

Warsztat Badacza danych harmono- gram

| Tydzień | Wykłady | Godz. | Ćwiczenia | Godz. |
|---------|---|--------|---|-------|
| 1 | 1 Wprowadzenie do data science 2 Typy modelowania i danych | 1 1 | Wstęp do ćw., dane i narzędzia do pracy, Python, Anaconda, Jupiter Notebook – wprowadzenie | 1 |
| 2 | 3 Typy regresji | 1 | Regresja liniowa (Python) – demo i praca | 2 |
| 3 | 4 Inne matematyczne podstawy d.s., | 1 | Regresja logistyczna (Python) – demo i praca | 2 |
| 4 | 5 Redukcja wielowymiarowości (PCA, LDA) | 1 | PCA (Python) – demo i praca | 2 |
| 5 | 6 K-NN i SVM | 1 | LDA (Python) – demo i praca | 2 |
| 6 | 7 Koncepcja Bayes’a w d. s. | 1 | K-NN w Python – demo i praca | 2 |
| 7 | 9 Miary oceny modelu – regresja | 1 | SVM dla 2 klas w Python – demo i praca | 2 |
| 8 | 9 Kolokwium 1 | 1 | SVM dla multiclass w R (Jup. Not.)– demo i praca | 2 |
| 9 | 10 Miary oceny modelu – klasyf. | 1 | Naïve Bayes w Python – demo + praca | 2 |
| 10 | 11 Drzewa decyzyjne | 1 | Kolokwium 1 + SVM dla multiclass w Python – praca | 2 |
| 11 | 12 Losowy Las | 1 | Drzewa Decyzyjne (Python)- demo i praca | 2 |
| 12 | | | Losowy Las (Python) - demo i praca | 3 |
| 13 | 13 Wstęp do sieci neuronowych | 1 | Losowy las (Python) - praca | 2 |
| 14 | 14 Wstęp do sieci neuronowych | 1 | ANN (Matlab) – demo | 2 |
| 15 | 15 Kolokwium 2 | 1 | ANN (Matlab) – praca, | 2 |

Losowy las z wizualizacją drzewa decyzyjnego i selekcja zmiennych

DEMO Problem: Baza danych plików demonstracyjnych **vlagunr-Phyto.csv**

Znalezienie parametrów środowiskowych i ich hierarchii (modelu drzewa), które najlepiej przewidują koncentrację **biomasy fitoplanktonu** w wodach Zalewu Wiślanego.

Praca własna: Baza danych plików roboczych (do zmiany) **vlagunr-Cyano.csv**

Zbuduj i przeanalizuj model lasu losowego na podstawie powyższego przykładu.

Rozwiąż problem: Znajdź parametry środowiskowe i ich hierarchię, które najlepiej **przewidują koncentrację biomasy sinic** w wodach Zalewu Wiślanego.

Tworzenie modelu drzewa

Obliczanie dokładności dla zestawów danych testowych i treningowych:

- dla wszystkich zmiennych
- dla 2 najważniejszych zmiennych
- dla najważniejszych zmiennych po 10% redukcji
- dla najważniejszych zmiennych po 3-krotnej losowej walidacji krzyżowej
- dla najważniejszych zmiennych po 3-krotnej gridowej walidacji krzyżowej siatki

Wizualizuj:

- 1) - Model drzewa
- 2) - Diagram ważności funkcji z redukcją mniej ważnych zmiennych
- 3) - Trzy wykresy z relacjami między zmienną celową (task) a 4 najważniejszymi zmiennymi

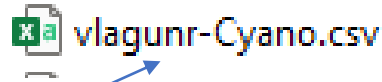
Zapisz skrypt jako pliki html i ipynb. Plik Html i wizualizacja modelu drzewa pokaz oceny

Losowy las z wizualizacją drzewa decyzyjnego i selekcja zmiennych

Otwórz **Webgraphviz**: <http://www.webgraphviz.com/?tab=map>

Regression
analysis in
Random
Forest

1. Otwórz JupiterNotebook
2. Ładuj zestawu danych z katalogu
3. Analizowanie kodu wzorca
4. Zmień zestaw danych na



5. Obliczanie dla zestawów danych testowych i treningowych: a. **całkowity model**, b. **dla dwóch najważniejszych cech**, c. **po usunięciu 10% najślabszych cech**:

1. Mean Absolute Error
2. Mean Absolute Error / Mean Ratio
3. Mean Squared Error
4. Explained Variance Square
5. Mean Absolute Percentage Error (MAPE)
6. $1 - \text{MAPE} = \text{Accuracy}$

Utwórz tabelkę z porównaniem tych miar dla prób testowej i treningowej.

6. Za pomocą **Webgraphviz** wizualizuj drzewo decyzyjne (znajdź kod (drzewo) w folderze Documents, otwórz **Webgraphviz** i wpisz kod do okna. Zapisz jako (w katalogu folderu) . Kod i wykres drzewa przekaz do oceny.

Wizualizuj:

Model drzewa

Diagram ważności funkcji z redukcją mniej ważnych zmiennych

Trzy wykresy z relacjami między zmienną celową (task) a 4 najważniejszymi zmiennymi

Pomiary błędów przewidywania (w regresji)

Absolute Errors

Absolute Prediction Error, APE

Bezwzględny błąd przewidywania, APE

$$e_t = (y_t - f_t^{(m)})$$

Gdzie :

y_t - zmierzona wartość w czasie t ,
 $f_t^{(m)}$ – przewidywana wartość w czasie t , uzyskane z zastosowania modelu prognostycznego m .

Zwany dalej indeksem modelu
(m) zostanie pominięty

Mean Absolute Error, MAE

Średni błąd bezwzględny, MAE

$$MAE = \frac{1}{n} \sum_{i=1}^n |e_i| = \overline{mean} |e_i|$$

Gdzie:

n –horyzont prognozy, średnia.

Mean Square Error, MSE

Średni błąd kwadratowy, MSE


$$MSE = \frac{1}{n} \sum_{i=1}^n (e_i^2) = \overline{mean} (e_i^2),$$

Root Mean Square Error, RMSE

Błąd średniej kwadratowej, RMSE

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (e_i^2)} = \sqrt{\overline{mean} (e_i^2)}$$

RMSE jest najpopularniejszą metryką oceny używaną w problemach z regresją. Wynika to z założenia, że błędy są bezstronne i podążają za rozkładem normalnym. Oto kluczowe punkty, które należy wziąć pod uwagę w RMSE:

- Moc "pierwiastka kwadratowego" umożliwia tej metryce pokazywanie dużych odchyłeń liczbowych.
- "Kwadratowy" charakter tego wskaźnika pomaga uzyskać bardziej wiarygodne wyniki, co zapobiega anulowaniu dodatnich i ujemnych wartości błędów. Innymi słowy, ta metryka trafnie wyświetla prawdopodobną wielkość terminu błędu.
-
- Pozwala uniknąć stosowania wartości błędu bezwzględnego, co jest wysoce niepożądane w obliczeniach matematycznych.
- Gdy mamy więcej próbek, rekonstrukcja rozkładu błędów za pomocą RMSE jest uważana za bardziej niezawodną.
-
- Na RMSE duży wpływ mają wartości odstające. Dlatego przed użyciem tych danych upewnij się, że z zestawu danych usunięto wartości odstające..
- W porównaniu do średniego błędu bezwzględnego, RMSE daje większą wagę i eliminuje duże błędy.
- Metryka RMSE jest podana przez: 
-

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

Ogólnie rzecz biorąc, bezwzględne miary błędu mają **następujące niedociągnięcia**.

- Główną wadą jest zależność skali. Dlatego jeśli zadanie prognozy obejmuje obiekty o różnych skalach lub wielkościach, nie można zastosować bezwzględnych miar błędu.
-
- Kolejną wadą jest duży wpływ wartości odstających w danych na prognozowaną ocenę wyników. Tak więc, jeśli dane zawierają wartości odstające o wartości maksymalnej (jest to powszechne w rzeczywistych zadaniach), wówczas bezwzględne miary błędów wykazują wartości zachowawcze.
- RMSE, MSE mają niską niezawodność: wyniki mogą się różnić w zależności od różnej frakcji danych

Miary oparte na błędach procentowych

Błędy procentowe są obliczane na podstawie wartości P_t

$$p_t = \frac{|e_t|}{y_t}$$

Accuracy = 1 - MAPE

Dokładność

Korzyść:

Błędy danych wyrażone w różnych skalach i jednostkach są porównywalne

Mean Absolute Percentage Error, MAPE

Średni bezwzględny błąd procentowy, MAPE

$$MAPE = \frac{1}{n} \sum_{i=1}^n 100 \cdot |p_i| = \underset{i=1,n}{mean}(100 \cdot |p_i|)$$

Root Mean Square Percentage Error, RMSPE

Główny średni kwadratowy błąd procentowy, RMSPE

$$RMSPE = \sqrt{\underset{i=1,n}{mean}(100 \cdot |p_i|)^2},$$

Niedociągnięć.

- Dzielenie przez zero, gdy rzeczywista wartość jest równa zero.
- Problem niesymetryczny - wartości błędów różnią się: czy przewidywana wartość jest większa czy mniejsza niż rzeczywista.
- Wartości odstające mają znaczący wpływ na wynik.

Wyjaśniona wariancja (zwana również wariancją wyjaśnioną) służy do pomiaru rozbieżności między modelem a rzeczywistymi danymi. Innymi słowy, jest to część całkowitej wariancji modelu, która jest wyjaśniona przez czynniki, które są faktycznie obecne i nie są spowodowane wariancją błędu.

Wyższe odsetki wyjaśnionej wariancji wskazują na silniejszą siłę kojarzenia. Oznacza to również, że dokonujesz lepszych prognoz (Rosenthal & Rosenthal, 2011).

$$r^2 = R^2 = \eta^2$$

Wyjaśniona wariancja może być oznaczona r^2 . W ANOVA nazywa się to eta do kwadratu (η^2), a w analizie regresji nazywa się to współczynnikiem determinacji (R^2). Te trzy terminy są zasadniczo synonimami, z wyjątkiem tego, że R^2 zakłada, że zmiany w zmiennej zależnej są spowodowane liniową relacją ze zmienną niezależną; η^2 nie ma tego podstawowego założenia.

Zmień
dataset

```
In [1]: # Pandas is used for data manipulation
import pandas as pd
```

```
In [2]: features = pd.read_csv('C:/Users/admin/Desktop/vlagunr-Phyto.csv')
```

```
In [3]: features.head(155)
```

Out[3]:

| | PSU | O2 | temp. | SS | SRP | DOP | PP | NH4N | NO3N | DON | PN | TN/TP | Fe | SiO4Si | Windspeedinsitu | Depth | DIN to DIP | PhytoBiomassC |
|-----|-------|-------|-------|--------|------|------|-------|-------|-------|-------|-------|--------|-------|--------|-----------------|-------|------------|---------------|
| 0 | 3.758 | 9.46 | 18.3 | 52.00 | 3.5 | 11.7 | 185.0 | 0.039 | 0.022 | 0.551 | 0.759 | 6.848 | 0.012 | 2.311 | 3.5 | 3.3 | 17.429 | 1.757 |
| 1 | 3.505 | 9.89 | 19.1 | 50.00 | 4.4 | 17.5 | 111.3 | 0.020 | 0.025 | 0.123 | 1.181 | 10.128 | 0.014 | 2.116 | 0.0 | 3.6 | 10.227 | 2.147 |
| 2 | 3.758 | 9.66 | 18.1 | 59.00 | 3.2 | 22.8 | 103.4 | 0.016 | 0.008 | 0.582 | 0.576 | 9.134 | 0.037 | 2.264 | 1.0 | 3.4 | 7.500 | 2.604 |
| 3 | 3.107 | 10.36 | 19.5 | 46.00 | 4.3 | 22.3 | 92.7 | 0.026 | 0.021 | 0.693 | 0.689 | 11.978 | 0.017 | 2.262 | 0.0 | 2.9 | 10.930 | 2.062 |
| 4 | 2.619 | 11.56 | 19.0 | 42.00 | 11.7 | 30.3 | 73.2 | 0.022 | 0.048 | 0.383 | 1.006 | 12.665 | 0.024 | 2.216 | 0.0 | 3.0 | 5.983 | 2.085 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 150 | 3.700 | 9.86 | 16.2 | 112.50 | 9.7 | 21.4 | 186.3 | 0.040 | 0.052 | 0.838 | 0.626 | 7.157 | 0.040 | 2.953 | 7.0 | 3.1 | 9.485 | 3.341 |
| 151 | 4.500 | 9.93 | 16.3 | 117.50 | 23.9 | 30.3 | 191.5 | 0.033 | 0.029 | 0.406 | 1.181 | 6.711 | 0.050 | 2.904 | 8.5 | 3.2 | 2.594 | 2.464 |
| 152 | 4.600 | 10.02 | 16.2 | 115.00 | 31.4 | 32.4 | 37.8 | 0.024 | 0.024 | 0.549 | 0.476 | 10.561 | 0.031 | 2.665 | 10.2 | 3.3 | 1.529 | 2.239 |
| 153 | 4.500 | 9.93 | 16.3 | 113.75 | 7.1 | 20.6 | 223.2 | 0.021 | 0.032 | 0.482 | 0.812 | 5.369 | 0.019 | 2.967 | 9.5 | 3.4 | 7.465 | 1.580 |
| 154 | 4.200 | 10.30 | 16.0 | 102.50 | 19.5 | 58.8 | 97.1 | 0.027 | 0.048 | 0.572 | 0.380 | 5.855 | 0.038 | 2.875 | 10.5 | 3.3 | 3.846 | 2.024 |

155 rows x 18 columns

```
In [4]: print('The shape of our features is:', features.shape)
"The shape of our features is: (155, 18)"
# Descriptive statistics for each column
features.describe()
```

The shape of our features is: (156, 18)

Out[4]:

| | PSU | O2 | temp. | SS | SRP | DOP | PP | NH4N | NO3N | DON | PN | TN/TP |
|-------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| count | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 | 156.000000 |
| mean | 3.356955 | 10.180705 | 17.350000 | 86.937115 | 15.035256 | 22.421154 | 134.547436 | 0.029429 | 0.037071 | 0.573897 | 0.765795 | 8.578487 |
| std | 0.758863 | 1.078504 | 3.122251 | 35.506823 | 11.841528 | 19.531893 | 51.210224 | 0.017682 | 0.020047 | 0.173958 | 0.316914 | 2.502329 |
| min | 1.400000 | 7.620000 | 10.600000 | 29.000000 | 0.100000 | 0.700000 | 20.900000 | 0.002000 | 0.007000 | 0.022000 | 0.049000 | 1.154000 |

Zmień
zmienną
docelową

jupyter Lab_RF_Phyto_Regression model-Copy1 Last Checkpoint: 28 minut temu (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

```
In [5]: # Use numpy to convert to arrays
import numpy as np
# Labels are the values we want to predict
labels = np.array(features['PhytoBiomassC'])
# Remove the Labels from the features
# axis 1 refers to the columns
features = features.drop('PhytoBiomassC', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)

In [6]: # Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.30, random_state = 42)

In [7]: print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)

Training Features Shape: (109, 17)
Training Labels Shape: (109,)
Testing Features Shape: (47, 17)
Testing Labels Shape: (47,)

In [8]: # The baseline predictions are the historical averages
baseline_preds = test_features[:, feature_list.index('temp.')]
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('temp. baseline error: ', round(np.mean(baseline_errors), 3))

temp. baseline error: 15.385

In [9]: # Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);
```


Wszystkie zmienne

jupyter Lab_RF_Phyto_Regression model-Copy1 Last Checkpoint: 29 minut temu (autosaved)



Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3

In [10]: # Use the forest's predict method on the test data

```
predictions = rf.predict(test_features)
# Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Test dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')
```

Test dataset Mean Absolute Error: 0.4996 degrees.

```
In [11]: test_labels.mean()
ratio = 100 * (np.mean(errors) / test_labels.mean())
print('Test dataset MAE/mean Ratio:', round(ratio, 5), '%')
```

Test dataset MAE/mean Ratio: 24.27669 %

```
In [12]: from sklearn.metrics import mean_squared_error
mse_test = mean_squared_error(test_labels, predictions)
mse_test
print('Test dataset Mean Squared Error:', round(mse_test, 5), 'degrees')
```

Test dataset Mean Squared Error: 0.4242 degrees

```
In [13]: from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(test_labels, predictions))
print('Test dataset Explained Variance Score:', round(EVS, 5), '%')
```

Test dataset Explained Variance Score: 31.91052 %

```
In [14]: # Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
print('Test dataset MAPE:', round(np.mean(mape), 3), '%')
```

Test dataset MAPE: 25.264 %

```
In [15]: # Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Test dataset Accuracy:', round(accuracy, 2), '%.')
```

Test dataset Accuracy: 74.74 %.

```
In [16]: # Use the forest's predict method on the train data
predictions = rf.predict(train_features)
# Calculate the absolute errors
errors = abs(predictions - train_labels)
# Print out the mean absolute error (mae)
print('Train dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')
```

Train dataset Mean Absolute Error: 0.28630 degrees.

```
In [17]: train_labels.mean()
ratio = 100 * (np.mean(errors) / train_labels.mean())
print('Train dataset MAE/mean Ratio:', round(ratio, 5), '%')
```

Train dataset MAE/mean Ratio: 9.55202 %

```
In [18]: from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(train_labels, predictions)
mse_train
print('Train dataset Mean Squared Error:', round(mse_train, 5), 'degrees')
```

Train dataset Mean Squared Error: 0.07338 degrees

```
In [19]: from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(train_labels, predictions))
print('Train dataset Explained Variance Score:', round(EVS, 5), '%')
```

Train dataset Explained Variance Score: 91.5201 %

```
In [20]: # Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / train_labels)
print('Train dataset MAPE:', round(np.mean(mape), 3), '%')
```

Train dataset MAPE: 12.048 %

```
In [21]: # Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Train dataset Accuracy:', round(accuracy, 2), '%.')
```

Train dataset Accuracy: 87.95 %.

```
In [22]: # Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 3)) for feature, importance in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:31} Importance: {}'.format(*pair)) for pair in feature_importances];
```

| | |
|-----------------|-------------------|
| Variable: PN | Importance: 0.233 |
| Variable: PSU | Importance: 0.131 |
| Variable: NH4N | Importance: 0.075 |
| Variable: NO3N | Importance: 0.075 |
| Variable: O2 | Importance: 0.067 |
| Variable: DON | Importance: 0.067 |
| Variable: temp. | Importance: 0.045 |

Wybierz
najważniejsze
2 zmienne

Dwie zmienne

jupyter Lab_RF_Phyto_Regression model-Copy1 Last Checkpoint: 31 minut temu (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run

| | |
|---------------------------|-------------------|
| Variable: SS | Importance: 0.027 |
| Variable: Windspeedinsitu | Importance: 0.026 |
| Variable: PP | Importance: 0.025 |
| Variable: Fe | Importance: 0.023 |
| Variable: SRP | Importance: 0.02 |

```
In [23]: # New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators=1000, random_state=42)
# Extract the two most important features
important_indices = [feature_list.index('PN'), feature_list.index('PSU')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]
```

```
In [24]: # Train the random forest
rf_most_important.fit(train_important, train_labels)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_labels)
# Display the performance metrics
print('Test dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')
```

Test dataset Mean Absolute Error: 0.57228 degrees.

```
In [25]: test_labels.mean()
ratio = 100 * (np.mean(errors) / test_labels.mean())
print('Test dataset MAE/mean Ratio:', round(ratio, 5), '%')
```

Test dataset MAE/mean Ratio: 27.80791 %

```
In [26]: from sklearn.metrics import mean_squared_error
mse_test = mean_squared_error(test_labels, predictions)
mse_test
print('Test dataset Mean Squared Error:', round(mse_test, 5), 'degrees')
```

Test dataset Mean Squared Error: 0.53733 degrees

```
In [27]: from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(test_labels, predictions))
print('Test dataset Explained Variance Score:', round(EVS, 5), '%')
```

Test dataset Explained Variance Score: 9.03189 %

```
In [28]: mape = np.mean(100 * (errors / test_labels))
accuracy = 100 - mape
print('Test dataset Accuracy:', round(accuracy, 3), '%.')
```

Test dataset Accuracy: 68.924 %.

```
In [29]: rf_most_important = RandomForestRegressor(n_estimators= 1000, random_state=42)
# Extract the two most important features
important_indices = [feature_list.index('PN'), feature_list.index('PSU')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]
```

```
In [30]: # Train the random forest
rf_most_important.fit(train_important, train_labels)
# Make predictions and determine the error
predictions = rf_most_important.predict(train_important)
errors = abs(predictions - train_labels)
# Display the performance metrics
print('Train dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')
```

Train dataset Mean Absolute Error: 0.23492 degrees.

```
In [31]: train_labels.mean()
ratio = 100 * (np.mean(errors) / train_labels.mean())
print('Train dataset MAE/mean Ratio:', round(ratio, 5), '%')
```

Train dataset MAE/mean Ratio: 10.8722 %

```
In [32]: from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(train_labels, predictions)
mse_train
print('Train dataset Mean Squared Error:', round(mse_train, 5), 'degrees')
```

Train dataset Mean Squared Error: 0.09626 degrees

```
In [33]: from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(train_labels, predictions))
print('Train dataset Explained Variance Score:', round(EVS, 5), '%')
```

Train dataset Explained Variance Score: 88.86874 %

```
In [34]: mape = np.mean(100 * (errors / train_labels))
accuracy = 100 - mape
print('Train dataset Accuracy:', round(accuracy, 3), '%.')
```

Train dataset Accuracy: 86.33 %.

localhost:8888/notebooks/Desktop/E-Learning%202021/Laboratory%20DST/Lab%2010%20Random%20Forest/RF_Phyto/Lab_RF_Phyto_Regression%20model-Copy1.ipynb#

Nowy folderNowa kartaHome Page - Select...Pierwsze krokiPekaoUWMData ScTlumYouTubeMapyGooglePocztaDeezerInfoTrinkaGnc+ GOKrzSoccerMusicConvertoKonfigurowanie sie...

jupyter Lab_RF_Phyto_Regression model-Copy1Last Checkpoint: 33 minuty temu (autosaved)

Python 3

FileEditViewInsertCellKernelWidgetsHelp

RunCode

In [35]:

```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
import matplotlib.pyplot as plt
# Set the style
plt.style.use('fivethirtyeight')
# List of x locations for plotting
x_values = list(range(len(importances)))
# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');

sorted_importances = [importance[1] for importance in feature_importances]
sorted_features = [importance[0] for importance in feature_importances]

# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)

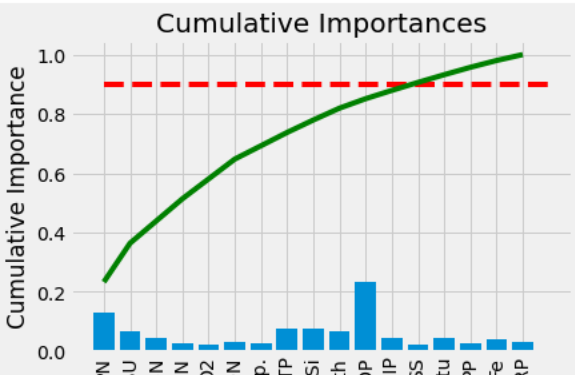
# Make a line graph
plt.plot(x_values, cumulative_importances, 'g-')

# Draw line at 90% of importance retained
plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'r', linestyle = 'dashed')

# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')

# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Cumulative Importances');
```

Cumulative Importances



Wpisz tu wyszukiwane słowa

17:3618.05.2021

Wybierz zmienne
po odrzuceniu 10%
najmniej ważnych
zmiennych

najważniejsze
zmienne (po
usunięciu 10% ze
skali ważności)

jupyter Lab_RF_Phyto_Regression model-Copy1 Last Checkpoint: 34 minuty temu (autosaved)



Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3

Run Code

```
In [36]: # Names of importances accounting for 90% of total importance
important_feature_names = ['PN', 'PSU', 'NH4N', 'O2', 'NO3N', 'DON', 'temp.', 'TN/TP', 'SiO4Si', 'Depth', 'DOP', 'DIN to DIP']

In [37]: # Find the columns of the most important features
important_indices = [feature_list.index(feature) for feature in important_feature_names]

In [38]: # Create training and testing sets with only the important features
important_train_features = train_features[:, important_indices]
important_test_features = test_features[:, important_indices]

In [39]: # Sanity check on operations
print('Important train features shape:', important_train_features.shape)
print('Important test features shape:', important_test_features.shape)

Important train features shape: (109, 12)
Important test features shape: (47, 12)

In [40]: # Use only the most important features
train_features = important_train_features[:, :]
test_features = important_test_features[:, :]

In [41]: # Update feature list for visualizations
feature_list = important_feature_names[:, :]

In [42]: # Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);

In [43]: # Use the forest's predict method on the test data
predictions = rf.predict(test_features)
# Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Test dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')

Test dataset Mean Absolute Error: 0.51881 degrees.

In [44]: test_labels.mean()
ratio = 100 * (np.mean(errors) / test_labels.mean())
print('Test dataset MAE/mean Ratio:', round(ratio, 5), '%')
```


localhost:8888/notebooks/Desktop/E-Learning%202021/Laboratory%20DST/Lab%2010%20Random%20Forest/RF_Phyto/Lab_RF_Phyto_Regression%20model-Copy1.ipynb#

Nowy folderNowa kartaHome Page - Select...Pierwsze krokiPekaoUWMData ScTlumYouTubeMapyGooglePocztaDeezerInfoTrinkaGnc+ GOKrzSoccerMusicConvertioKonfigurowanie sie...

jupyter Lab_RF_Phyto_Regression model-Copy1Last Checkpoint: 34 minuty temu (autosaved)

Python 3

FileEditViewInsertCellKernelWidgetsHelp

RunCode

In [45]:

```
from sklearn.metrics import mean_squared_error
mse_test = mean_squared_error(test_labels, predictions)
mse_test
print('Test dataset Mean Squared Error:', round(mse_test, 5), 'degrees')
```

Test dataset Mean Squared Error: 0.44903 degrees

In [46]:

```
from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(test_labels, predictions))
print('Test dataset Explained Variance Score:', round(EVS, 5), '%')
```

Test dataset Explained Variance Score: 28.95811 %

In [47]:

```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
print('Test dataset MAPE:', round(np.mean(mape), 3), '%')
```

Test dataset MAPE: 26.333 %

In [48]:

```
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Test dataset Accuracy:', round(accuracy, 2), '%.')
```

Test dataset Accuracy: 73.67 %.

In [49]:

```
# Use the forest's predict method on the train data
predictions = rf.predict(train_features)
# Calculate the absolute errors
errors = abs(predictions - train_labels)
# Print out the mean absolute error (mae)
print('Train dataset Mean Absolute Error:', round(np.mean(errors), 5), 'degrees.')
```

Train dataset Mean Absolute Error: 0.20562 degrees.

In [50]:

```
train_labels.mean()
ratio = 100 * (np.mean(errors) / train_labels.mean())
print('Train dataset MAE/mean Ratio:', round(ratio, 5), '%')
```

Train dataset MAE/mean Ratio: 9.51643 %

In [51]:

```
from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(train_labels, predictions)
mse_train
print('Train dataset Mean Squared Error:', round(mse_train, 5), 'degrees')
```

Train dataset Mean Squared Error: 0.07228 degrees

Wpisz tu wyszukiwane słowa

17:3718.05.2021

Desktop/E-Learning 2021/Laboratory%20DST/Lab%2010%20Random%20Forest/RF_Phyto/Lab_RF_Phyto_Regression%20model-Copy1.ipynb#

Nowy folder Nowa karta Home Page - Select... Pierwsze kroki Pekao UWM Data Sc Tium YouTube Mapy Google Poczta Deezer Info Trinkas nc+ GO Krz Soccer Music Convertio Konfigurowanie sie...

Jupyter Lab_RF_Phyto_Regression model-Copy1 Last Checkpoint: 36 minut temu (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [52]: `from sklearn.metrics import explained_variance_score
EVS = 100 * (explained_variance_score(train_labels, predictions))
print('Train dataset Explained Variance Score:', round(EVS, 5), '%')`

Train dataset Explained Variance Score: 91.64774 %

In [53]: `# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / train_labels)
print('Train dataset MAPE:', round(np.mean(mape), 3), '%')`

Train dataset MAPE: 11.892 %

In [54]: `# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Train dataset Accuracy:', round(accuracy, 2), '%.')`

Train dataset Accuracy: 88.11 %.


```
[55]: import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
plt.style.use('classic')
import seaborn as sns
sns.set()
data = pd.read_csv('C:/Users/admin/Desktop/vlagunr-Phyto.csv')
data.head()
vars = ['PhytoBiomassC', 'PN', 'PSU', 'NH4N', 'NO3N']
df = DataFrame (vars, columns=['Column_Name'])
g = sns.PairGrid(data, vars=['PhytoBiomassC', 'PN', 'PSU', 'NH4N', 'NO3N'],
                 hue='PhytoBiomassC', palette='RdBu_r')
g.map(plt.scatter, alpha=0.8)

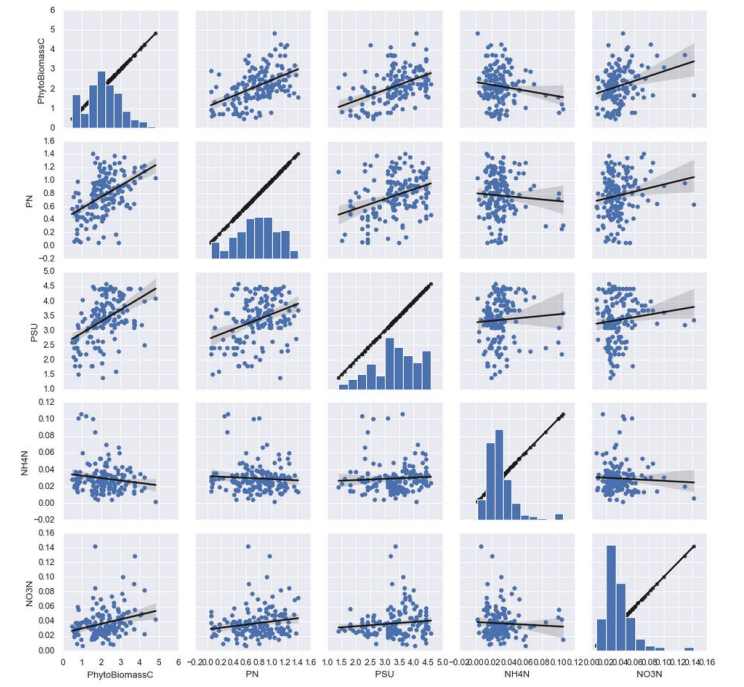
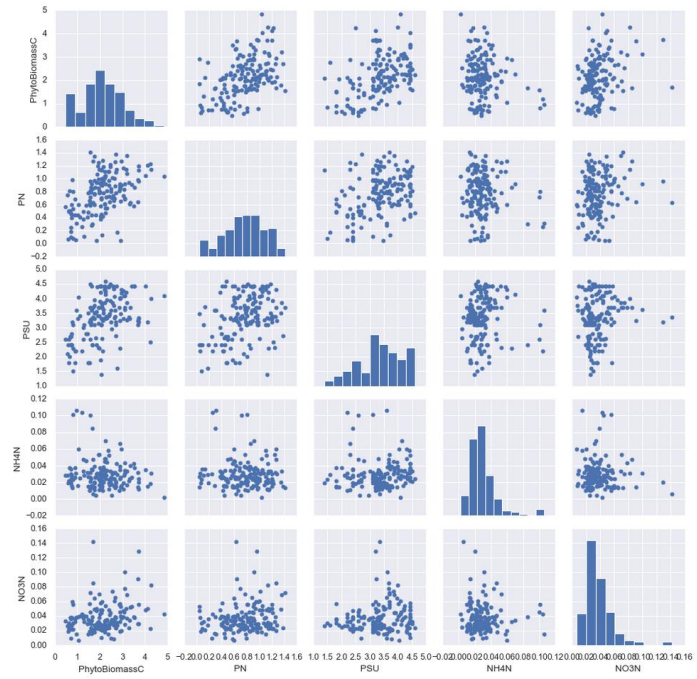
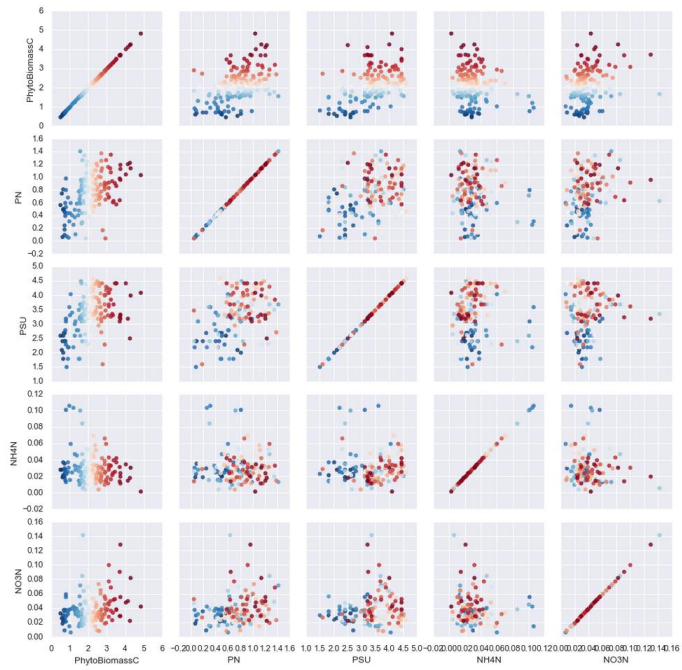
g = sns.PairGrid(data, vars=['PhytoBiomassC', 'PN', 'PSU', 'NH4N', 'NO3N'])
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)

g = sns.PairGrid(data, vars=['PhytoBiomassC', 'PN', 'PSU', 'NH4N', 'NO3N'])
g.map(sns.regplot, color=".1")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)

g.add_legend();
```

Trzy wykresy z relacjami
między zmienną celową
(task) a 4 najważniejszymi
zmiennymi





Webgraphviz – wizualizacja drzewa

```
In [55]: # Import tools needed for visualization
from sklearn.tree import export_graphviz
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3, min_samples_leaf=16)
rf_small.fit(train_features, train_labels)
# Extract the small tree
tree_small = rf_small.estimators_[5]
# Save the tree as a png image
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = important_feature_names, rounded = True, precision = 3)
```

```
In [56]: from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
```

Cross Validation dla Random Forest model: RandomSearchCV i GridSearchCV

RandomSearchCV ma ten sam cel co **GridSearchCV**: oba zostały zaprojektowane w celu znalezienia najlepszych parametrów w celu ulepszenia modelu.

Jednak w **RandomSearchCV** nie wszystkie parametry są testowane. Zamiast tego wyszukiwanie jest losowe, a wszystkie inne parametry są utrzymywane na stałym poziomie, podczas gdy parametry, które testujemy, są zmienne.

Główną różnicą między praktyczną implementacją tych dwóch metod jest to, że w **RandomSearchCV** możemy użyć **n_iter**, (n iteracji) aby określić, ile wartości parametrów chcemy pobrać próbki i przetestować. Zaleca się ustawienie **n_iter** na co najmniej 100

.
Dzięki **GridSearchCV**, wywołując metodę **best_params_** masz gwarancję uzyskania najlepszych wyników modelu w ramach wartości testowych, ponieważ przetestuje ona każdą z przekazanych wartości.

Jednak w przypadku **RandomSearchCV** im więcej próbek przetestujesz z zestawu wartości, tym bardziej pewne będzie wyszukiwanie - ale nigdy nie będzie w 100% pewne (chyba że przetestujesz każdą wartość z zestawu możliwych wartości). Statystycznie rzecz biorąc, możemy być dość pewni, że najlepsze znalezione parametry są rzeczywiście najlepszą kombinacją optymalnych parametrów, ponieważ wyszukiwanie jest całkowicie losowe.

ZADANIE: Wykonaj **obie CV** dla bazy **vlagunr-Cyano.csv**, zamieść tabelkę z porównaniem obu CV dla prób testowej i treningowej. Który model daje najwyższą dokładność?

Parametry modelu RF

```
In [66]: from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

Parameters currently in use:

```
{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 'warn',
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

RandomSearchCV

Parametry

RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Losowy wybór
najlepszych parametrów
(3-fold CV)

Model bazowy

n-iter = 100

Parametry wyjściowe
RandomSearchCV po
dopasowaniu 3 folds do
każdej ze 100 iteracji

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(train_features, train_labels)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 33 tasks      | elapsed: 12.6s
[Parallel(n_jobs=-1)]: Done 154 tasks    | elapsed: 49.6s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 1.5min finished
C:\Users\user\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:814: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                  estimator=RandomForestRegressor(bootstrap=True,
                                                    criterion='mse',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators='warn',
                                                    n_jobs=None, oob_score=False,
                                                    random_state=None),
                  param_distributions={'bootstrap': [True, False],
                                      'max_depth': [10, 20, 30, 40, 50, 60,
                                                    70, 80, 90, 100, 110,
                                                    None],
                                      'max_features': ['auto', 'sqrt'],
                                      'min_samples_leaf': [1, 2, 4],
                                      'min_samples_split': [2, 5, 10],
                                      'n_estimators': [200, 400, 600, 800,
                                                    1000, 1200, 1400, 1600,
                                                    1800, 2000]},
                  pre_dispatch='2*n_jobs', random_state=42, refit=True,
                  return_train_score=False, scoring=None, verbose=2)
```


RandomSearchCV

Dla próby
testowej

Wyniki dla base
model i random
model

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    ratio = 100 * (np.mean(errors) / test_labels.mean())
    mse_test = mean_squared_error(test_labels, predictions)
    EVS = 100 * (explained_variance_score(test_labels, predictions))
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Test dataset Model Performance')
    print('Test dataset Mean Absolute Error: {:.4f}degrees.'.format(np.mean(errors)))
    print('Test dataset MAE/mean Ratio: {:.4f}%.'.format(ratio))
    print('Test dataset Mean Squared Error: {:.4f}degrees.'.format(mse_test))
    print('Test dataset Explained Variance Score: {:.2f}%.'.format(EVS))
    print('Test dataset Accuracy: {:.2f}%.'.format(accuracy))
```

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
```

```
base_ratio = evaluate(base_model, test_features, test_labels)
base_mse_test = evaluate(base_model, test_features, test_labels)
base_EVS = evaluate(base_model, test_features, test_labels)
base_accuracy = evaluate(base_model, test_features, test_labels)
```

```
best_random = rf_random.best_estimator_
```

```
random_ratio = evaluate(best_random, test_features, test_labels)
random_mse_test = evaluate(best_random, test_features, test_labels)
random_EVS = evaluate(best_random, test_features, test_labels)
random_accuracy = evaluate(best_random, test_features, test_labels)
```

```
Test dataset Model Performance
Test dataset Mean Absolute Error: 0.5273degrees.
Test dataset MAE/mean Ratio: 25.6209%.
Test dataset Mean Squared Error: 0.4279degrees.
Test dataset Explained Variance Score: 33.90%.
Test dataset Accuracy: 71.28%.
```


RandomSearchCV

Dla próby
treningowej

Wyniki dla base
model i random
model

```
def evaluate(model, train_features, train_labels):
    predictions = model.predict(train_features)
    errors = abs(predictions - train_labels)
    ratio = 100 * (np.mean(errors) / train_labels.mean())
    mse_test = mean_squared_error(train_labels, predictions)
    EVS = 100 * (explained_variance_score(train_labels, predictions))
    mape = 100 * np.mean(errors / train_labels)
    accuracy = 100 - mape
    print('Train dataset Model Performance')
    print('Train dataset Mean Absolute Error: {:.4f}degrees.'.format(np.mean(errors)))
    print('Train dataset MAE/mean Ratio: {:.4f}%.'.format(ratio))
    print('Train dataset Mean Squared Error: {:.4f}degrees.'.format(mse_test))
    print('Train dataset Explained Variance Score: {:.2f}%.'.format(EVS))
    print('Train dataset Accuracy: {:.2f}%.'.format(accuracy))
```

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
```

```
base_ratio = evaluate(base_model, train_features, train_labels)
base_mse_test = evaluate(base_model, train_features, train_labels)
base_EVS = evaluate(base_model, train_features, train_labels)
base_accuracy = evaluate(base_model, train_features, train_labels)
```

```
best_random = rf_random.best_estimator_
```

```
random_ratio = evaluate(best_random, train_features, train_labels)
random_mse_test = evaluate(best_random, train_features, train_labels)
random_EVS = evaluate(best_random, train_features, train_labels)
random_accuracy = evaluate(best_random, train_features, train_labels)
```

```
Train dataset Model Performance
Train dataset Mean Absolute Error: 0.2341degrees.
Train dataset MAE/mean Ratio: 10.8355%.
Train dataset Mean Squared Error: 0.0955degrees.
Train dataset Explained Variance Score: 89.08%.
```

Grid Search CV

parametry

po
dopasowaniu 3
folds do każdej
ze 288 iteracji

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 500, 800, 2000]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
# Fit the grid search to the data
grid_search.fit(train_features, train_labels)
```

Fitting 3 folds for each of 288 candidates, totalling 864 fits

```
GridSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [80, 90, 100, 110],
                          'max_features': [2, 3], 'min_samples_leaf': [3, 4, 5],
                          'min_samples_split': [8, 10, 12],
                          'n_estimators': [100, 500, 800, 2000]}},
             verbose=2)
```

GridSearchCV

Dla próby
testowej

Wyniki dla base
model i random
model

```
# test dataset results
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    ratio = 100 * (np.mean(errors) / test_labels.mean())
    mse_test = mean_squared_error(test_labels, predictions)
    EVS = 100 * (explained_variance_score(test_labels, predictions))
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Test dataset Model Performance')
    print('Test dataset Mean Absolute Error: {:.4f}degrees.'.format(np.mean(errors)))
    print('Test dataset MAE/mean Ratio: {:.4f}%.'.format(ratio))
    print('Test dataset Mean Squared Error: {:.4f}degrees.'.format(mse_test))
    print('Test dataset Explained Variance Score: {:.2f}%.'.format(EVS))
    print('Test dataset Accuracy: {:.2f}%.'.format(accuracy))

grid_ratio = evaluate(best_random, test_features, test_labels)
grid_mse_test = evaluate(best_random, test_features, test_labels)
grid_EVS = evaluate(best_random, test_features, test_labels)
grid_accuracy = evaluate(best_grid, test_features, test_labels)
```

```
Test dataset Model Performance
Test dataset Mean Absolute Error: 0.5226degrees.
Test dataset MAE/mean Ratio: 25.3952%.
Test dataset Mean Squared Error: 0.4614degrees.
Test dataset Explained Variance Score: 28.67%.
Test dataset Accuracy: 73.46%.
Test dataset Model Performance
Test dataset Mean Absolute Error: 0.5226degrees.
```

GridSearchCV

Dla próby
treningowej

Wyniki dla base
model i random
model

```
#train dataset results
def evaluate(model, train_features, train_labels):
    predictions = model.predict(train_features)
    errors = abs(predictions - train_labels)
    ratio = 100 * (np.mean(errors) / train_labels.mean())
    mse_test = mean_squared_error(train_labels, predictions)
    EVS = 100 * (explained_variance_score(train_labels, predictions))
    mape = 100 * np.mean(errors / train_labels)
    accuracy = 100 - mape
    print('Train dataset Model Performance')
    print('Train dataset Mean Absolute Error: {:.4f}degrees.'.format(np.mean(errors)))
    print('Train dataset MAE/mean Ratio: {:.4f}%.'.format(ratio))
    print('Train dataset Mean Squared Error: {:.4f}degrees.'.format(mse_test))
    print('Train dataset Explained Variance Score: {:.2f}%.'.format(EVS))
    print('Train dataset Accuracy: {:.2f}%.'.format(accuracy))

grid_ratio = evaluate(best_random, train_features, train_labels)
grid_mse_test = evaluate(best_random, train_features, train_labels)
grid_EVS = evaluate(best_random, train_features, train_labels)
grid_accuracy = evaluate(best_grid, train_features, train_labels)
```

```
Train dataset Model Performance
Train dataset Mean Absolute Error: 0.0695degrees.
Train dataset MAE/mean Ratio: 3.2188%.
Train dataset Mean Squared Error: 0.0092degrees.
Train dataset Explained Variance Score: 98.93%.
Train dataset Accuracy: 95.97%.
Train dataset Model Performance
Train dataset Mean Absolute Error: 0.0695degrees.
Train dataset MAE/mean Ratio: 3.2188%.
Train dataset Mean Squared Error: 0.0092degrees.
Train dataset Explained Variance Score: 98.93%
```