XAutoML Project 1

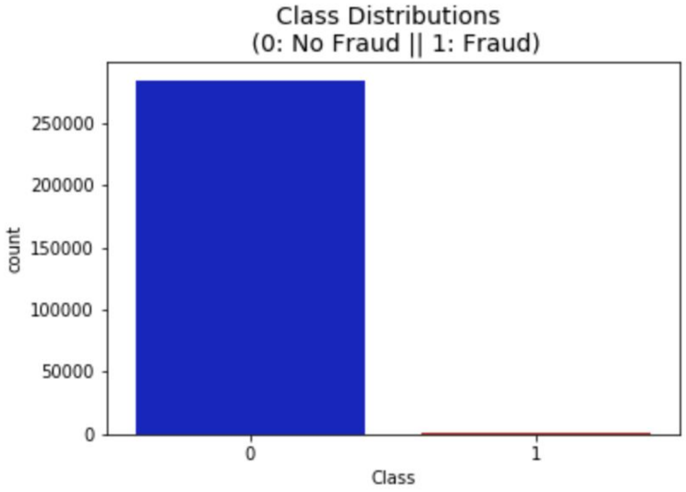
Olexandr Syzonov, Mykyta Luzan, Elena Novikova

Outline

1. Datasets description
2. Baseline
3. Search space configuration
4. AutoML pipeline
5. Final comparison

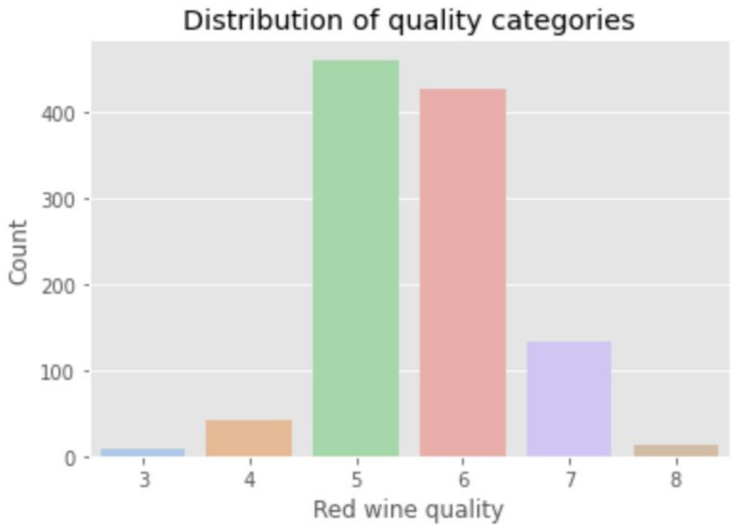
Dataset: Credit Card Fraud Detection

- Highly unbalanced: 0.172% of positive class
- 30 numerical input variables
- Input variables: Time, Amount and PCA features
- 284807 observations
- No missing values



Dataset: Red Wine Quality

- 6 categories that were converted to binary label
- 12 numerical input variables
- Input variables: pH, alcohol, acidity, etc
- 1599 observations
- No missing values



Baseline: Evaluation

- Stratification to deal with unbalanced data
- Train/val and test hold-out
- 5 Fold Cross Validation over train/val subsets
- Metric: F1-score

```
results = np.zeros((len(models), N_FOLDS))
cur_fold = 0
np.random.seed(RANDOM_SEED)

for train_index, test_index in skf.split(X_train, y_train):
    X_train_cv, X_val = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_cv, y_val = y_train.iloc[train_index], y_train.iloc[test_index]

    for i, clf in tqdm(enumerate(models)):
        if clf not in parallelize:
            clf = clf().fit(X_train_cv, y_train_cv)
        else:
             clf = clf(n_jobs=-1).fit(X_train_cv, y_train_cv)
        score = f1_score(y_val, clf.predict(X_val))
        results[i, cur_fold] = score
    cur_fold += 1
```

Baseline: Models

Classifier	fold_1	fold_2	fold_3	fold_4	fold_5
DummyClassifier	0.0000	0.0000	0.0000	0.0000	0.0000
LinearDiscriminantAnalysis	0.8138	0.8310	0.7973	0.8387	0.7945
QuadraticDiscriminantAnalysis	0.1177	0.1155	0.1050	0.1241	0.1108
LogisticRegression	0.6626	0.6887	0.7250	0.7013	0.6708
DecisionTreeClassifier	0.7532	0.7895	0.7843	0.7086	0.8052
KNeighborsClassifier	0.0976	0.1628	0.1395	0.1395	0.0494
SVC	0.0000	0.0000	0.0000	0.0000	0.0000
RandomForestClassifier	0.8429	0.8467	0.8276	0.8477	0.8611
ExtraTreesClassifier	0.8451	0.8633	0.8571	0.8533	0.8531
GaussianNB	0.2103	0.2519	0.2217	0.2589	0.2361
MLPClassifier	0.5565	0.5455	0.2453	0.0488	0.2375
PassiveAggressiveClassifier	0.0241	0.0000	0.0964	0.0000	0.0000

Credit Card Fraud Dataset

Classifier	fold_1	fold_2	fold_3	fold_4	fold_5
DummyClassifier	0.5942	0.5474	0.5468	0.4924	0.5115
LinearDiscriminantAnalysis	0.7849	0.7566	0.7581	0.7407	0.7518
QuadraticDiscriminantAnalysis	0.8100	0.6889	0.7347	0.7405	0.7577
LogisticRegression	0.7910	0.7564	0.7286	0.7388	0.7528
DecisionTreeClassifier	0.7941	0.7560	0.6940	0.7639	0.7518
KNeighborsClassifier	0.6567	0.6716	0.6809	0.7174	0.6816
SVC	0.7256	0.7108	0.6982	0.7239	0.7009
RandomForestClassifier	0.8271	0.7823	0.7970	0.8185	0.8223
ExtraTreesClassifier	0.8582	0.7698	0.8000	0.8172	0.8281
GaussianNB	0.7519	0.6641	0.7418	0.7528	0.7391
MLPClassifier	0.7755	0.7247	0.7397	0.7569	0.7410
PassiveAggressiveClassifier	0.6990	0.6972	0.7040	0.6972	0.6943

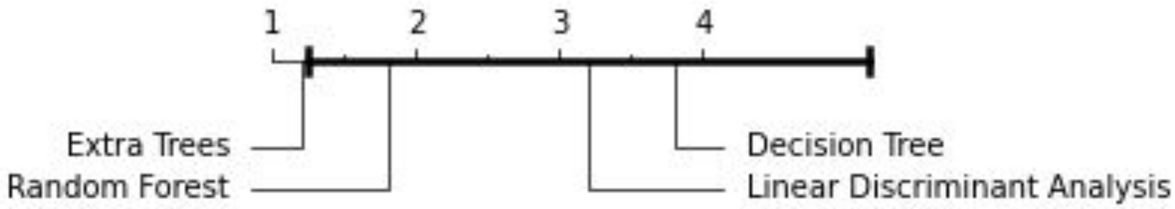
Red wine Quality Dataset

Baseline: Friedman’s and Post-hoc Nemenyi tests

H0: Same distributions of scores over 5 folds

H1: Different distribution -> at least one is different

	r1	r2	r3	r4	r5	mean
LinearDiscriminantAnalysis	3.0	3.0	3.0	3.0	4.0	3.2
DecisionTreeClassifier	4.0	4.0	4.0	4.0	3.0	3.8
ExtraTreesClassifier	1.0	1.0	1.0	1.0	2.0	1.2
RandomForestClassifier	2.0	2.0	2.0	2.0	1.0	1.8



Baseline: McNemar test

H0: both models have the same performance

H1: performances of the two models are not equal

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	56939	2
nr_incorrect_clf2	2	19

B + C = 4

Credit Card Fraud Dataset

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	247	10
nr_incorrect_clf2	9	54

B + C = 19

Red wine Quality Dataset

Extra Trees Classifier

sklearn.ensemble.ExtraTreesClassifier(n_estimators=100, *, **criterion**='gini', **max_depth**=None, **min_samples_split**=2, **min_samples_leaf**=1, min_weight_fraction_leaf=0.0, **max_features**='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, **bootstrap**=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, **class_weight**=None, ccp_alpha=0.0, max_samples=None)

Search Space

```
search_space={
    'max_depth': hp.randint('max_depth', 1, 200),
    'min_samples_split':hp.randint('min_samples_split', 2, 5),
    'min_samples_leaf':hp.randint('min_samples_leaf', 1, 20),
    'criterion':hp.choice('criterion', ['gini', 'entropy']),
    'max_features':hp.choice('max_features', ['sqrt', 'log2', 0.9]),
    'bootstrap':hp.choice('bootstrap', [True, False]),
    "class_weight": hp.choice('class_weight', ['balanced', 'balanced_subsample'])
}
```

Sequential Model Based Optimization

- Objective function: Extra Trees Classifier f1 score
- Domain space: Search Space
- Hyperparameter optimization function: Tree-structured Parzen Estimator
- Trials: f1 score

Hyperopt: 10 iterations

Random Search

- Time: 4min 24s
- Time per trial: 26.45s/trial
- Max f1 score: 0.15
- Number of Trials to attain the max score: 10

TPE

- Time: 2min 39s
- Time per trial: 15.99s/trial
- Max f1 score: 0.11
- Number of Trials to attain the max score: 9

```python
def optmimize(datasets, f1_baseline, budget_list, opt_method_list, use_cv=True, early_stopping=True):
    X_train, X_test, y_train, y_test = datasets
    final_params = {}
    final_scores = {}
    try:
        for budget in budget_list:
            for opt_method in opt_method_list:
                print((budget, opt_method))
                # tune model
                params, _ = run_experiment(search_space, budget, use_cv=use_cv, method=opt_method)
                final_params[(budget, opt_method)] = params

                # evaluate model
                clf = ExtraTreesClassifier(random_state=42, **params, n_jobs=-1).fit(X_train, y_train)
                y_pred_opt = clf.predict(X_test)
                f1 = f1_score(y_test, y_pred_opt)
                final_scores[(budget, opt_method)] = f1
                print(f"F1-score: {f1}")

                # early stopping condition, because we are looking for the smallest time budget
                if early_stopping and f1 > f1_baseline:
                    A = ((y_pred_baseline == y_test) & (y_pred_opt == y_test)).sum()
                    B = ((y_pred_baseline != y_test) & (y_pred_opt == y_test)).sum()
                    C = ((y_pred_baseline == y_test) & (y_pred_opt != y_test)).sum()
                    D = ((y_pred_baseline != y_test) & (y_pred_opt != y_test)).sum()
                    result = mcnemar([[A, B], [C, D]])
                    alpha = 0.05
                    if B + C > 20 and result.pvalue < alpha:
                        print(f"Model with time budget {budget} and {opt_method} optimization algo beat the baseline!")
                        return (budget, opt_method), final_params, final_scores
                    else:
                        print("F1 is better, but not statistically")
    except KeyboardInterrupt:
        print("Interrupted.")

    print("No model outperformed the baseline")
    return None, final_params, final_scores
```

```python
def optmimize(datasets, f1_baseline, budget_list, opt_method_list, use_cv=True, early_stopping=True):
    X_train, X_test, y_train, y_test = datasets
    final_params = {}
    final_scores = {}
    try:
        for budget in budget_list:
            for opt_method in opt_method_list:
                print((budget, opt_method))
                # tune model
                params, _ = run_experiment(search_space, budget, use_cv=use_cv, method=opt_method)
                final_params[(budget, opt_method)] = params

                # evaluate model
                clf = ExtraTreesClassifier(random_state=42, **params, n_jobs=-1).fit(X_train, y_train)
                y_pred_opt = clf.predict(X_test)
                f1 = f1_score(y_test, y_pred_opt)
                final_scores[(budget, opt_method)] = f1
                print(f"F1-score: {f1}")

                # early stopping condition, because we are looking for the smallest time budget
                if early_stopping and f1 > f1_baseline:
                    A = ((y_pred_baseline == y_test) & (y_pred_opt == y_test)).sum()
                    B = ((y_pred_baseline != y_test) & (y_pred_opt == y_test)).sum()
                    C = ((y_pred_baseline == y_test) & (y_pred_opt != y_test)).sum()
                    D = ((y_pred_baseline != y_test) & (y_pred_opt != y_test)).sum()
                    result = mcnemar([[A, B], [C, D]])
                    alpha = 0.05
                    if B + C > 20 and result.pvalue < alpha:
                        print(f"Model with time budget {budget} and {opt_method} optimization algo beat the baseline!")
                        return (budget, opt_method), final_params, final_scores
                    else:
                        print("F1 is better, but not statistically")
    except KeyboardInterrupt:
        print("Interrupted.")

    print("No model outperformed the baseline")
    return None, final_params, final_scores
```

```python
def optmimize(datasets, f1_baseline, budget_list, opt_method_list, use_cv=True, early_stopping=True):
    X_train, X_test, y_train, y_test = datasets
    final_params = {}
    final_scores = {}
    try:
        for budget in budget_list:
            for opt_method in opt_method_list:
                print((budget, opt_method))
                # tune model
                params, _ = run_experiment(search_space, budget, use_cv=use_cv, method=opt_method)
                final_params[(budget, opt_method)] = params

                # evaluate model
                clf = ExtraTreesClassifier(random_state=42, **params, n_jobs=-1).fit(X_train, y_train)
                y_pred_opt = clf.predict(X_test)
                f1 = f1_score(y_test, y_pred_opt)
                final_scores[(budget, opt_method)] = f1
                print(f"F1-score: {f1}")

                # early stopping condition, because we are looking for the smallest time budget
                if early_stopping and f1 > f1_baseline:
                    A = ((y_pred_baseline == y_test) & (y_pred_opt == y_test)).sum()
                    B = ((y_pred_baseline != y_test) & (y_pred_opt == y_test)).sum()
                    C = ((y_pred_baseline == y_test) & (y_pred_opt != y_test)).sum()
                    D = ((y_pred_baseline != y_test) & (y_pred_opt != y_test)).sum()
                    result = mcnemar([[A, B], [C, D]])
                    alpha = 0.05
                    if B + C > 20 and result.pvalue < alpha:
                        print(f"Model with time budget {budget} and {opt_method} optimization algo beat the baseline!")
                        return (budget, opt_method), final_params, final_scores
                    else:
                        print("F1 is better, but not statistically")
    except KeyboardInterrupt:
        print("Interrupted.")

    print("No model outperformed the baseline")
    return None, final_params, final_scores
```

```python
def optmimize(datasets, f1_baseline, budget_list, opt_method_list, use_cv=True, early_stopping=True):
    X_train, X_test, y_train, y_test = datasets
    final_params = {}
    final_scores = {}
    try:
        for budget in budget_list:
            for opt_method in opt_method_list:
                print((budget, opt_method))
                # tune model
                params, _ = run_experiment(search_space, budget, use_cv=use_cv, method=opt_method)
                final_params[(budget, opt_method)] = params

                # evaluate model
                clf = ExtraTreesClassifier(random_state=42, **params, n_jobs=-1).fit(X_train, y_train)
                y_pred_opt = clf.predict(X_test)
                f1 = f1_score(y_test, y_pred_opt)
                final_scores[(budget, opt_method)] = f1
                print(f"F1-score: {f1}")

                # early stopping condition, because we are looking for the smallest time budget
                if early_stopping and f1 > f1_baseline:
                    A = ((y_pred_baseline == y_test) & (y_pred_opt == y_test)).sum()
                    B = ((y_pred_baseline != y_test) & (y_pred_opt == y_test)).sum()
                    C = ((y_pred_baseline == y_test) & (y_pred_opt != y_test)).sum()
                    D = ((y_pred_baseline != y_test) & (y_pred_opt != y_test)).sum()
                    result = mcnemar([[A, B], [C, D]])
                    alpha = 0.05
                    if B + C > 20 and result.pvalue < alpha:
                        print(f"Model with time budget {budget} and {opt_method} optimization algo beat the baseline!")
                        return (budget, opt_method), final_params, final_scores
                    else:
                        print("F1 is better, but not statistically")
    except KeyboardInterrupt:
        print("Interrupted.")

    print("No model outperformed the baseline")
    return None, final_params, final_scores
```

```python
def optmimize(datasets, f1_baseline, budget_list, opt_method_list, use_cv=True, early_stopping=True):
    X_train, X_test, y_train, y_test = datasets
    final_params = {}
    final_scores = {}
    try:
        for budget in budget_list:
            for opt_method in opt_method_list:
                print((budget, opt_method))
                # tune model
                params, _ = run_experiment(search_space, budget, use_cv=use_cv, method=opt_method)
                final_params[(budget, opt_method)] = params

                # evaluate model
                clf = ExtraTreesClassifier(random_state=42, **params, n_jobs=-1).fit(X_train, y_train)
                y_pred_opt = clf.predict(X_test)
                f1 = f1_score(y_test, y_pred_opt)
                final_scores[(budget, opt_method)] = f1
                print(f"F1-score: {f1}")

                # early stopping condition, because we are looking for the smallest time budget
                if early_stopping and f1 > f1_baseline:
                    A = ((y_pred_baseline == y_test) & (y_pred_opt == y_test)).sum()
                    B = ((y_pred_baseline != y_test) & (y_pred_opt == y_test)).sum()
                    C = ((y_pred_baseline == y_test) & (y_pred_opt != y_test)).sum()
                    D = ((y_pred_baseline != y_test) & (y_pred_opt != y_test)).sum()
                    result = mcnemar([[A, B], [C, D]])
                    alpha = 0.05
                    if B + C > 20 and result.pvalue < alpha:
                        print(f"Model with time budget {budget} and {opt_method} optimization algo beat the baseline!")
                        return (budget, opt_method), final_params, final_scores
                    else:
                        print("F1 is better, but not statistically")
    except KeyboardInterrupt:
        print("Interrupted.")

    print("No model outperformed the baseline")
    return None, final_params, final_scores
```

Monitoring: Credit Scores Fraud

```
(100, 'tpe')
100%|██████████| 100/100 [15:33<00:00,  9.33s/trial, best loss: -0.819672131147541]
F1-score: 0.88268156424581
(200, 'tpe')
100%|██████████| 200/200 [34:48<00:00, 10.44s/trial, best loss: -0.8153846153846154]
F1-score: 0.8936170212765957
F1 is better, but not statistically
(500, 'tpe')
 45%|████▌     | 225/500 [40:08<49:04, 10.71s/trial, best loss: -0.8153846153846154]
Interrupted.
```

```
(500, 'tpe')
100%|██████████| 500/500 [1:30:26<00:00, 10.85s/trial, best loss: -0.8292682926829268]
F1-score: 0.8777777777777778
(1000, 'tpe')
100%|██████████| 1000/1000 [2:33:06<00:00,  9.19s/trial, best loss: -0.8292682926829268]
F1-score: 0.8764044943820225
(2000, 'tpe')
 60%|█████▉    | 1198/2000 [3:18:57<2:13:11,  9.96s/trial, best loss: -0.8153846153846154]
Interrupted.
No model outperformed the baseline
```

Final results: Credit Scores Fraud

Baseline F1: 0.8913 Optimized best F1: 0.8936

Final results: Credit Scores Fraud

Baseline F1: 0.8913 Optimized best F1: 0.8936

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	56940	2
nr_incorrect_clf2	2	18

Final results: Credit Scores Fraud

Baseline F1: 0.8913 Optimized best F1: 0.8936

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	56940	2
nr_incorrect_clf2	2	18

	r1	r2	r3	r4	r5	mean
0	3.0	2.0	2.0	1.5	2.0	2.1
1	2.0	1.0	1.0	1.5	1.0	1.3
2	1.0	3.0	3.0	3.0	3.0	2.6

Final results: Credit Scores Fraud

Baseline F1: 0.8913 Optimized best F1: 0.8936

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	56940	2
nr_incorrect_clf2	2	18

	r1	r2	r3	r4	r5	mean
0	3.0	2.0	2.0	1.5	2.0	2.1
1	2.0	1.0	1.0	1.5	1.0	1.3
2	1.0	3.0	3.0	3.0	3.0	2.6

```
final_params[(200, 'tpe')]
```

```
{'bootstrap': False,
 'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 129,
 'max_features': 0.9,
 'min_samples_leaf': 3,
 'min_samples_split': 2}
```

Monitoring: Red Wine Quality

```
(10, 'random')
100%|██████████| 10/10 [00:18<00:00,  1.80s/trial, best loss: -0.82168574877736]
F1-score: 0.8367952522255192
F1 is better, but not statistically
(20, 'random')
100%|██████████| 20/20 [00:39<00:00,  1.97s/trial, best loss: -0.8197646891820621]
F1-score: 0.8143712574850298
(50, 'random')
100%|██████████| 50/50 [01:30<00:00,  1.80s/trial, best loss: -0.81801586794]
F1-score: 0.8214285714285714
(100, 'random')
100%|██████████| 100/100 [02:57<00:00,  1.77s/trial, best loss: -0.8251873077524987]
F1-score: 0.8245614035087719
(200, 'random')
100%|██████████| 200/200 [06:20<00:00,  1.90s/trial, best loss: -0.8251873077524987]
F1-score: 0.8245614035087719
(500, 'random')
100%|██████████| 500/500 [16:16<00:00,  1.95s/trial, best loss: -0.82168574877736]
F1-score: 0.8367952522255192
F1 is better, but not statistically
(1000, 'random')
  1%|          | 10/1000 [00:20<34:38,  2.10s/trial, best loss: -0.8251873077524987]
Interrupted.
```

Final results: Red Wine Quality

Baseline F1: 0.8259 Optimized best F1: 0.8367

Final results: Red Wine Quality

Baseline F1: 0.8259 Optimized best F1: 0.8367

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	253	12
nr_incorrect_clf2	8	47

Final results: Red Wine Quality

Baseline F1: 0.8259 Optimized best F1: 0.8367

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	253	12
nr_incorrect_clf2	8	47

	r1	r2	r3	r4	r5	mean
0	1.5	1.5	1.5	1.5	1.5	1.5
1	1.5	1.5	1.5	1.5	1.5	1.5
2	3.0	3.0	3.0	3.0	3.0	3.0

Final results: Red Wine Quality

Baseline F1: 0.8259 Optimized best F1: 0.8367

	nr_correct_clf1	nr_incorrect_cl1
nr_correct_clf2	253	12
nr_incorrect_clf2	8	47

	r1	r2	r3	r4	r5	mean
0	1.5	1.5	1.5	1.5	1.5	1.5
1	1.5	1.5	1.5	1.5	1.5	1.5
2	3.0	3.0	3.0	3.0	3.0	3.0

```
final_params[(10, 'random')]
```

```
({'bootstrap': False,
  'class_weight': 'balanced',
  'criterion': 'entropy',
  'max_depth': 55,
  'max_features': 0.9,
  'min_samples_leaf': 1,
  'min_samples_split': 2},
```

THANKS!

Any questions?