Predizione di CO nell'aria

Lo scopo di questo studio è quello di predire la quantità di CO in un mg/m^3 d'aria in un istante di tempo data la presenza di altri elementi chimici correlati usando diversi modelli di regressione; la variabile da predire sarà quindi ricavabile da una funzione continua.

Caricamento delle librerie necessarie

Per cominciare importiamo le librerie necessarie per l'elaborazione dei dati:

- **NumPy:** necessaria per eseguire operazione su vettori e matrici di N dimensioni
- **Pandas:** per il caricamento e la gestione di dati sotto forma di tabelle
- **matplotlib:** fornisce funzioni per la creazione di diversi tipi di gravici

Dopo aver importanto le varie librerie abilitiamo la creazione dei grafici direttamente sul notebook usando l'istruzione *%matplotlib inline*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

Caricamento del dataset

Procediamo caricando il dataset in un dataframe di pandas. Il dataset viene sotto forma di zip per poi venire estratto usando il modulo zipfile.

```
import os.path
file_zip_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00360/AirQualityUCI
file_zip_name = "AirQualityUCI.zip"
file_data_name = "AirQualityUCI.csv"

if not os.path.exists(file_zip_name):
    from urllib.request import urlretrieve
    urlretrieve(file_zip_url, file_zip_name)
    from zipfile import ZipFile
    with ZipFile(file_zip_name) as f:
        f.extractall()

file = "/content/" + file_data_name;

airQuality = pd.read_csv(file,sep=";");
```

```
airQuality.head(10)
```

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)
0	10/03/2004	18.00.00	2,6	1360.0	150.0	11,9	1046.0	166.0
1	10/03/2004	19.00.00	2	1292.0	112.0	9,4	955.0	103.0
2	10/03/2004	20.00.00	2,2	1402.0	88.0	9,0	939.0	131.0
3	10/03/2004	21.00.00	2,2	1376.0	80.0	9,2	948.0	172.0
4	10/03/2004	22.00.00	1,6	1272.0	51.0	6,5	836.0	131.0
5	10/03/2004	23.00.00	1,2	1197.0	38.0	4,7	750.0	89.0
6	11/03/2004	00.00.00	1,2	1185.0	31.0	3,6	690.0	62.0
7	11/03/2004	01.00.00	1	1136.0	31.0	3,3	672.0	62.0
8	11/03/2004	02.00.00	0,9	1094.0	24.0	2,3	609.0	45.0
9	11/03/2004	03.00.00	0.6	1010.0	19.0	1.7	561.0	-200.0

Ci accorgiamo che il dataset contiene due colonne prive di significato, quindi procediamo a rimuoverle.

```
airQuality.drop(["Unnamed: 15","Unnamed: 16"], axis= 1, inplace=True)
```

Analizziamo ora com'è strutturato il dataset:

```
airQuality.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           9357 non-null   object 
 1   Time           9357 non-null   object 
 2   CO(GT)         9357 non-null   object 
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 non-null   float64
 5   C6H6(GT)       9357 non-null   object 
 6   PT08.S2(NMHC)  9357 non-null   float64
 7   NOx(GT)        9357 non-null   float64
 8   PT08.S3(NOx)   9357 non-null   float64
 9   NO2(GT)        9357 non-null   float64
 10  PT08.S4(NO2)   9357 non-null   float64
 11  PT08.S5(O3)    9357 non-null   float64
 12  T              9357 non-null   object 
```

```
 13  RH             9357 non-null   object
 14  AH             9357 non-null   object
dtypes: float64(8), object(7)
memory usage: 1.1+ MB
```

Le colonne del datafrane hanno il seguente significato:

- Date: la data in cui è stata effettuata la rilevazione dei dati
- Time: l'orario della rilevazione
- CO(GT): Quantità di CO in un mg/m^3 d'aria misurata ogni ora
- PT08.S1(CO): valore del sensore ad ossido di stagno rappresentante il valore nominale previsto di CO
- NMHC(GT): idrocarburi non metanici presenti in un microg/m^3
- C6H6(GT): concentrazione di benzene in un microg/m^3
- PT08.S2(NMHC): valore del sensore in titanio rappresentante il valore nominale previsto di NMHC
- NOx(GT): concentrazione di NOx in ppb
- PT08.S3(NOx): valore del sensore in ossido di tungsteno rappresentante il valore nominale previsto di NOx
- NO2(GT): concentrazione di NO2 in un microg/m^3
- PT08.S4(NO2): valore del sensore in ossido di tungsteno rappresentante il valore nominale previsto di NO2
- PT08.S5(O3): valore del sensore in ossido di indio rappresentante il valore nominale previsto di O3
- T: temperatura media durante la rilevazione
- RH: umidità relativa
- AH: umidità assoluta

Le colonne PT08.** contengono valori misurati da sensori che reagiscono alla presenza di specifici elementi chimici e permettono di stimare una possibile quantià di tale elemento, tuttavia da soli non sufficienti a dare una predizione esatta, per tanto utilizzeremo modelli di regressione lineare e polinomiale.

Ci accorgiamo anche che alcuni campi, sebbene contengano valori numerici, vengono considerati come object; questo avviene poichè utilizzano il formato decimale con la virgola al posta del punto.
Per correggere è sufficiente sostituire i punti con la virgola e viceversa e poi usare il metodo di pandas *to_numeric* per convertire tutte le stringhe numeriche in numeri.
Per sprecare meno spazio in memoria possiamo convertire la colonna Time in una colonna di tipo categorico; questo tornerà utile per poter effettuare la regressione usando anche questa colonna.

```
airQuality["CO(GT)"] = airQuality["CO(GT)"].str.replace(",",".",regex=False)
airQuality["C6H6(GT)"] = airQuality["C6H6(GT)"].str.replace(",",".",regex=False)
airQuality["T"] = airQuality["T"].str.replace(",",".",regex=False)
```

```
airQuality["RH"] = airQuality["RH"].str.replace(",",".",regex=False)
airQuality["AH"] = airQuality["AH"].str.replace(",",".",regex=False)
airQuality = airQuality.apply(pd.to_numeric, errors='ignore')
airQuality['Time'] = pd.Categorical(airQuality['Time'], categories=["00.00.00","01.00.00","02
                                                                    "19.00.00","20.00.00","21
airQuality.sort_values('Time',inplace=True)
```

Analisi preliminare

Ora che il nostro dataset è stato caricato correttamente possiamo effettuare un'analisi preliminare, visualizzando le medie, le deviazioni standard, i percentili, i minimi e massimi e il conteggio degli elementi distinti.

```
airQuality.describe()
```

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	P
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	
mean	-34.207524	1048.990061	-159.090093	1.865683	894.595276	168.616971	
std	77.657170	329.832710	139.789093	41.380206	342.333252	257.433866	
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	
25%	0.600000	921.000000	-200.000000	4.000000	711.000000	50.000000	
50%	1.500000	1053.000000	-200.000000	7.900000	895.000000	141.000000	
75%	2.600000	1221.000000	-200.000000	13.600000	1105.000000	284.000000	
max	11.900000	2040.000000	1189.000000	63.700000	2214.000000	1479.000000	

Quando in una rilevazione un valore è mancante viene assegnato -200.0; poichè non è possibile in nessun modo ricostruire i valori mancanti, siamo costretti ad eliminare tutte le righe che li contengono.

```
airQuality.dropna(inplace=True, how="all")
for column in airQuality:
  if airQuality[column].dtype == np.float64:
    airQuality = airQuality[airQuality[column] > - 200]
```

```
airQuality.describe()
```

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.
count	827.000000	827.000000	827.000000	827.000000	827.000000	827.000000	82
mean	2.353567	1207.879081	231.025393	10.771100	966.116082	143.501814	96
std	1.409496	241.816997	208.461912	7.418134	266.424557	81.829717	26
min	0.300000	753.000000	7.000000	0.500000	448.000000	12.000000	46
25%	1.300000	1017.000000	77.000000	4.800000	754.000000	81.000000	76
50%	2.000000	1172.000000	157.000000	9.100000	944.000000	128.000000	92

Abbiamo a disposizione 827 righe differenti, per cui i dati sono sufficientemente variegati.
Proviamo ora a calcolare il coefficiente di correlazione di pearson delle sole colonne numeriche per capire quali colonne siano effettivamente più utili per la predizione del monossido di carbonio.

```
airQuality.corr(method='pearson')
```

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT
CO(GT)	1.000000	0.936261	0.887167	0.972660	0.958426	0.951342	
PT08.S1(CO)	0.936261	1.000000	0.781747	0.931368	0.936346	0.922885	
NMHC(GT)	0.887167	0.781747	1.000000	0.897928	0.875061	0.811182	
C6H6(GT)	0.972660	0.931368	0.897928	1.000000	0.984834	0.927304	
PT08.S2(NMHC)	0.958426	0.936346	0.875061	0.984834	1.000000	0.926633	
NOx(GT)	0.951342	0.922885	0.811182	0.927304	0.926633	1.000000	
PT08.S3(NOx)	-0.823728	-0.829577	-0.774237	-0.848850	-0.910651	-0.814297	
NO2(GT)	0.861432	0.866579	0.728052	0.846743	0.885023	0.857425	
PT08.S4(NO2)	0.939921	0.945020	0.848489	0.960811	0.957883	0.912724	
PT08.S5(O3)	0.882943	0.935011	0.761905	0.896978	0.909100	0.893381	
T	0.318261	0.324815	0.366976	0.418409	0.445615	0.238395	
RH	-0.105157	-0.039570	-0.160257	-0.178410	-0.193333	-0.041975	
AH	0.295591	0.407038	0.282142	0.313415	0.325333	0.270679	

La prima cosa che ci appare evidente è come le varie variabili siano collegate tutte fra di loro; questo fatto sarà da tenere presente quando dovremo scegliere quale modello sia il migliore.

Notiamo come l'umidità non influisca in modo particolare sulla quantità di monossido di carbonio presente nell'aria, così come la temperatura.

La colonna PT08.S1(CO) è particolarmente legata alla colonna CO(GT) come è giusto aspettarsi; anche la presenza degli altri elementi chimici risulta essere correlata alla presenza di CO.

Possiamo quindi scartare le colonne T, RH e AH poichè poco influenti e rischiano di rendere il modello poco preciso. Teniamo presente che in questa fase abbiamo calcolato l'indice di correlazione di Pearson delle sole colonne numeriche, quindi le colonne Date e Time sono state

Il giorno in cui è stata effettuata la registrazione è a livello di logica poco influente poichè il traffico resta mediamente costante tutti i giorni dell'anno; le poche eccezioni rischierebbero di rovinare il modello.
Lo stesso discorso non si può invece fare per l'orario in cui viene effettuata la registrazione: ci aspettiamo per esempio che nelle ore di maggior traffico (es. 18:00) ci sia un picco di monossido di carbonio nell'aria; procediamo ora ad analizzare l'indice di correlazione di Pearson dell'orario e della colonna CO(GT).

```
dummies = pd.get_dummies(airQuality['Time'])
df = pd.DataFrame()
df.insert(0,"CO",airQuality["CO(GT)"])
df = pd.concat([df,dummies],axis=1)
df.corr(method='pearson')
```

	CO	00.00.00	01.00.00	02.00.00	03.00.00	04.00.00	05.00.00	06.00.
CO	1.000000	-0.115352	-0.162694	-0.200895	NaN	-0.153868	-0.254043	-0.2004
00.00.00	-0.115352	1.000000	-0.047494	-0.047494	NaN	-0.027734	-0.047494	-0.0474
01.00.00	-0.162694	-0.047494	1.000000	-0.046835	NaN	-0.027349	-0.046835	-0.0468
02.00.00	-0.200895	-0.047494	-0.046835	1.000000	NaN	-0.027349	-0.046835	-0.0468
03.00.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
04.00.00	-0.153868	-0.027734	-0.027349	-0.027349	NaN	1.000000	-0.027349	-0.0273
05.00.00	-0.254043	-0.047494	-0.046835	-0.046835	NaN	-0.027349	1.000000	-0.0468
06.00.00	-0.200479	-0.047494	-0.046835	-0.046835	NaN	-0.027349	-0.046835	1.0000
07.00.00	0.000490	-0.047494	-0.046835	-0.046835	NaN	-0.027349	-0.046835	-0.0468

Sorprendentemente l'orario della rivelazione è completamente trascurabile per la predizione del monossido di carbonio.

Rimuoviamo quindi tutte le colonne che abbiamo identificato come non necessarie per la regressione.

```
airQuality = airQuality.drop('T',1)
airQuality = airQuality.drop('RH',1)
airQuality = airQuality.drop('AH',1)
airQuality = airQuality.drop('Time',1)
airQuality = airQuality.drop('Date',1)
```

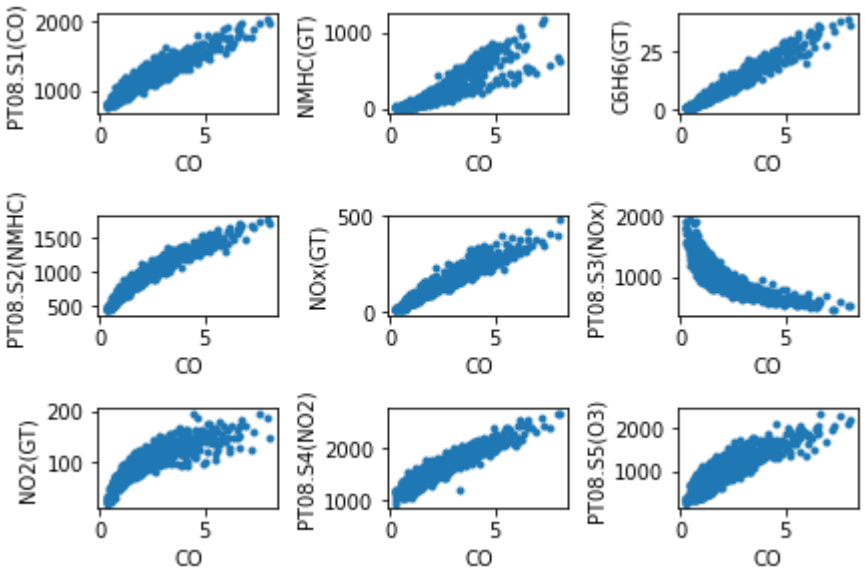
Costruzione dei modelli

Ora che abbiamo terminato l'analisi dei dati, possiamo costruire i diversi modelli di regressione; iniziamo valutando se sia più efficace un modello di regressione lineare oppure uno di regressione polinomiale.

Per aiutarci disegniamo il grafico di distribuzione fra la colonna CO(GT) e tutte le altre.

```
columnsNumber = len(airQuality.columns)
from math import sqrt
gridSize = int(sqrt(columnsNumber))
fig, ax = plt.subplots(gridSize, gridSize, constrained_layout = True)
i = 0
for column in airQuality:
  if column != "CO(GT)":
    x = int(i/gridSize)
    y = int(i%gridSize)
    ax[x,y].set_xlabel("CO")
    ax[x,y].set_ylabel(column)
```

```
    ax[x,y].plot(airQuality["CO(GT)"],airQuality[column],marker=".",linestyle="None")
    i += 1
plt.show()
```



È evidenziabile una leggera curvatura in alcuni dei grafici appena disegnati; questo ci suggerisce che una regressione polinomiale di grado due possa minimizzare l'errore sui dati, tuttavia anche un modello di grado uno potrebbe essere corretto.

Procediamo preparando il training ed il validation set usando il metodo hold-out.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
y = airQuality['CO(GT)']
X = airQuality.drop('CO(GT)',1)
X_train,X_val,y_train,y_val = train_test_split(X, y, train_size=0.7)
```

Linear regression

Ora che il training set è pronto possiamo addestrare i modelli; poichè i dati sono di grandezze diverse diremo alla classe LinearRegression di normalizzare i dati.

```
lrm = LinearRegression(normalize=True)
lrm.fit(X_train, y_train)
y_pred = lrm.predict(X_val)
print("Errore quadratico medio: " + str(np.mean((y_pred - y_val)**2)))
print("Errore relativo: "  + str(np.mean(np.abs(y_pred-y_val)/y_val)*100) + " %")
print("Indice R sui dati di addestramento: " + str(lrm.score(X_train, y_train)))
print("Indice R sui dati di validazione: " + str(lrm.score(X_val, y_val)))
```

```
Errore quadratico medio: 0.058136152842855233
Errore relativo: 8.533795341290192 %
```

```
Indice R sui dati di addestramento: 0.9738539448184231
Indice R sui dati di validazione: 0.969529656769152
```

Polynomial regression

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
prm = Pipeline([
                ("poly",
                 PolynomialFeatures(degree=2, include_bias=False)),
                ("linreg",LinearRegression(normalize=True))
])
prm.fit(X_train, y_train)
y_pred = prm.predict(X_val)
print("Errore quadratico medio: " + str(np.mean((y_pred - y_val)**2)))
print("Errore relativo: "  + str(np.mean(np.abs(y_pred-y_val)/y_val)*100) + " %")
print("Indice R quadro sui dati di addestramento: " + str(prm.score(X_train, y_train)))
print("Indice R quadro sui dati di validazione: " + str(prm.score(X_val, y_val)))
```

```
Errore quadratico medio: 0.05000179187321585
Errore relativo: 7.591480528829457 %
Indice R quadro sui dati di addestramento: 0.9833616060574004
Indice R quadro sui dati di validazione: 0.9737930412311148
```

Entrambi i modelli sono ottimi poichè l'indice R quadro è vicino ad 1, tuttavia sono anche molto simili rendendo difficile effettuare una preferenza; è possibile che esistano soluzioni migliori con un grado del polinomio più elevato? Utilizziamo la classe GridSearchCV su un range che va dal grado 1 al grado 10 e dividiamo il training set usando il KFold a 5 divisioni.

Polynomial regression con GridSearch

```
from sklearn.model_selection import GridSearchCV

pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression(normalize=True)),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': list(range(1,10))},
    scoring='r2',
    cv=5,
)
```

```
search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('poly',
                                        PolynomialFeatures(degree=2,
                                                           include_bias=False,
                                                           interaction_only=False,
                                                           order='C')),
                                       ('model',
                                        LinearRegression(copy_X=True,
                                                         fit_intercept=True,
                                                         n_jobs=None,
                                                         normalize=True))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'poly__degree': [1, 2, 3, 4, 5, 6, 7, 8, 9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
print("R quadro: " + str(search.score(X_val,y_val)))
print("Grado del polinomio migliore: " + str(search.best_params_["poly__degree"]))
```

```
R quadro: 0.9737930412311148
Grado del polinomio migliore: 2
```

Ridge regression

I risultati ottenuti con la regressione lineare e polinomiale sono ottimi, tuttavia potrebbero essere migliorati usando la ridge regression.

```
from sklearn.linear_model import Ridge
model = Ridge()
parameters = {'alpha': range(1,50)}
Ridge_reg= GridSearchCV(model, parameters, scoring='r2',cv=5)
Ridge_reg.fit(X_train, y_train)
print("Alpha migliore: " + str(Ridge_reg.best_params_['alpha']))
print("R quadro: " + str(Ridge_reg.score(X_val, y_val)))
```

```
Alpha migliore: 1
R quadro: 0.9695212527904744
```

Lasso regression

Si può effettuare un ulteriore tentativo di miglioramento utilizzando la regressione lasso; ricordiamo che la differenza sostanziale fra questa e la regressione ridge è che quest'ultima non

azzera mai i coefficienti; in altre parole la regressione lasso penalizza di più i coefficienti meno

```
from sklearn.linear_model import LassoCV
reg = LassoCV(cv=5, random_state=0, normalize=True).fit(X_train, y_train)
print("R quadro: " + str(reg.score(X_val, y_val)))
print("Alpha migliore: " + str(reg.alpha_))
print("Numero iterazioni: " + str(reg.n_iter_))
```

```
R quadro: 0.9686725917710726
Alpha migliore: 5.7463947729626e-05
Numero iterazioni: 262
```

Conclusioni

Entrambi i modelli di regressione lineare e polinomiale hanno un indice R quadro molto vicino ad uno, il che indica che essi sono molto precisi; il fatto che lo score sia altrettanto vicino all'uno usando il validation set ci fa capire che non sussiste alcun problema di overfitting. La differenza fra l'indice R quadro del modello basato su regressione lineare e di quello del modello basato su regressione polinomiale variano nell'ordine dei millesimi, rendendo difficile effettuare una scelta netta. Dovendoci affitare strettamente ai risultati della grid search allora opteremmo per il modello di regressione polinomiale di grado due.
Il modello ricavato dalla ridge regression è di un centesimo meno accurato (confrontando il coeffiente R2) rispetto agli, tuttavia ha la garanzia che, al variare della complessità dei dati del modello, esso non sia troppo aderente al training set rimanendo generico; per tale motivo il modello ridge è preferibile rispetto ai due precedenti.
L'ultimo modello creato è quello basato sulla regressiona lasso, anch'esso molto preciso e differente da quello precedente di un solo millesimo.

Riassumendo, abbiamo costruito quattro modelli con un'elevata precisione per predirre la quantità di monossido di carbonio nell'aria in un mg/m^3 d'aria conoscendo la composizione chimica di quest'ultima, soffisfancendo così gli obiettivi prefissati; dovendo effettuare una classifica dei modelli migliori partirei utilizzando il modello basato su regressione lasso, seguito da quello ridge, quello polinomiale di grado due ed infine quello lineare; i modelli di regressione polinomiale e di regressione lineare vengono posizionati per ultimi poichè non avendo applicata una regolarizzazione non risolvono il problema della collinearità delle variabili di input.

✓ 0 s data/ora di completamento: 12:50