Paper Evaluation

December 20, 2020

[1]:
```python
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 = False


# 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	0.594921	0.515567	0.205901	56243	918	3768	977	61906
ab	0.699748	0.638728	0.419178	56036	1125	2756	1989	61906
rf	0.781477	0.802295	0.574710	56489	672	2018	2727	61906
dt	0.783883	0.592739	0.602107	55198	1963	1888	2857	61906
et	0.778879	0.790766	0.570285	56445	716	2039	2706	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	0.828068	0.249294	0.874816	44661	12500	594	4151
	rus	0.811805	0.218075	0.887882	42055	15106	532	4213
ab	smote	0.855593	0.413469	0.806112	51735	5426	920	3825
	rus	0.887841	0.355694	0.912961	49314	7847	413	4332
rf	smote	0.843905	0.728216	0.709800	55904	1257	1377	3368
	rus	0.915836	0.398340	0.950896	50346	6815	233	4512
dt	smote	0.794038	0.586350	0.624658	55070	2091	1781	2964
	rus	0.877774	0.367381	0.881560	49958	7203	562	4183
et	smote	0.868622	0.677707	0.767545	55429	1732	1103	3642
	rus	0.914761	0.384935	0.956375	49910	7251	207	4538

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	0.233147	-0.266273	0.668915	-11582	11582	-3174	3174
	rus	0.216885	-0.297492	0.681981	-14188	14188	-3236	3236
ab	smote	0.155845	-0.225260	0.386934	-4301	4301	-1836	1836
	rus	0.188093	-0.283034	0.493783	-6722	6722	-2343	2343
rf	smote	0.062428	-0.074079	0.135090	-585	585	-641	641
	rus	0.134359	-0.403955	0.376185	-6143	6143	-1785	1785

dt	smote	0.010155	-0.006388	0.022550	-128	128	-107	107
	rus	0.093891	-0.225358	0.279452	-5240	5240	-1326	1326
et	smote	0.089743	-0.113058	0.197260	-1016	1016	-936	936
	rus	0.135882	-0.405831	0.386091	-6535	6535	-1832	1832

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

[10]:

```
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]:
```python
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
20	maxNestedBlocks	12	8	2	2	0	0
14	loc_method	11	4	2	2	2	1
15	uniqueWordsQty_method	7	0	4	1	2	0
10	methodsInvokedQty	7	0	0	4	0	3
23	cbo_method	4	0	2	1	1	0
3	dit	3	1	1	0	1	0
39	wmc_method	3	0	2	0	1	0
2	publicMethodsQty	3	0	1	1	1	0
41	returnsQty	3	0	0	2	1	0
13	variablesQty_method	3	0	0	1	0	2
9	rfc_method	3	0	0	0	2	1
27	publicFieldsQty	3	0	0	0	1	2
0	totalMethodsQty	2	1	0	1	0	0
56	cbo_class	2	0	0	0	1	1
16	parametersQty	1	1	0	0	0	0
1	abstractMethodsQty	1	0	1	0	0	0
31	methodsInvokedIndirectLocalQty	1	0	0	0	1	0
44	maxNestedBlocksQty	1	0	0	0	1	0
4	numbersQty_class	1	0	0	0	0	1
5	constructor_False	1	0	0	0	0	1
24	assignmentsQty_method	1	0	0	0	0	1
37	stringLiteralsQty_method	1	0	0	0	0	1
49	loc_class	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]:
```python
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                    (8599, 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                        (23793, 63)
zeppelin                      (10953, 63)
zookeeper                      (5279, 63)
```

6.1 Learning from all Apache projects

[14]:
```python
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]:
```
(388092, 63)
```

[15]:
```python
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]:
```python
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  \
5   0.645066  0.413277  0.697200  56082  1079  2784  1961  61906
28  0.613984  0.397893  0.688563  55974  1187  2857  1888  61906
18  0.637495  0.356164  0.669676  56200   961  3055  1690  61906
22  0.567834  0.351949  0.664857  55890  1271  3075  1670  61906
17  0.636476  0.324341  0.654482  56282   879  3206  1539  61906
10  0.646845  0.300316  0.643353  56383   778  3320  1425  61906
4   0.639578  0.293572  0.639920  56376   785  3352  1393  61906
3   0.675096  0.258799  0.624230  56570   591  3517  1228  61906
20  0.644840  0.259431  0.623785  56483   678  3514  1231  61906
12  0.641591  0.251633  0.619982  56494   667  3551  1194  61906
23  0.636946  0.251423  0.619763  56481   680  3552  1193  61906
29  0.675942  0.245732  0.617976  56602   559  3579  1166  61906
25  0.647092  0.243836  0.616398  56530   631  3588  1157  61906
24  0.613834  0.237513  0.612555  56452   709  3618  1127  61906
2   0.692065  0.231612  0.611529  56672   489  3646  1099  61906
9   0.527020  0.215806  0.599864  56242   919  3721  1024  61906
21  0.587156  0.202318  0.595255  56486   675  3785   960  61906
1   0.725649  0.188409  0.591248  56823   338  3851   894  61906
0   0.766825  0.170495  0.583096  56915   246  3936   809  61906
6   0.688634  0.151949  0.573123  56835   326  4024   721  61906
16  0.717573  0.144573  0.569925  56891   270  4059   686  61906
14  0.684275  0.117387  0.556445  56904   257  4188   557  61906
19  0.766091  0.102845  0.550119  57012   149  4257   488  61906
13  0.722222  0.087671  0.542436  57001   160  4329   416  61906
26  0.732075  0.081770  0.539643  57019   142  4357   388  61906
15  0.746959  0.064700  0.531440  57057   104  4438   307  61906
11  0.803468  0.029294  0.514350  57127    34  4606   139  61906
7   0.875817  0.028240  0.513954  57142    19  4611   134  61906
8   0.609756  0.010537  0.504989  57129    32  4695    50  61906
27  0.711538  0.007798  0.503768  57146    15  4708    37  61906

              project  training_size
```

5	cloudstack	52390
28	zeppelin	10953
18	oodt	6933
22	sqoop	3080
17	nutch	3321
10	helix	6787
4	bookkeeper	12711
3	archiva	5995
20	openmeetings	4839
12	jmeter	8599
23	storm	24208
29	zookeeper	5279
25	tez	8947
24	syncope	14915
2	ambari	21997
9	giraph	8039
21	reef	6150
1	accumulo	25458
0	apache-all	388092
6	commons-beanutils	1176
16	myfaces-tobago	3866
14	lens	6231
19	oozie	8821
13	knox	6821
26	thrift	1797
15	metamodel	4122
11	ignite	65181
7	cxf	33589
8	fluo	2094
27	tomcat	23793