|  |  |  |  |
| --- | --- | --- | --- |
|  | Politechnika Bydgoska im. J. J. Śniadeckich  Wydział Telekomunikacji,  Informatyki i Elektrotechniki | |  |
| **Przedmiot** | Analiza Regresji i Szeregów Czasowych | | |
| **Prowadzący** | dr inż. Damian Ledziński | | |
| **Temat** | Regresja Liniowa | | |
| **Student** | Cezary Tytko | | |
| **Nr lab.** | 2 | **Data wykonania** | 23.05.2024 |
| **Ocena** |  | **Data oddania spr.** |  |

Zgadnie z poleceniem wczytałem dane „california\_housing” i przeprowadziłem eksploracyjną analizę danych z wykorzystaniem narzędzia „ProfileReport”

1. housing = fetch\_california\_housing()

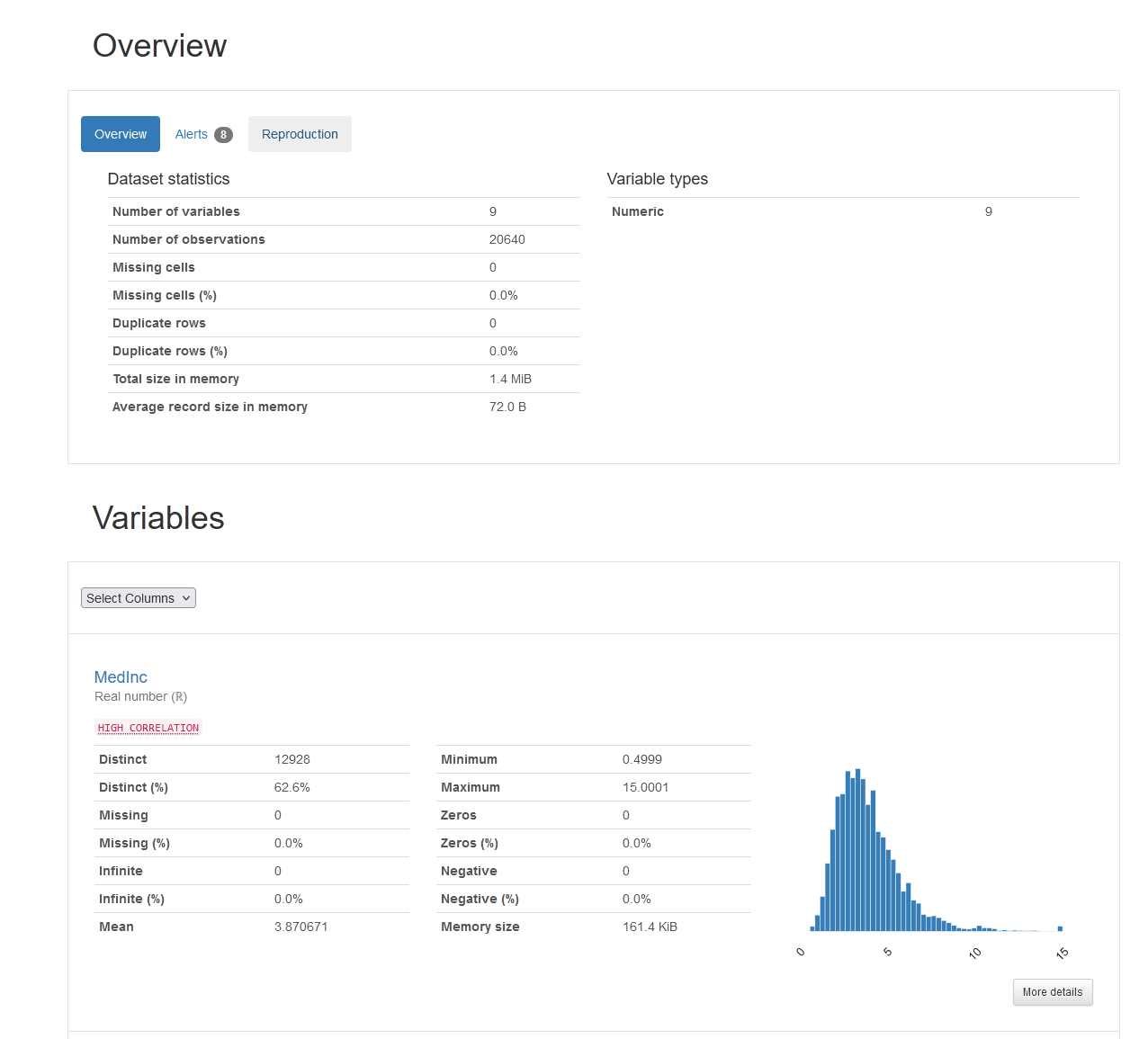
2. housing\_df = pd.DataFrame(data=housing.data, columns=housing.feature\_names)

3. housing\_df['target'] = housing.target

4. profile = ProfileReport(housing\_df, title='California Housing Prices - EDA Report', explorative=True)

5. profile.to\_file("california\_housing\_eda\_report.html")

6.



Następnie używając tych danych wytrenowałem 3 modele regresyjne(LinearRegression, RandomForestRegressor i MLPRegressor):

1. X\_train, X\_test, y\_train, y\_test = train\_test\_split(housing.data, housing.target, test\_size=0.2, random\_state=42)

2. linear\_reg = LinearRegression()

3. decision\_tree\_reg = RandomForestRegressor()

4. neural\_network\_reg = MLPRegressor(hidden\_layer\_sizes=(100, 50), activation='relu', solver='adam', max\_iter=1000)

5.

6. linear\_reg.fit(X\_train, y\_train)

7. decision\_tree\_reg.fit(X\_train, y\_train)

8. neural\_network\_reg.fit(X\_train, y\_train)

9.

10. linear\_reg\_pred = linear\_reg.predict(X\_test)

11. decision\_tree\_reg\_pred = decision\_tree\_reg.predict(X\_test)

12. neural\_network\_reg\_pred = neural\_network\_reg.predict(X\_test)

13.

14. linear\_reg\_mse = mean\_squared\_error(y\_test, linear\_reg\_pred)

15. decision\_tree\_reg\_mse = mean\_squared\_error(y\_test, decision\_tree\_reg\_pred)

16. neural\_network\_reg\_mse = mean\_squared\_error(y\_test, neural\_network\_reg\_pred)

17.

18. print("Mean Squared Error (MSE) dla regresji liniowej:", linear\_reg\_mse)

19. print("Mean Squared Error (MSE) dla drzewa decyzyjnego:", decision\_tree\_reg\_mse)

20. print("Mean Squared Error (MSE) dla modelu opartego na sieciach neuronowych:", neural\_network\_reg\_mse)

21.

I sprawdziłem ich skuteczność na trzech metrykach (MAE, RMSE i R2):

1. linear\_reg\_mae = mean\_absolute\_error(y\_test, linear\_reg\_pred)

2. decision\_tree\_reg\_mae = mean\_absolute\_error(y\_test, decision\_tree\_reg\_pred)

3. neural\_network\_reg\_mae = mean\_absolute\_error(y\_test, neural\_network\_reg\_pred)

4.

5. linear\_reg\_rmse = np.sqrt(mean\_squared\_error(y\_test, linear\_reg\_pred))

6. decision\_tree\_reg\_rmse = np.sqrt(mean\_squared\_error(y\_test, decision\_tree\_reg\_pred))

7. neural\_network\_reg\_rmse = np.sqrt(mean\_squared\_error(y\_test, neural\_network\_reg\_pred))

8.

9. linear\_reg\_r2 = r2\_score(y\_test, linear\_reg\_pred)

10. decision\_tree\_reg\_r2 = r2\_score(y\_test, decision\_tree\_reg\_pred)

11. neural\_network\_reg\_r2 = r2\_score(y\_test, neural\_network\_reg\_pred)

12.

13. print("Mean Absolute Error (MAE):")

14. print("Regresja liniowa:", linear\_reg\_mae)

15. print("Drzewo decyzyjne:", decision\_tree\_reg\_mae)

16. print("Sieć neuronowa:", neural\_network\_reg\_mae)

17.

18. print("\nRoot Mean Squared Error (RMSE):")

19. print("Regresja liniowa:", linear\_reg\_rmse)

20. print("Drzewo decyzyjne:", decision\_tree\_reg\_rmse)

21. print("Sieć neuronowa:", neural\_network\_reg\_rmse)

22.

23. print("\nR^2 score:")

24. print("Regresja liniowa:", linear\_reg\_r2)

25. print("Drzewo decyzyjne:", decision\_tree\_reg\_r2)

26. print("Sieć neuronowa:", neural\_network\_reg\_r2)

27.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Najlepszy okazał się las losowy (opisany jako wariant drzewa decyzyjnego)

Historyczne dane Forex:

Pobrałem historyczne dane dla pary walutowej EUR/PLN jako świeczki godzinowe z okresu 3 lat

EURPLN\_Candlestick\_1\_h\_BID\_01.05.2021-30.04.2024

Dodałem kolumnę z numerem dnia tygodnia i numerem godziny w ciągu dnia (dane przydatne ze względu na zamknięcie rynku w weekend)

1. df = pd.read\_csv('EURPLN\_Candlestick\_1\_h\_BID\_01.05.2021-30.04.2024.csv', parse\_dates=['Gmt time'],

2. date\_parser=lambda x: pd.to\_datetime(x, format='%d.%m.%Y %H:%M:%S.%f'))

3. # Dodaj kolumnę z dniem tygodnia

4. df['Day of Week'] = df['Gmt time'].dt.dayofweek

5. # Dodaj kolumnę z godziną w ciągu dnia

6. df['Hour of Day'] = df['Gmt time'].dt.hour

7. df = df.drop(columns=['Gmt time'])

8.

Specjalnie przygotowaną funkcją podzieliłem dane na zbiory: badany okres – docelowa wartość, funkcja ta przygotowuje dane w formacie przeznaczonym dla uczenia sieci LSTM, dlatego dodatkowo rozwinąłem macierz do wiersza, aby przekazać dane do prostego modelu drzewa decyzyjnego, aby pokazać działanie „m2cgen”, choć lepszym rozwiązaniem byłoby użycie sieci rekurencyjnej.

1. def GetTimeSeriesData(df: pd.DataFrame, target\_col: str, point\_per\_series: int, forward\_index: int, step\_next: int=5):

2. listNew = []

3. listValue = []

4. data\_range = point\_per\_series

5. data\_range\_index = data\_range - 1

6. full\_range = point\_per\_series + forward\_index

7. full\_range\_index = full\_range - 1

8. start = 0

9. print(df.\_\_len\_\_())

10. for j in range(start, df.\_\_len\_\_() - full\_range\_index, step\_next):

11. listTemp = []

12. for k in range(0, data\_range):

13. listTemp.append(df.iloc[k+j][df.columns.difference([target\_col])].tolist())

14. # listTemp.append(df[j + (k - start)])

15. listNew.append(listTemp)

16. listValue.append((df.iloc[full\_range\_index+j][target\_col] - df.iloc[data\_range\_index+j][target\_col]) >= 0)

17. # listValue.append((tab[j + stepForward][targetIndes] >= tab[j][targetIndes]).astype(int))

18. return np.array(listNew), np.array(listValue)

19.

1. Dane wcześniej ustandaryzowałem, choć dla drzewa decyzyjnego nie ma to znaczenia (od kolumny volume nie odjąłem średniej, aby zachować wartości 0 kiedy mamy brak handlu na rynku)

2. print(df.head())

3. columns\_to\_standardize = ['Open', 'High', 'Low', 'Close']

4. df\_standarized = df.copy()

5. # Inicjalizuj StandardScaler

6. scaler = StandardScaler()

7.

8. # Standaryzuj wybrane kolumny

9. df\_standarized[columns\_to\_standardize] = scaler.fit\_transform(df[columns\_to\_standardize])

10.

11. # Dla kolumny 'Volume' podziel wartości przez odchylenie standardowe

12. volume\_std = df['Volume'].std()

13. df\_standarized['Volume'] = df['Volume'] / volume\_std

14.

15. print(df\_standarized.head())

16.

1. time\_series\_data, time\_series\_Value = GetTimeSeriesData(df\_standarized, 'Open', 10, 2, 1)

2. print(time\_series\_data.shape)

3. print(time\_series\_Value.shape)

4.

Pobieramy 10 wierszy jako baza do analizy, kolumną przewidywaną jest „open” dwa wiersze w przód, skok do kolejnej kombinacji następuje co 1 wiersz

Obraz zawierający tekst, Czcionka, pismo odręczne, zrzut ekranu

Opis wygenerowany automatycznie

1. reshaped\_array = time\_series\_data.reshape(time\_series\_data.shape[0], -1)

2. print(reshaped\_array.shape)

3.



1. model = DecisionTreeRegressor(random\_state=42)

2. model.fit(X\_train, y\_train)

3.

4. # Przewidywanie na zbiorze testowym

5. y\_pred = model.predict(X\_test)

6.

7. # Ocena modelu

8. mse = mean\_squared\_error(y\_test, y\_pred)

9. print(f'Mean Squared Error: {mse}')

10.



Wynik nie był dal mnie istotny, ponieważ głównym celem było pokazanie działania biblioteki „m2cgen” w innym przypadku skorzystałbym z sieci neuronowych.

1. java\_code = m2c.export\_to\_java(model)

2.

3. # Zapisanie kodu Java do pliku

4. with open("LinearRegressionModel.java", "w") as f:

5. f.write(java\_code)

6.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Jak widać wszystko zadziałało i otrzymaliśmy wynik w postaci drzewa decyzyjnego zaimplementowanego w java, należy zauważyć że wyniki są w postaci danych po standaryzacji, przy wdrażaniu takiego modelu należy to uwzględnić i proces odwrócić

Wdrażane modelu na JForex:

Jest to stosunkowo proste zadanie jeżeli podejdziemy do tego rozsądnie, najpierw należy się zastanowić jakie dane możemy wyłuskać w trakcie działania strategii, następnie dla określonych danych, pobrać ich wartości historyczne i wytrenować no nich model,. Za pomocą biblioteki m2cgen możemy przenieść model do języka java, następnie zaimportować model na do strategii, wykonywać na jego postawie predykcji, a na podstawie tych predykcji podejmować działania.

Dla pokazania samej implementacji utworzyłem prostu model drzewa decyzyjnego, biorący wartości z 5 ostatnich godzin ceny zamknięcia, który próbuje przewidzieć czy za godzinę będzie wzrost czy spadek, jeżeli ma być wzrost to kupujemy, a jeżeli model przewiduje spadek to zamykamy:

1. import com.dukascopy.api.\*;

2. import java.util.\*;

3.

4. public class MyStrategy implements IStrategy {

5. private IEngine engine;

6. private IConsole console;

7. private IHistory history;

8. private IIndicators indicators;

9. private IContext context;

10. @Configurable("selectedInstrument:")

11. public Instrument instrument = Instrument.EURPLN;

12. private IOrder order;

13. private static final int WINDOW\_SIZE = 5;

14.

15. // Wygenerowany model drzewa decyzyjnego

16. private Model model;

17.

18. @Override

19. public void onStart(IContext context) throws JFException {

20. this.context = context;

21. this.engine = context.getEngine();

22. this.console = context.getConsole();

23. this.history = context.getHistory();

24. this.indicators = context.getIndicators();

25. this.model = new Model();

26. console.getOut().println("Strategy started.");

27. }

28.

29. @Override

30. public void onBar(Instrument instrument, Period period, IBar askBar, IBar bidBar) throws JFException {

31. if (!instrument.equals(this.instrument) || !period.equals(Period.ONE\_HOUR)) {

32. console.getOut().println("Return");

33. return;

34. }

35.

36. // Pobierz ostatnie 6 cen zamknięcia

37. List<Double> closePrices = new ArrayList<>();

38. for (int i = 0; WINDOW\_SIZE + 1 > i; i++) {

39. IBar bar = history.getBar(instrument, Period.ONE\_HOUR, OfferSide.BID, i);

40. closePrices.add(bar.getClose());

41. }

42. Collections.reverse(closePrices);

43.

44. // Oblicz różnice

45. double[] diffs = new double[WINDOW\_SIZE];

46. for (int i = 0; i < WINDOW\_SIZE; i++) {

47. diffs[i] = closePrices.get(i + 1) - closePrices.get(i);

48. }

49.

50. // Przewiduj zmianę na następną godzinę

51. double prediction = model.predict(diffs);

52.

53. // Jeśli przewidywana zmiana jest dodatnia, otwórz transakcję kupna

54. if (prediction > 0) {

55. if (order == null || order.getState() == IOrder.State.CLOSED) {

56. order = engine.submitOrder("Order\_" + System.currentTimeMillis(), instrument, IEngine.OrderCommand.BUY, 0.1);

57. console.getOut().println("Opened BUY order.");

58. }

59. }

60.

61. // Zamknij transakcję, jeśli wartość zacznie spadać

62. if (order != null && order.getState() == IOrder.State.FILLED) {

65. if (prediction < 0) {

66. order.close();

67. console.getOut().println("Closed order due to falling price.");

68. }

69. }

70. }

71.

72. @Override

73. public void onTick(Instrument instrument, ITick tick) throws JFException {

74. // Nie potrzebujemy implementacji dla onTick w tej strategii

75. }

76.

77. @Override

78. public void onMessage(IMessage message) {

79. // Można tutaj obsłużyć różne komunikaty

80. }

81.

82. @Override

83. public void onAccount(IAccount account) {

84. // Można tutaj obsłużyć informacje o koncie

85. }

86.

87. @Override

88. public void onStop() throws JFException {

89. if (order != null && order.getState() == IOrder.State.FILLED) {

90. order.close();

91. }

92. console.getOut().println("Strategy stopped.");

93. }

94. public class Model {

95. public double predict(double[] input) {

96. double var0;

97. if (input[3] <= 0.06116499751806259) {

98. …

99.

Model się nie sprawdził, ale nie to było celem, celem było pokazanie jak wdrożyć taki model, kluczową kwestia jest to że wytrenowany model musi działać na tym samym instrumencie i parametrach, w przeciwnym wypadku wyniki można uznać za losowe, nawet jeśli byłby pozytywne.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie