Alma Mater Studiorum - Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Cosmo Catalog Analysis

Canghiari Matteo
Matricola 1032059
matteo.canghiari@studio.unibo.it

Statistica Numerica
Anno Accademico 2023/2024

Introduzione

Cosmo-Catalog-Analysis è un notebook python realizzato per eseguire un'analisi statistica relativa ad un dataset inerente a fenomeni celesti. L'obiettivo del progetto consiste nello sviluppo di un'indagine statistica in grado di mettere in luce gli aspetti salienti dell'insieme dei dati. I principali strumenti utilizzati per il caso di studio preso in esame saranno approfonditi nelle sezioni successive, tuttavia è possibile anticipare il contenuto del progetto proposto.

L'indagine è composta da alcuni passaggi chiave di un'analisi statistica, suddivisi in:

- **EDA**, acronimo di Exploratory Data Analysis, il quale costituisce il primo step da affrontare; ottenuto e scelto un dataset sono analizzati i dati contenuti in esso, in maniera tale da evidenziare le informazioni intrinseche e per pianificare lo studio da svolgere
- **Splitting**, fase in cui il dataset è suddiviso in due insiemi, rispettivamente in un train set, utilizzato per l'addestramento del modello di classificazione, e in un test set, impiegato per stabilire la bontà della predizione del modello
- **Regressione lineare**, in cui è stato condotto un problema supervisionato legato alla predizione di una variabile di output y di tipo numerico; lo studio è stato realizzato tra coppie di variabili fortemente correlate, sia positivamente che negativamente
- **Addestramento del modello**, passaggio essenziale affinché i modelli di classificazione, pertanto legati alla predizione di una variabile di output y di tipo categorico, possano stabilire i parametri che minimizzano la funzione di loss
- **Valutazione delle performance**, addestrati i modelli e definiti i parametri associati, lo step successivo prevede lo studio delle performance associate a ciascun algoritmo. La valutazione della bontà del modello avviene attraverso lo studio dei risultati successivi alla fase di predizione
- **Studio statistico dei risultati**, fase in cui sono impiegate tecniche sia di statistica descrittiva che inferenziale; in particolare per un K fissato, sono state ripetute le fasi di addestramento e di testing per ciascun modello di classificazione, in maniera tale da ottenere uno storico delle metriche valutative associate

Concludendo, il dataset scelto rivela alcuni aspetti caratteristici dei fenomeni celesti, composto da tutti i domini chiave per l'esplorazione del cosmo. Di seguito, sono riportate brevi descrizioni dei domini trattati, in cui si evidenzia:

1.	**Temperature**	Informazioni termiche della stella
2.	**Luminosity**	Brillantezza/luminosità della stella
3.	**Radius**	Dimensione spaziale della stella
4.	**Magnitude**	Luce intrinseca della stella
5.	**Star type**	Tipologia della stella
6.	**Star color**	Colore visibile della stella
7.	**Spectral Class**	Classe spettrale della stella

EDA

Come riportato dall'introduzione, la prima fase, per un qualsiasi progetto legato ad una indagine statistica, prevede l'analisi dei dati contenuti all'interno del dataset. Un dataset è una tabella di *osservazioni*, in cui le colonne rappresentano le *features*, mentre le righe rappresentano le differenti entità; dal punto di vista matematico un dataset è una matrice di dimensione $N \times d$.

Il dataset scelto ha una dimensione pari a 240×7, rispettivamente il numero di righe e di colonne. Le features si contraddistinguono in domini *numerici* e *categorici*, come riportato nella tabella seguente.

1.	**Temperature**	int64
2.	**Luminosity**	float64
3.	**Radius**	float64
4.	**Magnitude**	float64
5.	**Star type**	int64
6.	**Star color**	object
7.	**Spectral Class**	object

Nonostante la colonna **Star type** sia di tipo numerico non è plausibile un suo impiego all'interno di indagini statistiche, poichè utilizzata principalmente per categorizzare le differenti entità della matrice.

Il dataset non possiede alcun *dato mancante*, generalmente definito come *NaN*, dato che la sua dimensione non varia successivamente all'eliminazione di osservazioni mancanti. Inoltre, è stato eseguito un rapido controllo relativo alle variabili categoriche; spesso il loro utilizzo avviene per distinguere le moltiplici entità secondo caratteristiche comuni, introducendo delle vere e proprie classi.

```
classes = df["Spectral Class"]
_dict = df["Spectral Class"].value_counts().to_dict()

print(_dict)

```

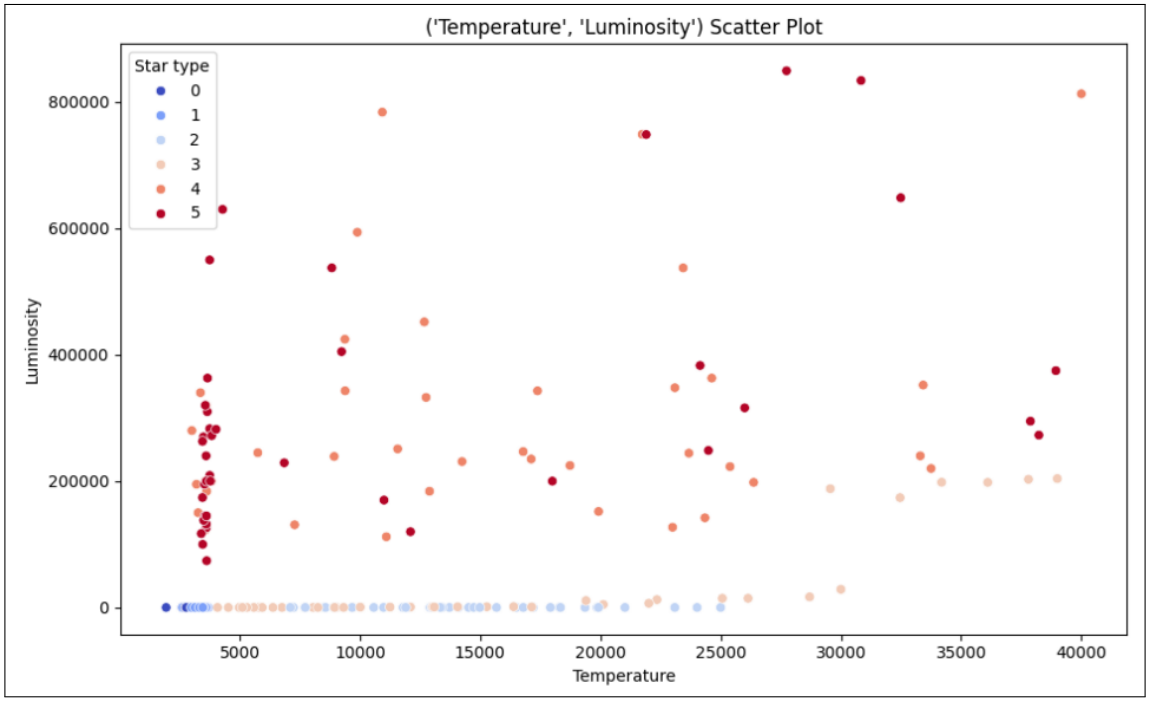
```
Out: {'M': 111, 'B': 46, 'O': 40, 'A': 19, 'F': 17, 'K': 6, 'G': 1}
```

Come da snippet di codice precedente, il risultato ottenuto esprime una certa proponderanza di determinate *classi*. A causa di tale sbilanciamento, una caratteristica simile potrebbe causare differenti problematiche associate alla predizione di modelli di classificazione, in questa circostanza è necessario accertarsi che lo splitting del dataset racchiudi ogni possibile associazione tra le differenti features, in modo tale che non si abbiano ripercussioni inerenti alla bontà dei modelli di classificazione.

Una volta completata l'analisi delle variabili categoriche, il controllo consecutivo è ricaduto sulle variabili numeriche, approfondendo in questo modo la comprensione dei dati. Sono state condotte tutte le tipologie di *indagini*, *univariate*, *bivariate* piuttosto che *multivariate*, ognuna delle quali caratterizzata dai propri strumenti grafici. I principali strumenti grafici impiegati per evidenziare le proprietà statistiche dei vari domini possono essere suddivisi in:

- **Boxplot**, strumento grafico usato per visualizzare la distribuzione di una variabile numerica, osservare i quartili ed identificare eventuali valori anomali, detti outlier
- **Istogramma**, strumento grafico per esaminare la distribuzione dei dati e le frequenze corrispondenti, raggrupate in intervalli definiti
- **Grafico a dispersione**, strumento visivo per osservare la relazione tra due variabili numeriche, in cui ogni punto rappresenta un'osservazione del dataset, avente coordinate pari ai valori delle due variabili
- **Matrice di correlazione**, matrice quadrata di dimensione pari al numero di variabili in input impiegate, ogni elemento che compone la matrice corrisponde al grado di correlazione tra due variabili, propriamente denominato *coefficente di correlazione*

A tal proposito, è interessante osservare la relazione catturata dal *grafico a dispersione* tra le features *Temperature* e *Luminosity*.



Il grafico a dispersione riportato rappresenta un *diagramma di Hertzsprung-Russell*, in quanto mostra la distribuzione dei corpi celesti in base alla loro temperatura e luminosità. Sebbene queste due grandezze fisiche dipendano strettamente da caratteristiche intrinseche della stella, come età, composizione chimica e massa, è evidente una suddivisione stratificata: nella parte inferiore si trovano le stelle più fredde e meno luminose, al centro quelle di temperatura e luminosità intermedie, mentre in alto sono collocate le stelle più calde e brillanti.

Ultimo passaggio, seppur fondamentale, riguarda l'analisi di valori anomali, detti *outliers*. Un outlier rappresenta un osservazione che si discosta significativamente rispetto ai valori contenuti all'interno della feature presa in considerazione. Una delle tecniche principali per valutare se un dato è un valore anomalo avviene tramite la definizione di un *range interquartile*, ottenuto tramite la differenza tra il terzo e il primo *quartile*. In semplici termini, il range esprime una certa variabilità in cui i dati possono essere considerati validi; se oltrepassato

il limite, inferiore oppure superiore, l'osservazione potrebbere rappresentare potenzialmente un valore anomolo.

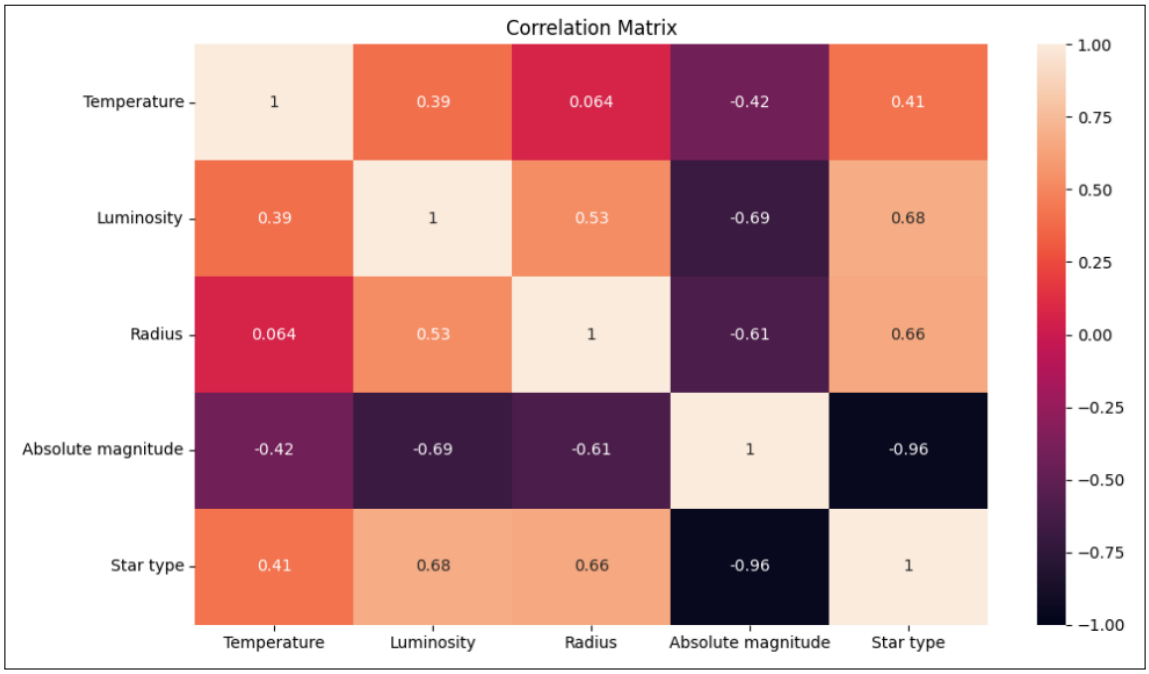
```python
def define_outliers(df: pd.DataFrame) -> tuple[Dict[str, int], pd.DataFrame]:
    _df = df.select_dtypes(include="number").copy()

    q1 = _df.quantile(0.25)
    q3 = _df.quantile(0.75)

    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    dict_outliers = {column.strip(): 0 for column in _df.columns}

    for column in _df.columns[1:]:
        for i in range(len(_df[column])):
            if _df.iloc[i][column] > upper_bound[column]:
                _df.iloc[i][column] = upper_bound[column]
                dict_outliers[column] += 1

            if _df.iloc[i][column] < lower_bound[column]:
                _df.iloc[i][column] = lower_bound[column]
                dict_outliers[column] += 1

    return (dict_outliers, _df)

```

```
Out: {'Temperature': 0, 'Luminosity': 12, 'Radius': 40, 'Absolute magnitude': 0, 'Star
    ↪ type': 0}
```

Regressione lineare

I modelli di *regressione lineare* sono implementati qualora il tipo della variabile di output è numerico. Formalmente, in statistica la regressione lineare permette di costruire un modello attraverso cui predire i valori di una variabile dipendente, Y, dati i valori di una o più variabili indipendenti, X.

A partire dai risultati ottenuti dalla *matrice di correlazione*, sono state scelti alcuni domini caratterizzati da un *coefficente di correlazione* r positivo, ossia maggiore di 0. Pertanto, il coefficente esprime il grado di dipendenza tra due features.



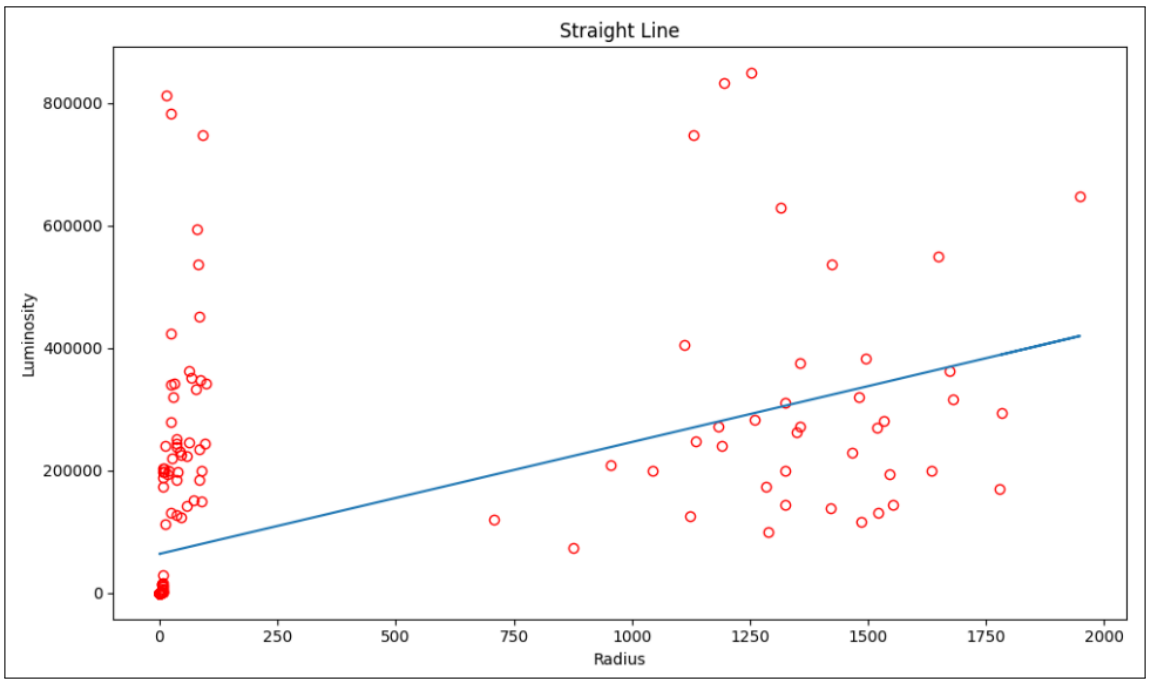
L'immagine propone la matrice di correlazione dei domini del dataset di carattere numerico; la colonna *Star type*, nonostante esprima una correlazione positiva nei confronti della maggior parte delle features che contraddistingue l'insieme di dati, non è stata impiegata per realizzare indagini statistiche, poichè è utilizzata principalmente per categorizzare le molteplici entità. A tal proposito, qualsiasi associazione rispetto al dominio *Absolute magnitude* presenta una dipendenza negativa. Questa caratteristica evidenzia un andamento inversamente proporzionale del dominio in esame rispetto agli altri. In particolare, all'aumentare dei valori della variabile *Absolute magnitude*, i dati delle altre colonne tenderanno a diminuire.

```
X_label = "Radius"
y_label = "Luminosity"

X = _df[[X_label]]
y = _df[y_label]

univariate_linear_regression(X_label, y_label, X, y)

```

```
Out:
```

```
In ordine:
 - m: [182.67977484]
 - q: 63864.43153126554
 - R^2: 0.27721880798570075
```

Lo snippet invoca il metodo indicato per eseguire una regressione lineare univariata, dato che la variabile di input coincide con un solo carattere del dataset. La funzione provvede, in completa autonomia, al calcolo computazionale degli *stimatori* dell'*equazione lineare*, quali *coefficente angolare* ed *intercetta*, mostrando i grafici relativi.

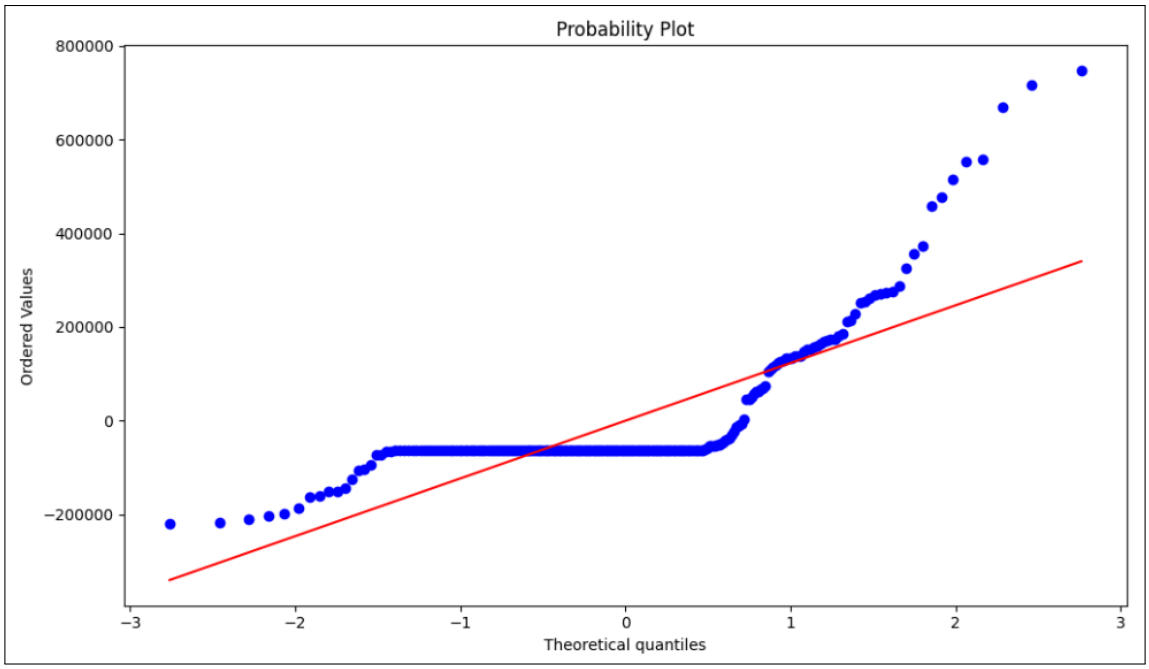


A livello visivo, i punti dovrebbero disporsi lungo la retta di equazione

$$y = 182.68x + 63864.432$$

secondo una certa omogeneità, affinché sia possibile affermare l'accuratezza dell'algoritmo di regressione lineare. Tuttavia, come riportato dall'immagine soprastante, le coppie (x, y) tendono ad uniformarsi in due insiemi suddivisi, testimoniando uno scarso rendimento da parte del modello. Il quadro delineato evidenzia le difficoltà nell'utilizzo di una funzione lineare capace di generare predizioni valide per la variabile di output designata.

La retta della regressione lineare rappresenta una semplificazione della realtà. Ciò implica che alcune caratteristiche del modello non possano emergere dalla raffigurazione geometrica; proprio per questa principale ragione è necessario implementare un'*analisi dei residui*. I residui costituiscono l'*errore di previsione* del modello di regressione, ottenuti mediante la differenza dei valori esatti rispetto ai valori predetti. La bontà del modello si muove di pari passo alla distribuzione dei residui. Uno score elevato della retta testimonia una distribuzione in cui circa la media è nulla, $\mu = 0$, e la varianza, σ^2, è costante, in modo tale che ciascuna predizione abbia sempre lo stesso grado di variabilità.



Il grafico definisce un *QQ-Plot*, ossia una raffigurazione dei quantili di una distribuzione. Il suo scopo principale consiste nel confrontare la distribuzione di una variabile osservata, in questa circostanza i *residui* del modello di regressione lineare, con la distribuzione della *normale*. Se le due distribuzioni presentano un andamento simile, i punti (x, y) dovrebbero allinearsi lungo alla retta. Pertanto, se i punti si discostano significativamente dalla retta, ciò indica che i residui non seguono una distribuzione normale. Quest'ultimo inciso valorizza la scarsa bontà del modello di regressione lineare implementato, data la notevole discrepanza, suggerendo la necessità di modelli alternativi più adatti alle osservazioni del dataset.

Addestramento del modello - Valutazione delle performance

Gli algoritmi di classificazione, come la *regressione logistica* e le *Support Vector Machines*, sono implementati qualora la variabile di output è di tipo categorico. In questo caso, la variabile di output potrà assumere soltanto un numero finito di valori, detti *classi*.

All'interno del progetto è stata attuata una conversione numerica di tali features, affinché i dati contenuti al loro interno fossero manipolabili dai modelli di classificazione.

```
encoder = preprocessing.LabelEncoder()
df_category = df.select_dtypes(include=["object", "category"])

print("In ordine:", end="\n")
for column in df_category.columns:
    _set = union_categorical_field(column, df)
    values = encoder.fit_transform([item for item in _set])
    _dict = {key: int(value) for key, value in zip(_set, values)}

    print(f"- Dominio \033[1m{column}\033[0m del dataset convertito in
          {_dict}", end="\n")

    df[column] = df_category[column].map(_dict)

```

```
Out:
In ordine:
- Dominio Star color del dataset convertito in {'White': 3, 'Red': 2, 'Blue-White': 1,
    ↪ 'Yellow-White': 4, 'Blue': 0}
- Dominio Spectral Class del dataset convertito in {'F': 2, 'A': 0, 'G': 3, 'M': 5, 'K': 4,
    ↪ 'O': 6, 'B': 1}
```

Successivamente allo snippet di codice dedito alla conversione numerica di colonne categoriche, è stata sviluppata la logica necessaria per usufruire dei modelli di classificazione.

La regressione logistica è un *classificatore lineare*, utilizzato qualora la variabile dipendente sia di natura categorica. L'obiettivo del modello è di stabilire la probabilità che un'osservazione, appartenente al dataset, possa generare una *classe* della variabile dipendente.

```
def logistic_regression(X_train, X_test, y_train, y_test):
    try:
        X_train = preprocessing.MinMaxScaler().fit_transform(X_train)
        X_test = preprocessing.MinMaxScaler().fit_transform(X_test)

        model = LogisticRegression(max_iter=100, multi_class="multinomial")
                .fit(X_train, y_train)

        y_pred = model.predict(X_test)
        accuracy = round(metrics.accuracy_score(y_test, y_pred)*100, 2)

        print(f"Accuratezza del modello di regressione logistica: {accuracy}")
    except Exception:
        print("I dataset inviati non rispettano i vincoli di dimensione")

```

A livello implementativo è possibile evidenziare due aspetti sostanziali, quali:

- **MinMaxScaler**, classe della libreria *preprocessing* di *sklearn*, utilizzata per normalizzare le variabili di input del modello di classificazione. La normalizzazione dei dati avviene affinché le features del dataset caratterizzate da scale differenti non influenzino il modello durante la fase di addestramento, evitando che l'algoritmo sia fondato su predeterminati bias
- **Parametri** della regressione logistica, in cui sono stati definiti il *numero di iterazioni* e la *tipologia* del modello. Brevemente, *max_iter* indica il numero massimo di iterazioni che il modello può eseguire per garantire la convergenza della *funzione sigmoide* rispetto alla variabile dipendente, mentre *multi_class* determina la modalità predittiva dell'algoritmo, tenendo conto della numerosità di classi associabili alla variabile di output

Conclusa la computazione del metodo, viene mostrata l'*accuratezza* del modello di regressione logistica confrontando i valori predetti rispetti ai dati esatti.

```
Out:
Accuratezza del modello di regressione logistica: 90.282
```

Tipicamente, le Support Vector Machines sono implementate qualora il dataset non sia *linearmente separabile*, ossia non esiste alcuna retta o iperpiano separatore in grado di suddividere nettamente le osservazioni della matrice. L'idea alla base dell'algoritmo SVM consiste nella semplificazione del dataset, mediante una certa funzione detta *kernel function*, affinché l'insieme di dati sia successivamente classificabile secondo *modelli SVC*, acronimo di Support Vector Classifier.

La bontà del modello dipende da due fattori principali, suddivisi in:

- **Kernel function**, semplifica e rende il dataset linearmente separabile. In tale ambito, non sono più impiegate rette oppure iperpiani separatori, ma sono introdotte *curve separatrici*, la cui forma è dettata dalla scelta della funzione nucleo
- **Iperparametri**, rappresentano i parametri associati al classificatore, come il *costo* per funzioni lineari, il *grado* dell'esponente per funzioni polinomiali, piuttosto che la *varianza* per funzioni esponenziali

```
def support_vector_machines(X_train, X_test, y_train, y_test):
    try:
        plt.figure(figsize=(20, 6))

        X_train = preprocessing.MinMaxScaler().fit_transform(X_train)
        X_test = preprocessing.MinMaxScaler().fit_transform(X_test)

        linear_model = svm.SVC(kernel="linear", C=1).fit(X_train, y_train)
        cf_matrix_linear = evaluate_kernel("Lineare", X_test, y_test, linear_model)

        plt.subplot(1, 3, 1)
        plt.title("Linear Confusion Matrix")
        plt.xlabel("True Values")
        plt.ylabel("Predicted Values")
        sns.heatmap(cf_matrix_linear, annot=True, cmap="Reds")

        poly_model = svm.SVC(kernel="poly", degree=3).fit(X_train, y_train)
        cf_matrix_poly = evaluate_kernel("Polinomiale", X_test, y_test, poly_model)
```

```
19
20          # Poly Confusion Matrix ...
21
22          rbf_model = svm.SVC(kernel="rbf").fit(X_train, y_train)
23          cf_matrix_rbf = evaluate_kernel("Esponenziale", X_test, y_test, rbf_model)
24
25          # Rbf Confusione Matrix ...
26
27          plt.tight_layout()
28          plt.show()
29      except Exception as e:
30          print("I dataset inviati non rispettano i vincoli di dimensione")
31
```

Al termine della computazione, sono mostrati il *misclassification ratio*, l'*accuratezza* e la *matrice di confusione* per ciascun modello. Il misclassification ratio è una metrica che esprime la percentuale totale dei punti mal classificati dal predittore; da cui, mediante la formula

$$Acc = 1 - MR(S),$$

è calcolata l'accuratezza, la quale manifesta la bontà del modello predittivo, nonostante non sia effettivamente una metrica.

```
1  def evaluate_kernel(type_model, X_test, y_true, model) -> metrics:
2
3      y_pred = model.predict(X_test)
4
5      MR = np.mean(y_pred != y_true)
6      accuracy = 1 - MR
7
8      print(f"{type_model}: \n - Misclassification ratio: {round(MR*100, 2)} \n
9      - Accuratezza del modello: {round(accuracy*100, 2)} \n")
10
11     return metrics.confusion_matrix(y_true, y_pred)
12
```

```
Lineare:
 - Misclassification ratio: 23.61
 - Accuratezza del modello: 76.39

Polinomiale:
 - Misclassification ratio: 5.56
 - Accuratezza del modello: 94.44

Esponenziale:
 - Misclassification ratio: 5.56
 - Accuratezza del modello: 94.44
```

Studio statistico dei risultati

Una sola esecuzione dei modelli, ottenuti tramite la Support Vector Machines, non è sufficiente per valutare la bontà degli algoritmi di classificazione, data l'aleatorietà dei dati utilizzati. Per tale ragione, sono state ripetute le fasi di addestramento e di testing per un numero intero K fissato.

L'obiettivo prevede di osservare l'andamento di un modello di classificazione introducendo la *cross-validation*; la convalida incrociata, cosiddetta *k-fold*, consiste nella suddivisione del train set in n parti di equivalente numerosità e, a ogni passo, si esclude iterativamente una sezione alla volta tentando di predirla con i gruppi di dati restanti, al fine di verificare la bontà del modello predittivo, evitando l'*overfit*. Inoltre, ad ogni iterazione è introdotto un certo grado di casualità durante la fase di *splitting* del dataset, in maniera tale che l'addestramento del modello avvenga sempre per suddivisioni differenti dell'insieme di dati.

```python
def evaluate_model(K: int, df: pd.DataFrame, _svm: svm.SVC) -> List[int]:
    print("Accuratezza del modello in ordine:", end="\n")

    accuracies = []
    for i in range(K):
        tuple_dataframes = split_dataframe(randomness=i,
                                           y_field="Spectral Class", df=df)

        X_train = tuple_dataframes[0]
        X_test = tuple_dataframes[1]

        y_train = tuple_dataframes[2]
        y_test = tuple_dataframes[3]

        model = _svm.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        scores = cross_val_score(_svm, X_train, y_train, scoring="accuracy",
                        cv=KFold(n_splits=i+1, shuffle=True, random_state=20+2*i))
        mean_accuracy = round(scores.mean()*100, 2)

        recall = metrics.recall_score(y_test, y_pred, average="weighted",
                                      zero_division=0)
        mean_recall = round(np.mean(recall)*100, 2)

        precision = metrics.precision_score(y_test, y_pred, average="weighted",
                                            zero_division=0)
        mean_precision = round(np.mean(precision)*100, 2)

        print(f"- Step {i+1}: \n\t - Precisione: {mean_precision} \n\t
        - Sensitivita': {mean_recall} \n\t- Accuratezza: {mean_accuracy}", end="\n")

        accuracies.append(mean_accuracy)

    return accuracies

```

```
Accuratezza del modello in ordine:
- Step 1:
    - Precisione: 84.3
    - Sensitività: 84.72
```

```
    - Accuratezza: 82.73
- Step 2:
    - ...
    - ...
    - ...
- ...
- Step 10:
    - Precisione: 86.08
    - Sensitività: 86.11
    - Accuratezza: 87.56

L'accuratezza media del modello linear risulta: 85.79
```

Il risultato evidenzia alcune metriche ottenute dal modello di classificazione secondo la *kernel function* di tipo *lineare*. Ad ogni iterazione del ciclo è riportata la *precisione*, la *sensitività* e l'*accuratezza* dell'algoritmo rispetto alla variabile di output y predetta. Al termine della funzione, è restituita la lista delle accuratezze estrapolate per ciascuna computazione.

Dalla lista è poi calcolata la *media* delle accuratezze, affinché sia definito un *intervallo di confidenza*. Similmente ai *quartili*, gli intervalli di confidenza esprimono un range di variabilità in cui ci si aspetta di inviduare un certo parametro; è necessario affermare che tale tecnica non indica la probabilità, in questa casistica, che la media delle accuretezze sia all'interno di tale intervallo, ma, semplicemente, stabilisce un limite inferiore e superiore in cui dovrebbe collocarsi la metrica presa in esame.

```python
def define_confidence_interval(_list):
    try:
        if check_types_list(float, _list):
            n = len(_list)

            std_accuracy = np.std(_list)
            mean_accuracy = np.mean(_list)

            confidence_interval = stats.t.interval(0.95, n-1,
                                    loc=np.mean(_list), scale=stats.sem(_list))

            print(f"L'intervallo di confidenza con livello di confidenza alpha=0.05
            risulta: [{confidence_interval[0]}, {confidence_interval[1]}]")
        else:
            raise Exception
    except Exception:
        print("La lista in input deve contenere solamente valori numerici")

```

```
L'intervallo di confidenza con livello di confidenza alpha=0.05 risulta:
    ↪ [84.40196998984591, 87.1780300101541]
```