tutorial_pandas_scikitlearn

October 24, 2025

1 Tutorial: Verkenning en Machine Learning met Pandas & Scikit-learn

Dit bewijsstuk dient voor criterium 2, het zelfstandig mobiliseren van verworven kennis en vaardigheden voor een goed experimenteel ontwerp/werkplan ten behoeve van de onderzoeksvraag.

In dit bewijsstuk laat ik zien dat ik oefen met data-preparatie, modelbouw en evaluatie van een ML-model (Logistic regression) op een oefendataset.

Verband met de hoofdvraag: Uiteindelijk wil ik een model ontwikkelen dat, op basis van mutatie- en methylatieprofielen van tumorcellijnen, kan voorspellen of een patiënt baat heeft bij een specifieke therapie. Deze oefening helpt me benodigde vaardigheden onder de knie te krijgen en de stappen van een ML-workfkow te oefenen, van verkennen van de data tot de evaluatie van een model.

1.0.1 Dataset: Wine Data

Voor deze oefening gebruik ik de Wine dataset uit scikit-learn, een ingebouwde dataset die informatie bevat over verschillende soorten wijn, inclusief hun chemischie eigenschappen en kwaliteitskenmerken. Het voordeel van een ingebouwde dataset is dat er geen externe downloads of databases nodig zijn, waardoor ik me volledig kan richten op de data-analyse en machine learning workflow.

1.0.2 Data laden

[16]:
```
from sklearn.datasets import load_wine

# dataset laden
wine_data = load_wine()

# bekijk de ruwe data
wine_data.data
```

[16]:
```
array([[1.423e+01, 1.710e+00, 2.430e+00, …, 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, …, 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, …, 1.030e+00, 3.170e+00,
        1.185e+03],
       …,
```

```
       [1.327e+01, 4.280e+00, 2.260e+00, …, 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, …, 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, …, 6.100e-01, 1.600e+00,
        5.600e+02]], shape=(178, 13))
```

Zoals hierboven te zien is, bestaat de data uit een N x M array (N samples, M features). Om dit overzichtelijker te maken, zet ik de data om naar een Pandas Dataframe. Dit maakt het eenvoudiger om de data te inspecteren, kolomnamen te zien en later bewerkingen uit te voeren.

In mijn eigen onderzoek zal de data een vergelijkbare structuur hebbben, maar met veel meer features. Elke rij (N) stelt dan één tumorcellijn voor, en elke kolom (M) een kenmerk van die cellijn. Deze kenmerken kunnen bijvoorbeeld bestaan uit duizenden methylatiesites en mutatiekenmerken. De uiteindelijke dataset zou dus bijvoorbeeld de vorm kunnen hebben van 300 cellijnen x 20.000 features)

[17]:
```
import pandas as pd

# data omzetten naar data frame met kolomnamen
wine_df = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)

# target toevoegen (de wijnsoort)
wine_df["target"] = wine_data.target

# Eerste 5 rijen bekijken
wine_df.head()
```

[17]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

De bovenstaande tabel toont de eerste 5 rijen van de dataset. Elke rij vertegenwoordit een wijnsoort,

en elke kolom is een eigenschap. De kolom 'target' bevat het label (0, 1 of 2), dat aangeeft tot welke wijnklasse het monster behoort.

Het is belangrijk om te begrijpen dat het toevoegen van een target-kolom van belang is bij supervised learning. Bij mijn eigen dataset zal deze targetkolom de responsewaarden bevatten, AUC)

1.0.3 Data verkennen

Het is belangrijk om de data eerst te verkennen. Pandas DataFrames zijn tweedimentsionale, gelabelde datastructuren met kolommen, rijen en data. Ze maken het eenvoudig om statistieken te bekijken, te filteren of grafieken te maken.

[18]:
```
# dataset verkenning
wine_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
```

De `info()`-methode geeft een overzicht van de DataFrame: - aantal rijen = 178 (aantal wijnmonsters) - aantal kolommen = 14 (13 features + 1 target) - Non-Null values: alle kolommen bevatten volledige data, er zijn geen missing values - datatype: features zijn `float64`, targer is `int64`

Belangrijke inzichten voor machine learning: 1. als alle data numeriek is, is het direct bruikbaar voor ML-algoritmes. 2. geen missing values, dus het is niet nodig om imputatie uit te voeren 3. target-kolom is duidelijk aanwezig (dit wordt het label voor supervised learning)

1.0.4 Beschrijvende statistiek en data-inspectie

Naast de basisinformatie uit `info()` is het handig om een statistische samenvatting van de data te bekijken. Pandas biedt hiervoor de `describe()`-methode. Deze genereerd statistieken als:

- count: aantal niet-lege waarden

- mean: gemiddelde
- std: standaarddeviatie
- min/max: minimum en maximum
- quartiles: 25%, 50%, 75% percentielwaarden

[19]:
```
# beschrijvende statistieken
wine_df.describe()
```

[19]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium
count	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573
std	0.811827	1.117146	0.274344	3.339564	14.282484
min	11.030000	0.740000	1.360000	10.600000	70.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000
50%	13.050000	1.865000	2.360000	19.500000	98.000000
75%	13.677500	3.082500	2.557500	21.500000	107.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000

	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
count	178.000000	178.000000	178.000000	178.000000
mean	2.295112	2.029270	0.361854	1.590899
std	0.625851	0.998859	0.124453	0.572359
min	0.980000	0.340000	0.130000	0.410000
25%	1.742500	1.205000	0.270000	1.250000
50%	2.355000	2.135000	0.340000	1.555000
75%	2.800000	2.875000	0.437500	1.950000
max	3.880000	5.080000	0.660000	3.580000

	color_intensity	hue	od280/od315_of_diluted_wines	proline
count	178.000000	178.000000	178.000000	178.000000
mean	5.058090	0.957449	2.611685	746.893258
std	2.318286	0.228572	0.709990	314.907474
min	1.280000	0.480000	1.270000	278.000000
25%	3.220000	0.782500	1.937500	500.500000
50%	4.690000	0.965000	2.780000	673.500000
75%	6.200000	1.120000	3.170000	985.000000
max	13.000000	1.710000	4.000000	1680.000000

	target
count	178.000000
mean	0.938202
std	0.775035
min	0.000000
25%	0.000000
50%	1.000000
75%	2.000000
max	2.000000

Deze statistieken geven snel inzicht in de distributie van de data en helpen bij het identifiveren van mogelijke afwijkingen of verschillende schalen tussen features.

1.0.5 Data weergave

Om een idee te krijgen van de daadwerkelijke waarden in elke kolom. kunnen we de eerste of laatste rijen van de DataFrame bekijken:

[20]:
```
# geef de laatste 5 rijen van de dataset
wine_df.tail()
```

[20]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	\
173	13.71	5.65	2.45	20.5	95.0	1.68	
174	13.40	3.91	2.48	23.0	102.0	1.80	
175	13.27	4.28	2.26	20.0	120.0	1.59	
176	13.17	2.59	2.37	20.0	120.0	1.65	
177	14.13	4.10	2.74	24.5	96.0	2.05	

	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	\
173	0.61	0.52	1.06	7.7	0.64	
174	0.75	0.43	1.41	7.3	0.70	
175	0.69	0.43	1.35	10.2	0.59	
176	0.68	0.53	1.46	9.3	0.60	
177	0.76	0.56	1.35	9.2	0.61	

	od280/od315_of_diluted_wines	proline	target
173	1.74	740.0	2
174	1.56	750.0	2
175	1.56	835.0	2
176	1.62	840.0	2
177	1.60	560.0	2

Hier is te zien dat de features op verschillende schalen staan. Algoritmes die gevoelig zijn voor de schaal van de data (zoals bijvoorbeeld Logistic Regression) kunnen hierdoor problemen krijgen. Daarom is het later vaak nodig om de data te normaliseren of standaardiseren voordat je een model traint.

Voor meer over pandas zie Python Pandas Tutorial: The Ultimate Guide for Beginners.

1.0.6 Data preprocessing

Nu de data is verkend is de volgende stap in een machine learning workflow om de data voor te bereieden zodat het model er goed mee kan werken. De data preprocessing kunnen we uitvoeren met Scikit-learn.

Real-world data kan rommelig zijn. Het kan missing values bevatten, overbodige kolommen, outliers of ruis. Het negeren van deze problemen leid er toe dat het model verkeerde patronen leert. “Garbage in, garbage out”.

Eerst splitsen we de dataset in: - Features (X): alle kolommen behalve de target - Labels (y): de target-kolom (wat we willen voorspellen)

[21]:
```
from sklearn.preprocessing import StandardScaler

# features en labels scheiden
X = wine_df[wine_data.feature_names].copy()  # alleen features
y = wine_df["target"].copy()                 # target
```

1.0.7 Standaardisatie van features

Er wordt gebruik gemaakt van `StandardScaler` om elke feature te standaardiseren.

[22]:
```
# scaler aanmaken en fitten
scaler = StandardScaler()
scaler.fit(X) # berekent het gemiddelde en de standaarddeviatie van elke feature

# features transformeren
X_scaled = scaler.transform(X) # hier wordt de berekening toegepast op de data␣
 ↪en wordt elke feature geschaalt

# print eerste voorbeeld van gestandaardiseerde waarden
print(X_scaled[0])
```

```
[ 1.51861254 -0.5622498   0.23205254 -1.16959318  1.91390522  0.80899739
  1.03481896 -0.65956311  1.22488398  0.25171685  0.36217728  1.84791957
  1.01300893]
```

De array die hier wordt weergegeven toot de gestandaardiseerde waarden van het eerste wijnmonster in de dataset. Elke waarde komt overeen met één feature maar nu geschaald zodat het gemiddelde van elke feature 0 is en de standaarddeviatie 1. De positieve waarden liggen hoger dan het gemiddelde, en de negatieve waarden liggen lager dan het gemiddelde.

1.0.8 Model Training:

Train/Test Split Voordat een machine learning model voorspellingen kan doen, moet het eerst getraind worden op data. Om te controleren of het model goed generaliseerd naar nieuwe, ongeziene data, moet de dataset worden opgesplitst in:

- Trainingsset (70%) voor het trainen van het model
- Testset (30%) voor het evalueren van de prestaties

[23]:
```
from sklearn.model_selection import train_test_split

# data splitsen in train en test
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(
    X_scaled,      # gestandaardiseerde features
    y,             # labels
    train_size=0.7,
    random_state=25  # voor reproduceerbaarheid
)
```

```
# controle van de splitsing
print(f"Train size: {round(len(X_train_scaled) / len(X) * 100)}% \n\
Test size: {round(len(X_test_scaled) / len(X) * 100)}%")
```

```
Train size: 70%
Test size: 30%
```

70% van de dat is nu trainingsdata en 30% is testdata. Het splitsen van de data zorgt ervoor dat de modelprestaties betrouwbaar kunnen worden geevalueerd. In mijn onderzoek zal ik dezelfde methoden gebruiken. (Eventueel 80%/20%)

Building the model (Logistic Regression) Nu de data is gesplitst en gestandaardiseerd, kan een machine learning model worden getraint. Voor dit voorbeeld gebruik ik **Logistic Regression**, een veelgebruikt algoritme voor classificatie. Dit model wil ik ook gebruiken bij mijn onderzoek.

Het doel van deze oefening is om het model te trainen om de juiste klasse (wijnsoort) te voorspellen op basis van features.

[24]:
```
from sklearn.linear_model import LogisticRegression

# model aanmaken
logistic_regression = LogisticRegression()

# train het model op de trainingsdata
logistic_regression.fit(X_train_scaled, y_train)

# voorspellingen maken op de testset
log_reg_preds = logistic_regression.predict(X_test_scaled)
```

De variabele `log_reg_preds` bevat de voorspelde klassen voor elke sample in de testset. Het model heeft geleerd van de trainingsdata en kan nu generaliseren naar ongeziene voorbeelden.

In mijn onderzoek zou ik hetzelfde kunnen doen met tumor- en methylatieprofielen als feature en therapieresonse als target.

1.0.9 Model evaluatie (Logistic Regression)

Nu het Logistic Regression model is getraind, kan ik gaan kijken hoe goed het presteert op de ongeziene data (testset). Hiervoor kan gebruik worden gemaakt van de volgende metrics:

- Precision: hoeveel de voorspellingen per klasse correct zijn
- Recall: hoeveel van de echte samples per klasse correct worden voorspeld
- F1-score: harmonisch gemiddelde van precision en recall
- Accuracy: percentage correcte voorspellingen

[ ]:
```

```

[25]:
```
from sklearn.metrics import classification_report

# evaluatie van LR
```

```
print("Logistic Regression Results:\n")
print(classification_report(y_test, log_reg_preds))
```

```
Logistic Regression Results:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      0.92      0.96        25
           2       0.86      1.00      0.92        12

    accuracy                           0.96        54
   macro avg       0.95      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54
```

Interpretatie van de resultaten **in de context van therapierespons**

De getoonde classificatiemetrics (precision, recall, F1-score, accuracy, macro- en weighted averages) geven inzicht in hoe goed het model de therapierespons van tumorcellijnen voorspelt op basis van mutatie- en methylatieprofielen.

Rijen per klasse (0, 1, 2): vertegenwoordigen de discrete responsklassen voor een specifiek medicijn, bijvoorbeeld:
- 0 = geen respons / resistent
- 1 = matige respons
- 2 = sterke respons

Precision per klasse: het percentage voorspellingen dat correct is binnen die responsklasse

Recall per klasse: het percentage echte cellijnen van die klasse dat correct wordt voorspeld

F1-score: gewogen combinatie van precision en recall, geeft een samenvatting van voorspellingskwaliteit per klasse

Accuracy: het totale percentage correcte voorspellingen over alle cellijnen.

Macro average: gemiddelde van alle klassen, geeft inzicht in de prestaties ongeacht klassegrootte.

Weighted average: gemiddelde van alle klassen, gewogen op basis van het aantal cellijnen per klasse.

Hoge scores betekenen dat het model veelbelovend is voor het selecteren van cellijnen die waarschijnlijk goed reageren, terwijl lagere scores aangeven waar het model verbeterd kan worden of waar bijvoorbeeld aanvullende biomarkers nodig zijn.

1.0.10 Afsluiting

In deze tutorial heb ik een eerste kennismaking gedaan met de mogelijkheden van scikit-learn. Ik heb geleerd hoe je data kunt voorbereiden, standaardiseren, splitsen in train- en testsets, een model kunt trainen (LR) en de prestaties kunt evalueren.

Dit vormt de basis voor het toepassen van ML op mijn eigen data.

Voor verdere verdieping in supervised learning en bredere mogelijkheiden van scikit-learn heb ik de volgende (betaalde) bron beschikbaar: Supervised Learning with scikit-learn.