WYKŁAD Z DEMONSTRACJĄ NA ŻYWO

Eksploracja danych i klasyfikacja

Od surowych danych do decyzji: predykcyjne utrzymanie ruchu w Pythonie, z asystentem AI

Zbiór danych: AI4I 2020 Predictive Maintenance (UCI)

Python · pandas · scikit-learn

PLAN

Plan wykładu

Start
Środowisko i narzędzia

Etap 1
EDA i audyt cech

Etapy 2–4
Podział, baseline, pierwszy model

Etapy 5–6
Ocena, metryki, walidacja krzyżowa

Etap 7
Dobór modelu i cechy

Finał
A gdybyśmy zaufali agentowi?

Etap 8
AGENTS.md i wnioski

Koniec
Pytania

Ciąg dalszy w części 2: regresja, klasteryzacja i detekcja anomalii.

START · ŚRODOWISKO

Ekosystem Pythona w jednym .venv

```
python -m venv .venv
source .venv/bin/activate
# Windows: .venv\Scripts\activate
pip install pandas numpy seaborn \
    scikit-learn ipykernel
pip freeze > requirements.txt
```

Biblioteka	Do czego
pandas	tabele, wczytywanie, EDA
numpy	obliczenia na tablicach
seaborn	wykresy (na matplotlib)
scikit-learn	modele, Pipeline, metryki, CV
ipykernel	kernel notebooka z .venv

Izolacja
Każdy projekt ma własne wersje pakietów.

Powtarzalność
requirements.txt odtwarza środowisko u każdego.

Bezpieczeństwo
Systemowy Python zostaje nietknięty.

START · NARZĘDZIA

VS Code, Codex i plik AGENTS.md

VS Code

Notebook .ipynb, kernel z .venv, wykresy obok kodu.

Codex

Agent w edytorze: czyta projekt, pisze i poprawia kod na nasze polecenie.

AGENTS.md

Stałe instrukcje dla agenta w repozytorium: środowisko, konwencje i zasady projektu.

```
# AGENTS.md
## Środowisko
- Python z .venv, pakiety z requirements.txt
- Pracujemy w notebooku lecture.ipynb
## Dane
- ai4i2020.csv, krótkie nazwy snake_case
- Target: failure

## Zasady ML
(uzupełnimy na końcu wykładu)
```

Wersja startowa pliku: zasady ML dopiszemy w finale

ZASADA NA DZIŚ

Kod z AI sprawdzamy przed uruchomieniem

Najpierw budujemy poprawny proces krok po kroku. Na końcu sprawdzamy, jaki kod napisze agent bez naszych wskazówek.

ZBIÓR DANYCH

AI4I 2020: 10 000 rekordów, 14 kolumn

10 000

rekordów (cykli procesu)

339

awarii (Machine failure = 1)

3,4%

odsetek klasy pozytywnej

Grupa	Kolumny	Opis
Identyfikatory	UDI, Product ID	numer rekordu i produktu
Typ produktu	Type	jakość L / M / H
Pomiary procesu	Air temperature [K] Process temperature [K] Rotational speed [rpm] Torque [Nm] Tool wear [min]	czujniki i stan narzędzia
Target	Machine failure	0 / 1
Tryby awarii	TWF, HDF, PWF, OSF, RNF	który typ awarii wystąpił

Źródło: UCI ML Repository, archive.ics.uci.edu/dataset/601 (Matzka, 2020)

Dane syntetyczne

POJĘCIE

Cechy (X) i target (y)

Cechy X				Target y
air_temp	rpm	torque	tool_wear	failure
298,1	1551	42,8	0	0
298,9	1408	46,3	3	0
302,6	1336	61,2	224	1

Wiersz

jeden przypadek: cykl procesu z pomiarami

Cechy (X)

to, co wiemy przed decyzją

Target (y)

to, co chcemy przewidzieć

Uczenie nadzorowane: model uczy się zależności X → y na przykładach ze znaną odpowiedzią, a potem przewiduje y dla nowych X.

ETAP 1 · EDA

Struktura danych

```
df = pd.read_csv("ai4i2020.csv")
df.columns = ["udi", "product_id", "type",
    "air_temp", "process_temp", "rpm",
    "torque", "tool_wear", "failure",
    "twf", "hdf", "pwf", "osf", "rnf"]
df.head()
df.info()       # typy, braki
df.describe()   # zakresy
df.isna().sum()
```

PROMPT DLA AI

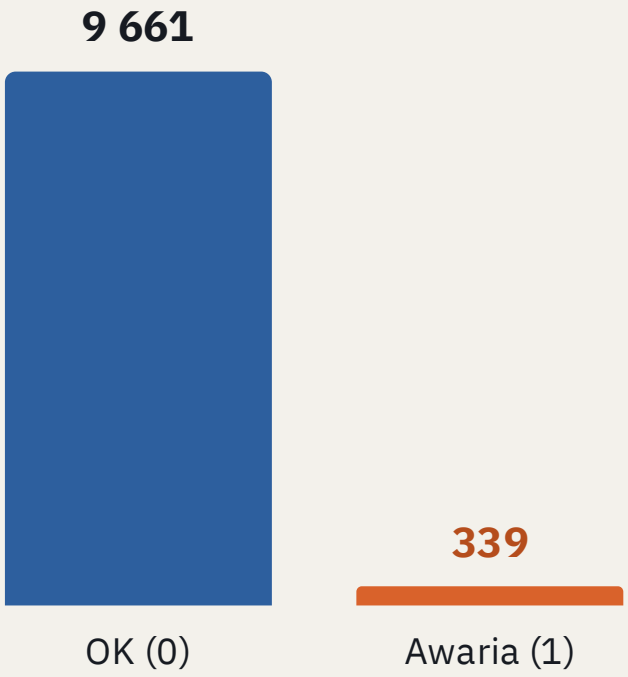
„Wczytaj ai4i2020.csv, nadaj kolumnom krótkie nazwy w snake_case i pokaż przegląd: typy, braki, statystyki.”

PYTANIA DO DANYCH

- Jakie mamy typy danych?
- Czy są braki?
- Jakie są zakresy wartości?

ETAP 1 · EDA

Ile jest awarii?



96,6%

accuracy modelu, który zawsze odpowiada „OK”

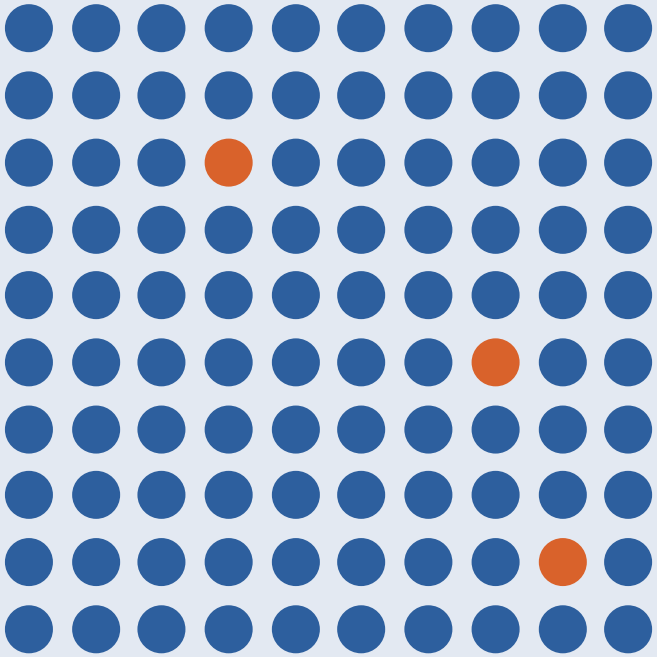
```
df["failure"].value_counts(normalize=True)
```

Czy klasy są zbalansowane? Czy 96,6% to dobry wynik?

Słupki w skali: 9 661 vs 339

POJĘCIE

Niezbalansowanie klas



100 przypadków: ok. 3 awarie

Jedna klasa jest dużo rzadsza od drugiej.

Model może nauczyć się ignorować rzadką klasę i nadal mieć wysokie accuracy.

ANALOGIA

Czujnik dymu, który nigdy nie piszczy, ma rację prawie każdego dnia. I jest bezużyteczny.

CO Z TYM ROBIMY

stratify przy podziale · metryki dla rzadkiej klasy · wagi klas lub próg

W tym zbiorze: ~3,4% awarii

ETAP 1 · EDA

Rozkłady cech i różnice między klasami

Histogramy

```
num = ["air_temp", "process_temp",
       "rpm", "torque", "tool_wear"]
df[num].hist(bins=40, figsize=(14, 8))
```

Skale? Outliery? Nietypowe rozkłady?

Boxploty względem targetu

```
sns.boxplot(df, x="failure", y="torque")
sns.boxplot(df, x="failure",
            y="tool_wear")
```

Czy klasy różnią się „na oko”?

PROMPT DLA AI

„Histogramy cech numerycznych, a potem boxploty torque i tool_wear względem failure.”

ETAP 1 · EDA

Gdzie skupiają się awarie?

Scatter rpm × torque

```
sns.scatterplot(df, x="rpm", y="torque",
    hue="failure", alpha=0.5)
```

Szukamy obszarów: czy awarie leżą na skrajach, przy bardzo niskiej lub bardzo wysokiej mocy?

Korelacje

```
sns.heatmap(df[num].corr(), annot=True,
    cmap="coolwarm")
```

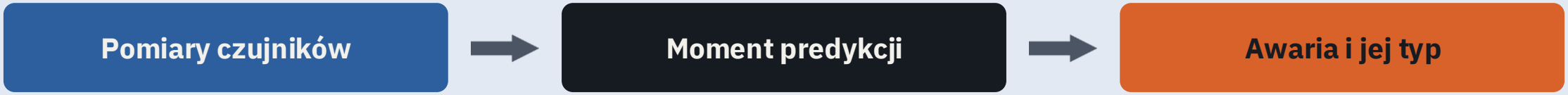
air_temp ↔ process_temp oraz rpm ↔ torque są silnie powiązane. Czy to redundancja?

PROMPT DLA AI

„Scatter rpm × torque pokolorowany według failure i heatmapa korelacji cech numerycznych.”

POJĘCIE

Wyciek danych (leakage)



Model może korzystać tylko z tego, co jest **przed** momentem predykcji.

Wyciek z przyszłości

Cecha powstaje dopiero po zdarzeniu, np. informacja o typie awarii.

Wyciek z testu

Coś liczone z udziałem danych testowych, np. skalowanie na całym zbiorze.

Analogia: egzamin z kluczem odpowiedzi na ławce. Świetny wynik, zero umiejętności.

ETAP 1 · AUDYT LEAKAGE

Audyt kolumn pod kątem wycieku

```
modes = ["twf", "hdf", "pwf", "osf", "rnf"]
df[modes + ["failure"]].corr()["failure"]
pd.crosstab(df["failure"],
    df[modes].sum(axis=1))
```

PROMPT DLA AI

„Sprawdź, jak kolumny twf, hdf, pwf, osf i rnf mają się do failure: korelacje i crosstab.”

Korelacja to za mało

O leakage decyduje to, kiedy informacja powstaje, a nie jak mocno koreluje z targetem.

Czy etykiety są spójne?

Crosstab pokaże, czy flagi trybów awarii zawsze zgadzają się z failure. Jeśli nie: komu wierzymy?

ETAP 1 · WYBÓR CECH

Czy ta informacja byłaby dostępna w momencie predykcji?

Kolumna	Decyzja	Dlaczego
failure	target (y)	to przewidujemy
twf, hdf, pwf, osf, rnf	DROP	opisują typ awarii, powstają po fakcie
udi, product_id	DROP	identyfikatory, nie właściwości procesu
type	one-hot	cecha kategoryczna L / M / H
air_temp … tool_wear	X	pomiary znane przed awarią

POJĘCIE

Zbiór treningowy i testowy

Train · 80%
tu model się uczy i tu podejmujemy wszystkie decyzje

Test · 20%
raz, na końcu

Analogia: uczysz się na zadaniach z ćwiczeń, ale egzamin musi mieć nowe zadania. Inaczej sprawdza pamięć, nie umiejętności.

Po co?

Żeby wiedzieć, jak model zadziała na danych, których nie widział.

Overfitting

Model zapamiętał train: błyszczy na nim i zawodzi na nowych danych.

Zasada

Skalowanie, wybór modelu i progu ustalamy na train. Testu nie dotykamy do końca.

W tym zbiorze: ok. 8 000 / 2 000 rekordów

ETAP 2 · PODZIAŁ DANYCH

Podział danych w kodzie

```
X = df.drop(columns=["udi", "product_id", "failure",
    "twf", "hdf", "pwf", "osf", "rnf"])
y = df["failure"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

stratify=y

Zachowuje ~3,4% awarii w obu częściach. Bez tego test może mieć ich wyraźnie mniej lub więcej.

random_state=42

Powtarzalny podział: ten sam wynik u wszystkich na sali.

Od teraz

X_test odkładamy do samego końca. Każda decyzja zapada na danych treningowych.

POJĘCIE

Klasyfikator: od cech do decyzji



`fit(X_train, y_train)`

uczenie na przykładach ze znaną odpowiedzią

`predict_proba(X)`

prawdopodobieństwo każdej klasy

`predict(X)`

gotowa decyzja po progu (domyślnie 0,5)

Próg to decyzja inżynierska, nie matematyczna. Wrócimy do niego w etapie 7.

ETAP 3 · BASELINE

Baseline: zawsze klasa większościowa

```
dummy = DummyClassifier(strategy="most_frequent")
dummy.fit(X_train, y_train)
print(classification_report(
    y_test, dummy.predict(X_test),
    zero_division=0))
```

≈0,97 accuracy

0 recall awarii

Wysokie accuracy, zero wykrytych awarii. Czy to dobry model?

POJĘCIE

Preprocessing: dane w formie dla modelu

Skalowanie

Cechy mają różne jednostki: K, rpm, Nm, min. Sprowadzamy je do wspólnej skali.

```
x' = (x − średnia) / std
# StandardScaler
```

Wymagane dla k-NN, SVM i k-means, pomocne przy regresji z regularyzacją, drzewom obojętne.

Kodowanie kategorii

Model liczy na liczbach, a type to litera L, M lub H.

```
M → [0, 0, 1]
# kolumny: H, L, M
```

One-hot: osobna kolumna 0/1 dla każdej wartości.

Braki danych

Uzupełniamy, np. medianą, albo usuwamy wiersze lub kolumny.

```
SimpleImputer(
    strategy="median")
```

W tym zbiorze braków nie ma.

fit tylko na train, transform na train i test. Pipeline pilnuje tego za nas.

ETAP 4 · PIERWSZY MODEL

Regresja logistyczna w jednym Pipeline

```
num = ["air_temp", "process_temp", "rpm",
       "torque", "tool_wear"]
pre = ColumnTransformer([
    ("num", StandardScaler(), num),
    ("cat", OneHotEncoder(), ["type"])])
model = Pipeline([
    ("pre", pre),
    ("clf", LogisticRegression(max_iter=1000))])
model.fit(X_train, y_train)
```

PROMPT DLA AI

„Pipeline: StandardScaler dla cech numerycznych, OneHotEncoder dla type, LogisticRegression. Ucz tylko na X_train.”

Dlaczego Pipeline?

Skaler uczy się tylko na train, także w każdym foldzie CV. Ta sama transformacja trafia na produkcję.

Czy jest lepszy od baseline?

ETAP 5 · OCENA

FP czy FN: który błąd kosztuje więcej?

	Predykcja: OK	Predykcja: awaria
Fakt: OK	**TN** sprawna maszyna, brak alarmu	**FP: fałszywy alarm** niepotrzebny przegląd lub postój
Fakt: awaria	**FN: przeoczona awaria** nieplanowany przestój, uszkodzenie	**TP** wykryta awaria, planowana interwencja

```
confusion_matrix(y_test, y_pred)
classification_report(y_test, y_pred)
```

PRZYPOMNIENIE

```
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1 = 2 · P · R / (P + R)
```

ROC-AUC: pole pod krzywą ROC, czyli prawdopodobieństwo, że losowa awaria dostanie wyższy wynik niż losowy przypadek OK. 0,5 to poziom losowy, 1 to model idealny.

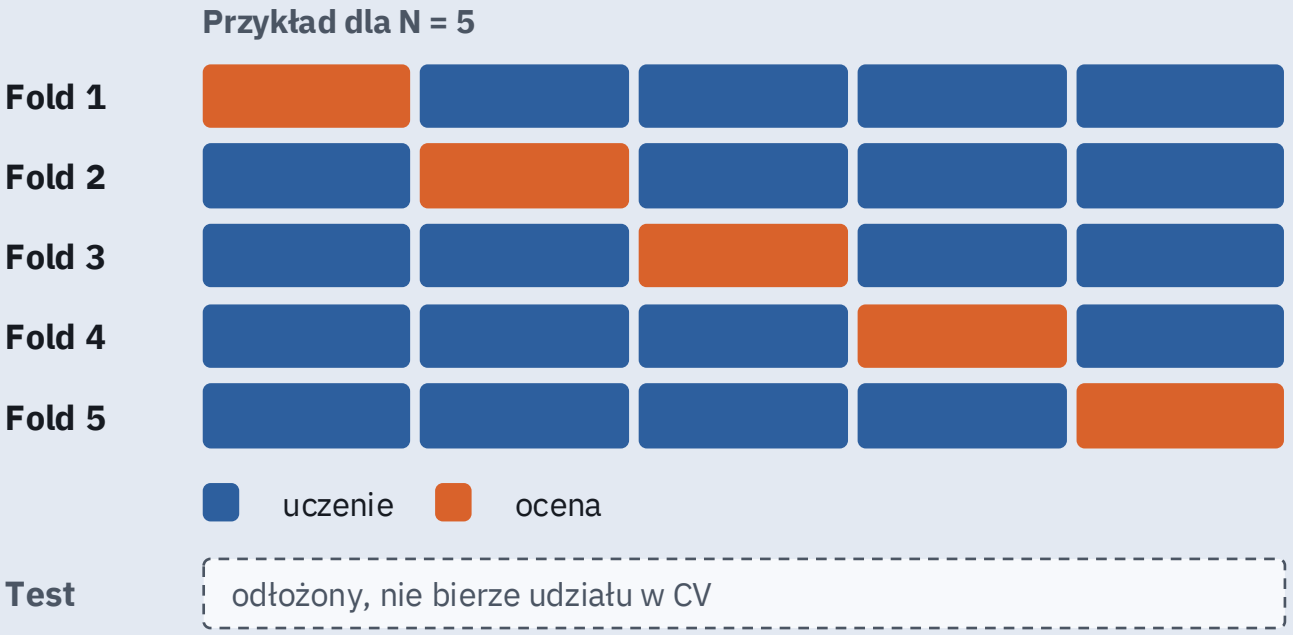
ETAP 5 · OCENA

Którą metrykę wybrać?

Metryka	Na jakie pytanie odpowiada	Pułapka
Accuracy	Jaki odsetek odpowiedzi jest trafny?	baseline ma ~97% bez wykrycia awarii
Precision	Ile alarmów było prawdziwych?	rośnie, gdy alarmujemy rzadko
Recall	Ile awarii wyłapaliśmy?	rośnie, gdy alarmujemy zawsze
F1	Kompromis precision i recall	zakłada równą wagę obu błędów
PR-AUC	Jak dobrze model szereguje ryzyko?	nie mówi, jaki próg wybrać
Koszt oczekiwany	Ile kosztują nas błędy?	wymaga przyjęcia kosztów FP i FN

POJĘCIE

Walidacja krzyżowa N-krotna



Jak?

Dzielimy train na N równych części. N razy: uczymy na N – 1 częściach, oceniamy na pozostałej.

Co dostajemy?

N wyników zamiast jednego: średnią i rozrzut.

Stratified

Każda część ma ten sam odsetek awarii.

Analogia: kilka krótkich sprawdzianów zamiast jednego. Jeden mógł być przypadkiem łatwy.

ETAP 6 · WALIDACJA KRZYŻOWA

Czy wynik zależy od jednego podziału?

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X_train, y_train,
    cv=cv, scoring="f1")
print(scores.round(3), scores.mean(), scores.std())
```

Zawsze jawny scoring=

Domyślnie cross_val_score liczy accuracy, czyli ocenia nas jak baseline.

Pipeline w CV

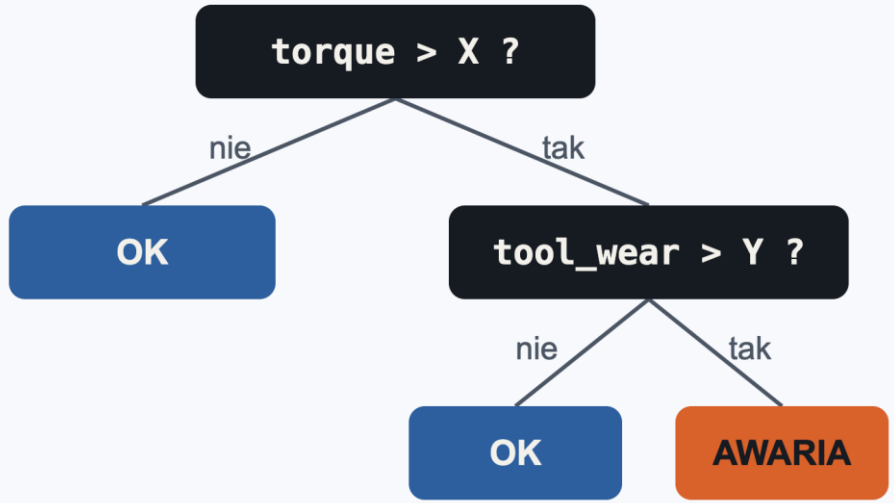
Skaler i encoder dopasowują się osobno w każdym foldzie. Bez przecieku z foldu walidacyjnego.

Średnia i rozrzut

Duże std oznacza, że wynik zależy od tego, które awarie trafiły do foldu.

POJĘCIE

Drzewo decyzyjne i las losowy



Drzewo

Seria pytań o progi na cechach. Algorytm sam wybiera pytania, które najlepiej rozdzielają klasy.

Ryzyko

Głębokie drzewo zapamiętuje dane treningowe. Ograniczamy głębokość lub liczbę przykładów w liściu.

Las losowy

Wiele drzew uczonych na losowych próbkach danych i cech. Decyzja przez głosowanie.

X i Y: progi znalezione przez algorytm

POJĘCIE

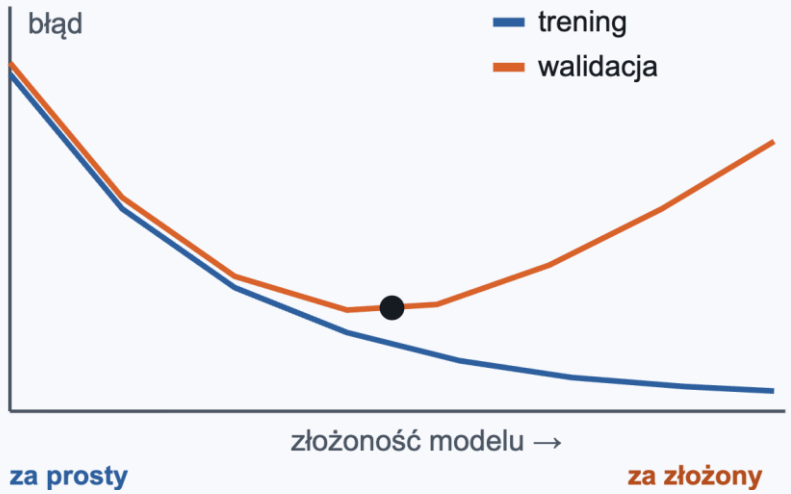
Parametry i hiperparametry

Parametry

Model uczy się ich z danych: współczynniki regresji, progi w drzewie.

Hiperparametry

Ustawiamy je przed uczeniem: siła regularyzacji C, głębokość drzewa, liczba drzew w lesie.



Hiperparametry dobieramy w walidacji krzyżowej na train. Test zostaje do samego końca.

ETAP 7 · HIPERPARAMETRY

Krzywa walidacji: głębokość drzewa

```
tree = Pipeline([("pre", pre),
                 ("clf", DecisionTreeClassifier(random_state=42))])
depths = [1, 2, 3, 4, 6, 8, 12, 16]
tr, va = validation_curve(tree, X_train, y_train, param_name="clf__max_depth",
    param_range=depths, cv=cv, scoring="f1")
plt.plot(depths, tr.mean(axis=1), label="trening")
plt.plot(depths, va.mean(axis=1), label="walidacja")

search = GridSearchCV(tree, {"clf__max_depth": depths,
    "clf__min_samples_leaf": [1, 5, 20]}, cv=cv, scoring="f1")
search.fit(X_train, y_train).best_params_
```

Gdzie kończy się underfitting, a zaczyna overfitting?

POJĘCIE

No free lunch: nie ma najlepszego algorytmu

Uśrednione po wszystkich możliwych problemach, żaden algorytm uczenia nie jest lepszy od innych.

1. Przewaga algorytmu bierze się z dopasowania jego założeń do problemu.
2. Dlatego model wybieramy na podstawie walidacji na danych.
3. Wiedza domenowa podpowiada założenia: progi → drzewa, zależność liniowa → regresja.

ANALOGIA

W warsztacie nie ma najlepszego klucza. Jest klucz pasujący do śruby.

Hipoteza: awarie w AI4I wygenerowano progami, więc drzewa mogą mieć przewagę. Sprawdzimy to w GridSearchCV.

POJĘCIE

Algorytmy klasyfikacji i ich założenia

Algorytm	Jak widzi dane	Mocne strony	Słabe strony
Regresja logistyczna	liniowa granica w przestrzeni cech	szybka, interpretowalna, mało danych	nieliniowość tylko przez inżynierię cech
k-NN	podobne przypadki mają podobną klasę	prosta idea, brak treningu	wymaga skalowania, wolna predykcja
Drzewo decyzyjne	seria progów na cechach	czytelne reguły, bez skalowania	łatwo się przeucza
Las losowy	głosowanie wielu drzew	mocny wybór domyślny, odporny	mniej czytelny, większy model
Gradient boosting	drzewa poprawiające błędy poprzedników	często najlepszy na danych tabelarycznych	więcej hiperparametrów do strojenia

Pytanie przy wyborze: jakie założenie pasuje do mojego problemu?

POJĘCIE

A co z SVM i sieciami neuronowymi?

SVM

Szuka granicy z jak największym marginesem między klasami. Jądra (np. RBF) pozwalają na granicę nieliniową.

+ małe i średnie zbiory, także przy wielu cechach

– wymaga skalowania, czuły na C i gamma, wolny przy dużej liczbie próbek

Sieci neuronowe

Same uczą się reprezentacji cech z surowych danych, warstwa po warstwie.

+ obrazy, dźwięk, tekst, surowe sygnały

– dużo danych i strojenia; na danych tabelarycznych zwykle nie wygrywają z zespołami drzew

Dane tabelaryczne → zespoły drzew
AI4I: zagregowane pomiary, np. temperatura, moment, zużycie

→

Surowy sygnał → sieci neuronowe
np. drgania łożyska próbkowane z wysoką częstotliwością

Wybór modelu zależy od postaci danych.

ETAP 7 · DOBÓR MODELU

Skalowanie: k-NN kontra drzewo

`df[num].std()`

Cecha	Odch. std.
air_temp	…
process_temp	…
rpm	…
torque	…
tool_wear	…

```
knn = KNeighborsClassifier(n_neighbors=5)
tree = DecisionTreeClassifier(max_depth=5, random_state=42)
for name, clf in [("k-NN", knn), ("drzewo", tree)]:
    for scaled in [False, True]:
        m = make_pipeline(StandardScaler(), clf) if scaled else clf
        s = cross_val_score(m, X_train[num], y_train,
                            cv=cv, scoring="f1")
        print(name, scaled, s.mean().round(3))
```

Model	Bez skalowania	Ze skalowaniem
k-NN	…	…
Drzewo decyzyjne	…	…

Bez skalowania k-NN widzi głównie obroty i zużycie narzędzia. Temperatury, istotne dla awarii HDF, mają znikomy wpływ.

Wartości uzupełniamy wspólnie w notebooku

ETAP 7 · DOBÓR MODELU

GridSearchCV: algorytm i hiperparametry

```
grid = [
    {"clf": [LogisticRegression(max_iter=1000, class_weight="balanced")],
     "clf__C": [0.01, 0.1, 1, 10]},
    {"clf": [RandomForestClassifier(random_state=42)],
     "clf__max_depth": [4, 8, None], "clf__min_samples_leaf": [1, 5]},
    {"clf": [GradientBoostingClassifier(random_state=42)],
     "clf__learning_rate": [0.05, 0.1], "clf__max_depth": [2, 3]},
    {"clf": [SVC(kernel="rbf", class_weight="balanced")],
     "clf__C": [0.1, 1, 10], "clf__gamma": ["scale", 0.1]}]
search = GridSearchCV(model, grid, cv=cv, scoring="f1")
search.fit(X_train, y_train)
search.best_params_, search.best_score_
```

Wyszukiwanie odbywa się wyłącznie w CV na train. Test dotykamy raz, dla zwycięzcy.

ETAP 7 · DOBÓR MODELU

Porównanie wyników wyszukiwania

Model	Najlepsze hiperparametry	F1 w CV (średnia ± std)
Regresja logistyczna	…	…
Las losowy	…	…
Gradient boosting	…	…
SVM (jądro RBF)	…	…

Rozrzut ma znaczenie

Różnica mniejsza niż rozrzut między foldami oznacza, że modele są praktycznie równe.

Prostszy przy remisie

Przy podobnym wyniku wybieramy model prostszy, szybszy i łatwiejszy do wyjaśnienia.

Optymizm wyboru

Najlepszy z wielu prób wynik jest lekko zawyżony. Dlatego potwierdzamy go na teście.

Wartości uzupełniamy wspólnie w notebooku

POJĘCIE

Inżynieria cech

Tworzymy nowe cechy z istniejących, żeby model łatwiej zobaczył zależność.

MODEL WIDZI OSOBNO

`torque, rpm`

Awaria, gdy moc jest za niska albo za wysoka: trudna granica dla modelu liniowego.

→

DAJEMY MU WPROST

`power = torque × rpm × 2π/60`

Jedna cecha z dwoma progami. Model ma znacznie prostsze zadanie.

Kombinacje

iloczyn, różnica, stosunek

Transformacje

logarytm, przedziały, potęgi

Wiedza domenowa

fizyka procesu, reguły technologa

Czas

dzień tygodnia, czas od przeglądu

Uwaga: nowa cecha też może przeciekać, np. gdy liczymy ją z przyszłości albo z targetu.

ETAP 7 · INŻYNIERIA CECH

Fizyka procesu jako cechy

Cecha	Wzór	Reguła awarii w generatorze
Moc P	torque × rpm × 2π / 60 [W]	PWF: P < 3500 W lub P > 9000 W
ΔT	process_temp − air_temp [K]	HDF: ΔT < 8,6 K i rpm < 1380
Obciążenie narzędzia	tool_wear × torque [min·Nm]	OSF (overstrain): próg ok. 11 / 12 / 13 tys. dla L / M / H

```
df["power"] = df["torque"] * df["rpm"] * 2 * np.pi / 60
df["dT"] = df["process_temp"] - df["air_temp"]
df["wear_torque"] = df["tool_wear"] * df["torque"]
```

Znasz fizykę procesu? Czasem nie potrzebujesz ML.

Progi: opis zbioru AI4I 2020 na UCI

FINAŁ · CODE REVIEW

A gdybyśmy bezkrytycznie zaufali agentowi?

NAIWNY PROMPT

„Zbuduj klasyfikator przewidujący awarię maszyny na tym zbiorze.”

Wasze zadanie

Przeczytajcie kod jak recenzent. Co jest nie tak?

Narzędzia

Wszystko z etapów 1–7: cechy, podział, metryki, CV.

Pytanie

Czy temu wynikowi można wierzyć?

FINAŁ · CODE REVIEW

Wynik naiwnego kodu: wyciek danych

```
X = df.drop(columns=["failure"])
# typowy naiwny kod: wszystko poza targetem trafia do X
```

Co widzimy

Wynik bliski idealnego, najlepszy w tabeli porównawczej.

Pytanie do sali

Dlaczego ten model jest bezwartościowy na produkcji?

Odpowiedź

Flagi trybów awarii znamy dopiero po awarii. W chwili predykcji ich nie ma.

FINAŁ · PODSUMOWANIE REVIEW

Typowe błędy w kodzie z AI

Błąd	Dlaczego szkodzi	Etap
Tryby awarii (twf…rnf) w X	leakage: informacja powstaje po awarii	1
udi / product_id jako cechy	model uczy się identyfikatorów	1
Brak stratify w podziale	test może mieć inny odsetek awarii	2
fit() skalera przed podziałem	statystyki testu przeciekają do treningu	4
Domyślny scoring w CV	liczy accuracy, a przy 3% awarii to nic nie mówi	6
Hiperparametry lub próg strojone na teście	test przestaje być niewidziany	7

Kolumna „Etap” mówi, gdzie na wykładzie robiliśmy to poprawnie. Każdy błąd staje się regułą w AGENTS.md.

ETAP 8 · AGENTS.MD

AGENTS.md: reguły po code review

```
# AGENTS.md
## Środowisko · ## Dane (bez zmian)

## Zasady ML
- Nie używaj twf, hdf, pwf, osf, rnf ani identyfikatorów w X
- Podział: stratify=y, random_state=42
- Preprocessing tylko wewnątrz Pipeline
- W CV zawsze jawny scoring= (nie accuracy)
- Hiperparametry i próg stroimy w CV, nigdy na teście
- Raportuj recall, precision i F1, nie samo accuracy
```

Błąd znaleziony w code review zapisujemy jako regułę.

Agent stosuje ją w kolejnych zadaniach.

Sprawdzenie: ten sam naiwny prompt jeszcze raz. Co się zmieniło?

Reguły pochodzą z listy błędów znalezionych w finale

ETAP 8 · WNIOSKI

Wnioski: czy wdrażamy model?

Model	Recall	Precision	F1	Accuracy	Koszt
Baseline „zawsze OK”	0	—	0	≈0,97	…
Regresja logistyczna	…	…	…	…	…
LR + cechy fizyczne	…	…	…	…	…
Random Forest	…	…	…	…	…
Zwycięzca GridSearchCV	…	…	…	…	…
Naiwny prompt, bez review	…	…	…	…	…

Koszt błędów

Czy model obniża koszt względem obecnych przeglądów?

Stabilność

Czy wynik trzyma się na wszystkich foldach?

Dane z produkcji

Czy prawdziwe dane wyglądają jak te syntetyczne?

Wartości uzupełniamy wspólnie w notebooku

PODSUMOWANIE KLASYFIKACJI

Cały proces klasyfikacji w jednym kodzie

```
# 1. Dane i podział
X = df.drop(columns=["udi", "product_id", "failure",
    "twf", "hdf", "pwf", "osf", "rnf"])
y = df["failure"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)

# 2. Preprocessing i model w jednym Pipeline
num = ["air_temp", "process_temp", "rpm",
       "torque", "tool_wear"]
pre = ColumnTransformer([
    ("num", StandardScaler(), num),
    ("cat", OneHotEncoder(), ["type"])])
model = Pipeline([("pre", pre),
                  ("clf", LogisticRegression())])
```

```
# 3. Walidacja i dobór modelu
cv = StratifiedKFold(5, shuffle=True,
                     random_state=42)
grid = [
 {"clf": [LogisticRegression(max_iter=1000)],
  "clf__C": [0.1, 1, 10]},
 {"clf": [RandomForestClassifier(random_state=42)],
  "clf__max_depth": [4, 8, None]},
 {"clf": [SVC()], "clf__C": [0.1, 1, 10]}]
search = GridSearchCV(model, grid, cv=cv,
                      scoring="f1")
search.fit(X_train, y_train)

# 4. Jedna ocena na teście, dla zwycięzcy
best = search.best_estimator_
y_pred = best.predict(X_test)
print(classification_report(y_test, y_pred))
```

dane → podział → Pipeline → GridSearchCV z CV → jedna ocena na teście

PODSUMOWANIE

Podsumowanie części 1

1. Najpierw baseline. Bez niego nie wiesz, czy model cokolwiek umie.
2. Zanim użyjesz cechy, zapytaj, czy istnieje w momencie predykcji.
3. Przy niezbalansowanych klasach accuracy wprowadza w błąd.
4. Wiedza domenowa często daje więcej niż zmiana algorytmu.
5. Kod z AI czytaj jak cudzy pull request i sprawdzaj przed użyciem.