# Projekt 2

Julia Strzelczyk, Maksymilian Tabian, Łukasz Wyszomierski

25.01.2026

## 1 Wybór modeli do portfolio

Portfolio modeli zostało stworzone na podstawie benchmarku MementoML dostępnego na stronie kaggle.com. Ze strony pobrano tabelę zawierającą wartości AUC dla 7 popularnych algorytmów machine learning: CatBoot, Gradient Boosting, regresji logistycznej z regularyzacją elastic net, k-nearest neighbors, Random Forest, Ranger oraz XGBoost uzyskane na 39 zbiorach danych. W ramach każdego z tych algorytmów testowane było wiele kombinacji hiperparametrów. Każda konfiguracja modeli i hiperparametrów została wywołana po 20 razy na każdym ze zbiorów dla różnych podziałów na zbiór treningowy i testowy. Dla każdej kombinacji (model, zbiór danych, indeks użytych hiperparametrów) policzono średnią z AUC. W ten sposób uzyskano następujące wyniki

| | Model 1 | Model 2 | ... | Model $n$ |
|---|---|---|---|---|
| zbiór 1 | $AUC_{11}$ | $AUC_{12}$ | ... | $AUC_{1n}$ |
| zbiór 2 | $AUC_{21}$ | $AUC_{22}$ | ... | $AUC_{2n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| zbiór m | $AUC_{m1}$ | $AUC_{m2}$ | ... | $AUC_{mn}$ |

gdzie $AUC_{ij}$ oznacza średnią wartość AUC dla j-tego modelu na i-tym zbiorze.

Następnie średnie wartości AUC dla każdego zbioru danych zastąpiono rangami (dokonano rangowania po wierszach w powyższej tabeli). W ten sposób otrzymano następującą dane

| | Model 1 | Model 2 | ... | Model $n$ |
|---|---|---|---|---|
| zbiór 1 | $R_{11}$ | $R_{12}$ | ... | $R_{1n}$ |
| zbiór 2 | $R_{21}$ | $R_{22}$ | ... | $R_{2n}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| zbiór m | $R_{m1}$ | $R_{m2}$ | ... | $R_{mn}$ |

Do portfolio wybrano po 7 konfiguracji każdego z rozważanych siedmiu typów modeli, które miały najwyższą średnią rangę na rozważanych zbiorach danych. Ponieważ okazało się, że w przypadku modeli typu catboost w pliku z wartościami hiperparametrów nie ma indeksów odpowiadającym modelom, które miały najwyższą średnią z rang, dlatego wybrano 7 najlepszych konfiguracji modeli catboost, spośród tych, których wartości hiperparametrów były opisane. Podobna sytuacja miała miejsce z częścią modeli typu knn. Ostateczne zestawienie modeli, które weszły w skład portfolio zawiera tabela na końcu raportu.

## 2 Implementacja i interfejs klasy MiniAutoML

Rdzeniem rozwiązania jest klasa `MiniAutoML`, która zawiera ujednolicony interfejs do trenowania i ewaluacji modeli. Poniżej przedstawiono zestawienie wszystkich zaimplementowanych metod wraz z ich opisem działania.

### Konstruktor i inicjalizacja

```
__init__(self, models_config, random_state=123)
```

- **Parametry:**
  - `models_config` (list): lista słowników z konfiguracją modeli (nazwa, ścieżka do klasy, parametry) wczytana z pliku JSON.

- `random_state` (int, domyślnie 123): seed zapewniający powtarzalność wyników (wpływa m.in. na podział w obiekcie `KFold`).

- **Działanie:** inicjalizuje obiekt, przygotowując go do przechowywania wyników eksperymentów.

### Metoda trenująca (fit)

```
fit(self, X, y, cv_folds=5, use_ensemble=False, ensemble_size=3)
```

- **Parametry:**
  - `X` (pd.DataFrame / np.array): zbiór treningowy.
  - `y` (pd.Series / np.array): zmienna docelowa (target).
  - `cv_folds` (int, domyślnie 5): liczba foldów używanych w kroswalidacji.
  - `use_ensemble` (bool, domyślnie False): flaga decydująca o użyciu trybu Ensemble.
  - `ensemble_size` (int, domyślnie 3): liczba najlepszych modeli, które wejdą w skład Ensemble (używane tylko gdy `use_ensemble=True`).
- **Preprocessing Pipeline:** wewnątrz metody budowany jest automatycznie obiekt `ColumnTransformer`, który rozdziela przetwarzanie zmiennych:
  - Zmienne numeryczne: uzupełnianie braków średnią (`SimpleImputer`) i skalowanie (`StandardScaler`).
  - Zmienne kategoryczne: uzupełnianie braków modą i kodowanie One-Hot (`OneHotEncoder` z parametrem `handle_unknown='ignore'`).
  - Target ($y$): kodowany za pomocą `LabelEncoder`.
- **Selekcja i Trening:** metoda iteruje przez konfiguracje, dynamicznie ładuje klasy modeli i tworzy dla każdego osobny `Pipeline` (preprocesor + model). Jakość oceniana jest za pomocą `cross_val_score` (metryka: balanced accuracy). Na koniec, w zależności od parametru `use_ensemble`, fitowany jest jeden najlepszy `Pipeline` lub kilka wybranych modeli.

### Metody pomocnicze i predykcja

```
transform_y(self, y)
```

- **Parametry:** `y` – wektor etykiet (np. ze zbioru testowego).
- **Działanie:** przekształca surowe etykiety na format numeryczny przy użyciu dopasowanego wcześniej `LabelEncoder`.

```
predict_proba(self, X)
```

- **Parametry:** `X` – dane testowe.
- **Działanie:** zwraca prawdopodobieństwo klasy pozytywnej (1).
  - Tryb Single: zwraca wynik metody `predict_proba` (lub `decision_function` + sigmoid) dla najlepszego modelu.
  - Tryb Ensemble: oblicza średnią arytmetyczną prawdopodobieństw zwróconych przez wszystkie modele składowe (Soft Voting).

```
predict(self, X)
```

- **Parametry:** `X` – dane testowe.
- **Działanie:** wywołuje wewnętrznie `predict_proba` i dokonuje przypisaniua do klasy (0 lub 1) na podstawie thresholdu 0.5.

## 3 Strategie wyboru modelu lub Ensemblingu

Logika zaimplementowana w metodzie `fit` pozwala na elastyczne podejście do problemu klasyfikacji poprzez dwa główne tryby działania.

### 3.1 Tryb jednego modelu

Jest to tryb domyślny (`use_ensemble=False`). System przeprowadza screening modeli zdefiniowanych w JSON, oceniając je w 5-krotnej kroswalidacji (CV).

- **Kryterium wyboru:** średnia wartość *Balanced Accuracy* ze wszystkich foldów.
- **Finalizacja:** model o najwyższym wyniku (`best_model`) jest klonowany i trenowany ponownie na całym zbiorze treningowym, w ramach pełnego pipeline'u.

### 3.2 Tryb Ensemble

Aktywowany ustawieniem parametru `use_ensemble=True`. Ma na celu zwiększenie stabilności predykcji.

- **Selekcja:** system wybiera $N$ modeli (gdzie $N$ =`ensemble_size`) z góry rankingu utworzonego podczas CV.
- **Trening:** każdy z wybranych modeli jest niezależnie trenowany na pełnym zbiorze danych.
- **Soft Voting:** predykcja odbywa się poprzez uśrednianie prawdopodobieństw. Jeśli $P_i$ to prawdopodobieństwo zwrócone przez $i$-ty model, to wynik końcowy wynosi:

$$P(y = 1|X) = \frac{1}{N} \sum_{i=1}^{N} P_i(y = 1|X)$$

## 4 Przykłady działania

W celu weryfikacji poprawności działania klasy `MiniAutoML`, przeprowadzono serię eksperymentów na rzeczywistych zbiorach danych. Poniżej przedstawiono krok po kroku proces uczenia i ewaluacji na przykładzie zbioru *Blood Transfusion Service Center*, a także zestawienie wyników dla drugiego zbioru danych – *Credit-G*.

### 4.1 Blood Transfusion Service Center

Pierwszym etapem było pobranie danych z repozytorium OpenML oraz wczytanie konfiguracji modeli z pliku JSON. Dane podzielono na zbiór treningowy i testowy (proporcja 80:20). Następnie uruchomiono metodę `fit` w trybie domyślnym (*Single Model*).

```python
import openml
dataset = openml.datasets.get_dataset('blood-transfusion-service-center')
X, y, _, _ = dataset.get_data(dataset_format="dataframe", target=dataset.default_target_attribute)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 123, stratify=y)
```

✓ 2.5s Python

```python
with open("models.json", "r") as f:
    models_config = json.load(f)
```

✓ 0.0s Python

```python
# Single model
automl_single = MiniAutoML(models_config)
automl_single.fit(X_train, y_train)

# Zakodowanie y_test tak samo jak y_train
y_test_encoded = automl_single.transform_y(y_test)
```

✓ 2m 23.0s Python

```
Rozpoczynam trenowanie 49 modeli

--- Tryb SINGLE ---

kknn_1954 (balanced accuracy z CV: 0.6368)
```

Jak widać, system automatycznie przetworzył dane i spośród 49 konfiguracji wybrał model `kknn_1954` (k-Nearest Neighbors) jako najlepszy. Dodatkowo dostajemy informację, że model ten osiągnął wynik Balanced Accuracy w kroswalidacji na poziomie 0.6368.

W kolejnym kroku, na tych samych danych, uruchomiono proces w trybie *Ensemble* (`use_ensemble=True`).

```python
# Ensemble
automl_ensemble = MiniAutoML(models_config)
automl_ensemble.fit(X_train, y_train, use_ensemble=True)
```

✓ 2m 22.4s Python

```
Rozpoczynam trenowanie 49 modeli

--- Tryb ENSEMBLE ---

Uśredniono prawdopodobieństwa następujących 3 modeli:
-kknn_1954 (balanced accuracy z CV: 0.6368)
-randomForest_1744 (balanced accuracy z CV: 0.6359)
-kknn_1939 (balanced accuracy z CV: 0.6358)

<MiniAutoML Object | Status: Fitted (Ensemble: 3 models)>
```

System wyselekcjonował 3 najlepsze modele: dwa warianty k-NN oraz jeden Random Forest, tworząc z nich ensemble.

Ostatnim etapem była ewaluacja obu podejść na niezależnym zbiorze testowym. Porównano wyniki uzyskane przez najlepszy pojedynczy model oraz przez stworzony ensemble.

```python
# Sprawdzenie wyników
y_pred_single    = automl_single.predict(X_test)
y_pred_ensemble  = automl_ensemble.predict(X_test)

y_proba_single   = automl_single.predict_proba(X_test)
y_proba_ensemble = automl_ensemble.predict_proba(X_test)
```

✓ 0.1s Python

```python
acc_single   = balanced_accuracy_score(y_test_encoded, y_pred_single)
acc_ensemble = balanced_accuracy_score(y_test_encoded, y_pred_ensemble)

auc_single   = roc_auc_score(y_test_encoded, y_proba_single)
auc_ensemble = roc_auc_score(y_test_encoded, y_proba_ensemble)
```

✓ 0.0s Python

```python
print(f"Balanced accuracy Single Model:  {acc_single:.4f}")
print(f"Balanced accuracy Ensemble Model: {acc_ensemble:.4f}")

print(f"ROC AUC Single Model:  {auc_single:.4f}")
print(f"ROC AUC Ensemble Model: {auc_ensemble:.4f}")
```

✓ 0.0s Python

```
Balanced accuracy Single Model:  0.5154
Balanced accuracy Ensemble Model: 0.6316
ROC AUC Single Model:  0.7076
ROC AUC Ensemble Model: 0.7094
```

Przedstawione wyniki wskazują na znaczącą przewagę podejścia Ensemble w przypadku tego zbioru danych. Balanced Accuracy wzrosło z poziomu 0.5154 (Single) do 0.6316 (Ensemble).

## 4.2 Credit-G

Analogiczne obliczenia przeprowadzono dla drugiego zbioru danych – *Credit-G*, dotyczącego oceny ryzyka kredytowego. W tym przypadku:

- Tryb Single: najlepszym modelem okazał się Gradient Boosting (`gbm_1627`), osiągając na zbiorze testowym Balanced Accuracy równe 0.5810.
- Tryb Ensemble: komitet złożony z modeli GBM i XGBoost osiągnął wynik 0.5560.

W przeciwieństwie do pierwszego przykładu, tutaj lepsze rezultaty przyniósł pojedynczy, dobrze dobrany model.

# 5 Wnioski

Zbudowanie portfolio modeli na podstawie wyników benchmarku MementoML pozwoliło na efektywne wykorzystanie wiedzy o skuteczności różnych algorytmów bez konieczności przeprowadzania kosztownych eksperymentów od podstaw. Zastosowana w systemie `MiniAutoML` metoda selekcji, oparta na tworzeniu rankingu wyników kroswalidacji, umożliwia skuteczną ocenę i automatyczny wybór najlepszego rozwiązania dla nowego zbioru danych.

Przeprowadzone analizy potwierdziły, że skuteczność systemu jest ściśle uzależniona od jakości i różnorodności modeli zawartych w portfolio. Eksperymenty wykazały dwie kluczowe zależności:

1. Strategia Ensemble przynosi najlepsze rezultaty, gdy modele składowe są zróżnicowane (np. połączenie k-NN i Random Forest w zbiorze *Blood Transfusion*). Mechanizm *Soft Voting* pozwolił tam na znaczną redukcję błędu i poprawę wyniku *Balanced Accuracy* względem najlepszego pojedynczego modelu.

2. W sytuacjach, gdy modele w komitecie są zbyt jednorodne (np. wyłącznie modele oparte na drzewach w zbiorze *Credit-G*), ensembling nie gwarantuje poprawy wyników, a czasem może prowadzić do ich nieznacznego pogorszenia.

Ostatecznie, elastyczna architektura rozwiązania pozwalająca na wybór między trybem pojedynczym a komitetem sprawia, że system jest uniwersalnym narzędziem wstępnej eksploracji danych. Dzięki oparciu konfiguracji o zewnętrzny plik JSON, portfolio może być łatwo rozszerzane w przyszłości, co potencjalnie zwiększy szansę na znalezienie optymalnego modelu dla trudniejszych problemów klasyfikacyjnych.

Tabela 1: Modele w portfolio i wartości ich hiperparametrów

| **Model** | **Klasa** | **Hiperparametry** |
|---|---|---|
| `xgboost_1045` | `xgboost.XGBClassifier` | `booster=gbtree`<br>`subsample=0.867133233579807`<br>`max_depth=14`<br>`min_child_weight=1.39082881585674`<br>`colsample_bytree=0.565902150422335`<br>`colsample_bylevel=0.621011920087039`<br>`missing=0`<br>`n_estimators=109`<br>`learning_rate=0.0699401344515662`<br>`random_state=123`<br>`verbosity=0` |
| `xgboost_1049` | `xgboost.XGBClassifier` | `booster=gbtree`<br>`subsample=0.683306726277806`<br>`max_depth=7`<br>`min_child_weight=2.56567480883493`<br>`colsample_bytree=0.744482032582164`<br>`colsample_bylevel=0.859297586232424`<br>`missing=0`<br>`n_estimators=184`<br>`learning_rate=0.0617450318338613`<br>`random_state=123`<br>`verbosity=0` |
| `xgboost_1056` | `xgboost.XGBClassifier` | `booster=gbtree`<br>`subsample=0.966606075176969`<br>`max_depth=14`<br>`min_child_weight=1.07724400296566`<br>`colsample_bytree=0.952940160222352`<br>`colsample_bylevel=0.354075000621378`<br>`missing=0`<br>`n_estimators=282`<br>`learning_rate=0.0962582983602717`<br>`random_state=123`<br>`verbosity=0` |

| Model | Klasa | Hiperparametry |
|---|---|---|
| xgboost_1070 | xgboost.XGBClassifier | booster=gbtree<br>subsample=0.867935555521399<br>max_depth=8<br>min_child_weight=2.18921599683768<br>colsample_bytree=0.71023326292634<br>colsample_bylevel=0.690302629210055<br>missing=0<br>n_estimators=355<br>learning_rate=0.0324106605480031<br>random_state=123<br>verbosity=0 |
| xgboost_1079 | xgboost.XGBClassifier | booster=gbtree<br>subsample=0.693450994323939<br>max_depth=14<br>min_child_weight=1.32540148414612<br>colsample_bytree=0.373665936477482<br>colsample_bylevel=0.297132815979421<br>missing=0<br>n_estimators=268<br>learning_rate=0.057311228047118<br>random_state=123<br>verbosity=0 |
| xgboost_1109 | xgboost.XGBClassifier | booster=gbtree<br>subsample=0.92063701513689<br>max_depth=6<br>min_child_weight=1.01113620011384<br>colsample_bytree=0.771420868672431<br>colsample_bylevel=0.232633330486715<br>missing=0<br>n_estimators=757<br>learning_rate=0.0816841850176899<br>random_state=123<br>verbosity=0 |
| xgboost_1160 | xgboost.XGBClassifier | booster=gbtree<br>subsample=0.777657110360451<br>max_depth=10<br>min_child_weight=1.92624185774548<br>colsample_bytree=0.413046978227794<br>colsample_bylevel=0.892870987206698<br>missing=0<br>n_estimators=756<br>learning_rate=0.0403232295575335<br>random_state=123<br>verbosity=0 |
| gbm_1100 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=12,<br>learning_rate=0.0126652064734548,<br>subsample=0.746615828573704,<br>random_state=123 |
| gbm_1272 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=3,<br>learning_rate=0.0055493718844742,<br>subsample=0.814661961793899,<br>random_state=123 |

| Model | Klasa | Hiperparametry |
|---|---|---|
| gbm_1280 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=5,<br>learning_rate=0.0041592344641079,<br>subsample=0.602520916424692,<br>random_state=123 |
| gbm_1399 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=3,<br>learning_rate=0.0036416000430948,<br>subsample=0.735055895708501,<br>random_state=123 |
| gbm_1456 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=10,<br>learning_rate=0.0032459353881806,<br>subsample=0.749076421372592,<br>random_state=123 |
| gbm_1627 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=9,<br>learning_rate=0.0103241557434611,<br>subsample=0.665613202936947,<br>random_state=123 |
| gbm_1698 | sklearn.ensemble.<br>GradientBoostingClassifier | n_estimators=500, max_depth=5,<br>min_samples_leaf=21,<br>learning_rate=0.0065209006218249,<br>subsample=0.67775614913553,<br>random_state=123 |
| catboost_1002 | catboost.CatBoostClassifier | iterations=1000, depth=7,<br>l2_leaf_reg=5.27307284849919,<br>bagging_temperature=1.13389472512655,<br>learning_rate=0.0388196531746707,<br>random_seed=123, verbose=false,<br>allow_writing_files=false |
| catboost_1023 | catboost.CatBoostClassifier | iterations=692, depth=7,<br>l2_leaf_reg=3.48136531982443,<br>bagging_temperature=3.06071982652903,<br>learning_rate=0.35291108770475,<br>random_seed=123, verbose=false,<br>allow_writing_files=false |
| catboost_1062 | catboost.CatBoostClassifier | iterations=107, depth=7,<br>l2_leaf_reg=8.76092083835427,<br>bagging_temperature=1.11388447314018,<br>learning_rate=0.0018349654487865,<br>random_seed=123, verbose=false,<br>allow_writing_files=false |
| catboost_1071 | catboost.CatBoostClassifier | iterations=216, depth=10,<br>l2_leaf_reg=5.58699261457645,<br>bagging_temperature=0.37228198624532,<br>learning_rate=0.0066953271086871,<br>random_seed=123, verbose=false,<br>allow_writing_files=false |
| catboost_1074 | catboost.CatBoostClassifier | iterations=356, depth=7,<br>l2_leaf_reg=2.78350780890656,<br>bagging_temperature=0.811784329621578,<br>learning_rate=0.560753418765299,<br>random_seed=123, verbose=false,<br>allow_writing_files=false |

| Model | Klasa | Hiperparametry |
|---|---|---|
| catboost_1078 | catboost.CatBoostClassifier | iterations=175, depth=8, l2_leaf_reg=0.98059299074675, bagging_temperature=1.71692322580631, learning_rate=0.41470411140348, random_seed=123, verbose=false, allow_writing_files=false |
| catboost_1096 | catboost.CatBoostClassifier | iterations=1000, depth=10, l2_leaf_reg=7.40502179088432, bagging_temperature=2.11991119056939, learning_rate=0.0679622874811962, random_seed=123, verbose=false, allow_writing_files=false |
| glmnet_1171 | sklearn.linear_model. LogisticRegression | l1_ratio=0.729730096645653, C=376.40659441239603, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1291 | sklearn.linear_model. LogisticRegression | l1_ratio=0.867228663060814, C=303.28923951521966, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1595 | sklearn.linear_model. LogisticRegression | l1_ratio=0.805811213329434, C=211.78771937924165, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1716 | sklearn.linear_model. LogisticRegression | l1_ratio=0.724161460762843, C=271.07652733704543, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1881 | sklearn.linear_model. LogisticRegression | l1_ratio=0.912751579657197, C=422.2581622237663, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1889 | sklearn.linear_model. LogisticRegression | l1_ratio=0.829109395388514, C=229.45663442219336, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| glmnet_1966 | sklearn.linear_model. LogisticRegression | l1_ratio=0.900965292239562, C=393.03284034483806, penalty=elasticnet, solver=saga, max_iter=5000, random_state=123 |
| kknn_1031 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=34, p=2 |
| kknn_1939 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=33, p=2 |
| kknn_1952 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=32, p=2 |
| kknn_1954 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=31, p=2 |
| kknn_1978 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=30, p=2 |
| kknn_1994 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=28, p=2 |
| kknn_1999 | sklearn.neighbors. KNeighborsClassifier | n_neighbors=29, p=2 |

| Model | Klasa | Hiperparametry |
|---|---|---|
| ranger_1444 | sklearn.ensemble. RandomForestClassifier | n_estimators=600, min_samples_leaf=2, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1561 | sklearn.ensemble. RandomForestClassifier | n_estimators=200, min_samples_leaf=2, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1644 | sklearn.ensemble. RandomForestClassifier | n_estimators=400, min_samples_leaf=2, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1669 | sklearn.ensemble. RandomForestClassifier | n_estimators=500, min_samples_leaf=2, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1715 | sklearn.ensemble. RandomForestClassifier | n_estimators=300, min_samples_leaf=2, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1780 | sklearn.ensemble. RandomForestClassifier | n_estimators=800, min_samples_leaf=1, bootstrap=false, random_state=123, n_jobs=-1 |
| ranger_1958 | sklearn.ensemble. RandomForestClassifier | n_estimators=300, min_samples_leaf=1, bootstrap=false, random_state=123, n_jobs=-1 |
| randomForest_1328 | sklearn.ensemble. RandomForestClassifier | n_estimators=200, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1364 | sklearn.ensemble. RandomForestClassifier | n_estimators=800, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1744 | sklearn.ensemble. RandomForestClassifier | n_estimators=300, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1780 | sklearn.ensemble. RandomForestClassifier | n_estimators=400, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1826 | sklearn.ensemble. RandomForestClassifier | n_estimators=600, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1861 | sklearn.ensemble. RandomForestClassifier | n_estimators=700, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |
| randomForest_1906 | sklearn.ensemble. RandomForestClassifier | n_estimators=500, min_samples_leaf=1, bootstrap=true, random_state=123, n_jobs=-1 |