Proiect 1
Predicția datelor pentru
Rata Criminalității din America
1990-2022

Nuță Leonard-Florian
Facultatea de Automatică și Calculatoare
Grupa 333AA

Descriere proiect

Setul de date este obținut dintr-un fișier CSV folosind biblioteca Pandas în Python. În acest context, datele referitoare la rata criminalității în America în perioada 1990-2022 (31.12.1990 – 31.12.2022) sunt organizate în două coloane: "Year" (Anul) și "CrimeRate" (Rata criminalității). Aceste date sunt apoi folosite pentru a construi un model de regresie liniară. Datele sunt în număr de **33**.

Algoritmul utilizat:

1. **Încărcarea și organizarea datelor:**
    - Datele sunt încărcate dintr-un fișier CSV folosind biblioteca pandas. Coloanele "Year" și "CrimeRate" sunt extrase pentru a fi utilizate în analiză. Fișierul a fost creat de către mine care am aflat informațiile ratelor de pe Internet.

2. **Preprocesarea datelor (opțional):**
    - Datele pot fi prelucrate, de exemplu, standardizând anii și ratele criminalității. În acest exemplu, am utilizat StandardScaler din scikit-learn pentru a standardiza anii.

3. **Divizarea setului de date:**
    - Setul de date este împărțit în trei părți: un set de antrenare (60%), un set de validare (20%), și un set de testare (20%). Această divizare ajută la evaluarea performanței modelului pe date necunoscute. Datele (ani și rate ale criminalității) sunt împărțite cu ajutorul funcției **train_test_split** din Scikit-learn. Părțile importante sunt **X_train**, **y_train** pentru antrenare, **X_valid**, **y_valid** pentru validare, și **X_test**, **y_test** pentru testare.

4. **Antrenarea modelului:**
    - Se utilizează o regresie liniară din scikit-learn pentru a antrena modelul pe setul de antrenare. În acest exemplu, nu s-au aplicat tehnici de regularizare, cum ar fi Lasso sau Ridge. Se va aplica Ridge ca scop al micșorării erorii medii pătratice.

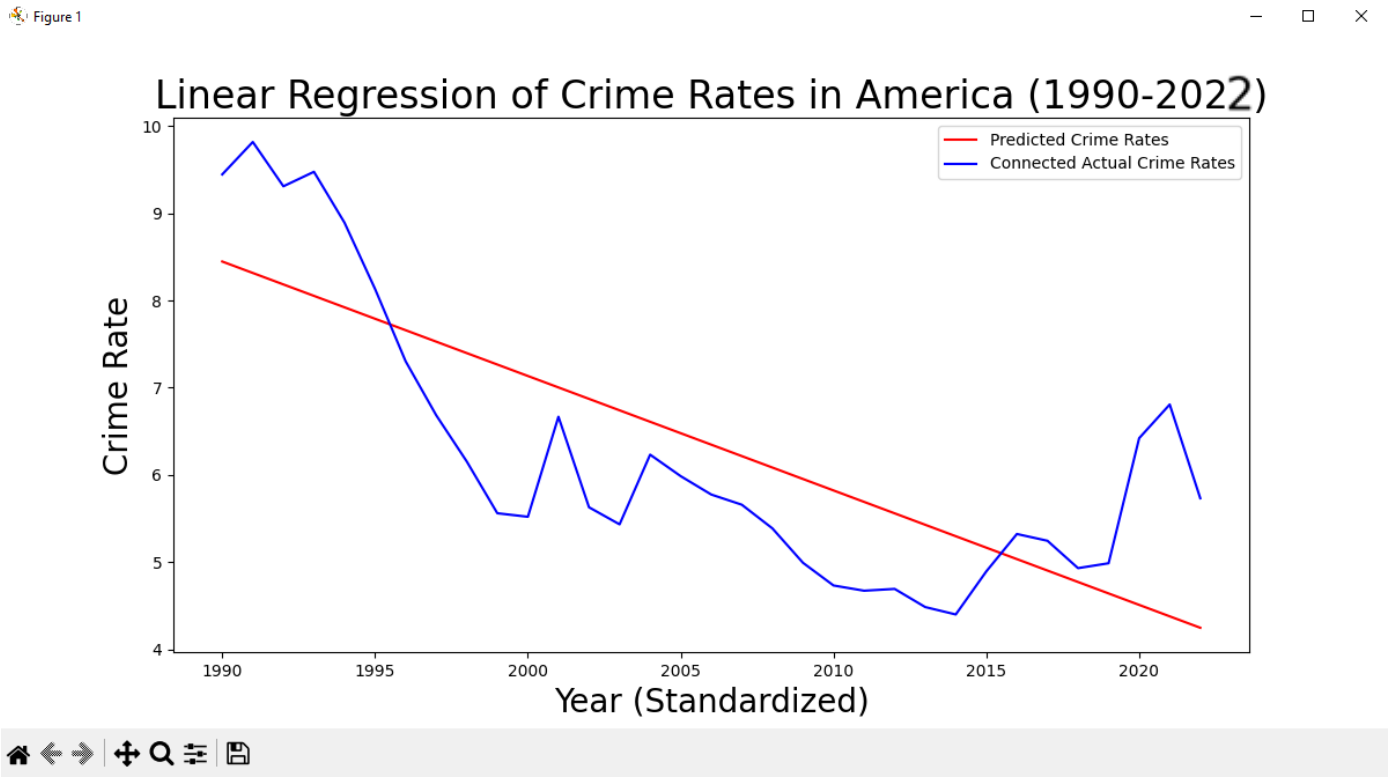
5. **Evaluarea modelului:**
    - Performanța modelului este evaluată pe setul de validare și pe setul de testare folosind eroarea medie pătratică (Mean Squared Error).

6. **Vizualizarea rezultatelor:**
    - Rezultatele sunt vizualizate cu ajutorul bibliotecii Matplotlib. Se afișează o linie roșie care prezice ratele criminalității pe întregul interval de ani și o linie albastră care conectează punctele reale ale ratei criminalității.

Secvență de cod și rezultate

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
# Load crime rate data from CSV
crime_data = pd.read_csv('crime_data.csv')  # Replace 'crime_data.csv' with your actual file name
years = crime_data['Year'].values.reshape(-1, 1)
crime_rates = crime_data['CrimeRate'].values.reshape(-1, 1)
# Standardize the features (optional)
scaler = StandardScaler()
years_scaled = scaler.fit_transform(years)
# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(years_scaled, crime_rates, test_size=0.4,
random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
# Create and train a linear regression model
crime_model = LinearRegression()
crime_model.fit(X_train, y_train)
# Extend the X_test range to cover all years and scale it
X_test_full_range = np.arange(min(years), max(years) + 1).reshape(-1, 1)
X_test_full_range_scaled = scaler.transform(X_test_full_range)
# Make predictions on the full range of years
crime_predictions = crime_model.predict(X_test_full_range_scaled)
# Evaluate the model on the validation set
mse_valid = mean_squared_error(y_valid, crime_model.predict(X_valid))
print(f'Mean Squared Error on Validation Set: {mse_valid}')
# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, crime_model.predict(X_test))
print(f'Mean Squared Error on Test Set: {mse_test}')
# Plot only the line connecting actual crime rates without individual dots
plt.figure(figsize=(12, 6))
plt.plot(X_test_full_range, crime_predictions, label='Predicted Crime Rates', color='red')
plt.plot(years, crime_rates, 'o-', color='blue', markersize=0, label='Connected Actual Crime
Rates')
plt.xlabel('Year (Standardized)', fontsize=20)
plt.ylabel('Crime Rate', fontsize=20)
plt.title('Linear Regression of Crime Rates in America (1990-2022)', fontsize=24)
plt.legend()
plt.show()
```

Secvența de cod utilizată genereaza următorul rezultat.



De asemenea se afișează și Eroarea Medie Pătratică (RMS Error) pentru subsetul de Validare și pentru cel de Testare.

```
Mean Squared Error on Validation Set: 0.9598454391801946
Mean Squared Error on Test Set: 1.6989667389979821

Process finished with exit code 0
```

Regresia liniară este potrivită pentru scenarii în care există o relație liniară între caracteristici și variabila țintă. Dacă relația este mai complexă, pot fi luate în considerare alți algoritmi, cum ar fi regresia polinomială sau modele de învățare automată mai avansate. Cu toate acestea, pentru o relație liniară simplă, regresia liniară este adesea eficientă și ușor de interpretat.

Schimbări aduse algoritmului general

1. Feature Scaling

Pentru unii algoritmi de învățare automată, scalarea caracteristicilor poate îmbunătăți viteza de convergență și performanța. Totuși, regresia liniară nu este deosebit de sensibilă la scalarea caracteristicilor. In codul prezentat mai sus am aplicat această tehnică.

2. Ridge Regression

Scop: Minimizarea funcției de cost adăugând un termen de penalizare care este proporțional cu suma pătratelor coeficienților.

Funcția Obiectiv**:** Se minimizează $J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} \theta_i^2$

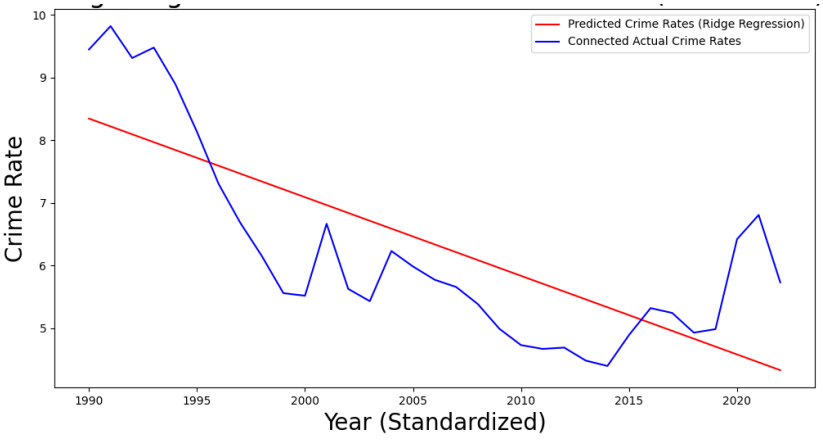
- θ sunt coeficienții modelului de regresie.
- MSE reprezintă Mean Squared Error, iar α este un parametru de reglare care controlează nivelul de regularizare.
- Efecte: Ridge Regression tinde să păstreze toți predictorii în model, dar îi penalizează în funcție de magnitudinea lor. Aceasta înseamnă că coeficienții nu sunt aduși exact la zero.

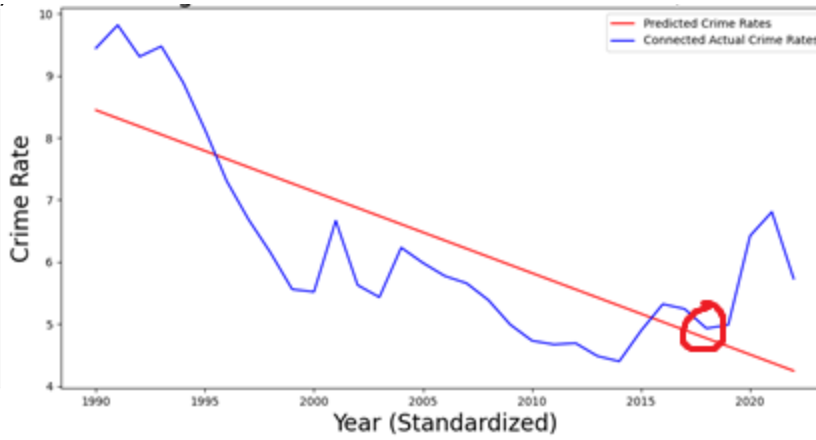
Ridge Regression (aplicat pe codul anterior)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Load crime rate data from CSV
crime_data = pd.read_csv('crime_data.csv')  # Replace 'crime_data.csv' with your actual file name
years = crime_data['Year'].values.reshape(-1, 1)
crime_rates = crime_data['CrimeRate'].values.reshape(-1, 1)

# Standardize the features (optional)
scaler = StandardScaler()
years_scaled = scaler.fit_transform(years)
```

```
# Split the data into training, validation, and test sets



X_train, X_temp, y_train, y_temp = train_test_split(years_scaled, crime_rates, test_size=0.4, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Create and train a Ridge regression model
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Extend the X_test range to cover all years and scale it
X_test_full_range = np.arange(min(years), max(years) + 1).reshape(-1, 1)
X_test_full_range_scaled = scaler.transform(X_test_full_range)

# Make predictions on the full range of years
crime_predictions = ridge_model.predict(X_test_full_range_scaled)

# Evaluate the model on the validation set
mse_valid = mean_squared_error(y_valid, ridge_model.predict(X_valid))
print(f'Mean Squared Error on Validation Set: {mse_valid}')

# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, ridge_model.predict(X_test))
print(f'Mean Squared Error on Test Set: {mse_test}')

# Plot only the line connecting actual crime rates without individual dots
plt.figure(figsize=(12, 6))
plt.plot(X_test_full_range, crime_predictions, label='Predicted Crime Rates (Ridge Regression)', color='red')
plt.plot(years, crime_rates, 'o-', color='blue', markersize=0, label='Connected Actual Crime Rates')
plt.xlabel('Year (Standardized)', fontsize=20)
plt.ylabel('Crime Rate', fontsize=20)
plt.title('Ridge Regression of Crime Rates in America (1990-2022)', fontsize=24)
plt.legend()
plt.show()
```

Rezultate comparate






```
Mean Squared Error on Validation Set: 0.9597590818924814
Mean Squared Error on Test Set: 1.642929608714052

Process finished with exit code 0
```

```
Mean Squared Error on Validation Set: 0.9598454391801946
Mean Squared Error on Test Set: 1.6989667389979821

Process finished with exit code 0
```

Ridge fără Ridge

Se observă o scădere a MSE in cazul aplicării Regresiei Ridge.

Biblioteci Python utilizate:

- **numpy**: Pentru manipularea și lucrul cu matrice și vectori în contextul datelor numerice.
- **pandas**: Pentru manipularea datelor tabulare și încărcarea datelor din fișiere CSV.
- **scikit-learn**: Pentru implementarea algoritmilor de învățare automată, în acest caz, pentru regresia liniară și divizarea setului de date.
- **matplotlib**: Pentru vizualizarea rezultatelor prin intermediul graficelor.

Rezultate obținute:

- Modelul de regresie liniară este antrenat pe setul de antrenare și evaluat pe setul de validare și testare.
- Rezultatele sunt vizualizate într-un grafic, care include linia roșie pentru ratele criminalității prezise și linia albastră pentru ratele reale conectate.
- Performanța modelului este evaluată prin intermediul mean squared error (MSE) pe seturile de validare și testare.

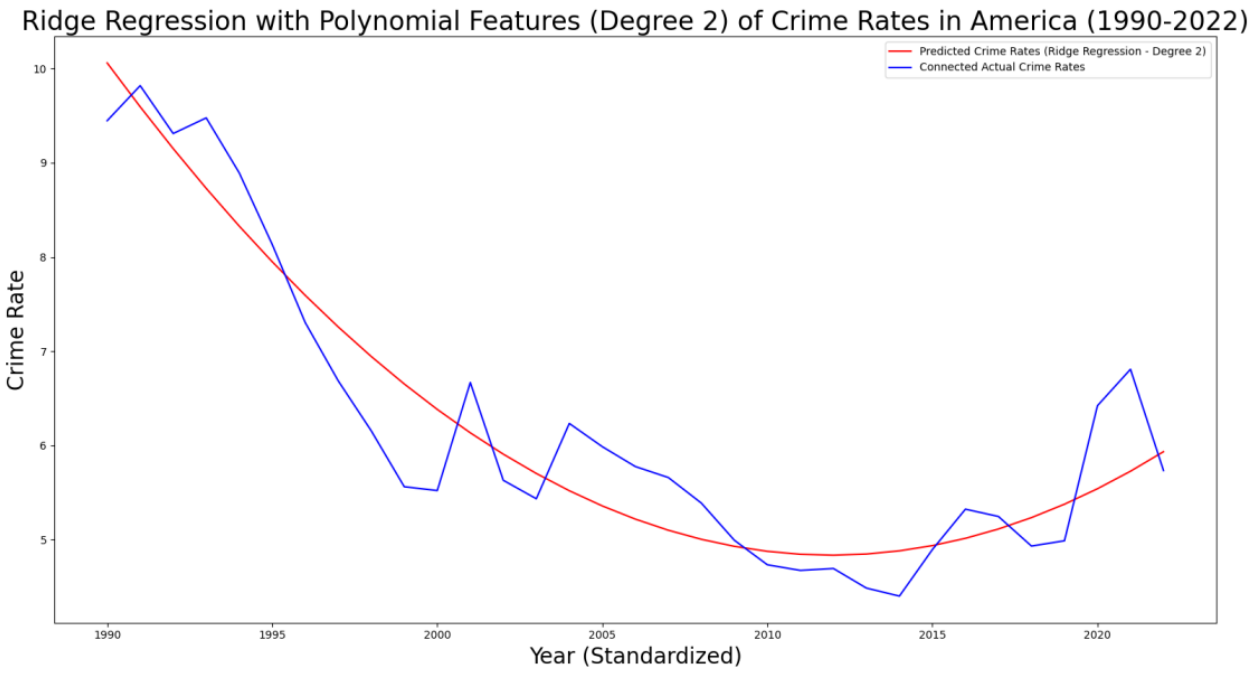
Alți algoritmi aduși în discuție

Regresia Polinomială

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Load crime rate data from CSV
crime_data = pd.read_csv('crime_data.csv')  # Replace 'crime_data.csv' with your actual file name
years = crime_data['Year'].values.reshape(-1, 1)
crime_rates = crime_data['CrimeRate'].values.reshape(-1, 1)

# Standardize the features (optional)
scaler = StandardScaler()
years_scaled = scaler.fit_transform(years)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(years_scaled, crime_rates, test_size=0.4, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Create polynomial features
degree = 2  # You can adjust the degree of the polynomial
poly_features = PolynomialFeatures(degree=degree, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)
X_valid_poly = poly_features.transform(X_valid)
X_test_poly = poly_features.transform(X_test)

# Create and train a Ridge regression model
ridge_model = Ridge(alpha=1.0)  # You can adjust the alpha parameter for different levels of regularization
ridge_model.fit(X_train_poly, y_train)
```

```
# Extend the X_test range to cover all years and scale it
X_test_full_range = np.arange(min(years), max(years) + 1).reshape(-1, 1)
X_test_full_range_scaled = scaler.transform(X_test_full_range)
X_test_full_range_poly = poly_features.transform(X_test_full_range_scaled)

# Make predictions on the full range of years
crime_predictions = ridge_model.predict(X_test_full_range_poly)

# Evaluate the model on the validation set
mse_valid = mean_squared_error(y_valid, ridge_model.predict(X_valid_poly))
print(f'Mean Squared Error on Validation Set: {mse_valid}')

# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, ridge_model.predict(X_test_poly))
print(f'Mean Squared Error on Test Set: {mse_test}')

# Plot only the line connecting actual crime rates without individual dots
plt.figure(figsize=(12, 6))
plt.plot(X_test_full_range, crime_predictions, label=f'Predicted Crime Rates (Ridge Regression - Degree {degree})', color='red')
plt.plot(years, crime_rates, 'o-', color='blue', markersize=0, label='Connected Actual Crime Rates')
plt.xlabel('Year (Standardized)', fontsize=20)
plt.ylabel('Crime Rate', fontsize=20)
plt.title(f'Ridge Regression with Polynomial Features (Degree {degree}) of Crime Rates in America (1990-2022)', fontsize=24)
plt.legend()
plt.show()
```

Rezultat



Observăm o reducere semnificativă a **Erorii Medii Pătratice.**

```
Mean Squared Error on Validation Set: 0.2512109713962271
Mean Squared Error on Test Set: 0.46556796075048407
```

Rezultatul este în relație în special cu numele algoritmului utilizat, predicția fiind de forma unei parabole (gradul este luat ca 2), așa cum se poate observa în graficul de mai sus.

MSE pe Setul de Validare: 0.2512:

- Un MSE mai mic indică faptul că, în medie, predicțiile modelului sunt mai aproape de valorile reale din setul de validare.
- MSE relativ scăzut sugerează că modelul de regresie polinomial se potrivește bine datelor de validare, capturând modelele subiacente în relația dintre caracteristici (ani) și variabila țintă (ratele de criminalitate).

MSE pe Setul de Testare: 0.4656:

- Un MSE mai mare pe setul de testare în comparație cu setul de validare poate indica faptul că modelul este mai puțin eficient în generalizarea către date noi, nevăzute anterior.
- Este esențial să monitorizăm diferența dintre valorile MSE pe setul de validare și cel de testare. O creștere semnificativă a MSE pe setul de testare ar putea sugera supradaaptarea, unde modelul devine prea specific pentru datele de antrenare.

Despre Regresia Polinomială:

- Regresia polinomială este un algoritm flexibil care poate captura relații neliniare între caracteristici și variabila țintă.
- Extinde **regresia liniară** prin introducerea de caracteristici polinomiale, permițând modelului să se potrivească modelelor de date mai complexe.
- Alegerea gradului polinomial este **crucială**. Un grad mai mare poate duce la **supraadaptare**, capturând zgomotul din datele de antrenare, în timp ce un grad mai mic poate duce la **subadapatare**, pierzând modele importante.

De ce s-a ales gradul 2 în acest caz ?

Simplicitate: Gradele polinomiale mai mici sunt adesea preferate din motive de simplicitate, mai ales dacă datele nu prezintă relații foarte complexe.

Evitarea Overfitting: Un grad polinomial prea mare poate duce la supradaptare (overfitting), adică modelul se adaptează prea bine la zgomotul din datele de antrenare și poate avea o performanță slabă pe datele noi.

Prin utilizarea unui grad polinomial de 2, se încearcă obținerea unui echilibru între capacitatea modelului de a se potrivi datelor de antrenare și capacitatea de a generaliza la noi date, evitând totuși o complexitate prea mare.

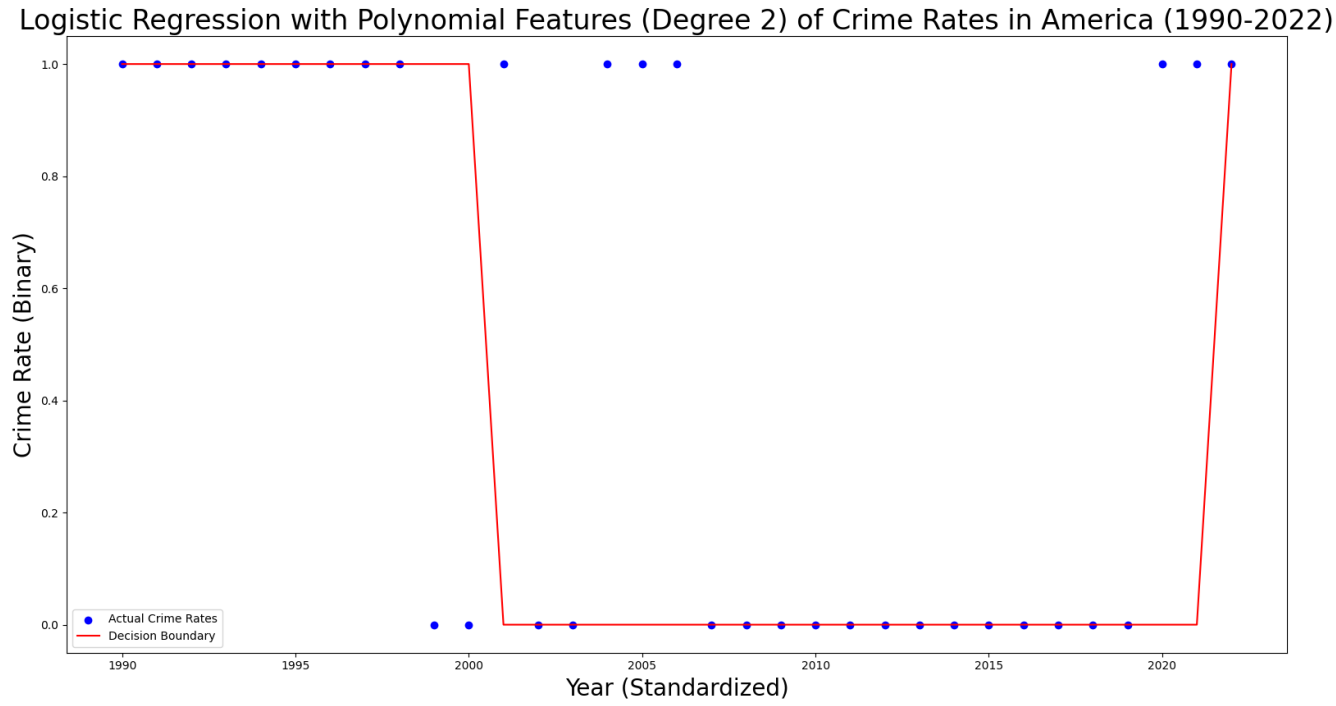
Concluzie despre algoritm

- MSE scăzut pe setul de validare sugerează că modelul de regresie polinomial se comportă bine pe datele de antrenare, capturând tendințele subiacente.

- MSE mai mare pe setul de testare sugerează preocupări cu privire la capacitatea modelului de a generaliza la date noi. Este esențial să evaluăm dacă modelul suferă de supraadaptare și dacă sunt necesare ajustări, cum ar fi reglarea gradului polinomial, pentru o generalizare mai bună.

Regresia Logistică

În cazul regresiei logistice a trebuit să modificăm putin cerințele problemei impuse, și anume să prezicem dacă ratele criminalității din fiecare an sunt ridicate sau scăzute, așadar nu vom compara acest algoritm cu ceilalți.



Set de validare

- Precizie: 1.0
- Recall: 0.6667
- Scor F1: 0.8

Set de test

- Precizie: 0.6667
- Recall: 0.5
- Scor F1: 0.5714

De asemenea, obținem matricile de confuzie împreună cu Erorile Pătratice Medii:

```
Accuracy on Validation Set: 0.8571428571428571
Confusion Matrix on Validation Set:
[[4 0]
 [1 2]]
Accuracy on Test Set: 0.5714285714285714
Confusion Matrix on Test Set:
[[2 1]
 [2 2]]

Process finished with exit code 0
```

Setul de Validare:

- **Acuratețe**: 85.71% - Modelul a prezis corect clasa pentru aproximativ 85.71% din setul de validare.
- **Precizie**: 100% - Toate instanțele prezise ca fiind pozitive au fost efectiv pozitive.
- **Recall**: 66.67% - Modelul a identificat aproximativ 66.67% din instanțele pozitive reale.
- **Scor F1**: 80% - Media aritmetică a preciziei și recall-ului este 80%.

Setul de Testare:

- **Acuratețe**: 57.14% - Modelul a prezis corect clasa pentru aproximativ 57.14% din setul de testare.
- **Precizie**: 66.67% - Două treimi dintre instanțele prezise ca fiind pozitive au fost efectiv pozitive.
- **Recall**: 50% - Modelul a identificat 50% din instanțele pozitive reale.
- **Scor F1**: 57.14% - Media aritmetică a preciziei și recall-ului este 57.14%.

Concluzie despre algoritm:

- Modelul se comportă bine în ceea ce privește precizia atât pe setul de validare, cât și pe cel de testare, indicând rate reduse de fals pozitiv.
- Cu toate acestea, recall-ul este relativ mai mic, în special pe setul de testare, sugerând că modelul ratează o parte semnificativă a instanțelor pozitive.
- Scorul F1 oferă un echilibru între precizie și recall, iar valorile sale indică o performanță moderată a modelului în ansamblu.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load crime rate data from CSV
crime_data = pd.read_csv('crime_data.csv')  # Replace 'crime_data.csv' with your actual file name
years = crime_data['Year'].values.reshape(-1, 1)
crime_rates = crime_data['CrimeRate'].values

# Create a binary target variable based on some threshold (e.g., median)
threshold = np.median(crime_rates)
crime_labels = (crime_rates > threshold).astype(int)

# Standardize the features (optional)
scaler = StandardScaler()
years_scaled = scaler.fit_transform(years)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(years_scaled, crime_labels, test_size=0.4, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Create polynomial features
degree = 2  # You can adjust the degree of the polynomial
poly_features = PolynomialFeatures(degree=degree, include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)
X_valid_poly = poly_features.transform(X_valid)
X_test_poly = poly_features.transform(X_test)

# Create and train a Logistic regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train_poly, y_train)

# Extend the X_test range to cover all years and scale it
X_test_full_range = np.arange(min(years), max(years) + 1).reshape(-1, 1)
```

```
X_test_full_range_scaled = scaler.transform(X_test_full_range)

# Plot decision boundary
plt.figure(figsize=(12, 6))
plt.scatter(X_test_full_range, crime_labels, color='blue', label='Actual Crime Rates')
plt.xlabel('Year (Standardized)', fontsize=20)
plt.ylabel('Crime Rate (Binary)', fontsize=20)
plt.title(f'Logistic Regression with Polynomial Features (Degree {degree}) of Crime Rates in America (1990-2022)', fontsize=24)

# Plot decision boundary
X_range_poly = poly_features.transform(X_test_full_range_scaled)
decision_boundary = logistic_model.predict(X_range_poly)
plt.plot(X_test_full_range, decision_boundary, label='Decision Boundary', color='red')

plt.legend()
plt.show()

# Evaluate the model on the validation set
accuracy_valid = accuracy_score(y_valid, logistic_model.predict(X_valid_poly))
confusion_matrix_valid = confusion_matrix(y_valid, logistic_model.predict(X_valid_poly))
print(f'Accuracy on Validation Set: {accuracy_valid}')
print(f'Confusion Matrix on Validation Set:\n{confusion_matrix_valid}')

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, logistic_model.predict(X_test_poly))
confusion_matrix_test = confusion_matrix(y_test, logistic_model.predict(X_test_poly))
print(f'Accuracy on Test Set: {accuracy_test}')
print(f'Confusion Matrix on Test Set:\n{confusion_matrix_test}')
```

Aspecte de Luat în Considerare:

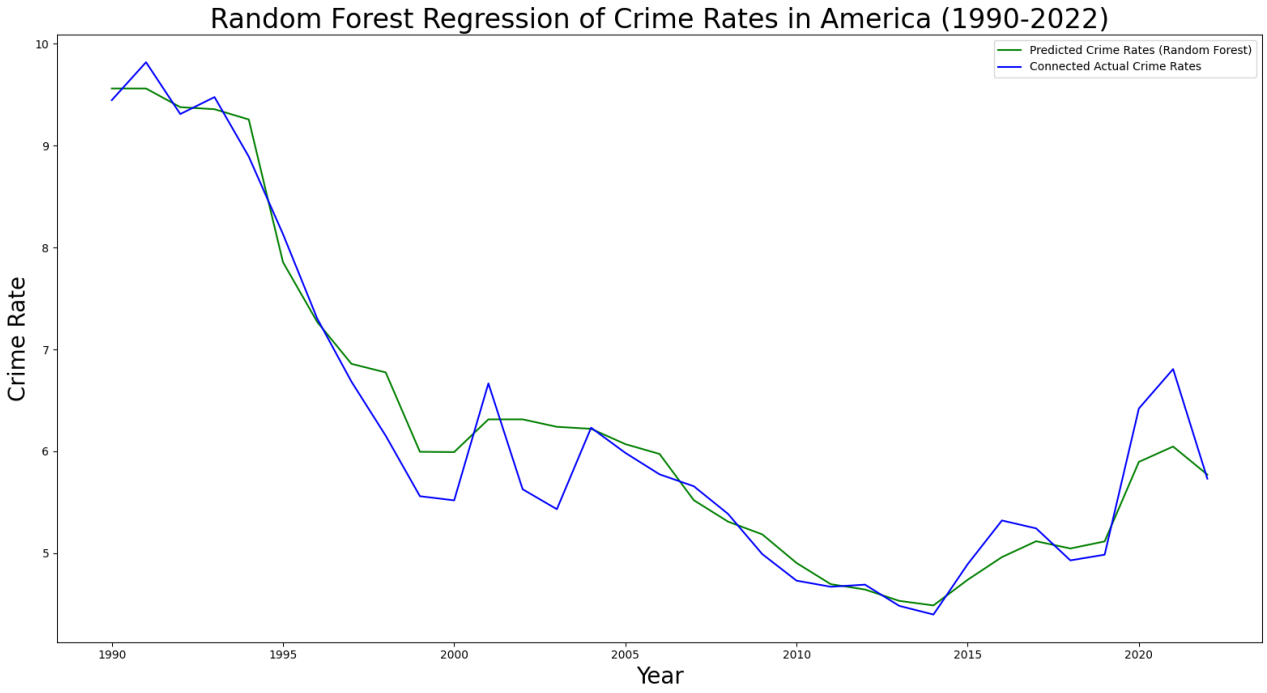
Alegerea pragului pentru binarizarea ratelor de criminalitate poate afecta rezultatele. În codul furnizat mai sus, folosim mediana ca prag, care este o abordare.

Random Forest

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Load crime rate data from CSV
crime_data = pd.read_csv('crime_data.csv')  # Replace 'crime_data.csv' with your actual file name
years = crime_data['Year'].values.reshape(-1, 1)
crime_rates = crime_data['CrimeRate'].values.reshape(-1, 1)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(years, crime_rates, test_size=0.4, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Reshape y_train and y_test to avoid DataConversionWarning
y_train = y_train.ravel()
y_test = y_test.ravel()

# Create and train a Random Forest regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust the number of estimators
rf_model.fit(X_train, y_train)

# Extend the X_test range to cover all years
X_test_full_range = np.arange(min(years), max(years) + 1).reshape(-1, 1)

# Make predictions on the full range of years
crime_predictions = rf_model.predict(X_test_full_range)

# Evaluate the model on the validation set
mse_valid = mean_squared_error(y_valid, rf_model.predict(X_valid))
print(f'Mean Squared Error on Validation Set: {mse_valid}')

# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, rf_model.predict(X_test))
```

```
print(f'Mean Squared Error on Test Set: {mse_test}')

# Plot only the line connecting actual crime rates without individual dots
plt.figure(figsize=(12, 6))
plt.plot(X_test_full_range, crime_predictions, label='Predicted Crime Rates (Random Forest)', color='green')

# Connect the actual crime rates with a line and make dots disappear
plt.plot(years, crime_rates, 'o-', color='blue', markersize=0, label='Connected Actual Crime Rates')

plt.xlabel('Year', fontsize=20)
plt.ylabel('Crime Rate', fontsize=20)
plt.title('Random Forest Regression of Crime Rates in America (1990-2022)', fontsize=24)
plt.legend()
plt.show()
```

Rezultate



Rezultatele se dovedesc a fi de-a dreptul impresionante folosind **Random Forest**, prin urmare **Regresia Liniară** va fi comparată cu **Random Forest**.

```
Mean Squared Error on Validation Set: 0.17669633378642655
Mean Squared Error on Test Set: 0.21385679134885585
```

Random Forest:

- MSE pe Setul de Validare: 0.1767
- MSE pe Setul de Test: 0.2139

Regresia Liniară:

- MSE pe Setul de Validare: 0.9598
- MSE pe Setul de Test: 1.6429

Pentru ambele seturi, de **validare** și **test**, metoda **Random Forest** depășește semnificativ **Regresia Liniară** în ceea ce privește **MSE**. Valori **mai mici** ale **MSE** indică faptul că modelul **Random Forest** oferă **predicții mai precise** în comparație cu **Regresia Liniară**.
Algoritmul **Random Forest** este o metodă de învățare de ansamblu care construiește mai mulți **arbori de decizie** și combină predicțiile acestora. Este cunoscut pentru capacitatea sa de a captura relații complexe în date și de a gestiona **modele non-liniare**.

Comparație Random Forest cu Regresia Liniară:

Avantaje Random Forest:

- Random Forest este un algoritm puternic care poate captura relații complexe, fiind potrivit pentru seturile de date cu modele intricate.
- Este mai puțin predispus la supraadaptare în comparație cu un singur arbore de decizie datorită naturii sale de ansamblu.
- Random Forest nu se bazează pe asumpția liniarității, făcându-l mai flexibil în capturarea modelelor diverse ale datelor.

Provocări în Regresia Liniară:

- Regresia Liniară presupune o relație liniară între caracteristici și variabila țintă. Dacă relația subiacentă este non-liniară, cum se întâmplă adesea, Regresia Liniară poate să nu o captureze eficient.
- Performanța modelului poate fi limitată în fața modelelor complexe sau non-liniare ale datelor.

Analiză algoritm – puncte cheie

1. **Arbori de Decizie Individuali:**

Un arbore de decizie poate fi reprezentat ca o serie de divizări binare bazate pe caracteristici, conducând la noduri terminale (frunze) cu valori prezise. Pentru un singur arbore de decizie, să notăm:

- X ca fiind caracteristicile de intrare.
- T ca fiind arborele de decizie.
- y_i ca fiind rezultatul prezis pentru instanța i.

Matematic, predicția y_i pentru o instanță X_i folosind un arbore de decizie T poate fi reprezentată ca:

$$y_i = f(X_i, T)$$

2. **Ansamblul de Arbori de Decizie (Random Forest):**

În Random Forest, sunt creați mai mulți arbori de decizie antrenați pe diferite submulțimi ale datelor de antrenare (bootstrapping) și folosind submulțimi aleatoare de caracteristici la fiecare divizare. Fie T_1, T_2, ... , T_n care reprezintă arborii de decizie individuali din ansamblu.

Predicția generală pentru o instanță X_i într-o problemă de regresie este media predicțiilor din toți arborii:

$$\hat{y}_i = \frac{1}{n} \sum_{j=1}^{n} f(X_i, T_j)$$

3. **Procesul de Antrenare:**

Random Forest implică antrenarea fiecărui arbore de decizie în mod independent și apoi agregarea predicțiilor lor. Matematic, procesul de antrenare a unui Random Forest implică optimizarea parametrilor arborilor individuali pe baza datelor de antrenare.

Fie D setul de date de antrenare. Scopul este să găsească un set de arbori de decizie T_1, T_2, ... , T_n care minimizează eroarea medie pătratică totală (MSE).

$$\text{MIN} \ \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \hat{y}_i)^2$$

unde (y_i ^) este rezultatul prezis pentru instanța i pe baza ansamblului de arbori de decizie.

4. **Hiperparametri**

Random Forest are hiperparametri care pot fi ajustați, cum ar fi numărul de arbori în ansamblu n, adâncimea maximă a fiecărui arbore și numărul de caracteristici luate în considerare la fiecare divizare. Ajustarea implică găsirea setului optim de hiperparametri care duce la cea mai bună performanță globală pe datele de validare sau test.

În codul rulat am lucrat cu numărul implicit de arbori furnizat de biblioteca specifică, și anume 100.

Algoritmul Random Forest combină predicțiile mai multor arbori de decizie pentru a crea un model de ansamblu robust și precis. Reprezentarea matematică implică predicțiile individuale ale arborilor de decizie și agregarea acestora în ansamblu. Procesul de antrenare vizează optimizarea parametrilor arborilor individuali pentru a minimiza eroarea generală de predicție.

Biblioteci utilizate

- **LogisticRegression**: Clasă pentru regresia logistică.
- **PolynomialFeatures**: Clasă pentru generarea de caracteristici polinomiale.
- **RandomForestRegressor**: Clasă pentru regresia cu Random Forest.

Concluzie finală

Valorile mai mici ale MSE pentru metoda **Random Forest** indică faptul că aceasta oferă predicții **mai precise** în comparație cu **Regresia Liniară** pentru **setul de date dat**.

Capacitatea **Random Forest** de a gestiona relații **non-liniare** și de a captura modele complexe îl face o opțiune preferabilă atunci când se lucrează cu seturi de date care prezintă astfel de **caracteristici**.