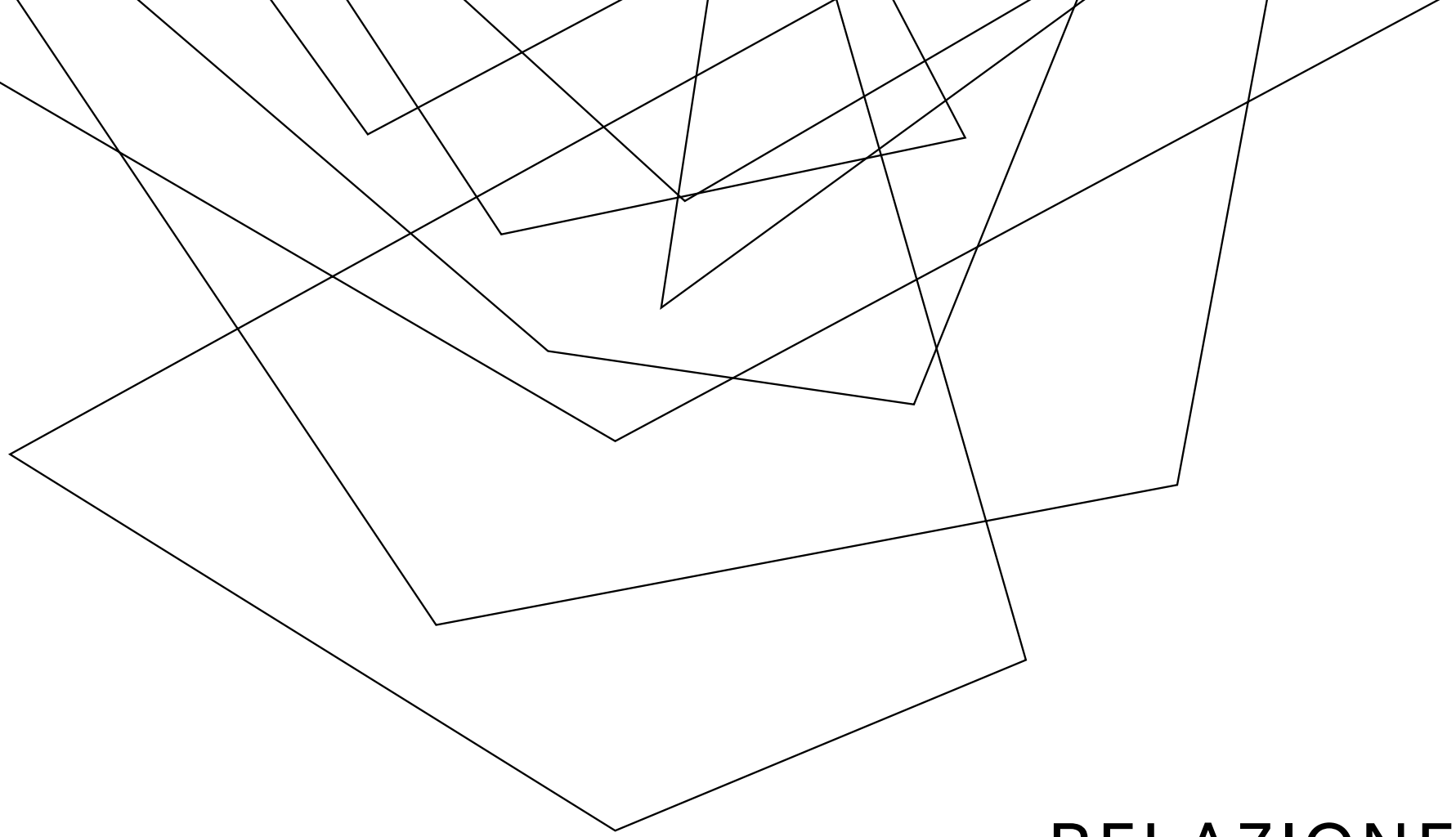


RELAZIONE D’ESAME

MACHINE LEARNING

Cecci Riccardo – Matr: 20023915

REGRESSIONE

Algoritmo: Regressione lineare con discesa del gradiente

Dataset: Automobili

INFORMAZIONI GENERALI

- Obiettivo: predizione di un valore numerico
- Il modello di apprendimento è **supervised**
- Tipi di regressione:
  - **Univariata:** l'approssimazione sarà definita da una retta

$$\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 x}$$

  - **Multivariata:** l'approssimazione sarà definita da un iperpiano

$$\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_n x_n}$$

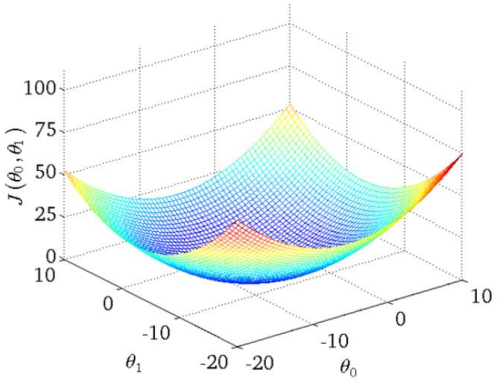
$$\boldsymbol{h_\theta(x) = \theta^T x}$$

FUNZIONE DI COSTO

- Usata per valutare la bontà del modello
- Obiettivo: minimizzare la funzione di costo $J(\theta)$:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

- $J(\theta)$ è una parabola **convessa** → ha un solo minimo (assoluto)



DISCESA DEL GRADIENTE

- Si vuole minimizzare la funzione di costo → bisogna sapere se la funzione sta crescendo o decrescendo in modo da dirigersi verso il minimo
- Idea: usare la derivata (parziale)
  - **Derivata > 0:** funzione crescente
  - **Derivata < 0:** funzione decrescente
- Derivo rispetto ai θ_n:

$$J'(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)\, x_j^i$$

DISCESA DEL GRADIENTE (CONT'D)

- Si usa la derivata della funzione di costo per aggiornare i θ in modo iterativo

Repeat{

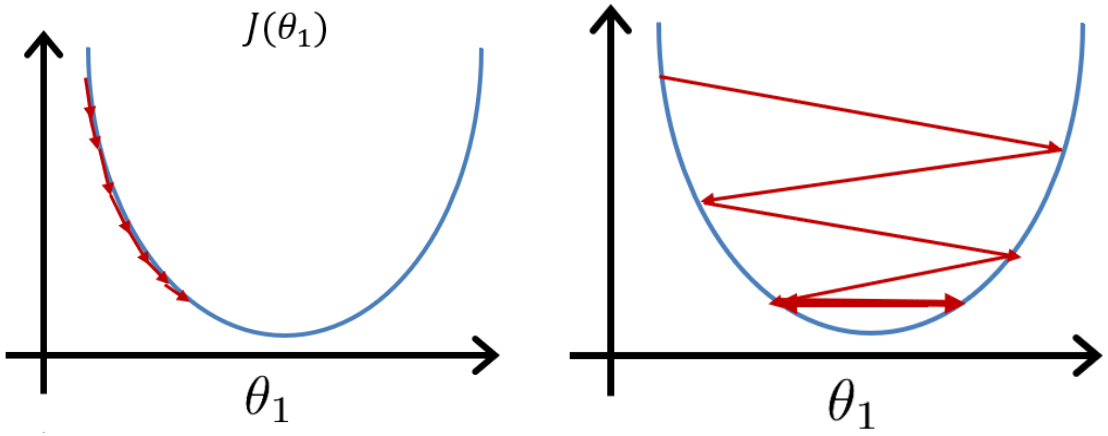
$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

}

- α è la **learning rate**

LEARNING RATE α

- Controlla la velocità di apprendimento
- Deve essere ben bilanciata:
  - α troppo grande → $J(\theta)$ potrebbe non decrementare e, in caso estremo, potrebbe anche divergere
  - α troppo piccolo → convergenza lenta



IMPLEMENTAZIONE DELLA DISCESA DEL GRADIENTE

```
def gradientDescent(alpha, tolerance, theta, x, y, m):

    previousCost = 1000000000000000
    while True:
        h = np.dot(x, theta) # ipotesi, ovvero i valori predetti
        #Discesa del gradiente
        theta = theta - ((alpha * (1/m)) * np.dot((h - y).T, x).T)
        j_cost = (np.sum((h - y)**2))/(2*m)
        diff = previousCost - j_cost
        if diff < tolerance:
            # se il cambiamento nei theta è stato minimale l'algoritmo si ferma
            break
        else:
            # aggiornamento costo
            previousCost = j_cost

    return theta
```

PRE-PROCESSING DEI DATI (WEKA)

- Operazioni svolte:
  - Eliminazione di istanze con target nullo
  - Rimpiazzamento dei **missing values**
  - Trasformazionedi tutte le features in **features numeriche**
  - **Standardizzazione** in modo che tutti i dati abbiano
    - **Media 0**
    - **Varianza 1**

$$x_n = \frac{x_n - \mu_n}{s_n}$$

PRESTAZIONI DEL MODELLO

- Per testare il modello è importante non utilizzare gli stessi dati su cui il modello è stato allenato
- Il dataset viene diviso in:
  - Training set: usato per l'apprendimento
  - Test set: usato per la valutazione
- Sono stati usati due metodi per la suddivisione del dataset:
  - Split 70/30
  - Cross Validazione

CALCOLO DELLE PREDIZIONI

- Il calcolo delle predizioni è una combinazione lineare fra le features e il vettore dei θ

```
def predictions(x_test, theta):
    pred = np.dot(x_test, theta)
    return pred
```

VALUTAZIONE DEL MODELLO

METRICHE

```python
def modelEvaluation(prediction, target):
    meanTarget = np.mean(target)

    #R2
    sse = np.sum((prediction-target)**2)
    sst = np.sum((target - meanTarget)**2)
    r2 = 1- (sse/sst)

    #Mean absolute error
    mae = np.mean(np.abs(prediction - target))

    #RMSE
    rmse = np.sqrt((np.sum((prediction - target)**2)) / len(prediction))

    return (r2, mae, rmse)
```

- R^2: Indica quanto della variabilità dei dati è spiegato dalle variabili nel modello

$$R^2 = 1 - \frac{ss_e}{ss_t}$$

$$ss_e = \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

$$ss_t = \sum_{i=1}^{m} (y^i - \bar{y})^2$$

- Mean Absolute Error

$$MAE = \frac{1}{m} \sum_{i=1}^{m} |h_\theta(x^i) - y^i|$$

- Root Mean Squared Error

$$\sqrt{\frac{\sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2}{m}}$$

TRAINING
SPLIT 70/30

- Si è suddiviso il dataset in:
  - **Training set:** 70% dei dati, usato in fase di apprendimento
  - **Test set:** 30% dei dati, usato in fase di test

```
training_set = dataset.head(round(len(dataset)*(70/100)))
test_set = dataset.tail(len(dataset) - len(training_set)).reset_index(drop=True)
```

```
x = training_set.drop('price', axis = 1).values
y = training_set['price'].values
m = len(training_set)

#inizializzazione dei theta
# genero un array di theta casuali inizializzati in base ai valori massimi e minimi dei dati
# target (y) e in base al numero di features nella matrice delle features (x)
theta = np.random.uniform(-(np.max(y)), np.max(y), size=x.shape[1])

finalTheta = gradientDescent(alpha=alpha, tolerance=tolerance, theta=theta, x=x, y=y, m=m)
```

TEST
SPLIT 70/30

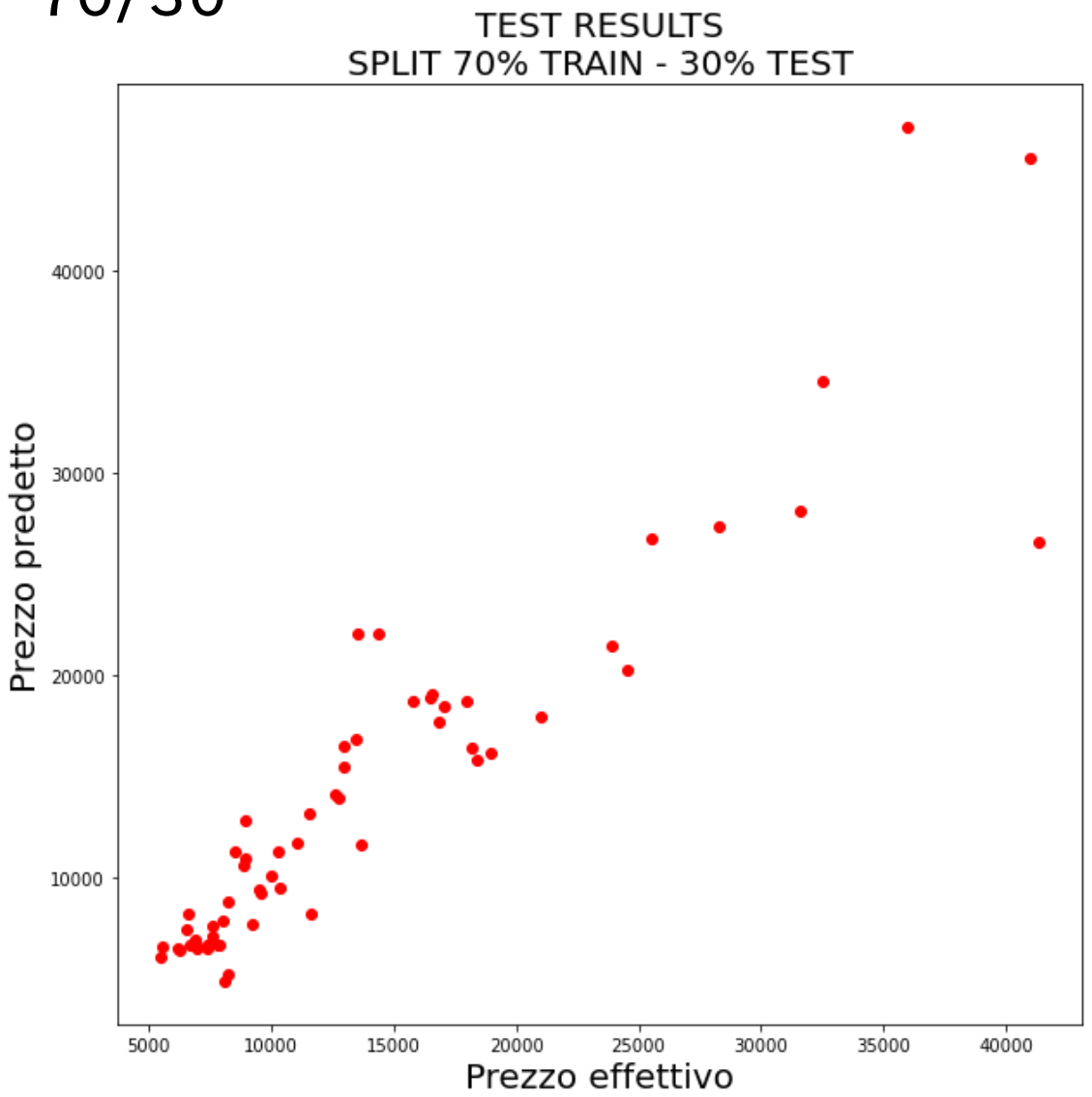
- Si è quindi usato il rimanente 30% dei dati per valutare il modello

```
#Valutazione
x_test = test_set.drop('price', axis = 1).values
predizione = predictions(x_test, finalTheta)
y_dataset = test_set['price'].values

r2, mae, rmse = modelEvaluation(prediction=predizione, target=y_dataset)
```

RISULTATI OTTENUTI

SPLIT 70/30

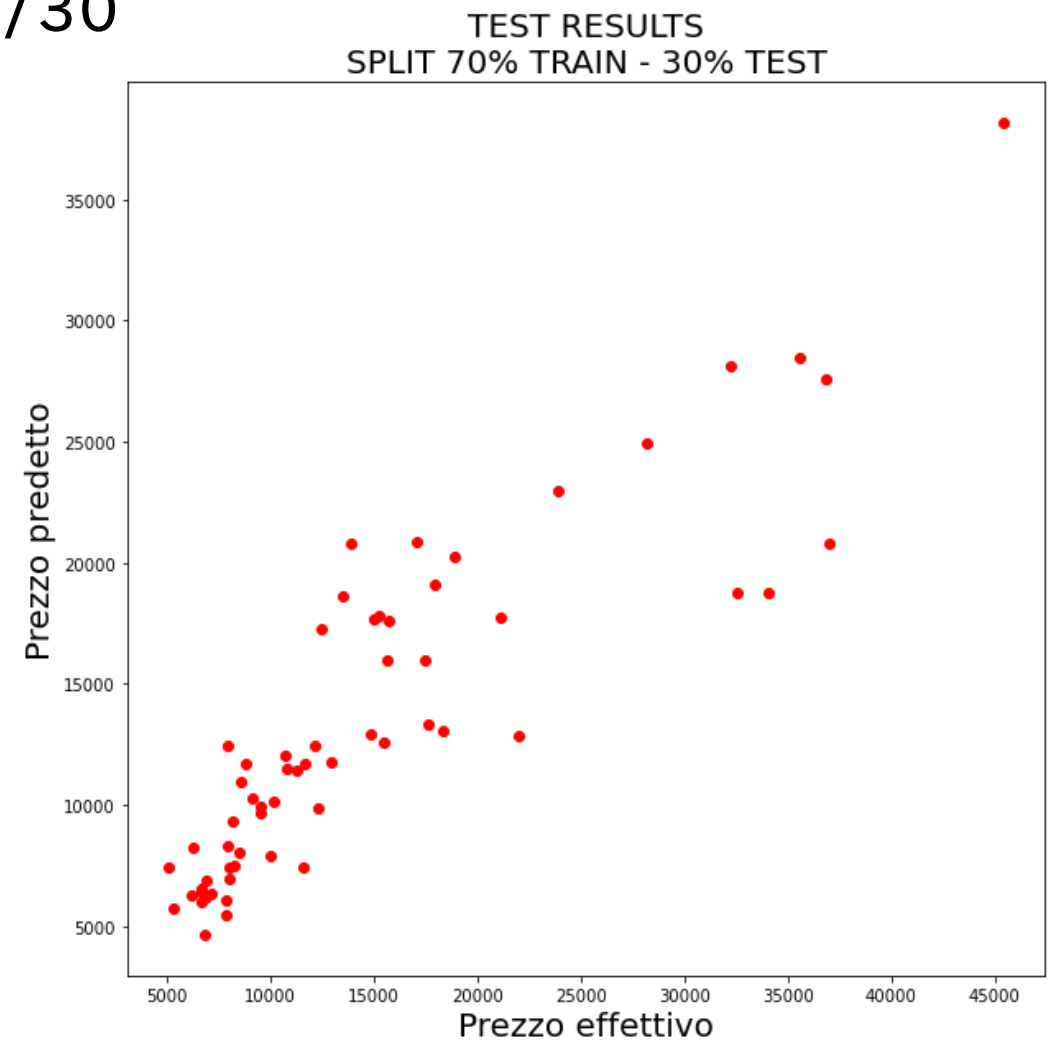
```
VALUTAZIONE DEL MODELLO TRAMITE SPLIT 70/30
########## MODELLO ##########

LGND: price = Theta * feature

price =
        13394.95214 * bias +
        88.38132 * symboling +
        -226.41427 * normalized-losses +
        -971.12837 * make +
        -2815.86861 * fuel-type +
        907.80329 * aspiration +
        361.29307 * num-of-doors +
        -392.04385 * body-style +
        -190.91468 * drive-wheels +
        1523.97461 * engine-location +
        817.19553 * wheel-base +
        -507.58954 * length +
        594.7819 * width +
        459.08793 * height +
        1449.7789 * curb-weight +
        -254.20866 * engine-type +
        1424.1512 * num-of-cylinders +
        4614.7462 * engine-size +
        -261.19594 * fuel-system +
...

R2 = 0.84434
Mean absolute error = 2215.35751
Root Mean Squared Error = 3424.78571
```



RISULTATI SCIKIT-LEARN

SPLIT 70/30

```
(scikit-learn) - VALUTAZIONE DEL MODELLO TRAMITE SPLIT 70/30
########## MODELLO ##########

LGND: price = Theta * feature

price =
        -156.48848 * symboling +
        -389.70055 * normalized-losses +
        -1148.38268 * make +
        2346.44335 * fuel-type +
        1235.31326 * aspiration +
        -366.98634 * num-of-doors +
        -404.41418 * body-style +
        -584.51593 * drive-wheels +
        -0.0 * engine-location +
        469.06136 * wheel-base +
        1683.28608 * length +
        -74.37712 * width +
        165.5025 * height +
        1080.78942 * curb-weight +
        192.80566 * engine-type +
        1510.1367 * num-of-cylinders +
        3386.3421 * engine-size +
        -1018.61062 * fuel-system +
        -503.942 * bore +
...

R2 = 0.75362
Mean Absolute Error = 2918.68401
Root Mean Squared Error = 4594.87981
```

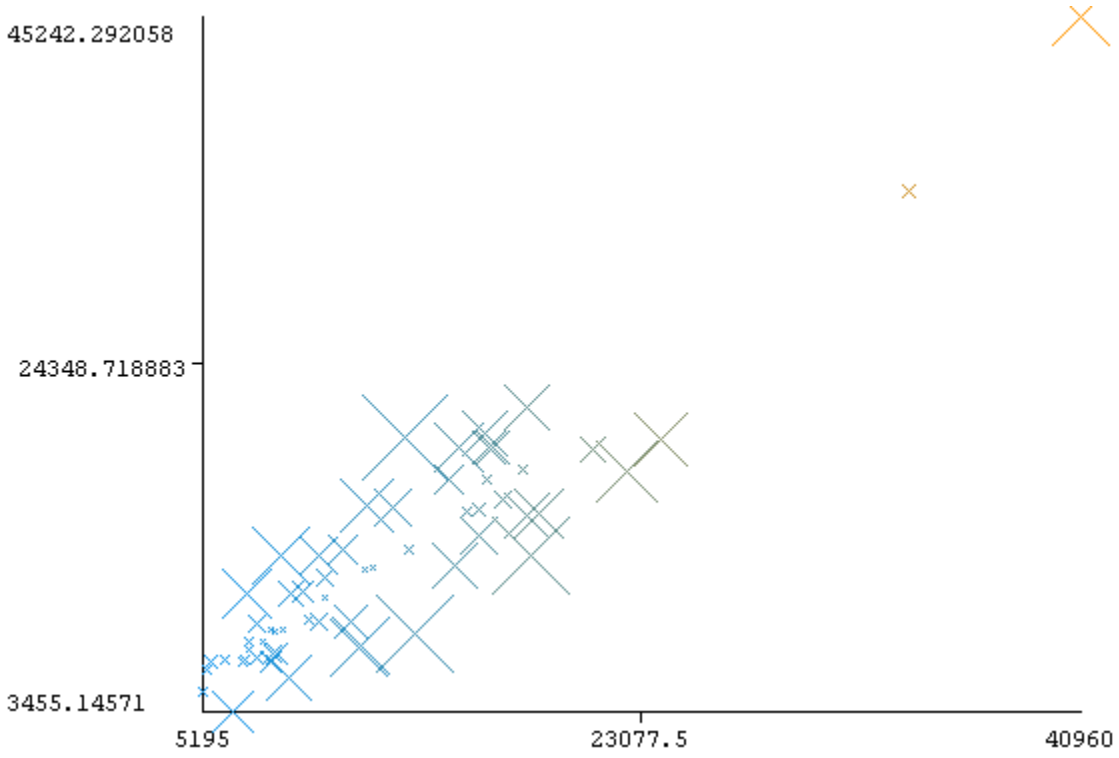


RISULTATI WEKA

SPLIT 70/30

```
Linear Regression Model

price =

   -989.5124 * make +
   1435.1087 * fuel-type +
    798.591  * aspiration +
   -364.3183 * body-style +
   -614.8027 * drive-wheels +
   1592.7488 * engine-location +
    402.81   * length +
    820.2342 * width +
    625.6361 * height +
   1354.9452 * curb-weight +
   1450.5204 * num-of-cylinders +
   3913.4632 * engine-size +
   -701.8827 * fuel-system +
   -664.6538 * stroke +
   -774.8616 * compression-ratio +
    725.2787 * peak-rpm +
  -1284.0913 * city-mpg +
   1658.6958 * highway-mpg +
  13270.0957
```

Correlation coefficient	0.9284
Mean absolute error	2019.6657
Root mean squared error	2616.1165



TRAINING

CROSS-VALIDAZIONE

- Perché usare la Cross-Validazione invece dello split 70/30?
  - sfrutta meglio i dati, minimizzando l'incertezza dovuta a una singola divisione casuale
  - Non c’è una divisione «statica» dei dati
- Cross-validazione K-Fold
  - Si divide il dataset in K sotto-dataset (fold)
  - Si usa un fold come **test** e tutti gli altri per **training**
  - Si ripete per ogni fold
  - Infine, si fa la media dei risultati

RISULTATI
CROSS-VALIDAZIONE

Risutati ottenuti

```
******* TEST CON CROSS-VALIDAZIONE *******
########## MODELLO ##########

LGND: price = Theta * feature

price =
        13185.33884 * bias +
        194.34995 * symboling +
        -464.61559 * normalized-losses +
        -1068.89491 * make +
        2863.27709 * fuel-type +
        788.81714 * aspiration +
        37.61788 * num-of-doors +
        -231.19123 * body-style +
        -670.68624 * drive-wheels +
        1429.12723 * engine-location +
        424.80253 * wheel-base +
        187.32459 * length +
        838.13768 * width +
        300.90602 * height +
        1185.10044 * curb-weight +
        57.60723 * engine-type +
        1284.19179 * num-of-cylinders +
        4083.79844 * engine-size +
...
R2 = 0.85136
Mean absolute error = 2094.67976
Root Mean Squared Error = 2811.50312
```

Scikit-Learn

```
R2 = 0.69439
Mean Absolute Error = 2903.06933
Root Mean Squared Error = 3879.80389
```

Weka

```
Linear Regression Model

price =

    -989.5124 * make +
    1435.1087 * fuel-type +
     798.591  * aspiration +
    -364.3183 * body-style +
    -614.8027 * drive-wheels +
    1592.7488 * engine-location +
     402.81   * length +
     820.2342 * width +
     625.6361 * height +
    1354.9452 * curb-weight +
    1450.5204 * num-of-cylinders +
    3913.4632 * engine-size +
    -701.8827 * fuel-system +
    -664.6538 * stroke +
    -774.8616 * compression-ratio +
     725.2787 * peak-rpm +
   -1284.0913 * city-mpg +
    1658.6958 * highway-mpg +
   13270.0957
```

```
=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.929
Mean absolute error                   2063.5527
Root mean squared error               2944.5578
```

CLASSIFICAZIONE

Algoritmo: K-NN

Dataset: NASA - Nearest Earth Objects

INFORMAZIONI GENERALI

- Classificazione = predizione di una classe
- Metodo **supervised**

- Ci sono 2 tipi di apprendimento:
  - **Eager learning:** viene costruito un modello esplicito a partire dai dati del dataset. Avendo il modello, i dati possono essere scartati e le predizioni verranno fatte dando i dati in input al modello
  - **Lazy learning:** i dati non vengono scartati ma vengono conservati e faranno parte del modello. Infatti, le predizioni vengono fatte confrontando le istanze incognite con quelle memorizzate

K-NEAREST NEIGHBOR (K-NN)

- Considera ogni istanza come un punto in uno spazio n-dimensionale
- Definisce i vicini in base alla **distanza**

- Svolge:
  - **Classificazione:** recupera la classe più rappresentata tra i k vicini
  - **Regressione:** assegna il valore medio dei valori dei k vicini

DISTANZA & PESO

- La distanza può essere riassunta con la formula di Minkowsky:

$$d(x,y) = (\sum_{i=1}^{d} |x_i - y_i|^p)^{\frac{1}{p}}$$ se $p = 2$ → distanza euclidea

```
def euclideanDistance(ex, query):
    distance = np.linalg.norm(ex - query)
    return distance
```

- Ha senso che elementi più vicini influenzino di più nella determinazione della predizione
- Il peso lo si può definire come inversamente proporzionale alla distanza

$$w_{iq} = \frac{1}{d(x_i, y_q)}$$

```
def weightByDistance(distance):
    weight = 1/distance
    return weight
```

ASSEGNAZIONE DELLA CLASSE

- Come si assegna un oggetto incognito ad una classe?
  - Classe più rappresentata fra i k nearest neighbor (se non pesati)
  - Nel caso in cui si utilizzi il peso:

$$score(c_k, x_q) = \frac{\sum_{i:c(x_i)=c_k} w_{iq}}{\sum_{i=1}^{n} w_{iq}}$$

  - Si assegna x_q alla classe con score più alto

IMPLEMENTAZIONE K-NN

```python
def kNN_weighted(examples, query, k):
    class_distance = []
    query_values = query.values
    query_weight = weightByDistance(np.linalg.norm(examples.drop('hazardous', axis=1) - query_values, axis=1))
    class_distance = list(zip(query_weight, examples['hazardous']))

    class_distance.sort(reverse=True)  # Ordinamento basato sui pesi

    freq_hazardous = sum(w for w, c in class_distance[:k] if c == True)
    freq_not_hazardous = sum(w for w, c in class_distance[:k] if c == False)

    if freq_hazardous > freq_not_hazardous:
        return True
    else:
        return False
```

IMPLEMENTAZIONE K-NN (CONT'D)

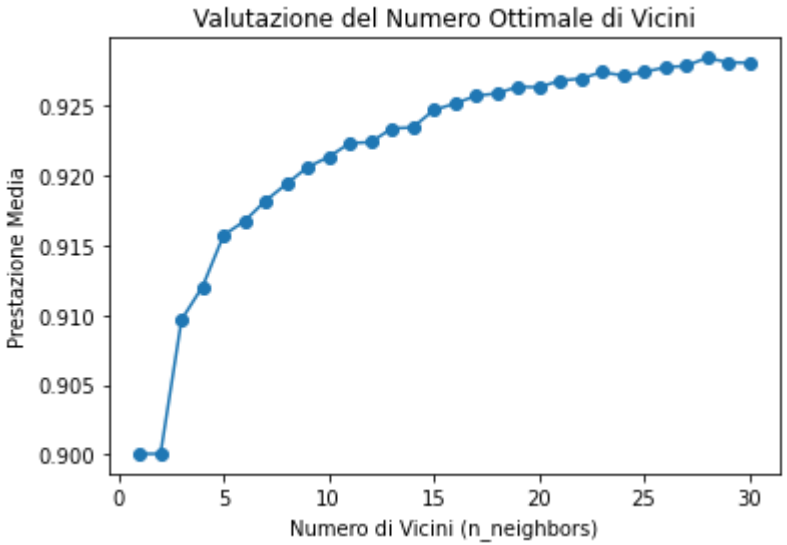
- Viene calcolata la distanza euclidea tra *query* e ciascun esempio nel dataset *examples*
- In base alla distanza, si calcolano i pesi per ogni esempio con la funzione *weightByDistance*
- Viene creata una lista di coppie, (peso calcolato, classe ("hazardous" o "not hazardous")) dell'istanza corrispondente nell'insieme di addestramento
- La lista di coppie viene ordinata in ordine decrescente in base ai pesi
- Viene calcolata la somma dei pesi delle prime k coppie (vicini più prossimi) per entrambe le classi ("hazardous" e "not hazardous"). E' il voto ponderato basato sulla presenza di vicini appartenenti a ciascuna classe
- Si confronta la somma dei pesi dei vicini appartenenti alla classe "hazardous" con quella dei vicini appartenenti alla classe "not hazardous".
- La query viene quindi classificata in base alla classe più rappresentata

PRE-PROCESSING DEI DATI (PYTHON)

- Nello script *edit_csv* si è provato il pre-processing dei dati tramite python
- Processing effettuato:
  - Rimozione delle righe con nome «594913 'Aylo'chaxnim (2020 AV2)» che causavano dei problemi di compatibilità
  - Rimozione dei duplicati
  - Rimozione delle colonne *id* e *name* in quanto non rilevanti per il calcolo della distanza
  - Si sono rese numeriche tutte le features nominali *(orbiting_body*, *sentry_object)*
  - Normalizzazione dei dati in un range [0,1]

SCELTA DEL PARAMETRO K

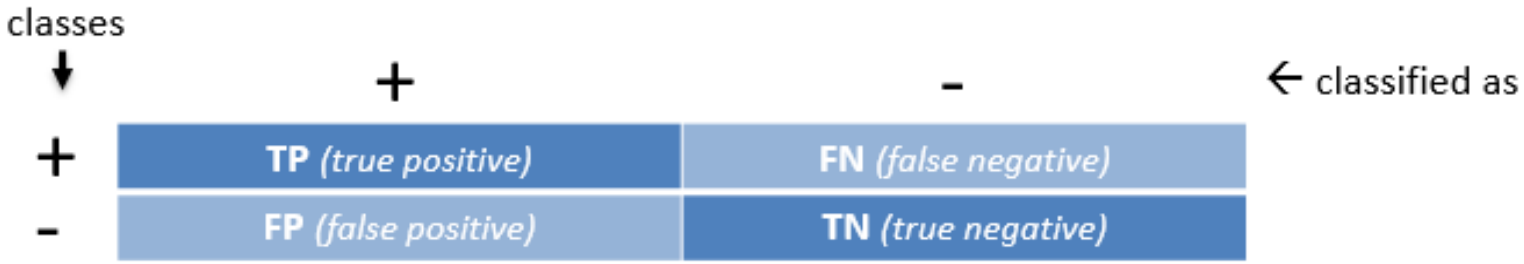
- Per scegliere il numero migliore di vicini, si sono valutate le prestazioni del modello incrementando k



- Con k troppo grande → **Underfit**
- Con k troppo piccolo → **Overfit**
- Si è quindi optato per k = 20

METRICHE PER LA VALUTAZIONE DEL MODELLO

- Per valutare le prestazioni del modello si sono usate diverse statistiche:
- Innanzitutto, si è partiti dalla **matrice di confusione**



METRICHE PER LA VALUTAZIONE DEL MODELLO

- **Precisione**: percentuale di casi classificati correttamente tra tutti quelli classificati positivi

$$P = \frac{TP}{TP + FP}$$

- **Recall**: percentuale di casi classificati correttamente tra tutti quelli veramente positivi

$$R = \frac{TP}{TP + FN}$$

- **F1 Score:** unisce precisione e recall

$$F_1 = 2\frac{PR}{P + R}$$

METRICHE PER LA VALUTAZIONE DEL MODELLO

- **Accuratezza**: capacità del modello di classificare correttamente

$$a = \frac{TP + TN}{m}$$

- **Coefficiente Kappa di Cohen:** permette di misurare la confidenza nell'accuratezza
  - K ≈ 0 → accuratezza non significativa (classi sbilanciate)
  - K ≈ 1 → accuratezza significativa

$$k = \frac{a - p_e}{1 - p_e} \qquad p_e = p_+ + p_-$$

$$p_- = \frac{TN + FP}{m} * \frac{TN + FN}{m} \qquad p_+ = \frac{TP + FN}{m} * \frac{TP + FP}{m}$$

IMPLEMENTAZIONE METRICHE DI VALUTAZIONE

```python
def confusionMatrix(classi):
    # True hazardous
    TH = classi.loc[(classi['hazardous'] == True) & (classi['hazardous_predicted'] == True)].reset_index(drop=True).shape[0]
    # False hazardous
    FH = classi.loc[(classi['hazardous'] == False) & (classi['hazardous_predicted'] == True)].reset_index(drop=True).shape[0]
    # False not hazardous
    FNH = classi.loc[(classi['hazardous'] == True) & (classi['hazardous_predicted'] == False)].reset_index(drop=True).shape[0]
    # True not hazardous
    TNH = classi.loc[(classi['hazardous'] == False) & (classi['hazardous_predicted'] == False)].reset_index(drop=True).shape[0]
    print(TH, FH, FNH, TNH)
    conf_matrix = pd.DataFrame({"True": [TH, FH], "False": [FNH, TNH]}, index = ["True", "False"])

    return(conf_matrix, TH, FH, FNH, TNH)
```

```python
#coeff Kappa
p_hazardous = ((TH + FNH)/m) * ((TH + FH)/m)
p_not_hazardous = ((TNH + FH)/m) * ((TNH + FNH)/m)
p_e = p_hazardous + p_not_hazardous
k_coeff = (accuracy - p_e) / (1 - p_e)
print("Kappa coefficient => " + str(round(k_coeff, 5)))

print("\n######## STATISTICHE DELLE DUE CLASSI ########")
print("\n++++++++ HAZARDOUS ++++++++")
precision = TH / (TH + FH)
print("Precisione = " + str(round(precision, 5)))
recall = TH / (TH + FNH)
print("Recall = " + str(round(recall, 5)))
F1_score = 2 * ((precision * recall) / (precision + recall))
print("F1 score = " + str(round(F1_score, 5)))

print("\n-------- NOT HAZARDOUS -------")
precision = TNH / (TNH + FNH)
print("Precisione = " + str(round(precision, 5)))
recall = TNH / (TNH + FH)
print("Recall = " + str(round(recall, 5)))
F1_score = 2 * ((precision * recall) / (precision + recall))
print("F1 score = " + str(round(F1_score, 5)))
```

```python
m = TH + FH + FNH + TNH

# classificazioni corrette
correct_classifications = TH + TNH
accuracy = correct_classifications / m
accuracy_percentage = accuracy * 100
print("Accuratezza del modello (istanze classificate correttamente) => " +

#classificazioni errate
incorrect_classifications = FH + FNH
inaccuracy = incorrect_classifications / m
inaccuracy_percentage = inaccuracy * 100
print("Imprecisione del modello (istanze classificate incorrettamente) =>
```

TEST – SPLIT 70/30
RISULTATI

Risultati ottenuti

```
------- MATRICE DI CONFUSIONE -------
142 53 561 7471
       True  False
True    142    561
False    53   7471
Accuratezza del modello (istanze classificate correttamente) => 92.53677
N° classificazioni corrette =>7613
Imprecisione del modello (istanze classificate incorrettamente) => 7.46323
N° classificazioni sbagliate =>614
Kappa coefficient => 0.28991

######## STATISTICHE DELLE DUE CLASSI ########

++++++++ HAZARDOUS ++++++++
Precisione = 0.72821
Recall = 0.20199
F1 score = 0.31626

-------- NOT HAZARDOUS -------
Precisione = 0.93015
Recall = 0.99296
F1 score = 0.96053
```

Risultati scikit-learn

```
******* RISULTATI CON SCIKIT-LEARN *******

------- MATRICE DI CONFUSIONE -------
       True  False
True    163    527
False    66   7471

Correctly Classified Instances (accuratezza): 7634 (92.79203%)
Incorrectly Classified Instances: 593 (7.20797%)
Kappa statistic: 0.32659

=== Detailed Accuracy By Class ===

--- CLASSE HAZARDOUS ---

Precision: 0.71179
Recall: 0.23623
F1 score: 0.35473
Fallout: 0.00876
Specificity: 0.99124

--- CLASSE NOT HAZARDOUS ---

Precision: 0.93411
Recall: 0.99124
F1 score: 0.96183
Fallout: 0.76377
Specificity: 0.23623
```

ANALISI DEI RISULTATI

- Analizzando i risultati si nota che:
  - Il modello ha un'accuratezza alta ma un coefficiente kappa basso

```
Kappa coefficient => 0.28991
```

```
Kappa statistic: 0.32659
```

  - Questo ci porta a dire che le classi sono sbilanciate, il che è confermato da entrambe la matrici di confusione

Matrice di confusione calcolata

```
------- MATRICE DI CONFUSIONE -------
       True  False
True    142    561
False    53   7471
```

Matrice di confusione di scikit-learn

```
------- MATRICE DI CONFUSIONE -------
       True  False
True    163    527
False    66   7471
```

Si nota come (fortunatamente) la stragrande maggioranza degli oggetti sia stata classificata come *non hazardous*

Ciò porta ad avere un coefficiente kappa estremamente basso

UNSUPERVISED

Algoritmo: Clustering (k-means)

Dataset: Country data

UNSUPERVISED LEARNING

- Ogni istanza del training set ha solo l'insieme degli attributi, non ci sono label che indicano la predizione corretta

- Con il **clustering** si cercano di individuare gruppi di elementi
  - Senza avere le label «soluzione»

K-MEANS

- E’ un algoritmo di **clustering**
- Si basa sulla **distanza** → elementi vicini appartengono allo stesso gruppo
- Bisogna specificare a priori il numero di cluster

K-MEANS

- **Stato iniziale**: randomico, si scelgono **k centroidi**

1. Calcolo della distanza di ogni punto dai centroidi
2. Assegno i punti ai centroidi più vicini
3. Cambio i centroidi → il nuovo centroide sarà la media dei punti
4. Go back to 1

IMPLEMENTAZIONE K-MEANS

```python
def assign_to_clusters(data, centroids):
    clusters = []
    for point in data:
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        cluster = np.argmin(distances)
        clusters.append(cluster)
    return clusters
```

```python
def update_centroids(data, clusters, num_clusters):
    new_centroids = np.zeros((num_clusters, data.shape[1]))
    for cluster in range(num_clusters):
        cluster_points = [data[i] for i in range(len(data)) if clusters[i] == cluster]
        if cluster_points:
            new_centroids[cluster] = np.mean(cluster_points, axis=0)
    return new_centroids
```

```python
num_clusters = 3
initial_centroids = np.array([m1[i] for i in np.random.choice(range(len(m1)), num_clusters, replace=False)])

max_iterations = 100
tolerance = 1e-4

centroids = initial_centroids
for i in range(max_iterations):
    old_centroids = centroids
    clusters = assign_to_clusters(m1, centroids)
    centroids = update_centroids(m1, clusters, num_clusters)
    if np.all(np.abs(centroids - old_centroids) < tolerance):
        break
```

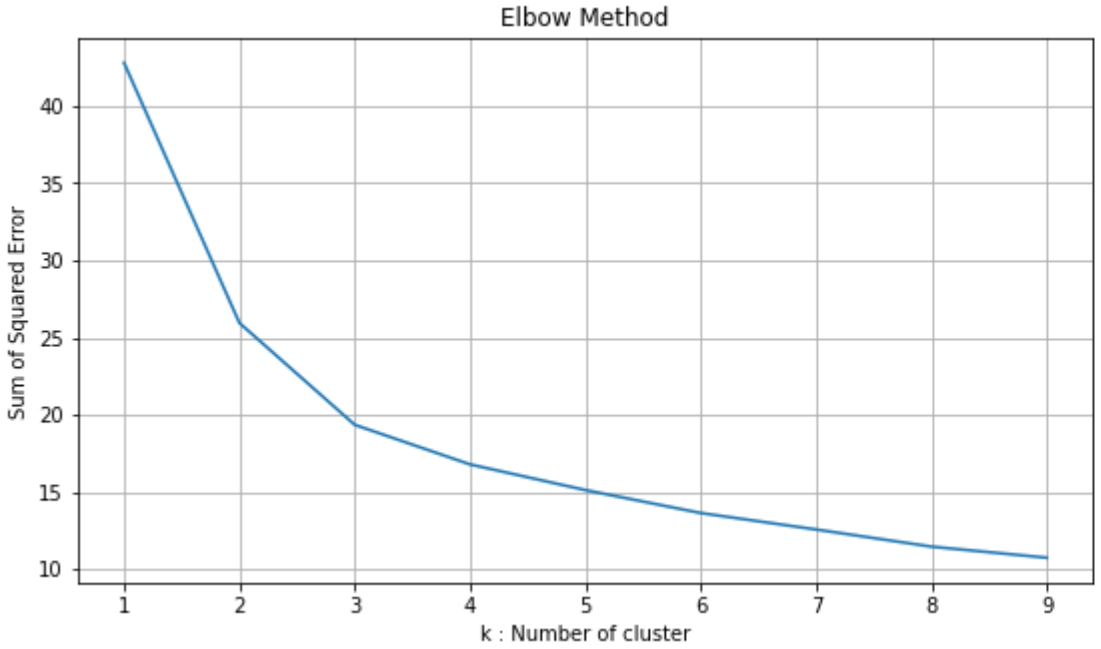
PRE-PROCESSING DEI DATI

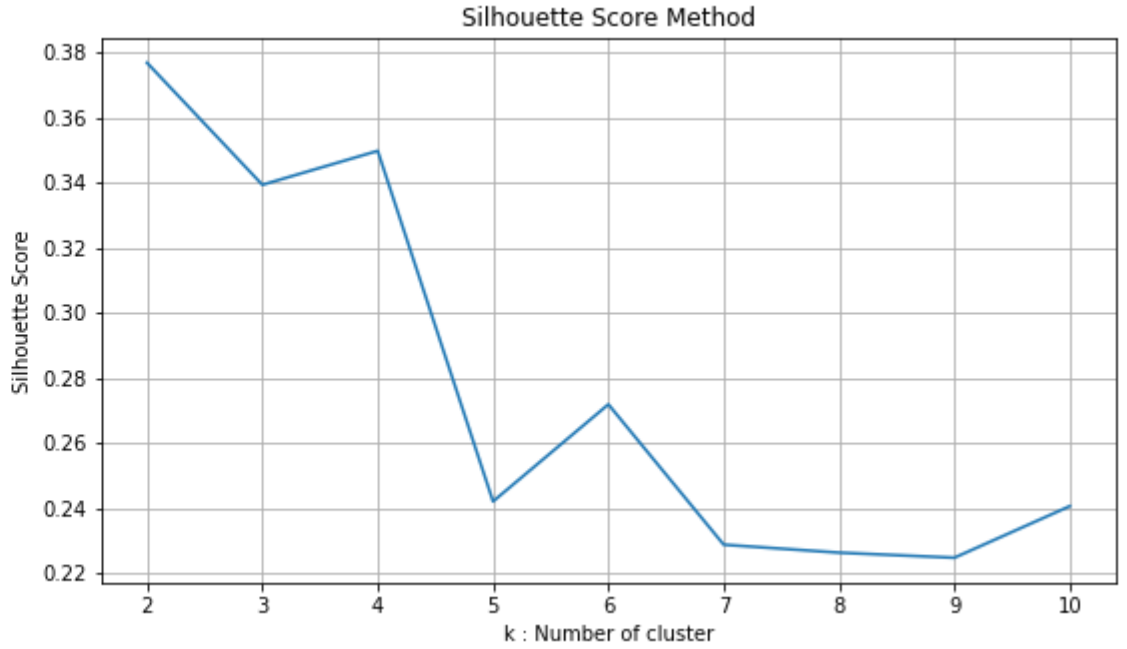
- Il dataset non presenta dati mancanti
- Si è applicata la normalizzazione a tutti i dati eccetto quelli nominali (*country*)

```
columns_to_normalize = df1.columns.difference(['country'])
scaler = MinMaxScaler()

# Applica la normalizzazione solo sulle colonne numeriche
df1[columns_to_normalize] = scaler.fit_transform(df1[columns_to_normalize])
```

NUMERO DI CLUSTER

- Per selezionare il numero di cluster ottimale si è sfruttato il metodo dell'elbow in combinazione con la silhouette





Si è quindi selezionato k = 3

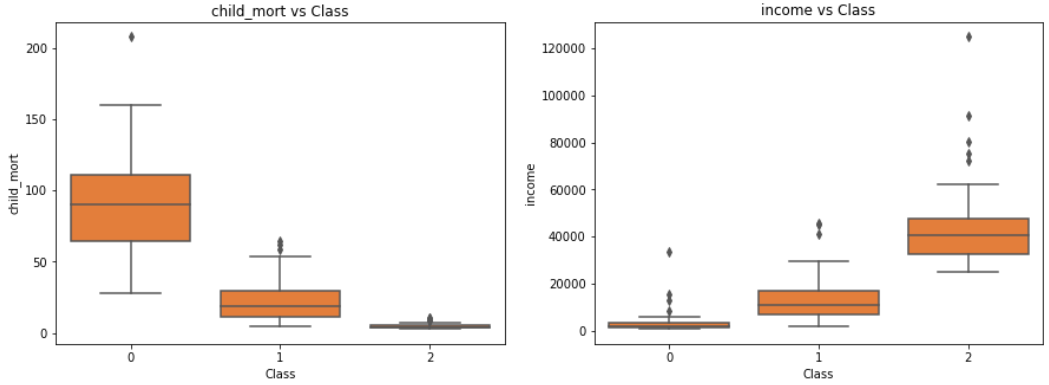
INTERPRETAZIONE DELLE CLASSI

- Il numero di cluster è stato scelto anche in base all'interpretazione che si può dare a ciascuna delle 3 classi individuate:
  a) Paesi che necessitano di aiuti
  b) Paesi che potrebbero aver bisogno di aiuti
  c) Paesi che non hanno bisogno di aiuti

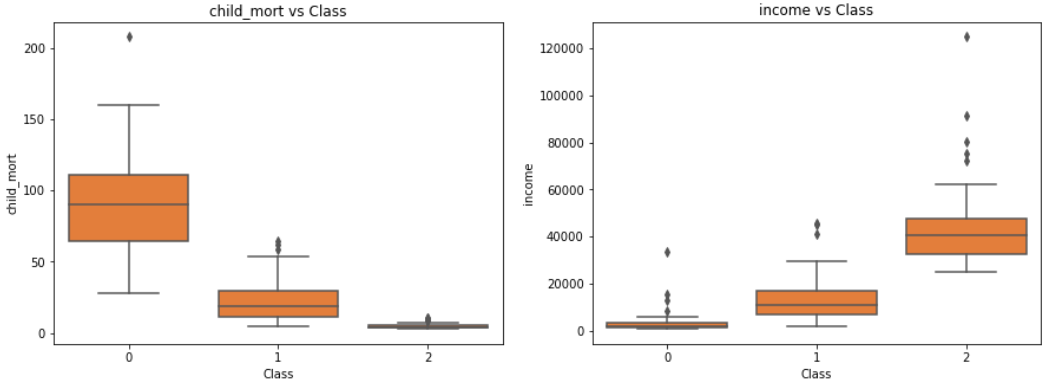
INDIVIDUAZIONE DEI CLUSTER

- La corrispondenza fra cluster e interpretazione è stata fatta valutando due parametri rilevanti per stimare lo sviluppo di un paese:
  - Mortalità infantile
  - Reddito netto per persona

Cluster ottenuti



Cluster individuati da scikit learn



INDIVIDUAZIONE DEI CLUSTER

- Si possono notare quindi le 3 classi:
  - Alta mortalità infantile & basso reddito → Paesi che necessitano di aiuti
  - Mortalità infantile e reddito medio-bassi → Paesi che potrebbero aver bisogno di aiuti
  - Bassa mortalità infantile & alto reddito → Paesi che non hanno bisogno di aiuti

VALUTAZIONE DEI RISULTATI

- I risultati sono stati valutati tramite:
  - Il plot di una cartina geografica colorata in base ai 3 cluster individuati
  - Il confronto degli indici di silhouette

SILHOUETTE

- E’ una misura utilizzata per capire la bontà dei cluster
- Confronta la distanza media degli elementi dello stesso cluster con la distanza media degli elementi di altri cluster

Distanza degli elementi dello stesso cluster

$$a(i) = \frac{1}{|c^i| - 1} \sum_{j \epsilon c^j, i \neq j} d(i,j)$$

Distanza degli elementi di un altro cluster

$$\mathrm{b}(i) = min_{i=j} \frac{1}{|c^i|} \sum_{j \epsilon c^j} d(i,j)$$

Calcolo silhouette

$$s(i) = \begin{cases} 1 - \dfrac{a(i)}{b(i)} & if\ a(i) < b(i) \\ 0 & if\ a(i) = b(i) \\ \dfrac{b(i)}{a(i)} - 1 & if\ a(i) > b(i) \end{cases}$$

IMPLEMENTAZIONE CALCOLO SILHOUETTE

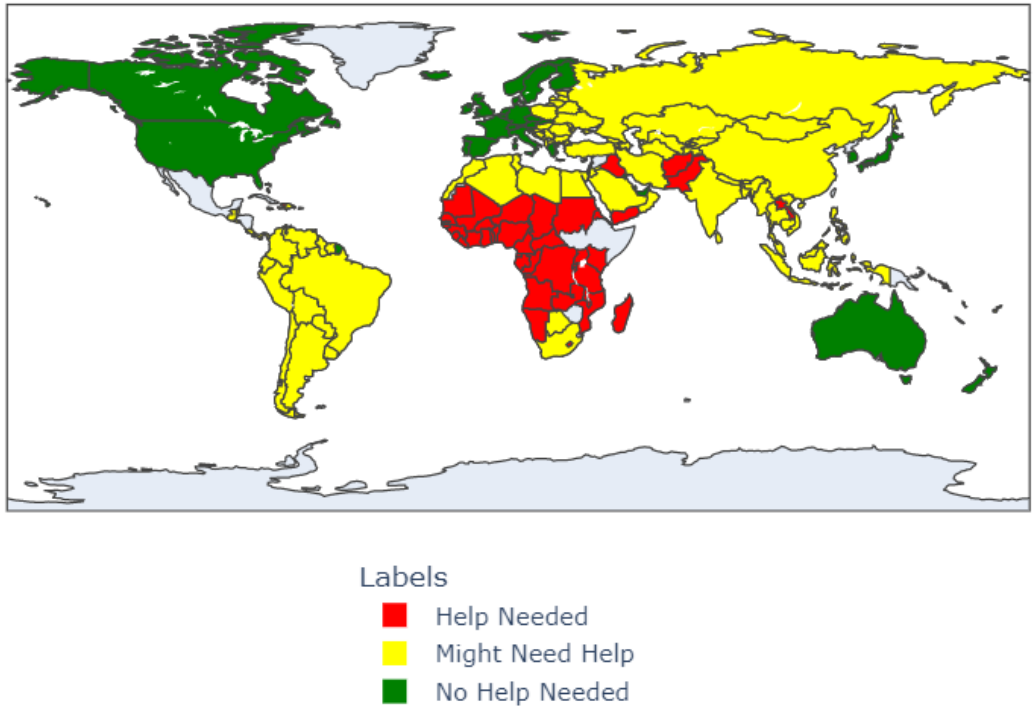
```python
def silhouette_coefficient(data, clusters):
    num_points = len(data)
    silhouette_values = []

    for i in range(num_points):
        point = data[i]
        cluster = clusters[i]
        cluster_points = [data[j] for j in range(num_points) if clusters[j] == cluster]
        a = np.mean([euclidean_distance(point, other_point) for other_point in cluster_points if not np.array_equal(point, other_point)])

        other_clusters = set(range(len(centroids))) - {cluster}
        b_values = []
        for other_cluster in other_clusters:
            other_cluster_points = [data[j] for j in range(num_points) if clusters[j] == other_cluster]
            b_values.append(np.mean([euclidean_distance(point, other_point) for other_point in other_cluster_points]))

        b = min(b_values)
        silhouette = (b - a) / max(a, b)
        silhouette_values.append(silhouette)

    return (silhouette_values)
```

```python
silhouette = silhouette_coefficient(m1, clusters)
print("Coefficiente di silhouette medio:", str(round(np.mean(silhouette), 4)))
```

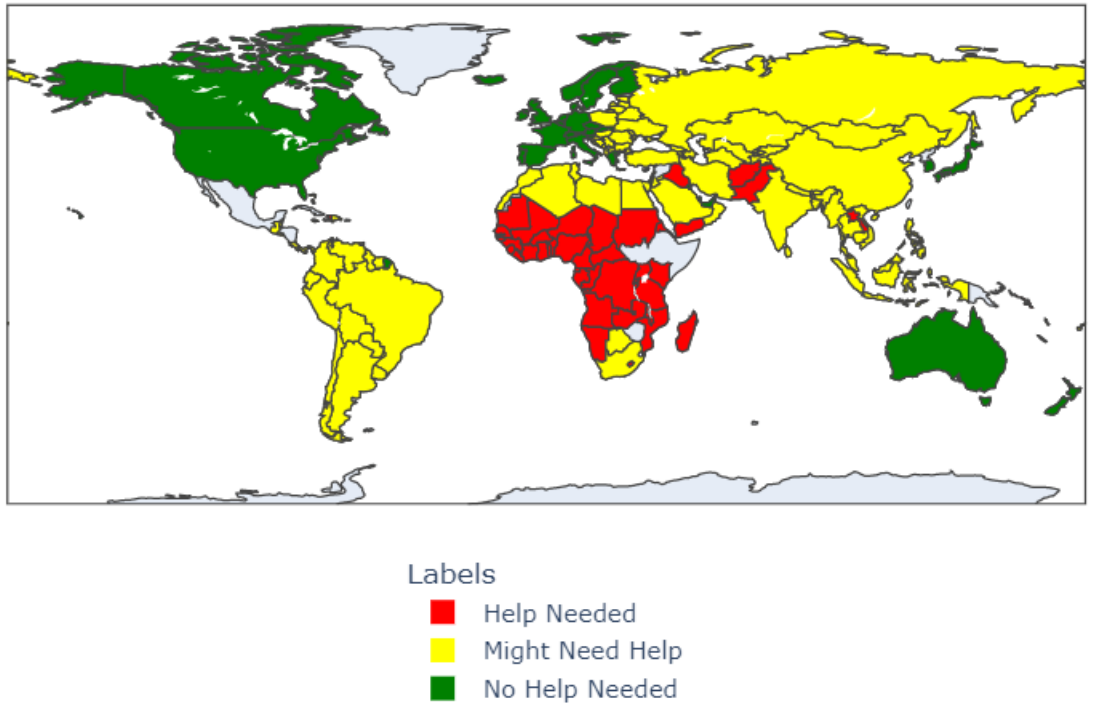
CONFRONTO DEI RISULTATI

CARTINA GEOGRAFICA

Risultati ottenuti

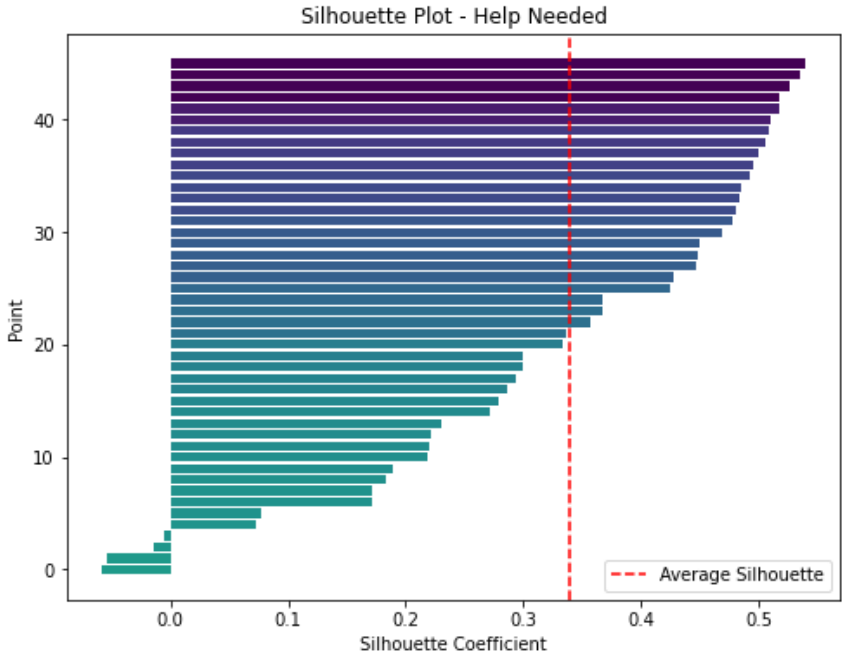


Risultati scikit learn

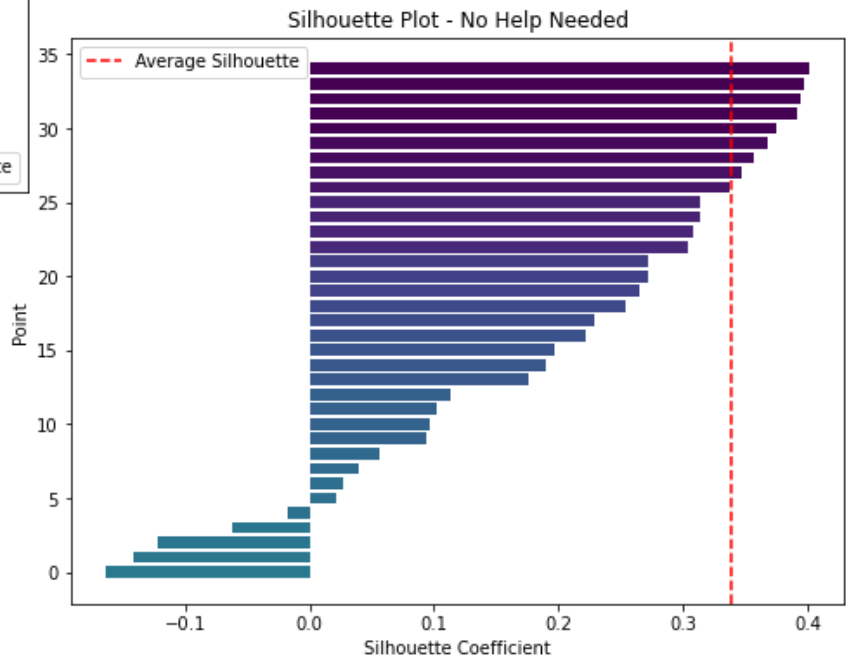


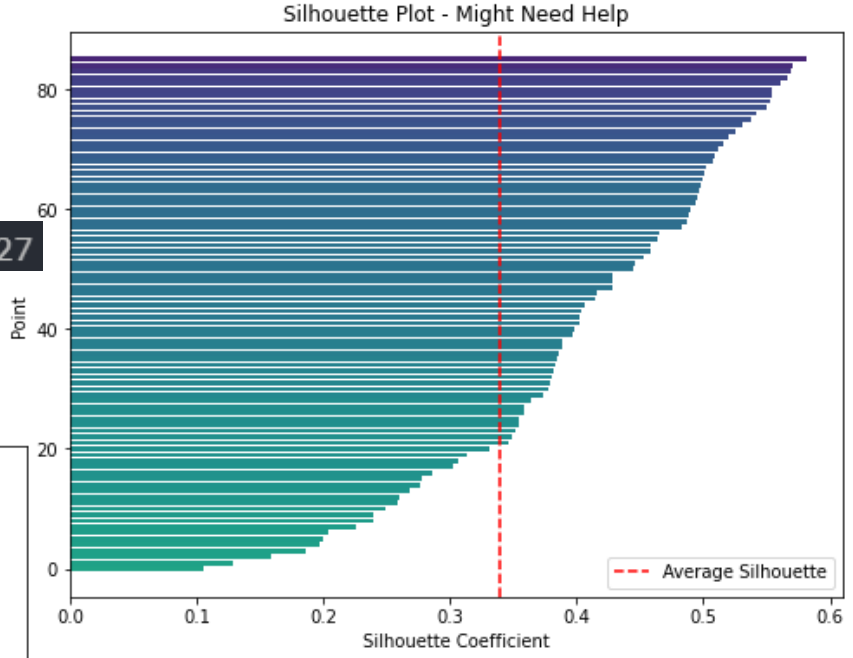
CONFRONTO DEI RISULTATI

SILHOUETTES – RISULTATI OTTENUTI



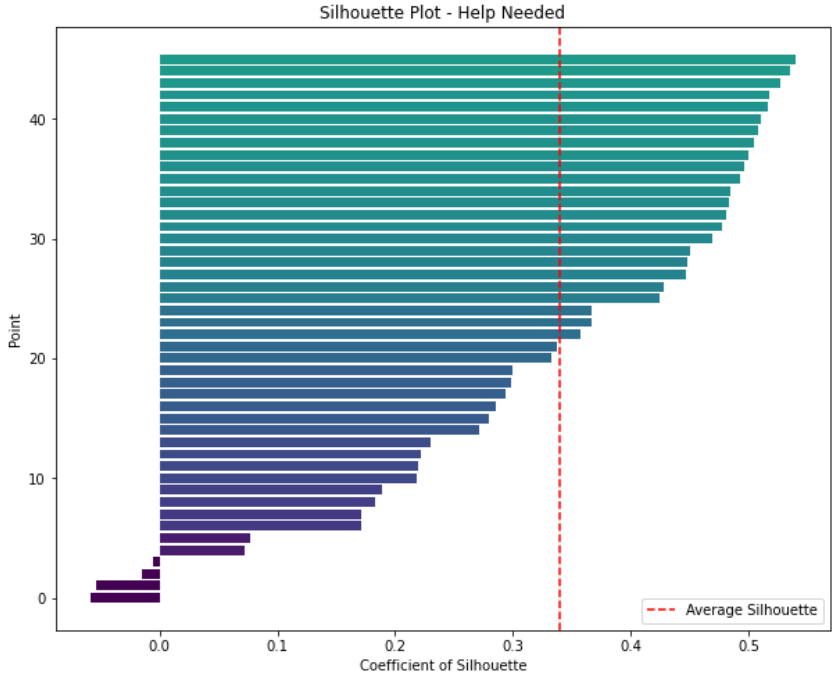

```
Coefficiente di silhouette medio: 0.3427
```





CONFRONTO DEI RISULTATI

SILHOUETTES – RISULTATI SCIKIT LEARN




```
Coefficiente di silhouette medio: 0.3393
```