

CN25 - Homework 3

Matteo Mazzetti 0001161552

1 Regressione lineare semplice

1.1 Problema test

Creiamo un problema test in Python con un vettore arbitrario alphaTrue tale che $\alpha_0 = 0$, $\alpha_i = i^{\frac{1}{2i}}$ per $i = 1..d$ e con n valori x_i equispaziati in [0,1].

```
def f(x, alpha):
    d=alpha.shape[0]-1
    y=0
    for i in range(d+1):
        y=y+alpha[i]*x**i
    return y
d=8
alphaTrue=np.zeros((d+1,))
for i in range(d+1):
    if(i==0): alphaTrue[i]=0
    else: alphaTrue[i]=np.sqrt(i)**(1/i)
n=15
x=np.linspace(0, 1, n)
e=np.random.normal(loc=0,scale=0.1, size=(n,))
y=np.zeros_like(x)
for i in range(n):
    y[i]=f(x[i],alphaTrue)+e[i]
def f(x, alpha):
    d=alpha.shape[0]-1
    y=np.zeros_like(x)
    for i in range(d+1):
        y=y+alpha[i]*x**i
    return y
def vandermonde(x,d):
    n=x.shape[0]
    X=np.zeros((n,d+1))
    for i in range(d+1):
        X[:,i]=x**i
    return X
X=vandermonde(x,d)
def SVD(X,y,d):
    n=y.shape[0]
    U,s,Vt=np.linalg.svd(X)
    Sigma=np.zeros((n,d+1))
    for i in range(d+1):
        Sigma[i,i]=s[i]
    alphaSVD=np.zeros((d+1,))
    for i in range(d+1):
        alphaSVD+=(U[:,i].T @ y)/s[i]*Vt[i,:]
    return alphaSVD
alphaSVD=SVD(X,y,d)
def residue(X,y,alpha):
    return np.linalg.norm(X @ alpha - y)
print(f"Residuo SVD: {residue(X, y, alphaSVD)}")
print(f"Residuo True (e): {residue(X, y, alphaTrue)}")
```

Eseguiamo lo snippet di codice precedente e utilizziamo una tabella per visualizzare gli output:

Test	n	d	alphaTrue	Residuo True (e)	Residuo SVD
1	6	5	(0, 1, 1.18920712, 1.20093696, 1.18920712, 1.17461894)	0.20034206385514466	5.1038709474956296e-14
2	10	2	(0, 1, 1.18920712)	0.2679803924132402	0.1086352499303631
3	10	5	(0, 1, 1.18920712, 1.20093696, 1.18920712, 1.17461894)	0.4252414548545332	0.22701582110711566
4	15	8	(0, 1, 1.18920712, 1.20093696, 1.18920712, 1.17461894, 1.16103667, 1.14911673, 1.13878863)	0.3876815795154283	0.1796314890672186

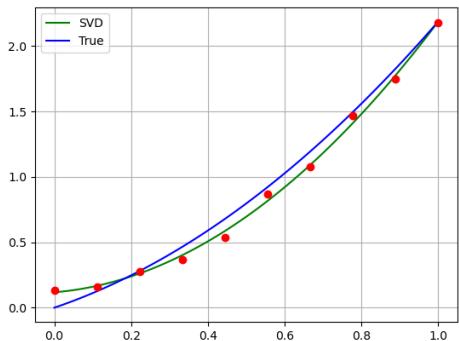


Figure 1: Test 2

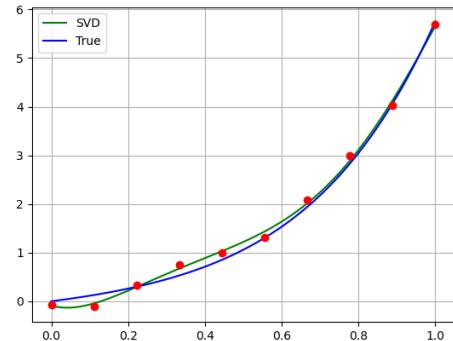


Figure 2: Test 3

1.2 Salary dataset

Usiamo ora le funzioni scritte in precedenza (vandermonde, SVD, f) per problema vero: troviamo la funzione che approssima meglio i dati del dataset Salary_Data.csv che mette in relazione gli anni di lavoro con il salario.

```
dataSet=pd.read_csv("dataSets/Salary_Data.csv")
print(f"Head dataset: \n{dataSet.head()}")
xRaw=dataSet["YearsExperience"].values
yRaw=dataSet["Salary"].values
print(f"Head dataset: \n{dataSet.head()}")
print(f"Descrizione dataset: \n{dataSet.describe()}")
xRaw=dataSet["YearsExperience"].values
yRaw=dataSet["Salary"].values
print(f"Numero di righe: {yRaw.shape[0]}")
d=5
X=vandermonde(xRaw,d)
alphaSVD=SVD(X,yRaw,d)
```

L'output di questo frammento di codice è:

```
Head dataset:
   YearsExperience      Salary
0            1.1    39343.0
1            1.3    46205.0
2            1.5    37731.0
3            2.0    43525.0
4            2.2    39891.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   YearsExperience    30 non-null      float64
 1   Salary             30 non-null      float64
dtypes: float64(2)
memory usage: 612.0 bytes
Descrizione dataset:
   YearsExperience      Salary
count      30.000000    30.000000
mean       5.313333    76003.000000
std        2.837888    27414.429785
min        1.100000    37731.000000
25%        3.200000    56720.750000
50%        4.700000    65237.000000
75%        7.700000    100544.750000
max       10.500000   122391.000000
Numero di righe:30
```

Vediamo al variare del grado (d) del polinomio approssimante come cambia la funzione.

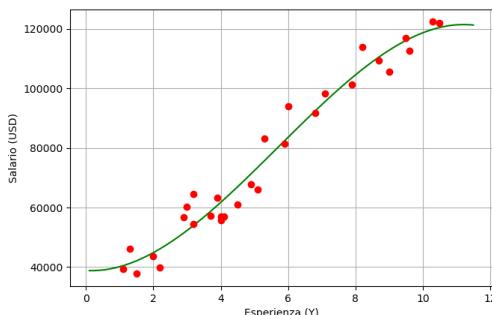


Figure 3: d=3

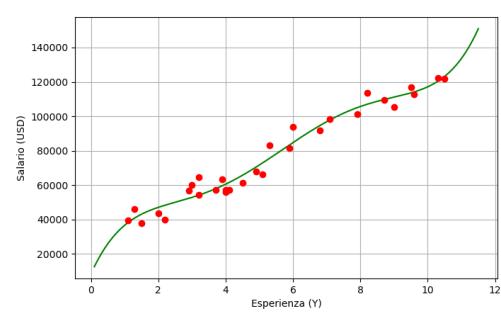


Figure 4: d=5

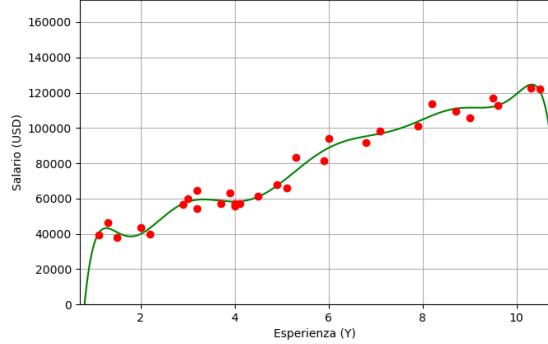


Figure 5: $d=10$

1.3 House Price

Lavoriamo adesso su un altro dataset, variando sempre il grado (d) del polinomio approssimante. Il dataset che usiamo adesso avrebbe 8 colonne, riduciamole a due e poi visualizziamo i grafici come abbiamo fatto prima.

```
dataSet2=pd.read_csv("dataSets/HousePrice.csv")
dataSet2.pop("Num_Bedrooms")
dataSet2.pop("Num_Bathrooms")
dataSet2.pop("Year_Built")
dataSet2.pop("Lot_Size")
dataSet2.pop("Garage_Size")
dataSet2.pop("Neighborhood_Quality")
xRaw=dataSet2["Square_Footage"].values
yRaw=dataSet2["House_Price"].values
d=3
X=vandermonde(xRaw,d)
alphaSVD=SVD(X,yRaw,d)
xMin=xRaw.min()
xMax=xRaw.max()
xx=np.linspace(xMin-1, xMax+1,10000)
yy=f(xx,alphaSVD)
```

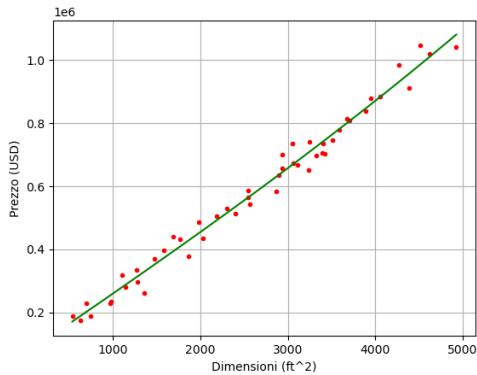


Figure 6: $d=3$

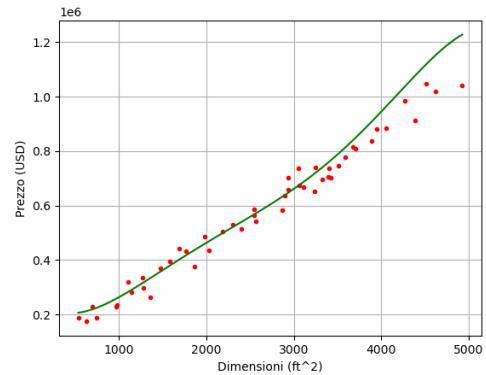


Figure 7: $d=5$

2 Regressione lineare multipla

Se lavoriamo con dataset con più di due colonne non possiamo utilizzare le funzioni scritte in precedenza, ci conviene utilizzare le funzioni della libreria python sklearn(1). Vediamo come fare analizzando un dataframe dedicato alle spese e rendite dell'advertising di quattro colonne: TV, radio, giornali, vendite le cui unità di misura sono gli USD.

```
dataSet=pd.read_csv("dataSets/Advertising.csv")
print(f"Head dataset: \n{dataSet.head()}")
print(f"Info dataset: \n {dataSet.info()}")
print(f"Descrizione dataset:\n{dataSet.describe()}")
xRaw=dataSet.drop("sales", axis=1)
yRaw=dataSet["sales"]
print(f"Numero di righe:{yRaw.shape[0]}")
print(f"Numero di features:{xRaw.shape[1]}\n")
xTrain, xTest, yTrain, yTest = train_test_split(xRaw, yRaw, test_size=0.2, random_state=89)
model = LinearRegression()
model.fit(xTrain, yTrain)
yPred = model.predict(xTest)
print("Mean Squared Error:", mean_squared_error(yTest, yPred))
print("R-squared:", r2_score(yTest, yPred))
print("\nEliminiamo la feature TV:")
xRaw=dataSet.drop(["sales","TV"], axis=1)
yRaw=dataSet["sales"]
xTrain, xTest, yTrain, yTest = train_test_split(xRaw, yRaw, test_size=0.2, random_state=89)
model = LinearRegression()
model.fit(xTrain, yTrain)
yPred = model.predict(xTest)
print("Mean Squared Error:", mean_squared_error(yTest, yPred))
print("R-squared:", r2_score(yTest, yPred))
print("\nEliminiamo la feature RADIO:")
xRaw=dataSet.drop(["sales","radio"], axis=1)
yRaw=dataSet["sales"]
xTrain, xTest, yTrain, yTest = train_test_split(xRaw, yRaw, test_size=0.2, random_state=89)
model = LinearRegression()
model.fit(xTrain, yTrain)
yPred = model.predict(xTest)
print("Mean Squared Error:", mean_squared_error(yTest, yPred))
print("R-squared:", r2_score(yTest, yPred))
print("\nEliminiamo la feature NEWSPAPER:")
xRaw=dataSet.drop(["sales","newspaper"], axis=1)
yRaw=dataSet["sales"]
xTrain, xTest, yTrain, yTest = train_test_split(xRaw, yRaw, test_size=0.2, random_state=89)
model = LinearRegression()
model.fit(xTrain, yTrain)
yPred = model.predict(xTest)
print("Mean Squared Error:", mean_squared_error(yTest, yPred))
print("R-squared:", r2_score(yTest, yPred))
```

Nella pagina successiva è riportato l'output del codice. Osserviamo da MSE e R² che la feature meno importante è NEWSPAPER, mentre il mezzo pubblicitario che ha il maggiore ritorno su investimento è la TELEVISIONE.

(1) Le funzioni di sklearn utilizzate sono:

- sklearn.linear_model.LinearRegression
- sklearn.model_selection.train_test_split
- sklearn.metrics.mean_squared_error
- sklearn.metrics.r2_score

Output dello snippet precedente:

```
Head dataset:
   TV  radio  newspaper  sales
0  230.1    37.8       69.2   22.1
1   44.5    39.3       45.1   10.4
2   17.2    45.9       69.3    9.3
3  151.5    41.3       58.5   18.5
4  180.8    10.8       58.4   12.9
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   TV          200 non-null    float64
 1   radio        200 non-null    float64
 2   newspaper    200 non-null    float64
 3   sales        200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
Info dataset:
None
Descrizione dataset:
   TV      radio  newspaper  sales
count  200.000000  200.000000  200.000000  200.000000
mean   147.042500  23.264000  30.554000  14.022500
std    85.854236  14.846809  21.778621  5.217457
min    0.700000  0.000000  0.300000  1.600000
25%   74.375000  9.975000  12.750000  10.375000
50%   149.750000  22.900000  25.750000  12.900000
75%   218.825000  36.525000  45.100000  17.400000
max   296.400000  49.600000  114.000000 27.000000
Numero di righe:200
Numero di features:3

Mean Squared Error: 3.5238438330657615
R-squared: 0.876306336753169

Eliminiamo la feature TV:
Mean Squared Error: 20.726196806168318
R-squared: 0.27247082175622517

Eliminiamo la feature RADIO:
Mean Squared Error: 7.534020295894862
R-squared: 0.7355414673514513

Eliminiamo la feature NEWSPAPER:
Mean Squared Error: 3.4050352713381216
R-squared: 0.880476744671727
```