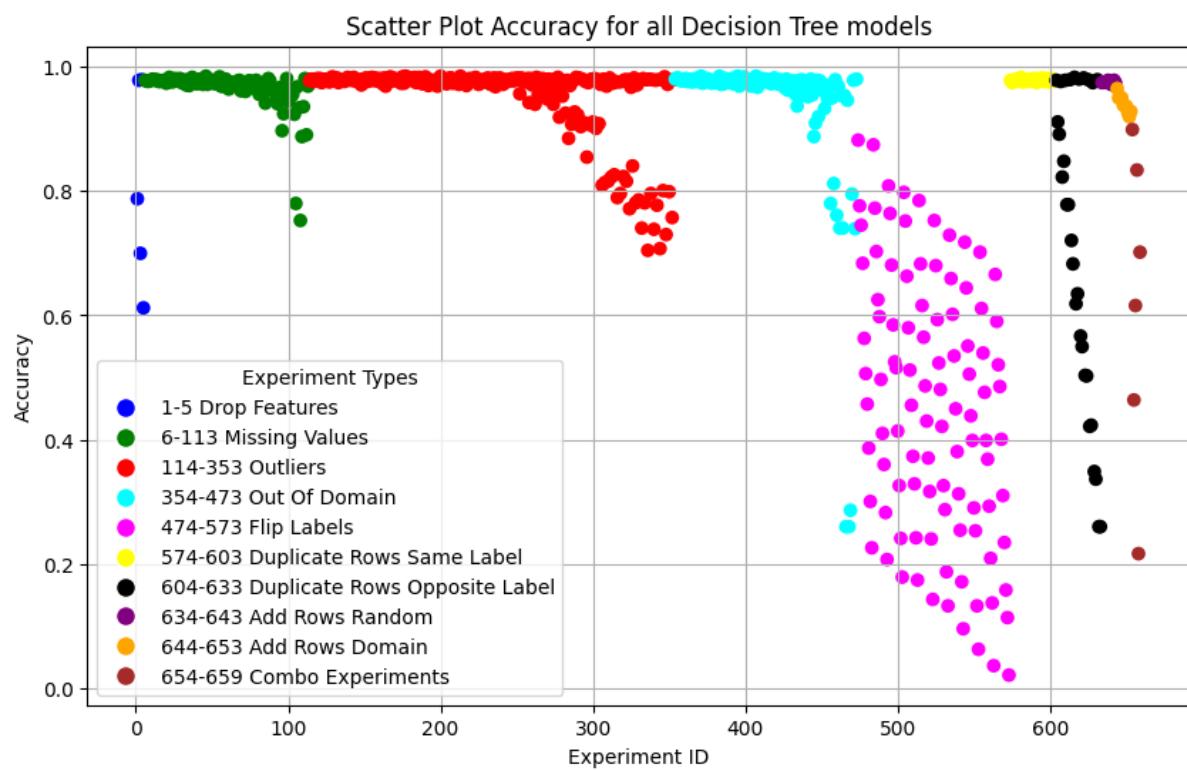
Sono stati generati ulteriori grafici complessivi sui vari esperimenti, questa volta analizzando singolarmente ogni metrica, al fine di avere un'idea della distribuzione dei valori più specifica. Sono riportati i grafici dell'accuracy per i modelli, ma sono presenti anche per le altre metriche nel notebook "experiments.ipynb".

SVM





Decision Tree



Quello che è stato analizzato con i Violin Plot è confermato nella visualizzazione di questi due grafici, dove effettivamente la maggior parte della distribuzione tende ad 1.

Viene però mostrato più nel dettaglio, soprattutto attraverso gli Scatter Plot, quali esperimenti vanno ad abbassare le metriche, notabile in modo evidente nello Scatter Plot del Decision Tree, dove l'esperimento Flip Labels porta al maggior decremento.

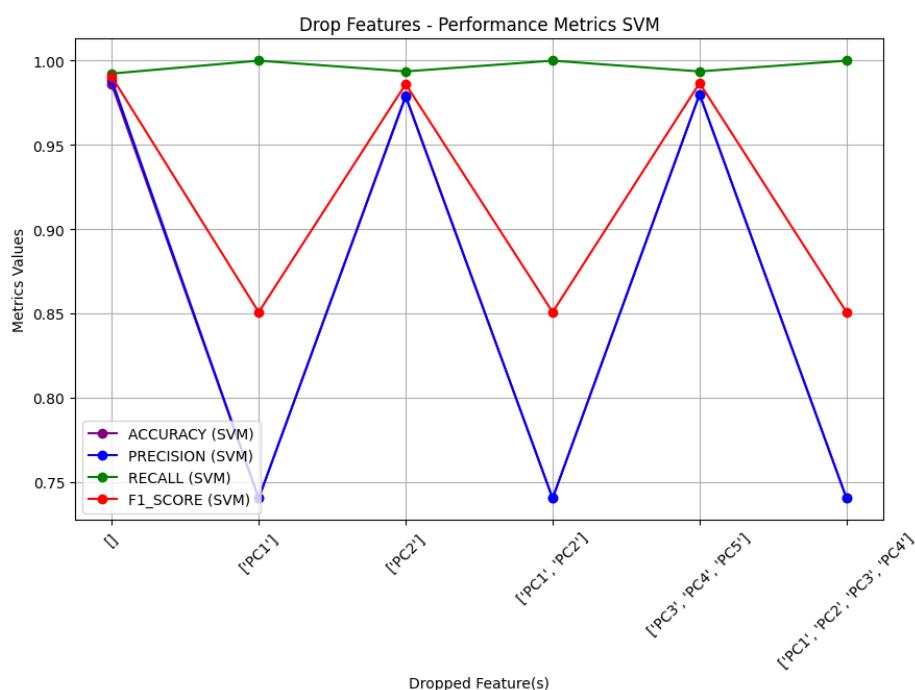
Il SVM invece tende più allo stabilizzarsi su determinati valori e rimanere pressoché costante per tipo di esperimento, come si può notare dagli outliers per SVM nello Scatter Plot.

Drop Features

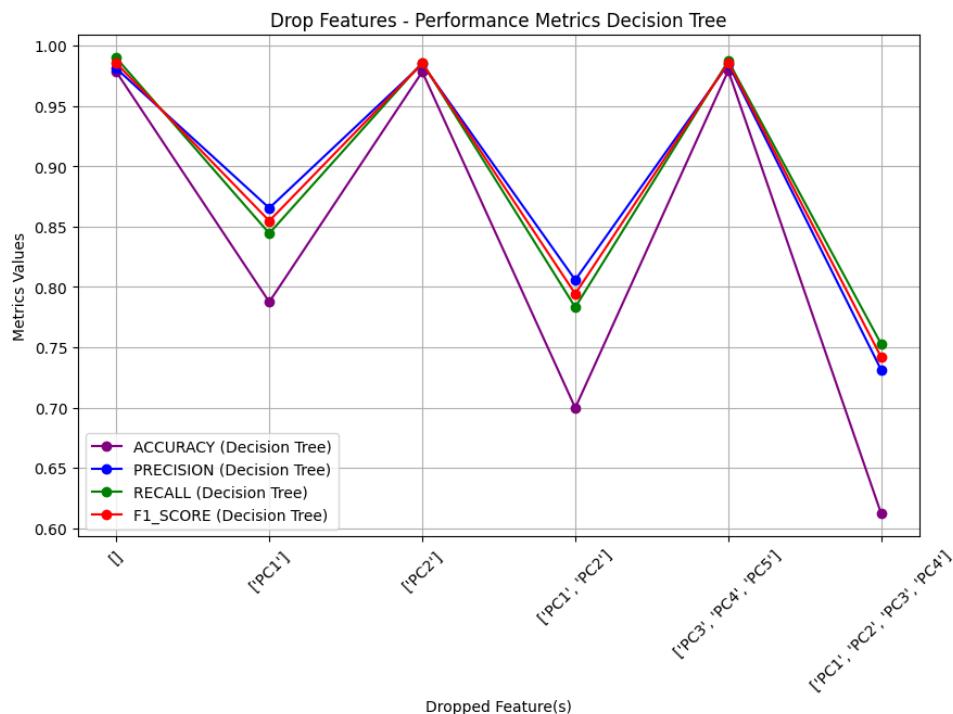
Il primo tipo di esperimento effettuato sul dataset è stato quello di droppare le features (differisce dall'introduzione di valori nulli in quanto la feature in esame non viene proprio considerata dal modello di Machine Learning).

Dato che il drop delle features viene effettuato con le componenti della PCA ci si aspetta un grande decremento delle prestazioni in relazione alla varianza spiegata di ogni componente, pertanto la mancanza di PC1 e/o PC2 dovrebbe influenzare maggiormente le prestazioni complessive.

SVM



Decision Tree



I tre grafici mostrano l'andamento delle prestazioni a seconda delle componenti eliminate, elencate in formato lista sull'asse x.

Abbiamo la lista vuota che indica il punto di partenza (quindi dataset pulito) delle metriche, che faranno da riferimento per i valori successivamente riscontrati.

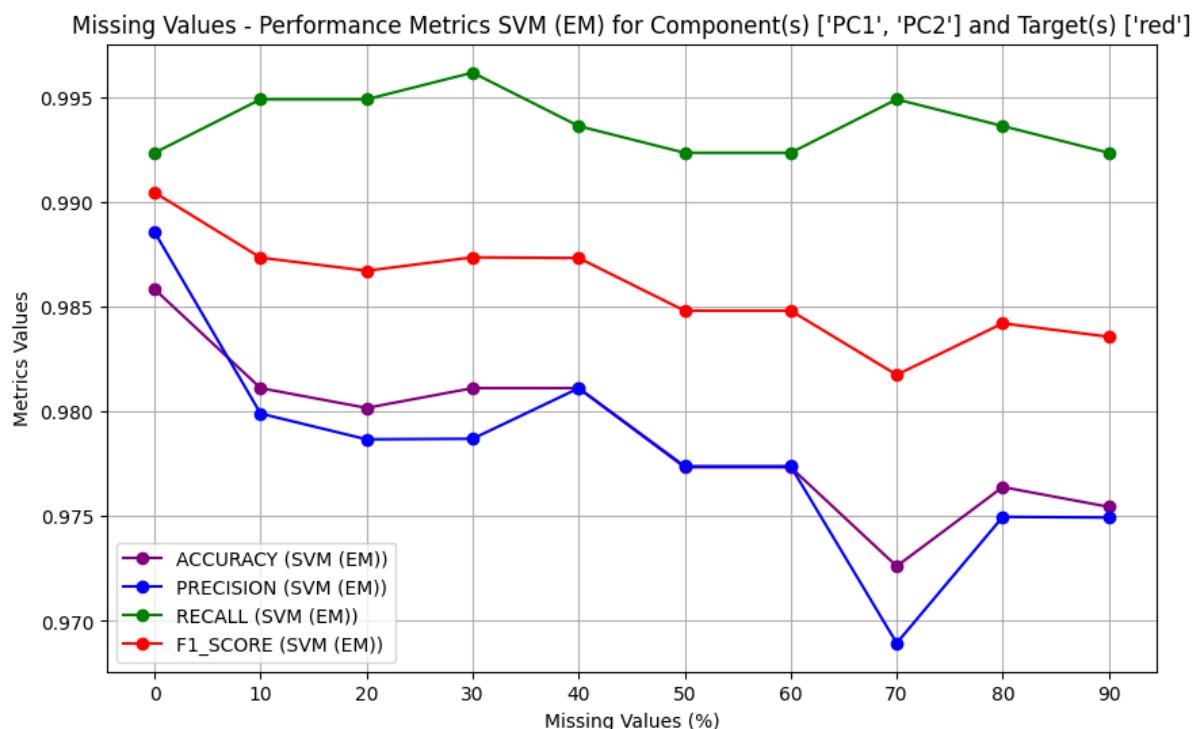
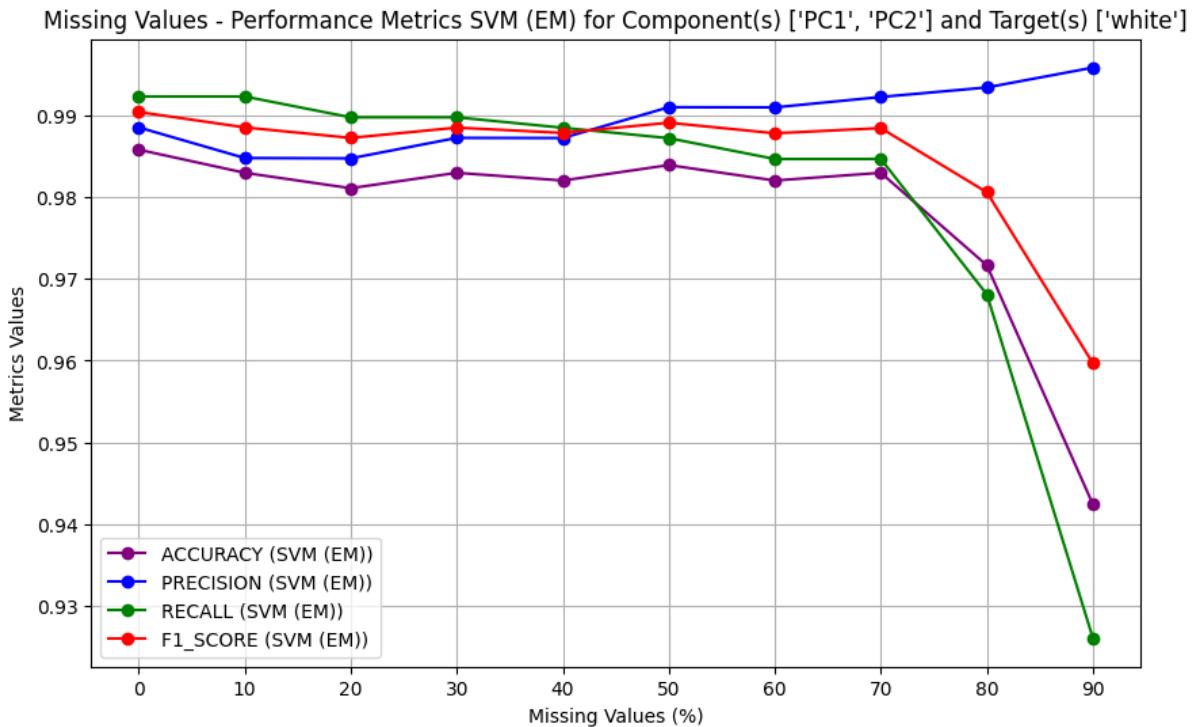
Come supposto introducendo l'esperimento, quando vengono eliminate le componenti PC1 e PC2 si hanno dei cali bruschi delle metriche, mentre se si considera PC3, PC4 e PC5 abbiamo un calo molto lieve. Si può notare inoltre, soprattutto per SVM, come le misure di recall e precision siano le più distanti, con recall tendente sempre all'1, quindi con una buona capacità di identificazione delle istanze positive ma, dato precision basso, anche di molti falsi positivi.

Missing Values

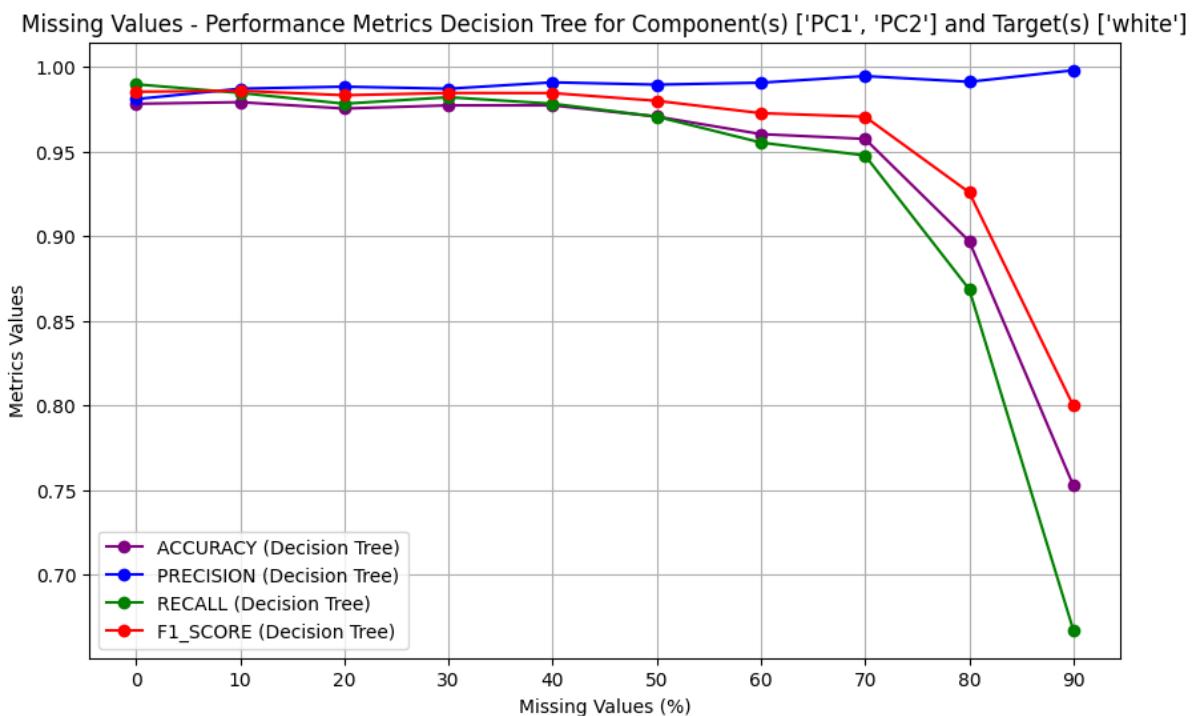
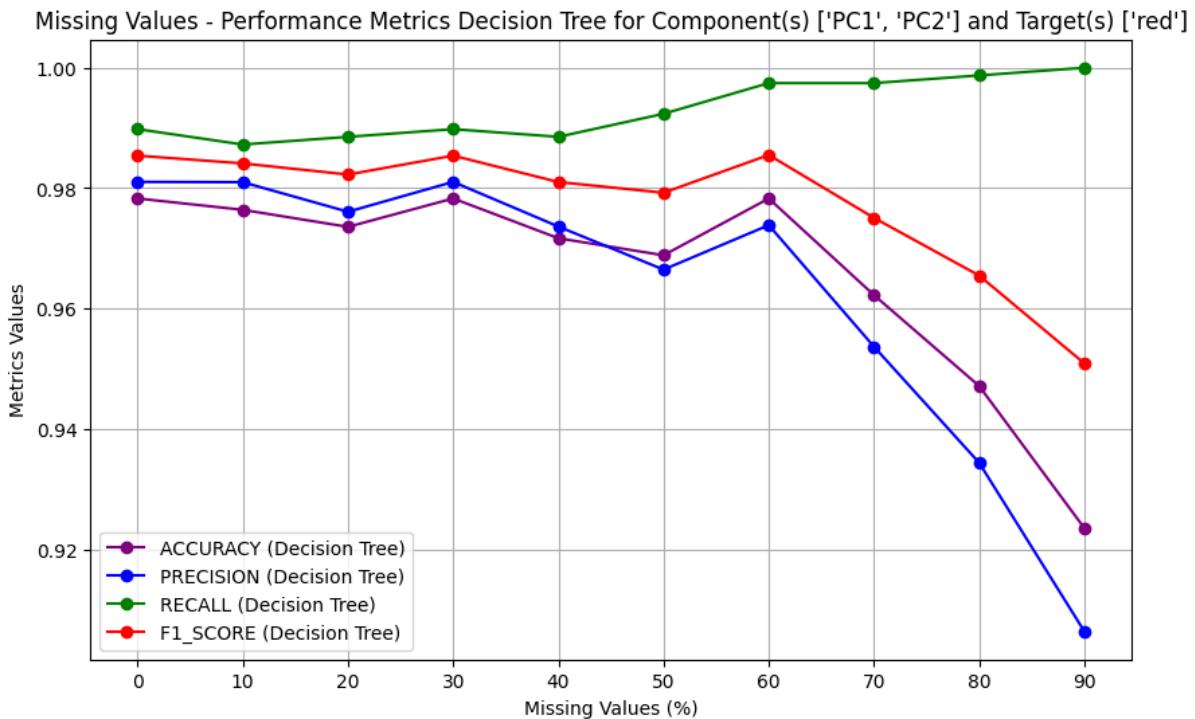
Il secondo tipo di esperimento effettuato sul dataset si è concentrato sull'introduzione dei valori mancanti, andando a violare come il drop delle features i principi di completezza. Quello che si aspetta con l'introduzione di valori mancanti è simile al drop delle features, più valori mancanti sono presenti su feature significative, minori saranno le prestazioni riscontrate. Sono stati

effettuati 108 diversi esperimenti coprendo tutte le combinazioni di percentuali, features e target, pertanto saranno riportati i grafici più significativi, in più sono stati esclusi i grafici che utilizzano **mean** in quanto presentano metriche peggiori che con **EM**, come rilevato da un'analisi sulle metriche riscontrate dagli esperimenti. **EM** e **mean** sono stati utilizzati in quanto SVM e la non supporta i missing values, infatti SVM fallisce in fit. **EM** e **mean** verranno trattati nel dettaglio nell'approfondimento dedicato.

SVM (EM)



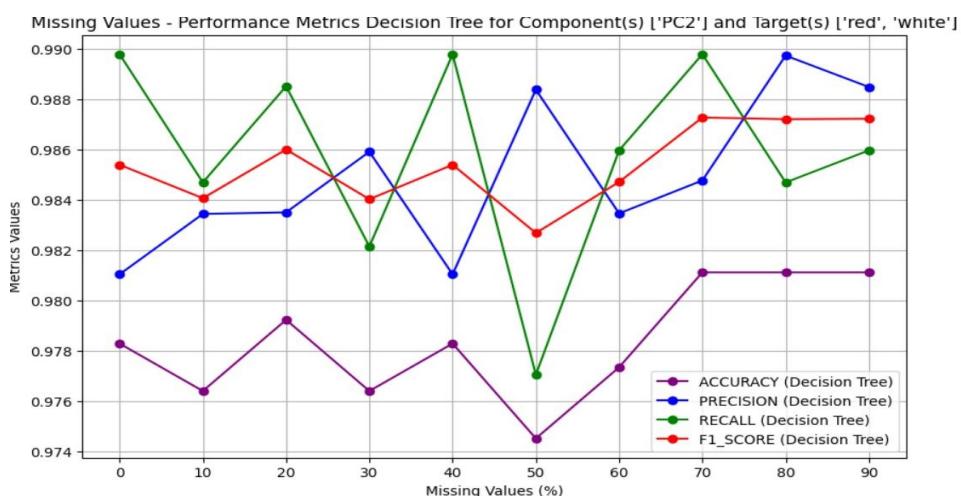
Decision Tree



Analizzando ogni grafico, si può notare come EM sia in grado di mantenere comunque delle ottime metriche, non scendendo sotto i 0.90 per SVM indipendentemente dalle features condizionate e il target. Abbiamo però sempre la tendenza di avere la precision o recall alto e viceversa.

Un comportamento più smooth lo possiamo notare sul Decision Tree, il quale supporta nativamente i valori mancanti. Considerando solo il target “red” notiamo che le prestazioni rimangono comunque ottime, dovute probabilmente al fatto che il target è sbilanciato in favore dei vini “white”, infatti analizzando il grafico dove si prende in considerazione il target “white” le prestazioni scendono fino a 0.70, in quanto essendo la maggioranza, i vini bianchi vengono più impattati.

Infine, il Decision Tree, analizzando in questo caso particolare PC2 e le due labels, mostra in modo evidente l’operazione di splitting, indicata dalle ondulazioni presenti dovute probabilmente ad una ricerca della feature corretta su cui approssimare i valori mancanti.



Nota sullo Splitting

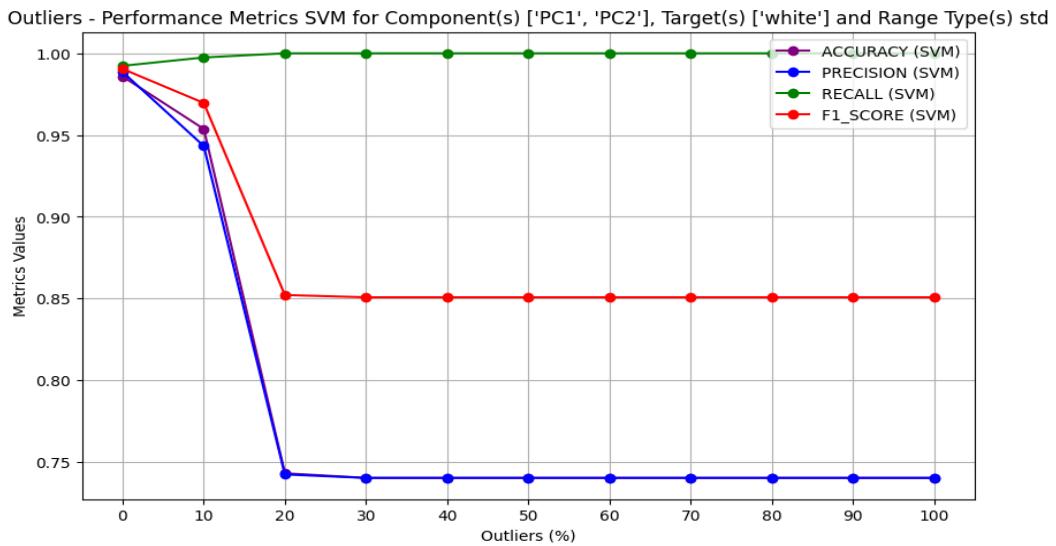
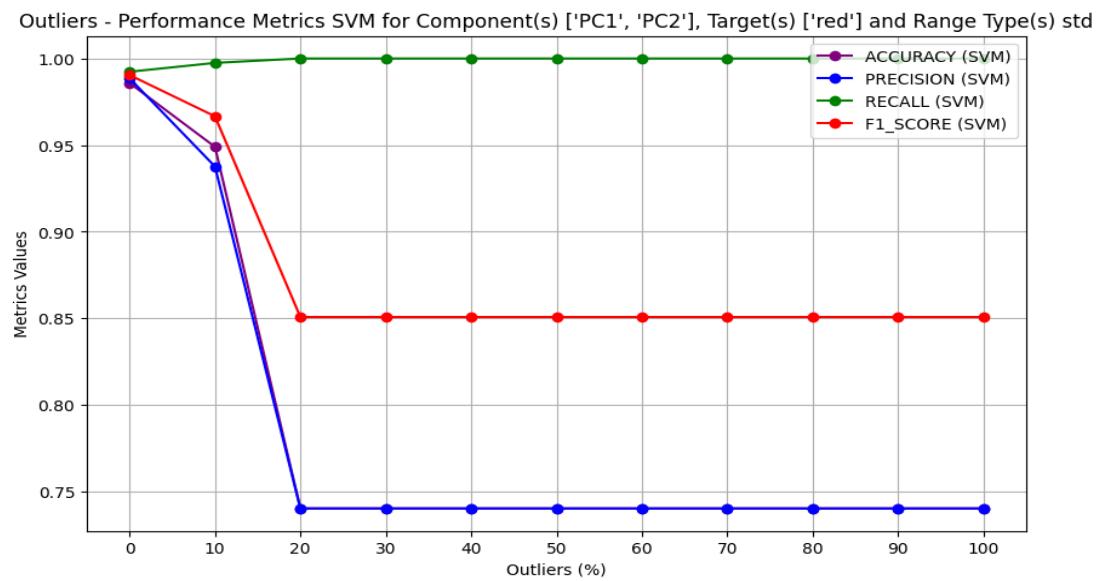
In sostanza, quando si crea uno split su un nodo basato su una feature che contiene valori mancanti abbiamo:

- Per ogni possibile split sui dati non mancanti (tenendo da parte i dati mancanti), lo splitter valuta quale split è migliore includendo tutti i valori mancanti in un nodo.
- Durante la predizione, le istanze con valori mancanti vengono assegnate al nodo corrispondente a quello utilizzato nello split trovato durante l’addestramento. Se entrambi i nodi hanno la stessa valutazione del criterio di split, il nodo destro è preferito per le istanze con valori mancanti.
- Se durante l’addestramento non sono stati osservati valori mancanti per una determinata feature, durante la fase di predizione i valori mancanti vengono mappati al nodo figlio che contiene il maggior numero di campioni.

Outliers

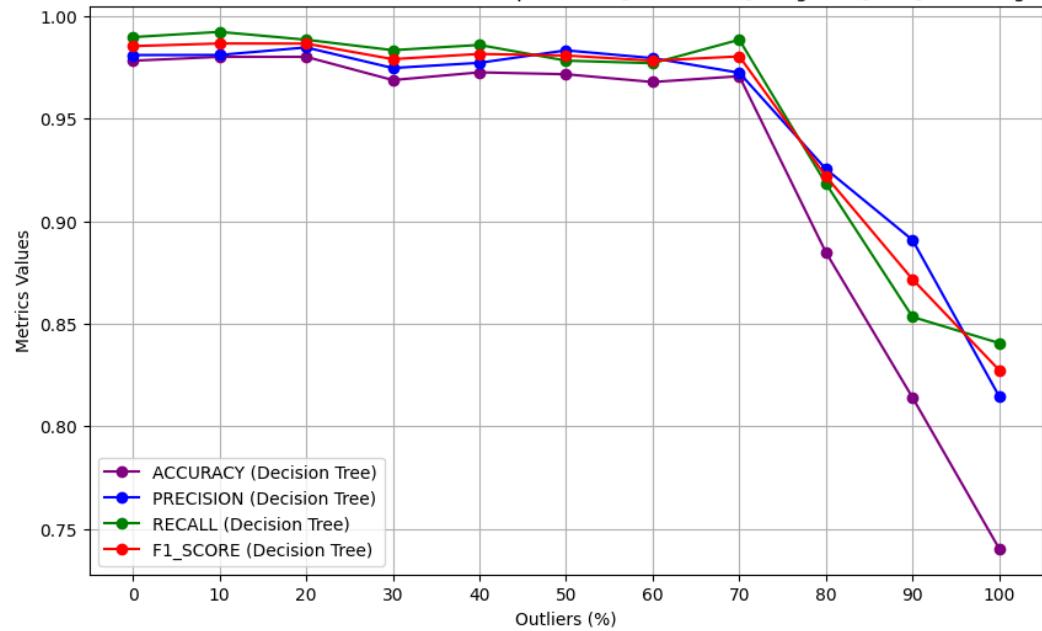
Il terzo tipo di esperimento effettuato sul dataset si è concentrato sull'introduzione degli outliers, violando la consistenza. Questi, sono stati introdotti basandosi sul calcolo di range utilizzando media/deviazione standard o IQR, similmente a come è stato fatto durante il controllo della consistenza, ma questa volta sulle componenti PCA. Come per i precedenti esperimenti ci si aspettano dei decrementi andando a sporcare le prime due componenti. Sono riportati i grafici più significativi su 240 esperimenti, analizzando le prime due componenti e i differenti target:

SVM

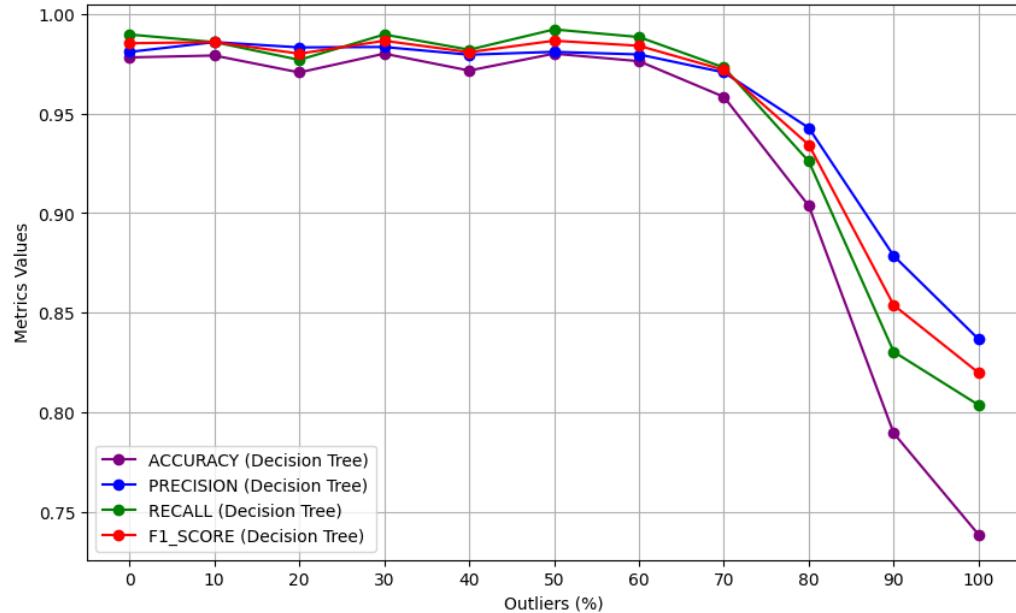


Decision Tree

Outliers - Performance Metrics Decision Tree for Component(s) ['PC1', 'PC2'], Target(s) ['red'] and Range Type(s) std



Outliers - Performance Metrics Decision Tree for Component(s) ['PC1', 'PC2'], Target(s) ['white'] and Range Type(s) std



In questo caso è stato analizzato come tipo di range media/deviazione standard, ma risultati simili sono ottenuti con IQR.

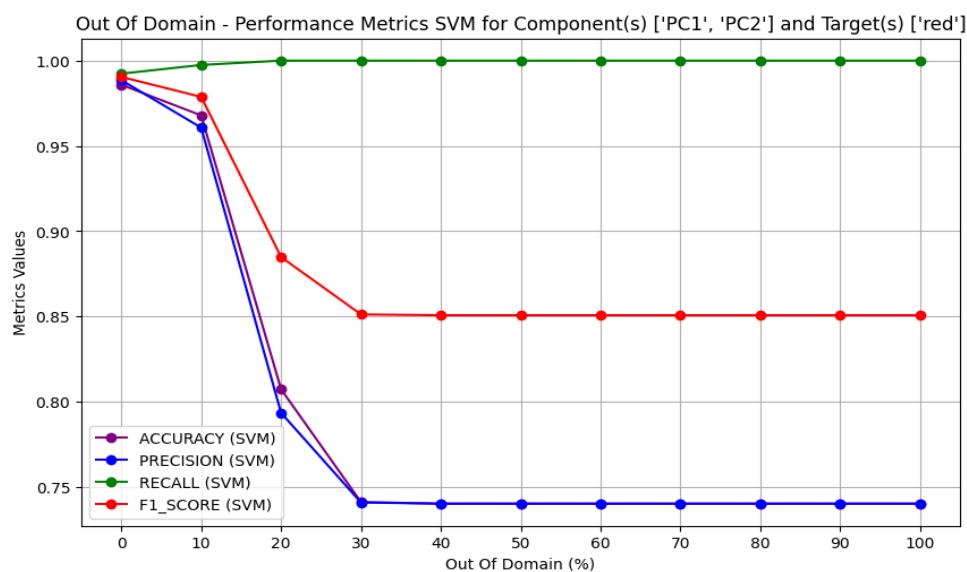
Da questi grafici si può notare un decremento delle prestazioni fino a poco meno di 0.75 per tutti i modelli, indipendentemente dal target, ma questa volta il Decision Tree ha una discesa più fluida, tollerando maggiormente gli outliers. La SVM invece arriva al “minimo” molto più velocemente.

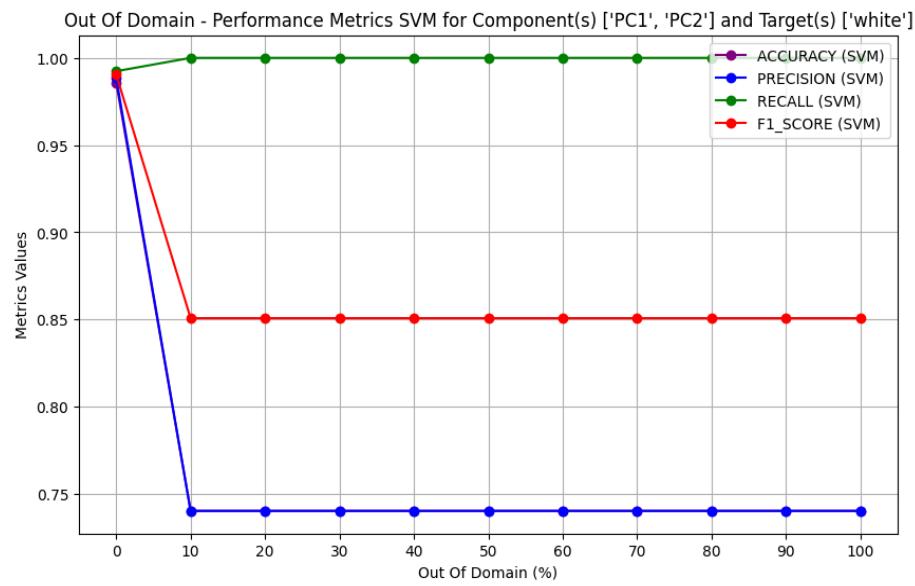
Out of Domain Values

Il quarto tipo di esperimento effettuato sul dataset si è focalizzato sull'introduzione di valori fuori dal dominio, o più precisamente valori molto alti che differiscono pesantemente dai valori attesi in PCA. La generazione di questi ultimi segue la stessa logica degli outliers, ma il threshold utilizzato è equivalente a 10, molto elevato. Come per gli esperimenti precedenti ci aspettiamo lo stesso comportamento complessivo.

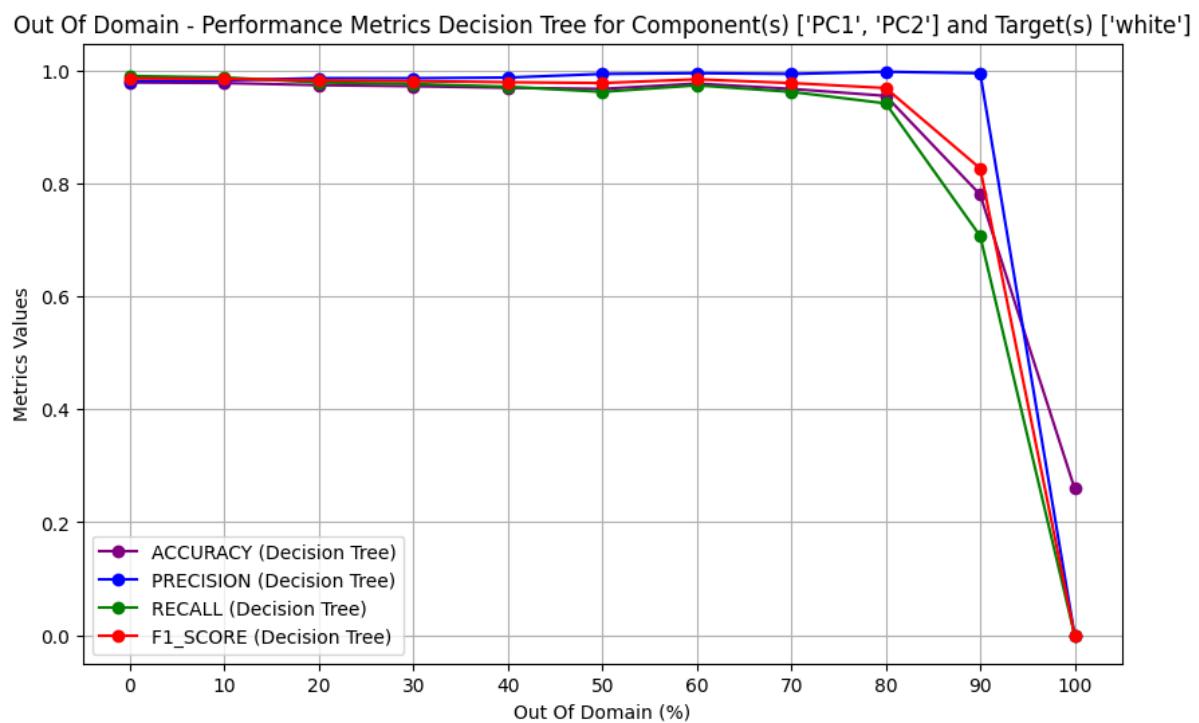
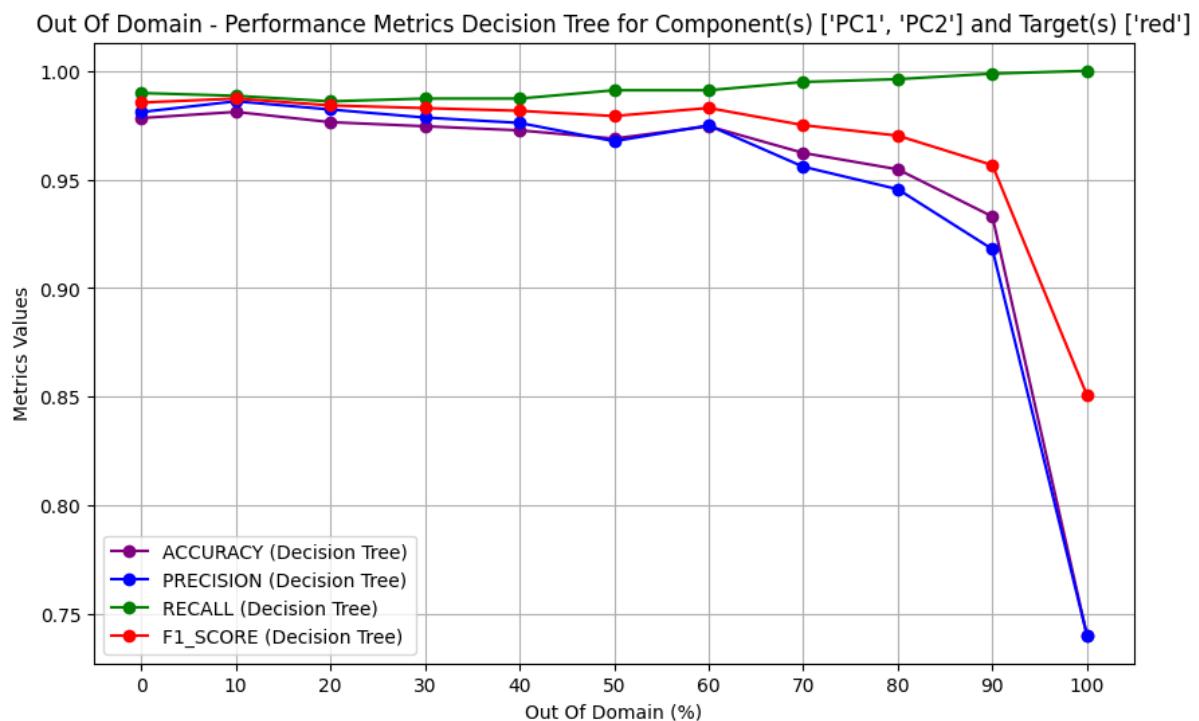
Sono riportati i grafici significativi per 120 esperimenti:

SVM





Decision Tree



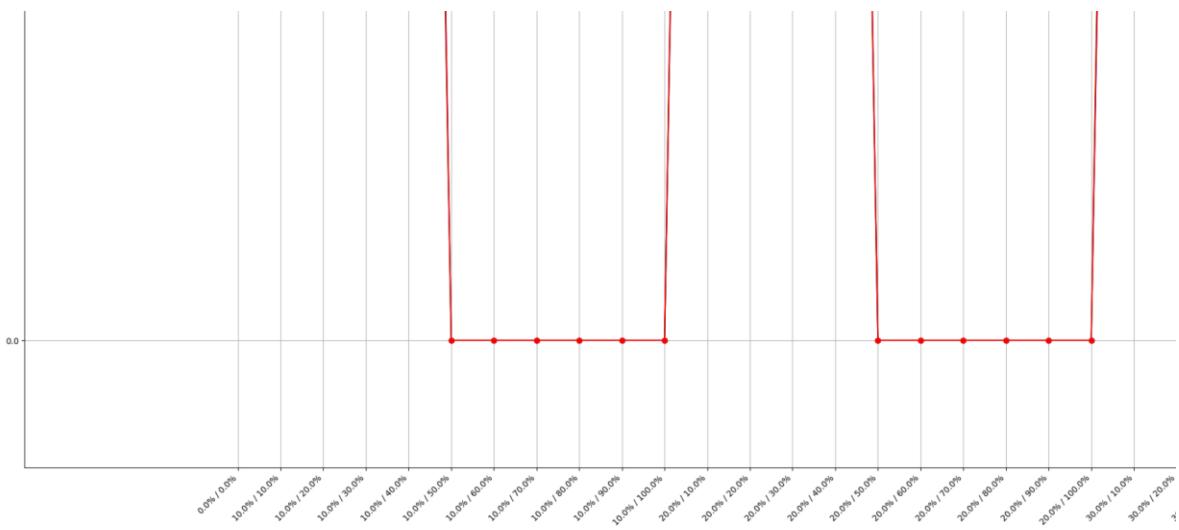
Il comportamento riscontrato dai modelli si trova abbastanza in linea con quello visto con gli outliers, ma con alcune differenze.

Il SVM è l'unico modello che fatica di più, rimanendo pressoché invariato, ma peggiorando nel caso di vini "white" (convergendo al minimo già a 0.10).

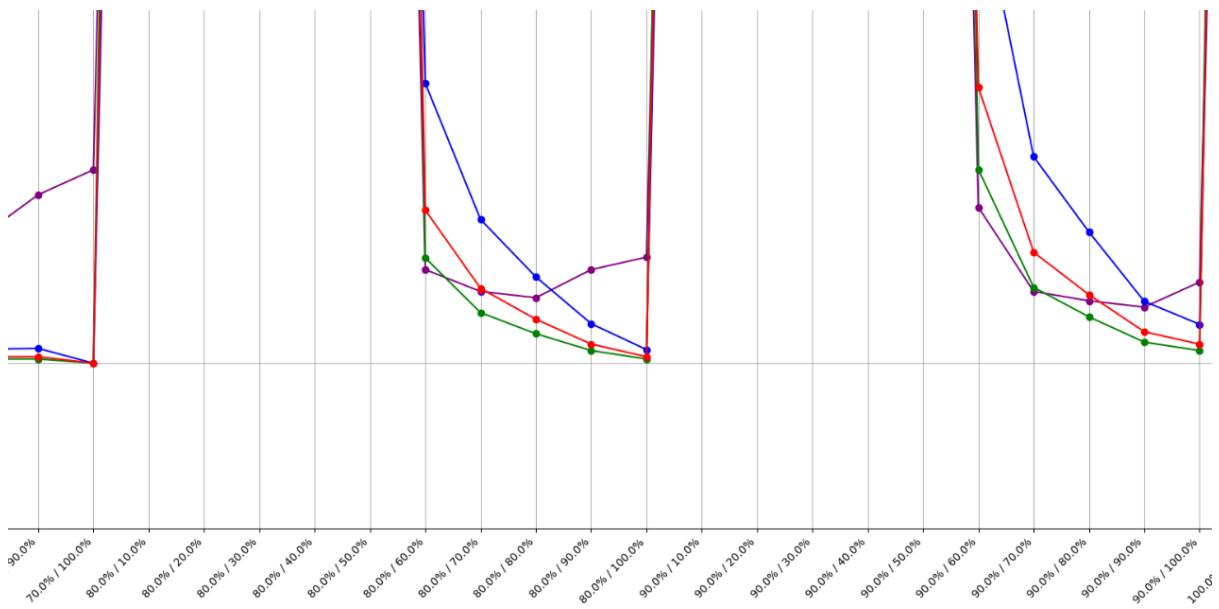
Il Decision Tree conferma il comportamento riscontrato con gli Outliers.

Flip Labels

Il quinto tipo di esperimento effettuato sul dataset è stato il capovolgimento dei valori del target type. Questo esperimento induce una grande fonte di incertezza, molto più pronunciata rispetto ai precedenti esperimenti, in quanto stiamo inducendo i modelli ad apprendere in modo completamente errato, considerando un vino bianco rosso e viceversa. Attraverso delle percentuali sono state invertite le label di tipo rosso e bianco per un totale di 100 esperimenti. Vengono riportate le sezioni di uno dei grafici (in quanto molto grandi e non visualizzabili correttamente in un foglio) del comportamento delle prestazioni. Ogni modello presenta lo stesso comportamento quindi verrà analizzato più nel dettaglio il valore delle metriche al variare della percentuale:

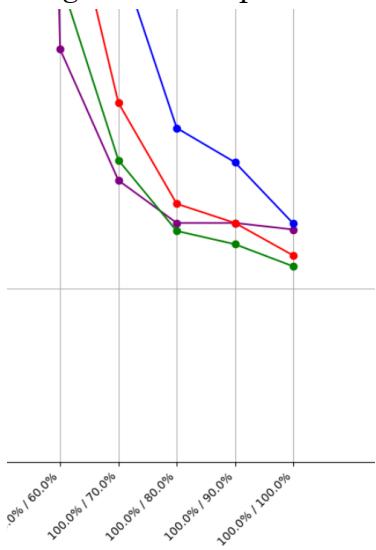


Tenendo fissa al una label e capovolgendo l'altra possiamo notare come ci sia un picco delle prestazioni a 0, rendendo i modelli completamente inutilizzabili. In particolare viene tenuta fissa "red" e capovolta "white".



Un comportamento più interessante lo abbiamo quando:

- La label “white” viene poco sporcata, ma “red” si, dove si verifica una perdita più graduale e non netta, convergendo a 0 solo al 100%, analizzato dal grafico di sopra.



- Quando entrambe le features vengono capovolte con alte percentuali (Es. 90% e 60%) si ha comunque un andamento meno netto e soprattutto con una convergenza leggermente maggiore di 0 (indicato dalla riga orizzontale). Questo si nota ancora di più nel seguente grafico, dove entrambe le labels sono completamente capovolte (in pratica tratta i vini rossi come bianchi e viceversa), non avendo accuracy e le altre metriche uguali a 1 il modello andrà a “sbagliare”, andando a predire correttamente.

Infine vi è riportato una sezione di grafico per il Decision Tree, in quanto, nonostante complessivamente abbia un comportamento equivalente agli altri

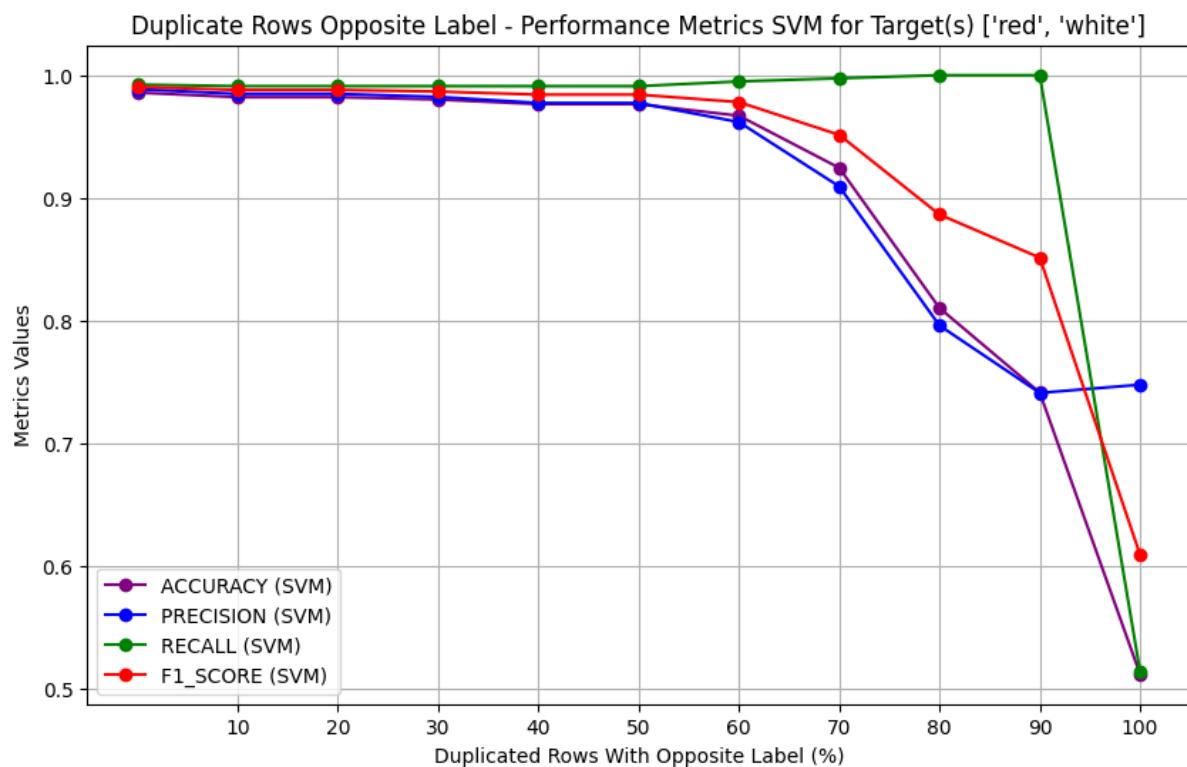
modelli, ha una discesa molto più graduale, similmente a delle “spine” per tutte le percentuali trattate prima:



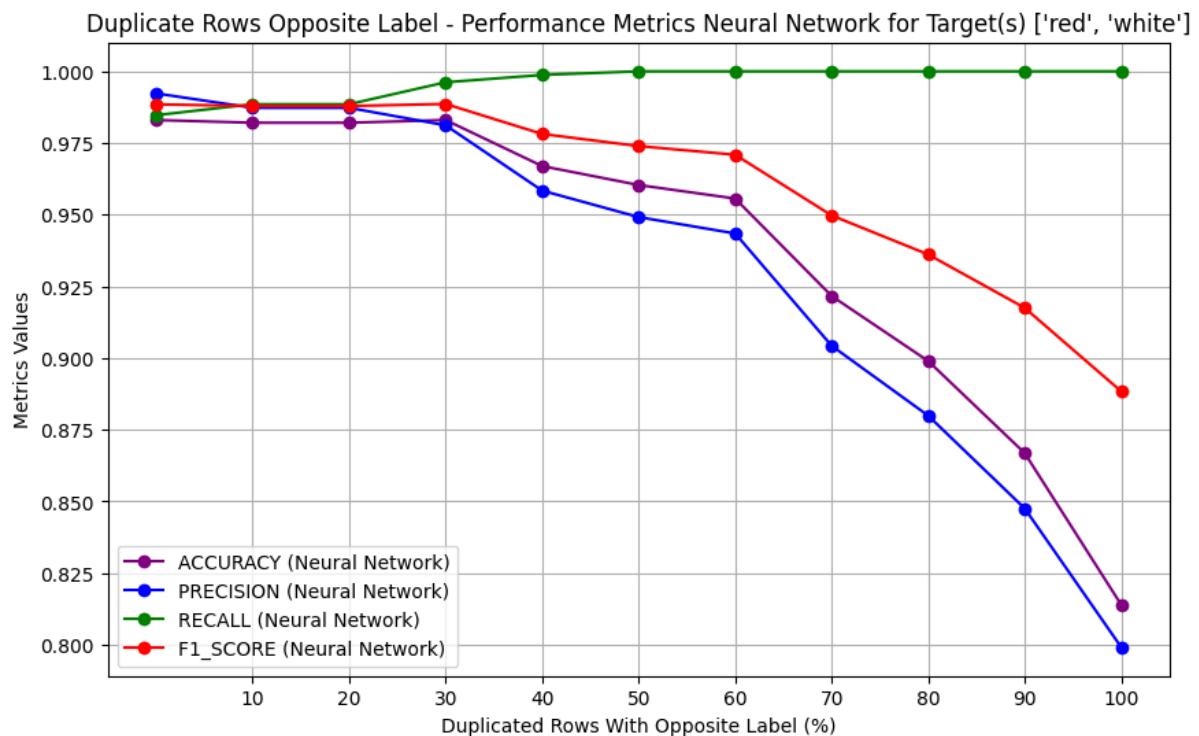
Duplicate Rows

In questo esperimento andiamo a vedere come la duplicazione delle istanze influenza le performance dei 3 modelli. Saranno presentate varie tipologie e combinazioni di questo esperimento. In particolare per ogni modello abbiamo duplicato una percentuale di istanze mantenendo la stessa label solo su vini rossi, successivamente solo su vini bianchi e poi sia su vini bianchi che rossi. In un secondo momento abbiamo ripetuto il tutto flippando la label. Di seguito vengono riportati i grafici più significativi per tipologia di esperimento, si guardi il notebook “experiments.ipynb” per poterli visionare tutti.

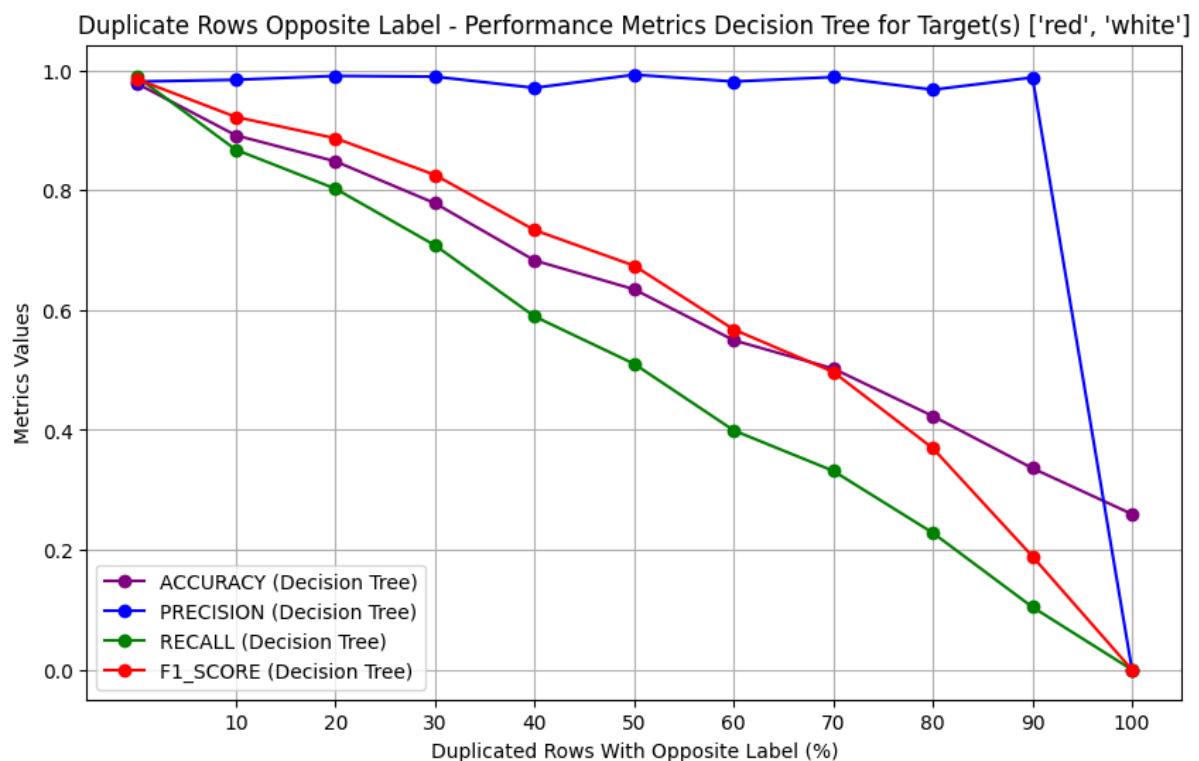
Duplicate Rows Opposite Label



Nel grafico soprastante sulle ascisse troviamo la percentuale di istanze flippate e sulle ordinate l'andamento delle metriche delle performance. La recall inaspettatamente migliora all'aumentare delle duplicazioni per poi calare improvvisamente quasi dimezzandosi al raggiungere del 100%. F1 score invece subisce un declino minimo fino al 70% per poi diminuire in maniera più marcata man mano che si avvicina al 100% raggiungendo quasi la Recall, la diminuzione di questa metrica era prevedibile. Anche Precision e Accuracy subiscono leggere diminuzioni fino al 60% seppur leggermente maggiori rispetto a F1 score, passato il 60% invece tendono a subire un declino più vertiginoso che va ad assestarsi a 0.75 circa per precision raggiunto il 90% mentre Accuracy diminuisce in maniera sempre più marcata fino a raggiungere recall.

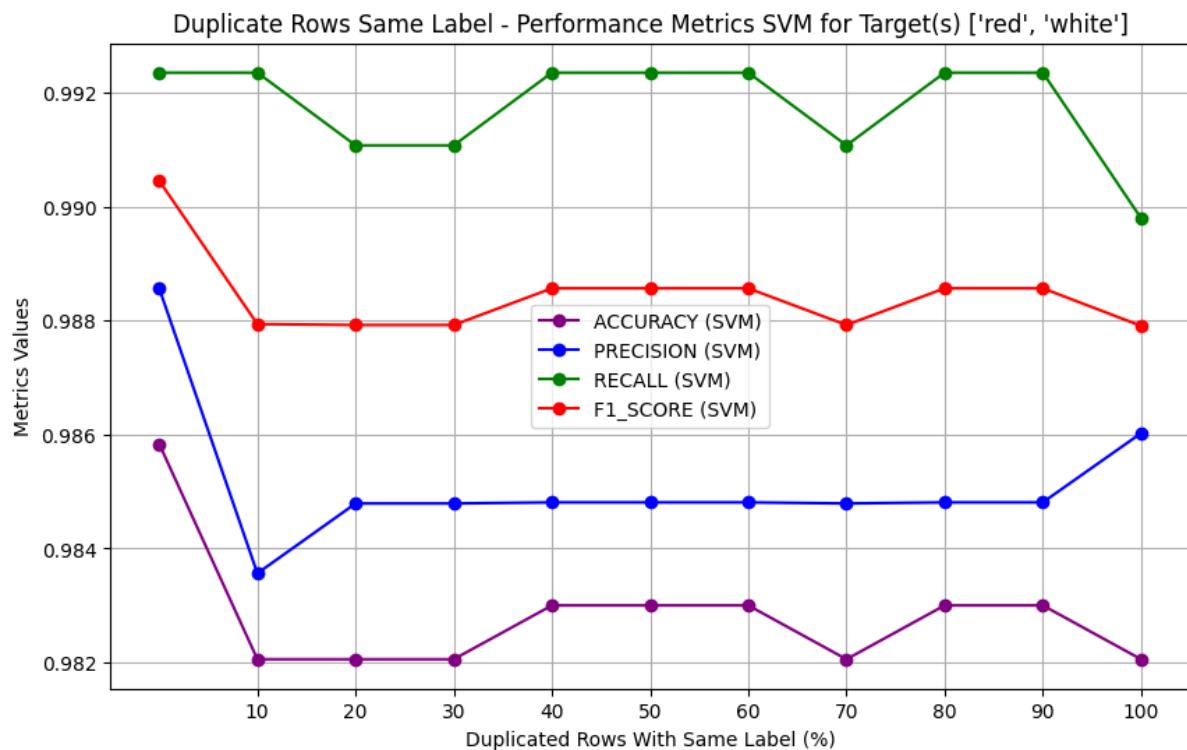


La recall come nella SVM tende ad aumentare al crescere della duplicazione delle righe. Le restanti tre metriche all'aumentare delle duplicazioni tendono a peggiorare, in maniera minima fino al 60%, più evidentemente dal 70% in poi. Precision e Accuracy perdono circa venti punti percentuali mentre F1 Score ne perde solamente dieci.

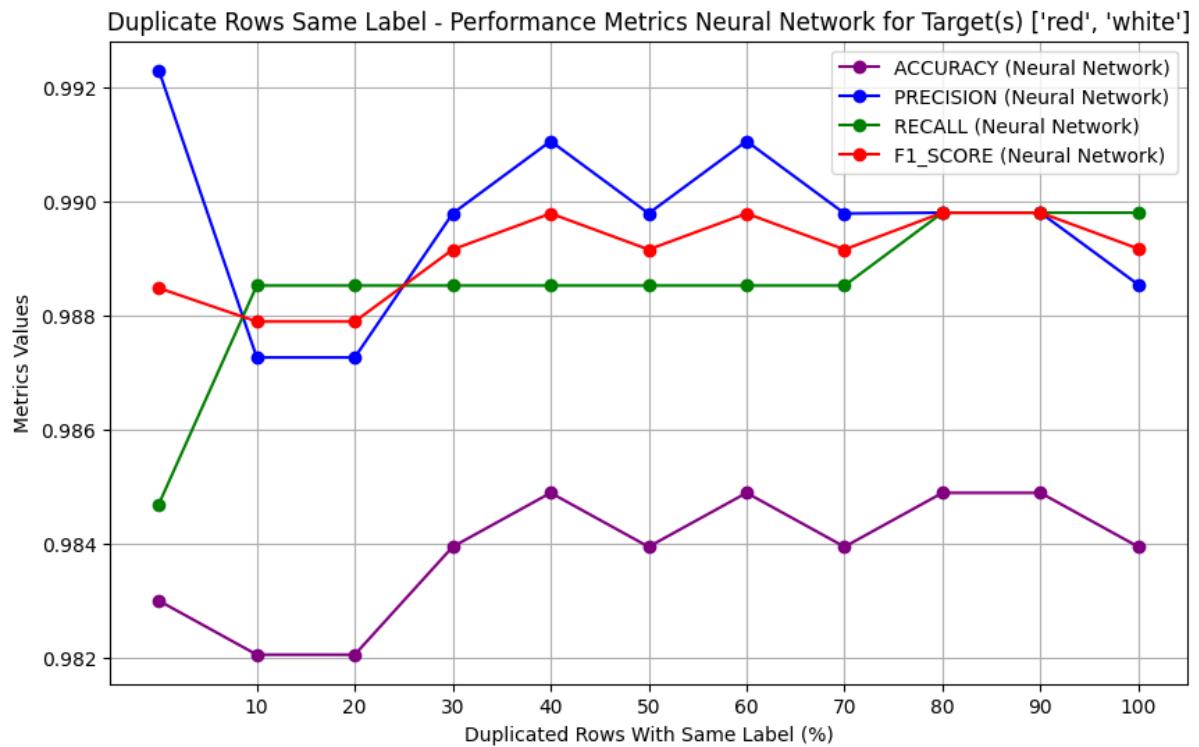


L'Albero Decisionale mostra dei risultati diversi dai due modelli appena descritti. Precision tende ad aumentare fino al 90% per poi andare a 0 al 100%. Le restanti tre metriche subiscono fin da subito un peggioramento continuo e costante che come per Precision termina in 0 ad eccezione di accuracy che va ad assestarsi a 0.28 circa.

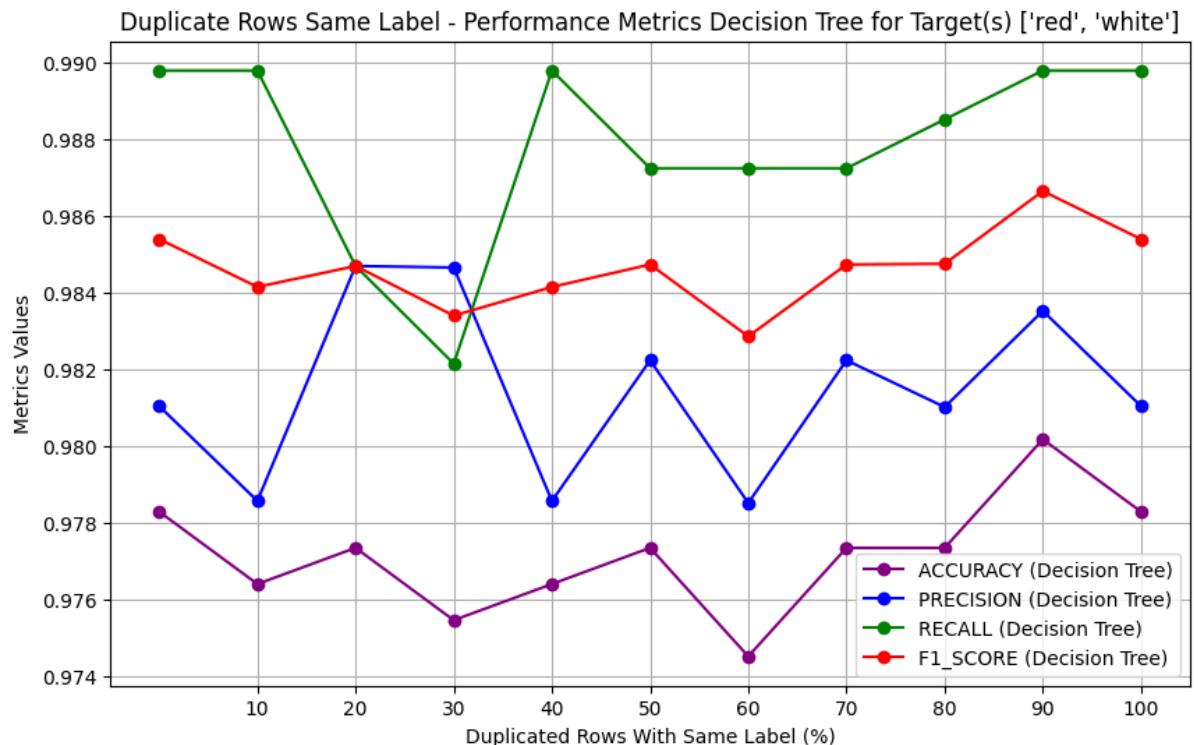
Duplicate Rows Same Label



Quando l'istanza viene duplicata senza cambiare l'etichetta, tutte le metriche hanno un leggerissimo decremento che è sempre minore o uguale dello 0.5%. Inizialmente era atteso un miglioramento/mantenimento delle performance, inaspettatamente c'è stato un calo minimo.



Le metriche subiscono cambiamenti non significativi. Accuracy, F1 Score e Recall tendono ad avere un miglioramento non significativo. Precision invece seppur in maniera non significativa peggiora leggermente.

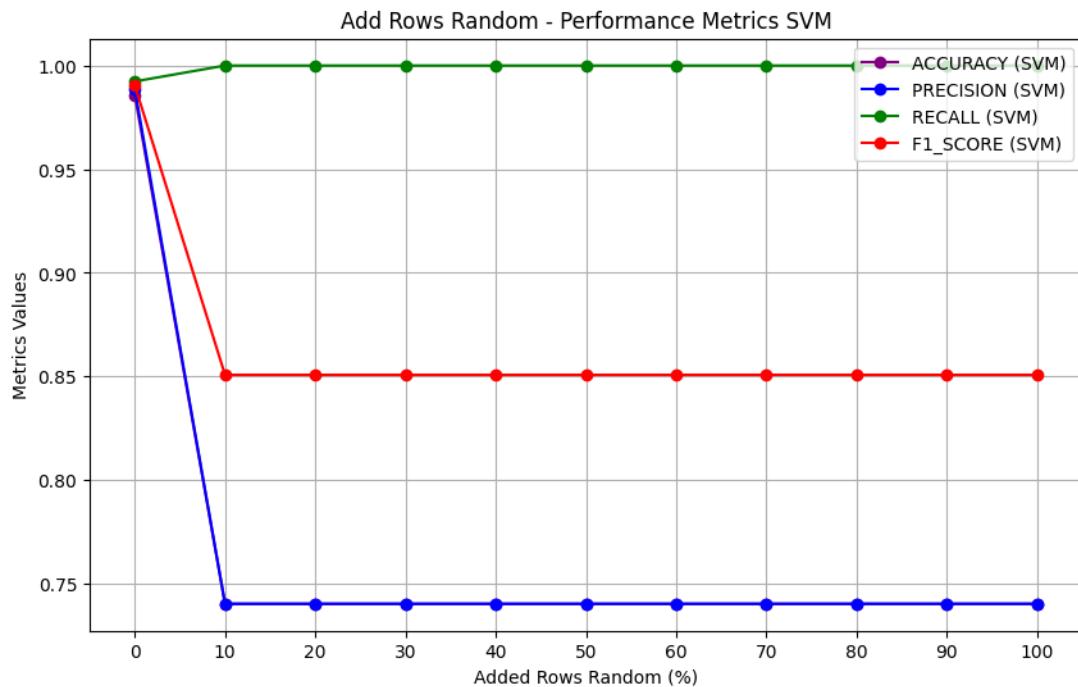


Le metriche subiscono cambiamenti non significativi. Tutte tendono a migliorare al crescere dei duplicati. In Recall e Precision si notano cambiamenti più repentini a percentuali basse.

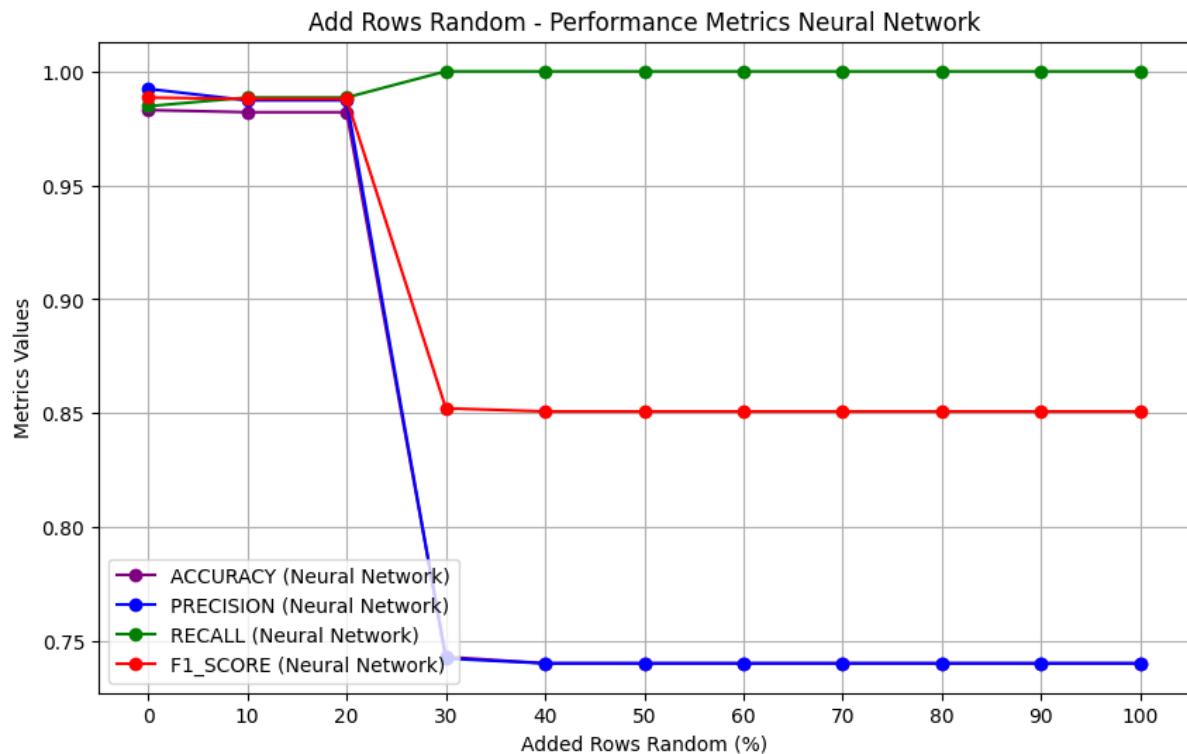
Add Rows

In questo esperimento andiamo a osservare come la generazione di nuove istanze nel dataset va ad impattare sulle performance. Verranno eseguiti due tipi di esperimenti: la prima metà di esperimenti genera delle righe in maniera randomica senza alcun tipo di vincolo, possono essere creati outlier e valori completamente fuori dominio*. La seconda metà di esperimenti crea istanze in maniera completamente casuale con l'unica differenza che i valori generati sono compresi nel proprio dominio*. Visto che viene utilizzata la PCA la quale è composta da valori continui non esiste un vero e proprio dominio. Per dominio* si intende la generazione di valori casuali di valori compresi tra Mean \pm 3 * Std.

Add Rows Random



Nel grafico soprastante notiamo un leggero miglioramento per la Recall, mentre per Accuracy (la quale non si vede perchè è completamente sovrapposta con Precision) e Precision vi è un significativo abbassamento delle metriche seppur totalmente stabile dopo il primo 10% di righe generate. F1 Score subisce anch'essa un peggioramento significativo seppur minore di 0.1 rispetto a Precision. Si nota che la SVM dopo un iniziale declino delle performance generale rimane stabile nonostante il continuo aumento di righe random aggiunte.



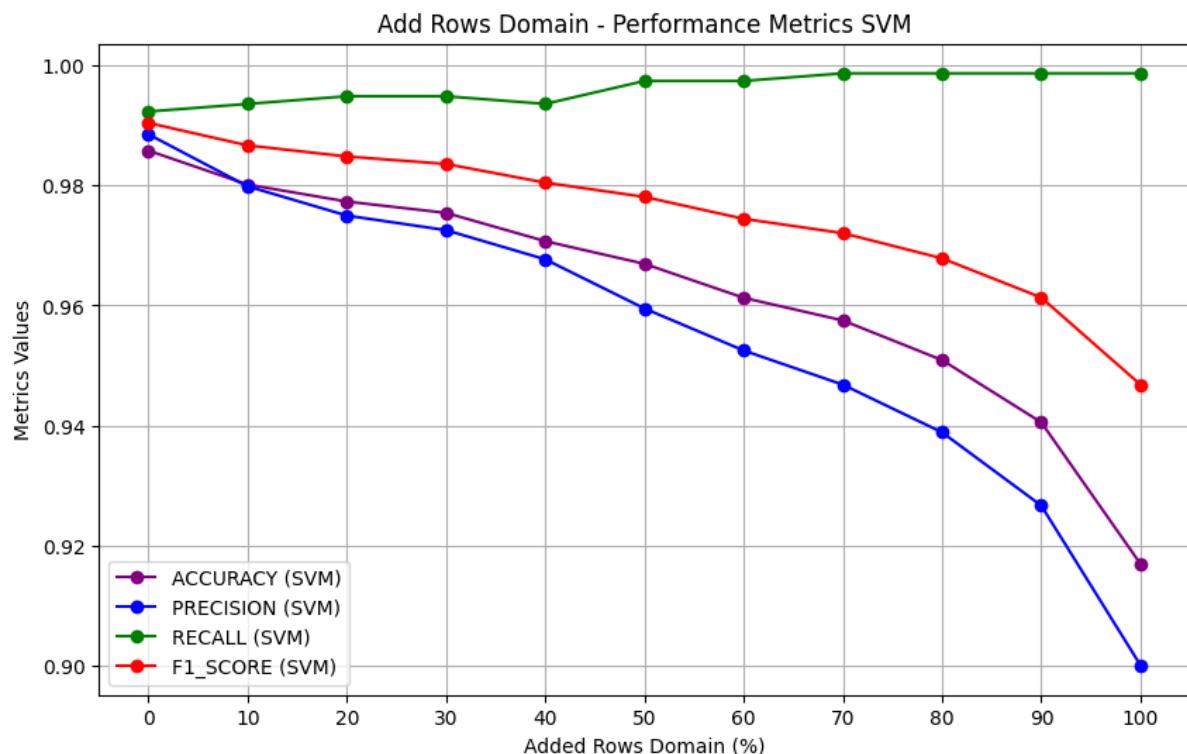
Nel grafico soprastante notiamo un leggero miglioramento per la Recall, mentre per Accuracy (la quale non si vede perchè è completamente sovrapposta con Precision) e Precision rimangono invariate per il primo 20%, dal 20% al 30% subiscono un importante calo che poi non subisce più alcun tipo di variazione. F1 Score rimane invariata per il primo 20%, dal 20% al 30% subisce un calo significativo ma meno impattante di Accuracy e Recall, successivamente non subisce più alcun tipo di variazione.

SVM subisce un iniziale generale calo delle prestazioni per poi rimanere stabile nonostante i continui aumenti di righe random. La stabilità delle performance al crescere delle nuove righe random fa sì che queste ultime non siano più un evento raro ma bensì del tutto normale, pertanto il modello riesce ad adattarsi e rimanere stabile.



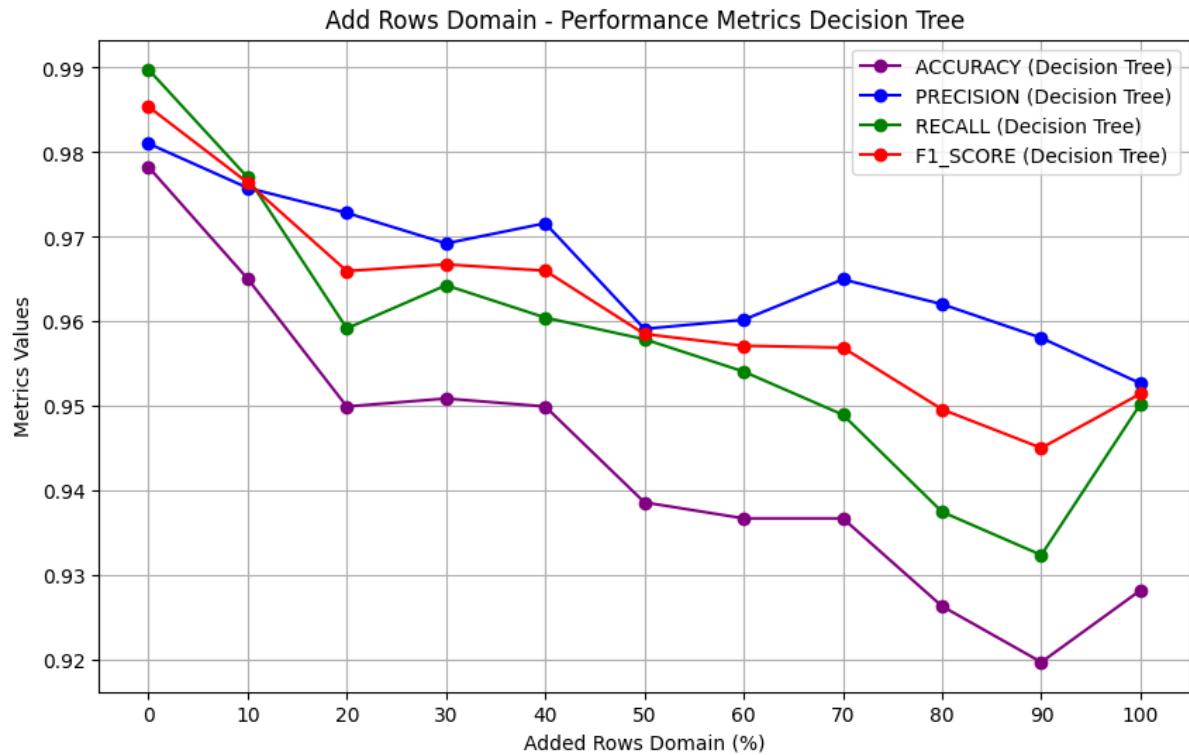
Il Decision Tree è il modello che meglio gestisce le righe random aggiunte e infatti subisce un leggero calo delle prestazioni ad eccezione di Precision. Guardando nell'insieme il grafico notiamo andamenti poco prevedibili per tutte le metriche prese in considerazione.

Add Rows Domain



Recall tende ad aumentare al crescere dell'aggiunta di righe. F1 Score tende sempre a diminuire in maniera lenta e costante ad eccezione della totale duplicazione di righe.

Accuracy e Precision seguono una diminuzione pressoché identica fino al 40%, successivamente Recall tende a diminuire più di Accuracy. Tutte le metriche diminuite continuano ad essere soddisfacenti.



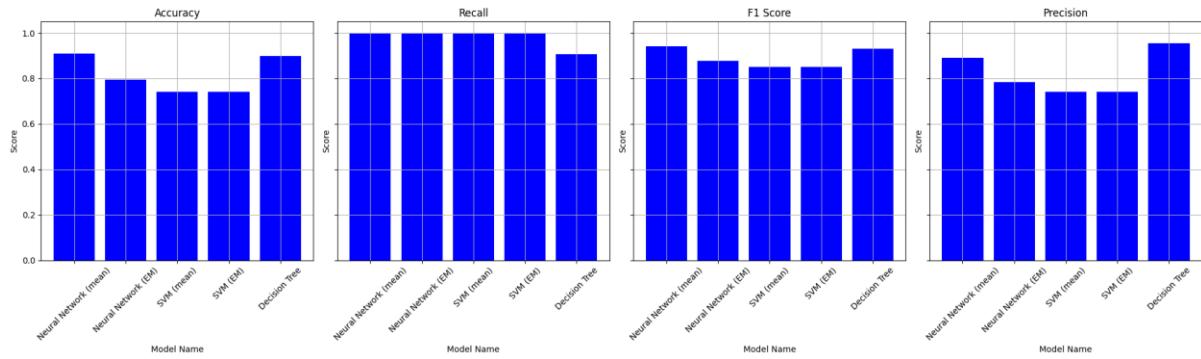
Il Decision Tree vede una diminuzione di tutte le metriche costante con qualche rialzo poco significativo. Generalmente c'è una diminuzione di pochi punti percentuali al crescere delle righe generate.

Confrontando i risultati di **ADD ROWS DOMAIN** con quelli di **ADD ROWS RANDOM** osserviamo che vi è un maggior calo nelle performance degli ultimi. Ciò è giustificabile dal fatto che in **ADD ROWS DOMAIN** aggiungiamo righe random diverse tra loro ma tutto sommato simili, invece in **ADD ROWS RANDOM** non poniamo alcun vincolo sulla generazione delle righe e di conseguenza è più probabile ottenere righe diverse tra loro e diverse dalle righe originali già presenti nel dataset.

Esperimenti Custom

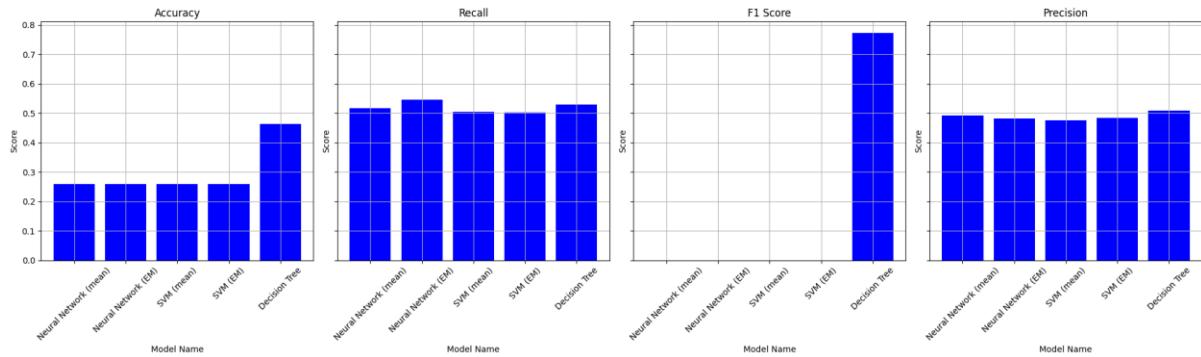
- PC3, PC4 e PC5 sono state dropate, è stato introdotto il 15% di missing values su PC1 e PC2, è stato introdotto un 5% di outliers su PC1, è stato

introdotto un **5%** di features fuori dominio su PC2, al **3%** dei vini bianchi e rossi sono state invertite le etichette, sono state duplicate il **10%** delle istanze mantenendo la **stessa** label e un **5%** cambiandola, sono state aggiunte 1.5% di righe completamente random e un **3%** di **righe** con valori compresi tra il massimo e il minimo di ogni features.



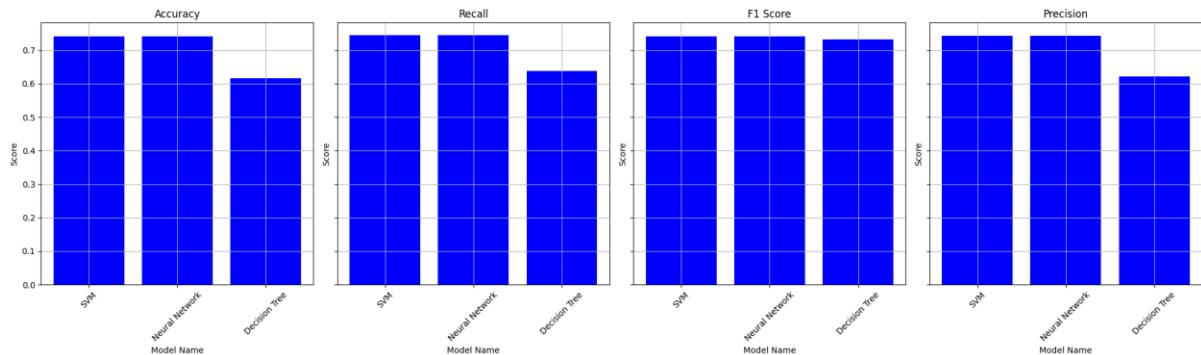
Accuracy e precision sono leggermente calate in media di circa il 10%, le restanti metriche invece sono rimaste invariate.

- Sono state dropgate **PC1** e **PC2**, alle restanti features sono stati aggiunti il **50%** di missing values, sono state **invertite** le label del **10%** dei vini rossi e del **45%** dei vini bianchi e sono state generate il **35%** di nuove righe senza alcun tipo di vincolo.



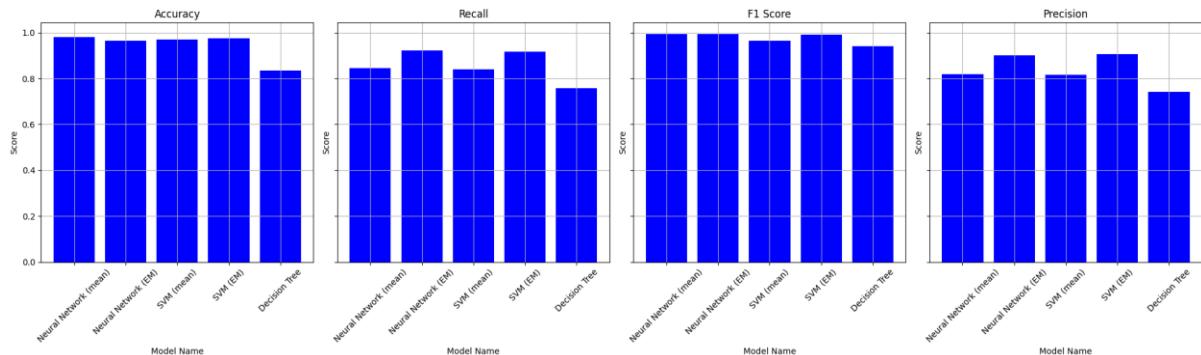
Avendo dropato le features più importanti e pesantemente sporco le rimanenti le prestazioni - come prevedibile - sono peggiorate. Il Decision Tree è riuscito ad adattarsi meglio a questi cambiamenti probabilmente grazie al fatto che si adatta già nativamente ai missing values.

- Sono state dropgate tutte le colonne tranne **PC5** e sono stati applicati il **5%** di outliers.



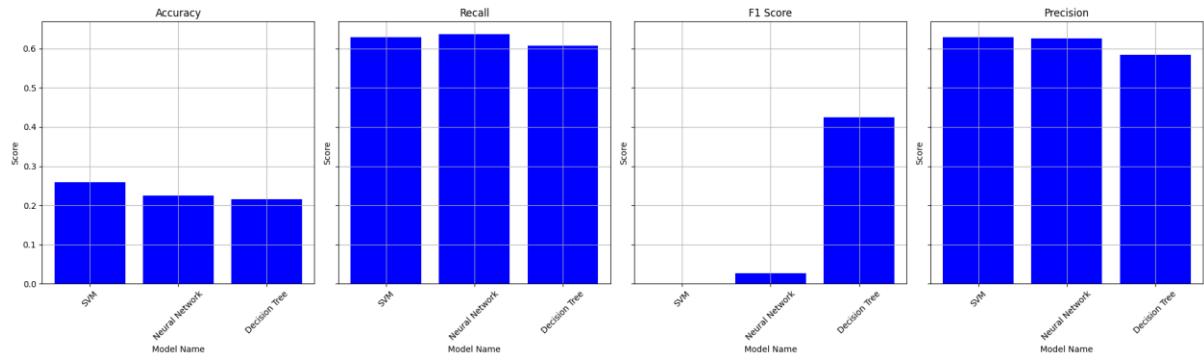
Nonostante avendo dropgate le features più significative tutti i modelli hanno saputo adattarsi riducendo le performance del **20%** circa.

- Sono stati introdotti il **30%** dei missing values su tutte le componenti della PCA, sono state flipgate il **10%** delle label dei vini rossi e il **15%** dei bianchi.



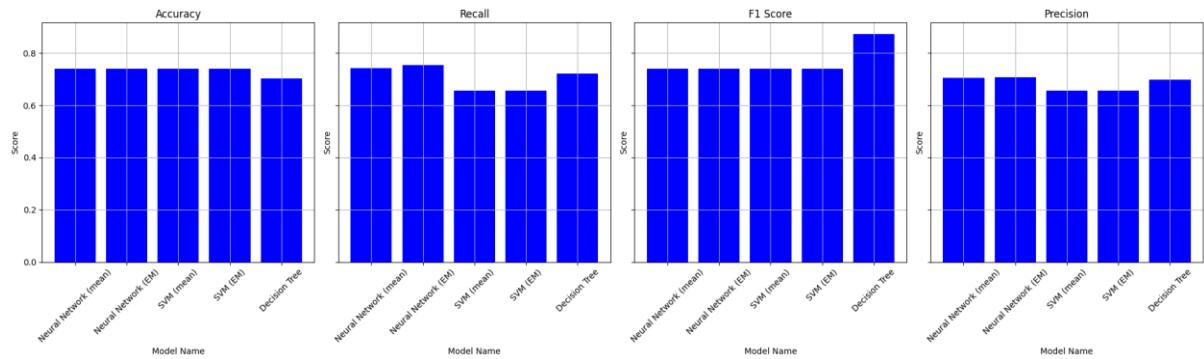
Tutti i modelli rispondono generalmente bene a questo test. Recall e Precision subiscono una leggera diminuzione.

- Viene dropgate PC1, vengono invertite tutte le label e vengono generate il 20% delle righe senza alcun vincolo



Le performance sono evidentemente peggiorate a causa del flip totale di tutte le istanze, l'accuracy è così bassa perché i modelli classificano

- Vengono **droppate PC2 e PC3**, sulle colonne restanti vengono aggiunti il **35%** di missing values, al **50%** delle istanze di **PC1, PC4 e PC5** sono stati aggiunti valori fuori dominio e sono state generate il **70%** delle righe con valori casuali controllati.



Le performance di ogni modello diminuiscono tutte del **20%** circa tranne che per F1 Score per il Decision Tree che ha subito una diminuzione minore.

Si noti che in alcuni esperimenti ci sono più modelli di altri. Ciò è dovuto al fatto SVM non supporta nativamente i missing values. Per ovviare questo problema abbiamo deciso di sostituire i missing values in due modi diversi: con media ed EM.

Conclusioni

Una volta terminata l'esecuzione e lo studio degli esperimenti possiamo andare a valutare effettivamente quanto siano state rispettate le dimensioni di qualità, garantendo ai modelli un dataset pulito e privo di ambiguità.

Consideriamo prima di tutto le [prestazioni](#) di partenza per i modelli:

- I modelli presentano metriche altissime, circa sul **0.98** per ogni metrica globale, indicando delle ottime prestazioni.
- Considerando l'approccio **naive** (ovvero senza tuning degli iperparametri), i modelli si comportano fin da subito perfettamente, questo indica una certa semplicità del dataset, che permette facilmente una discriminazione efficace del target nonostante sia sbilanciato.

Partendo dalle considerazioni sui modelli iniziali, andiamo ad analizzare complessivamente ciò che è stato ottenuto da ogni [esperimento](#), evidenziando gli esperimenti che hanno impattato di più (ciò è anche deducibile in modo chiaro dagli scatter plots mostrati nell'[analisi globale](#)):

- **Drop Features, Missing Values, Outliers, Out Of Domain Values e Duplicate Rows Same Label** portano una diminuzione delle metriche non trascurabile, ma non eccessiva, portandole circa a **0.75** di minimo, specialmente se toccate le prime due componenti PCA per quanto riguarda SVM. Il Decision Tree segue invece più un andamento a “cascata”, riscontrando effettivamente più variazioni in base ai parametri, ma fermandosi agli stessi valori.
- Quando gli esperimenti iniziano ad essere più complessi e impattanti abbiamo una diminuzione drastica delle prestazioni, arrivando all'impossibilità di apprendere dal dataset. Questo avviene maggiormente nell'esperimento **Flip Labels**, dove si va a portare ad un apprendimento completamente errato. L'aggiunta di righe duplicate e casuali, riportate dagli ultimi esperimenti **Duplicate Rows Opposite Label, Add Rows Random/Domain** portano invece delle grosse diminuzioni, non importanti come il flip del target ma con minimi che raggiungono poco più del **0.2**, inaccettabile considerando il punto di partenza delle metriche globali.
- Durante gli esperimenti le metriche che hanno mostrato un andamento più incerto sono state **Precision** e **Recall**, indotto dal fatto che i modelli andassero a classificare generalmente tutto come veri positivi, ma con un numero elevato di falsi positivi o negativi, in pratica classificando solo la classe predominante, o viceversa in base alla metrica (Precision alta = pochi falsi positivi, Recall alto = pochi falsi negativi).
- Dei modelli analizzati:

- Il **SVM** risulta quello più sensibile alla variazione della qualità del dataset, andando a convergere subito al minimo valore per ogni metrica se vengono impattate le componenti più importanti della PCA, però tende a rimanere invariato e costante, rimanendo sempre sul 0.75. Questo rapido cambiamento di prestazioni non è ideale.
- Il **Decision Tree** invece, come menzionato prima, segue un andamento più a “cascata”, dove ogni parametro scelto altera l'apprendimento. Il fatto di non convergere direttamente al minimo lo rende sicuramente più robusto degli altri modelli, ma più variabile.

Inoltre, nell'ambito dell'automatizzazione e delle pipeline, è cruciale riconoscere la potenziale variabilità nella distribuzione dei dati, nota come **drift** dei dati. Per gestire efficacemente questo fenomeno nel contesto reale, è essenziale impiegare sistemi di Continuous Integration e Continuous Delivery (CI/CD) integrati nelle pratiche di MLOps. È possibile, per esempio, configurare l'esecuzione automatica di una pipeline che, tramite sistemi di alerting, identifica tempestivamente un eventuale drift dei dati. Questo permette di riaddestrare i modelli e di prendere le misure necessarie per adeguarsi alle nuove condizioni dei dati, garantendo così la resilienza e l'efficacia del sistema nel tempo.

In conclusione quindi si può affermare ciò che è stato introdotto a lato teorico, ovvero le misure di **Data Quality** sono estremamente importanti e necessarie al fine di ottenere delle prestazioni accettabili dai modelli di apprendimento. Va infatti considerato come il dataset utilizzato in questo progetto fosse estremamente semplice per l'apprendimento e anche molto piccolo a livello di dimensioni (sui 5295 records), quindi gli stessi esperimenti effettuati su un dataset più complesso, di grandi dimensioni potrebbero portare risultati ancora più evidenti e disastrosi per l'apprendimento. Per quanto riguarda la parte di **Explainability** si è riusciti a dare un ottimo significato ai dati prodotti dalla PCA, nonché un approfondimento su quest'ultima, e di come ogni componente si relaziona all'impatto di apprendimento, specialmente la prima componente che contiene la maggior percentuale di varianza spiegata, confermando la sua importanza.

Appendice A

Principal Component Analysis

Quando applichiamo algoritmi di machine learning su un dataset, spesso ci troviamo in una situazione in cui il nostro dataset è composto da un numero elevato di features. Effettuare previsioni in questo tipo di situazioni può risultare spesso non computazionalmente sostenibile, sia per mancanza di tempo sia per mancanza di potenza di calcolo.

Principal Component Analysis (PCA) va a diminuire il numero di colonne del dataset senza compromettere una perdita di pattern significativa.

È ragionevole pensare che quando applichiamo la PCA ad un dataset, visto che stiamo riducendo il numero di informazioni, otterremo performance inferiori.

Quanto appena citato è vero ma una minima riduzione nelle performance è un buon trade-off per una riduzione del peso computazionale. L'idea principale che sta dietro alla PCA è semplice. Vengono costruite delle **nuove variabili sulla base di combinazioni lineari delle variabili originali**. Le nuove variabili ottenute sono **scorrelate tra di loro**. Nella prima nuova componente viene cercato di immettere il massimo numero possibile di informazioni; quello che non è stato possibile immettere nella prima componente verrà immesso nella seconda, quando anche la seconda componente risulterà saturata le rimanenti informazioni verranno messe nella terza colonna e così via...

L'intero processo è composto da 5 step:

1. Si effettua una **standardizzazione delle variabili** per far sì che le variabili che hanno un range di dominio più ampio di altre non prevarichino sulle altre, altrimenti andremmo incontro a bias.
2. Viene calcolata la **matrice delle covarianze**. Si vuole capire quanto ogni variabile si discosta dalla media per vedere se esistono correlazioni fra altre variabili. Se otteniamo covarianze positive significa che quando una variabile aumenterà anche l'altra lo farà, nel caso di valori negativi quando una variabile diventerà più grande l'altra diminuirà e viceversa.
3. Vengono calcolati **autovettori e autovalori** e ordinati in maniera decrescente.
4. Vengono scartati gli autovalori che forniscono meno informazioni e viene creato il **feature vector**, definito come matrice che ha come colonne gli autovettori delle componenti che decidiamo di tenere.
5. Per comporre i valori effettivi della PCA visto che finora sono state solo calcolate informazioni riguardo al dataset, si calcola il **prodotto**

$$V_{features}^T \cdot DStd^T$$

dove $V_{features}^T$ è il feature vector trasposto e $DStd^T$ è il dataset originale standardizzato trasposto.

Ciò che viene prodotto è la PCA. Si noti che ogni feature prodotta è priva di significato proprio ossia non ha interpretazione singola ma un'interpretazione globale.

Appendice B

SHapley Additive exPlanations (SHAP)

SHAP (SHapley Additive exPlanations) fornisce spiegazioni riguardanti il funzionamento di modelli di ML.

Le sue origini derivano dalla **teoria dei giochi**.

Viene usato nei task di classificazione/regressione, calcolando quanto ogni componente in input nel modello contribuisce nella generazione dell'output ed è inoltre possibile vedere come il modello si comporta rispetto specifici valori di determinate features.

Ha un unico svantaggio: richiede un tempo computazionale generalmente alto.

Sorge spontaneo domandarsi in quale momento dell'intero sviluppo del modello applicare SHAP. La risposta è alla fine di tutte le operazioni: generalmente come prima operazione viene importato il dataset e preprocessato; successivamente si effettuano training e test; segue poi un possibile tuning degli iperparametri ed infine si calcolano gli **SHAP values**.

La teoria sottostante il calcolo effettivo degli SHAP values va a misurare **quanto ogni componente di un team contribuisce nella realizzazione di un progetto**.

La formula per il calcolo del valore di un membro m è:

$$\varphi_m(v) = \frac{1}{p} \sum_S \frac{[v(S \cup \{m\}) - v(S)]}{\binom{p-1}{k(S)}} , \quad m = 1, 2, 3, \dots p$$

dove **v** è il valore, **p** è il numero di membri del team, **S** è un sottoinsieme del team **T**, **k(S)=|S|**.

Questa formula per essere valida deve rispettare le seguenti proprietà

- **Simmetria**: se due membri contribuiscono in modo uguale, allora i loro SHAP values devono essere uguali.
- **Linearità**: se consideriamo due progetti distinti, la somma dei valori SHAP per ciascun membro nei due progetti combinati deve essere uguale alla somma dei valori SHAP nei singoli progetti.
- **Efficienza**: la somma di tutti i valori SHAP deve essere uguale al valore totale del progetto.

Appendice C

Possibili Metodi di Imputazione

Nei dataset reali, tra i diversi possibili errori possiamo osservare cinque tipi di errore prevalenti: valori mancanti, outliers, duplicati, inconsistenze ed etichette errate.

In questa appendice ci concentriamo sui valori mancanti e sui possibili metodi di imputazione.

I valori mancanti occorrono quando non viene memorizzato alcun valore per le celle.

Essi possono essere rilevati in maniera naturale cercando entrate vuote o NaN (un placeholder comunemente utilizzato).

Abbiamo i seguenti metodi per risolvere i valori mancanti:

- **eliminazione**
 - si eliminano i record con valori mancanti (i.e. si droppano le righe)
- **imputazione semplice**
 - abbiamo 5 diversi modi
 - per i valori numerici
 - media
 - mediana
 - moda
 - per i valori categorici
 - moda (classe più frequente)
 - variabile fittizia “missing”
- altre tecniche, tra cui l'**imputazione iterativa** (o **imputazione multivariata**)
 - processo in cui ciascuna feature viene modellata come una funzione delle altre feature, ad esempio un problema di regressione in cui i valori mancanti vengono predetti
 - ciascuna feature viene imputata sequenzialmente, una dopo l'altra, consentendo di utilizzare i valori imputati in precedenza
 - è iterativa perché il processo si ripete diverse volte, permettendo di calcolare stime sempre migliori dei valori mancanti man mano che i valori mancanti di tutte le features vengono stimati
 - un esempio di metodo iterativo è l'algoritmo **Expectation Maximization (EM)**

Appendice D

Expectation Maximization

L'algoritmo Expectation-Maximization (EM) è un **metodo iterativo** utilizzato per trovare stime di massima verosimiglianza (MLE) o di massima a posteriori (MAP) di parametri in modelli statistici, dove il modello dipende da **variabili latenti non osservate** (variabili nascoste).

Anche se queste variabili non possono essere osservate direttamente, possiamo stimare la loro distribuzione condizionata data le osservazioni attuali e i parametri stimati finora.

L'algoritmo alterna tra due step principali: lo step di **aspettativa (E-step)** e lo step di **massimizzazione (M-step)**.

Nello step di **aspettativa**, si calcola il valore atteso della verosimiglianza logaritmica completa, considerando le variabili latenti, date le osservazioni attuali e le stime dei parametri. Matematicamente, si definisce la funzione Q come:

$$Q(\theta \mid \theta^{(t)}) = \mathbb{E}_{\mathbf{Z} \sim p(\cdot \mid \mathbf{X}, \theta^{(t)})} [\log p(\mathbf{X}, \mathbf{Z} \mid \theta)]$$

dove \mathbf{X} sono i dati osservati, \mathbf{Z} le variabili latenti e $\theta^{(t)}$ i parametri stimati all'iterazione t .

Nello step di **massimizzazione**, si aggiornano le stime dei parametri massimizzando la funzione Q :

$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta \mid \theta^{(t)})$$

Questo passaggio trova i parametri che massimizzano il valore atteso calcolato nella E-step, migliorando così la stima corrente dei parametri.

Svantaggi dell'algoritmo

- può convergere a **soluzioni subottimali**.
- la qualità delle stime dipende dalle **condizioni iniziali**
- può richiedere **molte iterazioni per convergere**.