Paper Evaluation

January 1, 2021

[1]:
```
import os
import shutil

import pandas as pd
import logpred_method as experiment

from sklearn.model_selection import train_test_split


# Use "FRACTION = None" for full dataset
FRACTION: float = None


# lr: Linear Regression
# ab: Ada Boost
# rf: Random Forest
# dt: Decision Tree
# et: Extra Trees
MODELS = ["lr", "ab", "rf", "dt", "et"]

# You can ignore features on the experiment
IGNORED_FEATURES = ["tryCatchQty_class", "tryCatchQty_method"]


# Hyperparameter tuning
TUNING_ENABLED = True


# Stores estimators and feature importances across experiments
ESTIMATORS = {}
FEATURE_IMPORTANCES = {}
```

1 Utilities

[2]:
```
def merge_scores(scores):
    """
    Returns a merged score from a sequence of scores.
    This is useful to see scores as Pandas DataFrames.

    Example:
        in  - [{"a": 1, "b": 2}, {"a": 10, "b": 20}]
        out - {"a": [1, 10], "b": [2, 20]}
    """
    merged = {k:[] for k in scores[0].keys()}
    for score in scores:
        for k, v in score.items():
            merged[k].append(v)

    return merged
```

2 Experiment CSV and Output directory

[3]:
```
csv_path = os.path.abspath(os.path.join("out", "dataset", "adyen-main",␣
↪"dataset_full.csv"))

X, y = experiment.load_dataset(csv_path, drops=IGNORED_FEATURES,␣
↪fraction=FRACTION)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=experiment.RANDOM_SEED
)

output_dir = os.path.abspath(os.path.join("out", "ml",␣
↪f"evaluation-tuning-{TUNING_ENABLED}"))
if os.path.exists(output_dir):
    shutil.rmtree(output_dir)
os.makedirs(output_dir)
```

3 RQ 1. What is the performance of machine learning models in predicting log placement in a large-scale enterprise system?

[4]:
```
from sklearn.dummy import DummyClassifier


def rq1():
    scores = []
    for model in MODELS:
        out = experiment.run(
```

```
            model,
            X_train=X_train,
            X_test=X_test,
            y_train=y_train,
            y_test=y_test,
            output_to=os.path.join(output_dir, f"rq1-{model}.log"),
            tuning_enabled=TUNING_ENABLED
        )
        estimator, score, fi = out
        scores.append(score)

        # Save to the global state this run
        ESTIMATORS[model] = estimator
        FEATURE_IMPORTANCES[model] = fi


    # Dummy baselines
    biased_guess = DummyClassifier(
        strategy="stratified",
        random_state=experiment.RANDOM_SEED
    )
    biased_guess.fit(X_train, y_train)
    bg_score = experiment.make_score(y_test, biased_guess.predict(X_test))
    scores.append(bg_score)

    random_guess = DummyClassifier(
        strategy="uniform",
        random_state=experiment.RANDOM_SEED
    )
    random_guess.fit(X_train, y_train)
    rg_score = experiment.make_score(y_test, random_guess.predict(X_test))
    scores.append(rg_score)

    return scores

rq1_scores = rq1()
```

[5]:
```
# data alignment for later merging
dummy_models = ["BG", "RG"]
for i, baseline_score in enumerate(rq1_scores[-2:]):
    baseline_score["balancing"] = "-"
    baseline_score["model"] = dummy_models[i]
    if "mean_fit_time" in rq1_scores[0].keys():
        baseline_score["mean_fit_time"] = 0
        baseline_score["std_fit_time"] = 0
        baseline_score["mean_test_score"] = 0
        baseline_score["std_test_score"] = 0
```

3.1 Results

[6]:
```
results_rq1 = pd.DataFrame.from_dict(merge_scores(rq1_scores)).
 ↪set_index(["model"])
results_rq1.reset_index().to_csv(
    os.path.join(output_dir, "rq1-results.csv"),
    index=False,
)
results_rq1["acc prec recall tn fp fn tp total".split(" ")]
```

[6]:

model	acc	prec	recall	tn	fp	fn	tp	total
lr	0.670649	0.650672	0.357218	56251	910	3050	1695	61906
ab	0.699748	0.638728	0.419178	56036	1125	2756	1989	61906
rf	0.794597	0.805250	0.601264	56471	690	1892	2853	61906
dt	0.736441	0.574207	0.503899	55388	1773	2354	2391	61906
et	0.765820	0.736127	0.547945	56229	932	2145	2600	61906
BG	0.501791	0.079933	0.080506	52764	4397	4363	382	61906
RG	0.498704	0.076281	0.497155	28595	28566	2386	2359	61906

4 RQ 2. What is the impact of different class balancing strategies on prediction?

[7]:
```
# Similar to rq1 but we include sampling in the experiment now.
def rq2():
    scores = []
    for model in MODELS:
        for balancing in ["smote", "rus"]:
            out = experiment.run(
                model,
                X_train=X_train,
                X_test=X_test,
                y_train=y_train,
                y_test=y_test,
                balancing=balancing,
                output_to=os.path.join(output_dir, f"rq2-{model}-{balancing}.
 ↪log"),
                tuning_enabled=TUNING_ENABLED
            )
            estimator, score, fi = out
            scores.append(score)

            # Save to the global state this run
            key = f"{model}-{balancing}"
            ESTIMATORS[key] = estimator
            FEATURE_IMPORTANCES[key] = fi
```

```
    return scores

rq2_scores = rq2()
```

4.1 Results

[8]:
```
results_rq2 = pd.DataFrame.from_dict(merge_scores(rq2_scores)).
 ↪set_index(["model", "balancing"])
results_rq2.reset_index().to_csv(
    os.path.join(output_dir, "rq2-results.csv"),
    index=False,
)
relevant_cols = "acc prec recall tn fp fn tp".split(" ")
results_rq2[relevant_cols]
```

[8]:

model	balancing	acc	prec	recall	tn	fp	fn	tp
lr	smote	0.875455	0.373657	0.872287	50223	6938	606	4139
	rus	0.876464	0.367216	0.878609	49977	7184	576	4169
ab	smote	0.874992	0.307692	0.922234	47315	9846	369	4376
	rus	0.887841	0.355694	0.912961	49314	7847	413	4332
rf	smote	0.892943	0.491093	0.859852	52933	4228	665	4080
	rus	0.918283	0.401525	0.954689	50409	6752	215	4530
dt	smote	0.859116	0.394763	0.822972	51174	5987	840	3905
	rus	0.885030	0.354974	0.906849	49342	7819	442	4303
et	smote	0.899221	0.446087	0.890200	51916	5245	521	4224
	rus	0.911649	0.372387	0.957218	49506	7655	203	4542

Comparative result to the baseline (no balancing). Positive value indicates improvement.

[9]:
```
results_rq2_rel = results_rq2.loc[MODELS, relevant_cols] - results_rq1.
 ↪loc[MODELS, relevant_cols]
results_rq2_rel.reset_index().to_csv(
    os.path.join(output_dir, "rq2-results-relative.csv"),
    index=False
)
results_rq2_rel
```

[9]:

model	balancing	acc	prec	recall	tn	fp	fn	tp
lr	smote	0.204806	-0.277015	0.515068	-6028	6028	-2444	2444
	rus	0.205815	-0.283456	0.521391	-6274	6274	-2474	2474
ab	smote	0.175243	-0.331036	0.503056	-8721	8721	-2387	2387
	rus	0.188093	-0.283034	0.493783	-6722	6722	-2343	2343
rf	smote	0.098346	-0.314157	0.258588	-3538	3538	-1227	1227
	rus	0.123687	-0.403725	0.353425	-6062	6062	-1677	1677

```
dt    smote      0.122676 -0.179444  0.319073 -4214  4214 -1514  1514
      rus        0.148590 -0.219233  0.402950 -6046  6046 -1912  1912
et    smote      0.133401 -0.290040  0.342255 -4313  4313 -1624  1624
      rus        0.145829 -0.363740  0.409273 -6723  6723 -1942  1942
```

5 RQ 3. What are the most recurring relevant features across models?

[10]:
```python
def rank_to_df(rank, top=3):
    cols = ["total"] + [i+1 for i in range(top)]
    data = pd.DataFrame.from_records(
        [(name, sum(count[:top]), *count[:top]) for name, count in rank.
 ↪items()],
        columns=["feature"] + cols
    )
    return data[data["total"] > 0].sort_values(by=cols, ascending=False)


def feature_importance_rank(selected_models):
    rank = {}
    for model in selected_models:
        ordered_features = sorted(
            FEATURE_IMPORTANCES[model],
            key=lambda pair: abs(pair[1]),
            reverse=True
        )
        for pos, feature_pair, in enumerate(ordered_features):
            feature = feature_pair[0]
            if feature not in rank.keys():
                rank[feature] = [0 for i in range(len(ordered_features))]
            rank[feature][pos] += 1
    return rank
```

5.1 Results

[11]:
```
fi = rank_to_df(
    feature_importance_rank(
        FEATURE_IMPORTANCES.keys()
    ),
    top=5
)
fi.to_csv(
    os.path.join(output_dir, "rq3-results.csv"),
    index=False
)
fi
```

[11]:

	feature	total	1	2	3	4	5
3	maxNestedBlocks	14	11	1	0	2	0
23	loc_method	9	1	2	1	3	2
50	methodsInvokedQty	8	0	2	1	1	4
18	cbo_method	6	1	2	2	1	0
9	uniqueWordsQty_method	6	0	1	3	2	0
27	wmc_method	5	0	3	1	0	1
40	variablesQty_method	4	0	1	0	1	2
2	type_interface	3	1	0	2	0	0
49	rfc_method	3	0	0	3	0	0
0	constructor_True	2	1	1	0	0	0
1	type_enum	2	0	1	0	1	0
22	returnsQty	2	0	0	1	1	0
12	cbo_class	2	0	0	0	1	1
4	constructor_False	2	0	0	0	0	2
7	publicMethodsQty	1	0	1	0	0	0
13	type_anonymous	1	0	0	1	0	0
10	abstractMethodsQty	1	0	0	0	1	0
47	publicFieldsQty	1	0	0	0	1	0
5	loopQty_method	1	0	0	0	0	1
20	loc_class	1	0	0	0	0	1
34	lcom	1	0	0	0	0	1

6 RQ 4. How well a model trained with open-source data can generalize to the context of a large-scale enterprise system?

[12]:
```
def selected_apache_projects():
    """
    Returns the name of the selected Apache projects as listed in the "out/
↪selection" directory.
    """
    selection_dir = os.path.abspath(os.path.join("out", "selection"))
    return sorted([
        selected.replace(".sh", "")
```

```
        for selected in os.listdir(selection_dir)
        if selected.endswith(".sh")
    ])


def load_X_y(project: str):
    dataset_path = os.path.abspath(
        os.path.join("out", "dataset", project, "dataset_full.csv")
    )
    X_apache, y_apache = experiment.load_dataset(
        dataset_path, drops=IGNORED_FEATURES
    )
    assert X.shape[1] == X.shape[1]

    return X_apache, y_apache


APACHE_PROJECTS = {
    project: load_X_y(project)
    for project in selected_apache_projects()
}

assert len(APACHE_PROJECTS) == 29
```

[13]:
```
for k, v in APACHE_PROJECTS.items():
    print(f"{k:20} {str(v[0].shape):>15}")
```

```
accumulo              (25458, 63)
ambari                (21997, 63)
archiva                (5995, 63)
bookkeeper            (12711, 63)
cloudstack            (52390, 63)
commons-beanutils      (1176, 63)
cxf                   (33589, 63)
fluo                   (2094, 63)
giraph                 (8039, 63)
helix                  (6787, 63)
ignite                (65181, 63)
jmeter                 (8597, 63)
knox                   (6821, 63)
lens                   (6231, 63)
metamodel              (4122, 63)
myfaces-tobago         (3866, 63)
nutch                  (3321, 63)
oodt                   (6933, 63)
oozie                  (8821, 63)
openmeetings           (4839, 63)
```

```
reef                    (6150, 63)
sqoop                   (3080, 63)
storm                  (24208, 63)
syncope                (14915, 63)
tez                     (8947, 63)
thrift                  (1797, 63)
tomcat                 (23789, 63)
zeppelin               (10953, 63)
zookeeper               (5279, 63)
```

6.1 Learning from all Apache projects

[14]:
```
X_apache_all = pd.concat(
    [X_apache for X_apache, _ in APACHE_PROJECTS.values()],
    ignore_index=True,
)
y_apache_all = pd.concat(
    [y_apache for _, y_apache in APACHE_PROJECTS.values()],
    ignore_index=True,
)

# Sum of entries must be equals to the number of final entries
assert sum([X.shape[0] for X, _ in APACHE_PROJECTS.values()]) == X_apache_all.
 ↪shape[0]

# apache dataset size, all together
X_apache_all.shape
```

[14]:
```
(388086, 63)
```

[15]:
```
def rq4():
    scores = []
    model = "rf"
    out = experiment.run(
        model,
        X_train=X_apache_all,
        X_test=X_test,
        y_train=y_apache_all,
        y_test=y_test,
        output_to=os.path.join(output_dir, f"rq4-{model}-apache-all.log"),
        tuning_enabled=TUNING_ENABLED
    )
    estimator, score, fi = out
    score["project"] = "apache-all"
    score["training_size"] = X_apache_all.shape[0]
    scores.append(score)
```

```
    # Save to the global state this run
    key = f"{model}-apache-all"
    ESTIMATORS[key] = estimator
    FEATURE_IMPORTANCES[key] = fi

    return scores


rq4_scores_all = rq4()
```

6.2 Learning from Projects Individually

[16]:
```
def rq4_individual():
    scores = []
    model = "rf"
    for project, Xy in APACHE_PROJECTS.items():
        out = experiment.run(
            model,
            X_train=Xy[0].drop(columns=["type"]),
            X_test=X_test.drop(columns=["type"]),
            y_train=Xy[1].drop(columns=["type"]),
            y_test=y_test.drop(columns=["type"]),
            output_to=os.path.join(output_dir, f"rq4-{model}-{project}.log"),
            tuning_enabled=TUNING_ENABLED
        )
        estimator, score, fi = out
        score["project"] = project
        score["training_size"] = Xy[0].shape[0]
        scores.append(score)

        # Save to the global state this run
        key = f"{model}-{project}"
        ESTIMATORS[key] = estimator
        FEATURE_IMPORTANCES[key] = fi

    return scores


rq4_scores_individual = rq4_individual()
```

6.3 Results

[17]:
```
results_rq4 = pd.DataFrame.from_dict(
    merge_scores(
        rq4_scores_all + rq4_scores_individual
    )
```

```
)
results_rq4.to_csv(
    os.path.join(output_dir, "rq4-results.csv"),
    index=False
)
results_rq4.drop(columns=["model", "balancing"]).sort_values(by="acc prec␣
↪recall".split(" "), ascending=False)
```

[17]:

	prec	recall	acc	tn	fp	fn	tp	total	\
28	0.558075	0.466807	0.718061	55407	1754	2530	2215	61906	
5	0.617372	0.458377	0.717397	55813	1348	2570	2175	61906	
18	0.524696	0.445522	0.706010	55246	1915	2631	2114	61906	
3	0.512254	0.405269	0.686618	55330	1831	2822	1923	61906	
10	0.456426	0.395153	0.678044	54928	2233	2870	1875	61906	
26	0.292957	0.434773	0.673834	52182	4979	2682	2063	61906	
22	0.509714	0.375975	0.672977	55445	1716	2961	1784	61906	
4	0.548428	0.360379	0.667874	55753	1408	3035	1710	61906	
17	0.591966	0.354057	0.666899	56003	1158	3065	1680	61906	
20	0.473761	0.344362	0.656305	55346	1815	3111	1634	61906	
29	0.615717	0.300527	0.642478	56271	890	3319	1426	61906	
12	0.613345	0.292518	0.638605	56286	875	3357	1388	61906	
21	0.482746	0.285985	0.630274	55707	1454	3388	1357	61906	
1	0.561856	0.275659	0.628907	56141	1020	3437	1308	61906	
24	0.523539	0.271865	0.625663	55987	1174	3455	1290	61906	
9	0.499037	0.272919	0.625088	55861	1300	3450	1295	61906	
23	0.508087	0.271444	0.624814	55914	1247	3457	1288	61906	
25	0.537975	0.268704	0.624774	56066	1095	3470	1275	61906	
0	0.634540	0.240042	0.614283	56505	656	3606	1139	61906	
13	0.455796	0.195574	0.588095	56053	1108	3817	928	61906	
16	0.697981	0.167545	0.580763	56817	344	3950	795	61906	
6	0.658053	0.161012	0.577033	56764	397	3981	764	61906	
14	0.603213	0.158272	0.574815	56667	494	3994	751	61906	
2	0.509960	0.134879	0.562060	56546	615	4105	640	61906	
15	0.616337	0.104953	0.549765	56851	310	4247	498	61906	
19	0.598997	0.100738	0.547570	56841	320	4267	478	61906	
7	0.634361	0.091043	0.543344	56912	249	4313	432	61906	
11	0.585366	0.075869	0.535704	56906	255	4385	360	61906	
27	0.480000	0.017703	0.508055	57070	91	4661	84	61906	
8	0.708333	0.003583	0.501730	57154	7	4728	17	61906	

	mean_fit_time	std_fit_time	mean_test_score	std_test_score	\
28	5.136771	0.303454	0.608672	0.055245	
5	34.540206	0.341172	0.688930	0.047541	
18	2.902078	0.088237	0.696361	0.052132	
3	2.432808	0.086360	0.603609	0.050725	
10	3.096748	0.018374	0.625165	0.051317	
26	0.597513	0.039189	0.579517	0.118321	

22	0.039282	0.000429	0.635258	0.045599
4	5.725520	0.107754	0.628419	0.012928
17	1.557437	0.047319	0.733949	0.032546
20	0.251455	0.017759	0.602822	0.058897
29	2.808430	0.056375	0.652493	0.044552
12	4.986435	0.068239	0.629827	0.019289
21	2.318175	0.044156	0.743132	0.030666
1	1.706350	0.083317	0.659124	0.062119
24	6.587307	0.180886	0.695074	0.043837
9	3.135745	0.120043	0.656631	0.057375
23	10.767351	0.419028	0.632447	0.043144
25	0.493675	0.019209	0.655741	0.033598
0	59.088807	2.092968	0.624406	0.015241
13	0.278699	0.015733	0.606928	0.044770
16	0.730303	0.010426	0.611807	0.052527
6	0.280362	0.010550	0.609434	0.100149
14	0.057817	0.001567	0.601587	0.075064
2	10.242492	0.064239	0.629535	0.037100
15	0.034103	0.000564	0.572548	0.046581
19	4.030439	0.065594	0.587377	0.044657
7	2.166840	0.103651	0.550008	0.026176
11	5.088855	0.188440	0.599730	0.034344
27	12.916080	0.909552	0.571598	0.048679
8	0.202116	0.001323	0.491624	0.073801

	project	training_size
28	zeppelin	10953
5	cloudstack	52390
18	oodt	6933
3	archiva	5995
10	helix	6787
26	thrift	1797
22	sqoop	3080
4	bookkeeper	12711
17	nutch	3321
20	openmeetings	4839
29	zookeeper	5279
12	jmeter	8597
21	reef	6150
1	accumulo	25458
24	syncope	14915
9	giraph	8039
23	storm	24208
25	tez	8947
0	apache-all	388086
13	knox	6821
16	myfaces-tobago	3866

6	commons-beanutils	1176
14	lens	6231
2	ambari	21997
15	metamodel	4122
19	oozie	8821
7	cxf	33589
11	ignite	65181
27	tomcat	23789
8	fluo	2094