Paper Evaluation

August 17, 2020

[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 = 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("data", "dataset.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]:
```
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

    return scores

rq1_scores = rq1()
```

3.1 Results

[5]:
```
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(" ")]
```

[5]:

model	acc	prec	recall	tn	fp	fn	tp	total
lr	0.607027	0.501586	0.233298	56061	1100	3638	1107	61906
ab	0.712096	0.645349	0.444468	56002	1159	2636	2109	61906
rf	0.791698	0.810508	0.594942	56501	660	1922	2823	61906
dt	0.795162	0.618868	0.622129	55343	1818	1793	2952	61906
et	0.789504	0.801658	0.591149	56467	694	1940	2805	61906

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

[6]:
```
# 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

[7]:
```
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]
```

[7]:

model	balancing	acc	prec	recall	tn	fp	fn	tp
lr	smote	0.823436	0.237579	0.881770	43734	13427	561	4184
	rus	0.819213	0.230795	0.882613	43203	13958	557	4188
ab	smote	0.878035	0.420490	0.853741	51578	5583	694	4051
	rus	0.900482	0.371991	0.931507	49699	7462	325	4420
rf	smote	0.857857	0.740161	0.737197	55933	1228	1247	3498
	rus	0.923443	0.407526	0.963119	50517	6644	175	4570
dt	smote	0.799209	0.594392	0.634352	55107	2054	1735	3010
	rus	0.881425	0.385404	0.879241	50508	6653	573	4172
et	smote	0.876866	0.690007	0.782929	55492	1669	1030	3715
	rus	0.920738	0.398047	0.962276	50256	6905	179	4566

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

[8]:
```
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
```

[8]:

model	balancing	acc	prec	recall	tn	fp	fn	tp
lr	smote	0.216409	-0.264007	0.648472	-12327	12327	-3077	3077
	rus	0.212186	-0.270791	0.649315	-12858	12858	-3081	3081
ab	smote	0.165939	-0.224859	0.409273	-4424	4424	-1942	1942
	rus	0.188386	-0.273358	0.487039	-6303	6303	-2311	2311
rf	smote	0.066159	-0.070347	0.142255	-568	568	-675	675
	rus	0.131745	-0.402982	0.368177	-5984	5984	-1747	1747
dt	smote	0.004047	-0.024476	0.012223	-236	236	-58	58
	rus	0.086264	-0.233464	0.257113	-4835	4835	-1220	1220
et	smote	0.087362	-0.111650	0.191781	-975	975	-910	910
	rus	0.131235	-0.403610	0.371128	-6211	6211	-1761	1761

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

[9]:
```
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

[10]:
```
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
```

[10]:

	feature	total	1	2	3	4	5
20	maxNestedBlocks	12	9	1	1	0	1
9	loc_method	11	5	2	1	2	1
15	uniqueWordsQty_method	8	0	2	4	1	1
25	cbo_method	7	0	2	2	2	1
12	methodsInvokedQty	7	0	1	2	2	2
56	wmc_method	4	0	2	1	1	0
0	totalMethodsQty	3	1	1	1	0	0
1	abstractMethodsQty	3	0	2	1	0	0
51	cbo_class	3	0	1	0	2	0
28	publicFieldsQty	3	0	0	0	0	3
44	returnsQty	2	0	1	0	1	0
2	publicMethodsQty	2	0	0	2	0	0
3	dit	2	0	0	0	1	1
11	rfc_method	2	0	0	0	1	1
7	returnQty	1	0	0	0	1	0
16	parametersQty	1	0	0	0	1	0
4	numbersQty_class	1	0	0	0	0	1
14	variablesQty_method	1	0	0	0	0	1
32	methodsInvokedIndirectLocalQty	1	0	0	0	0	1
46	maxNestedBlocksQty	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?

[11]:
```
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
```

[12]:
```
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                  (6790, 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

[13]:
```
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
```

[13]:
```
(388095, 63)
```

[14]:
```
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

[15]:
```
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

[16]:
```
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)
```

[16]:

	prec	recall	acc	tn	fp	fn	tp	total	\
5	0.646967	0.436038	0.708143	56032	1129	2676	2069	61906	
28	0.624456	0.392835	0.686612	56040	1121	2881	1864	61906	
22	0.602902	0.385248	0.682092	55957	1204	2917	1828	61906	
18	0.623209	0.375764	0.678452	56083	1078	2962	1783	61906	
17	0.624953	0.350474	0.666507	56163	998	3082	1663	61906	
10	0.660502	0.316122	0.651317	56390	771	3245	1500	61906	
12	0.644743	0.293361	0.639972	56394	767	3353	1392	61906	
4	0.646319	0.292308	0.639515	56402	759	3358	1387	61906	
25	0.659722	0.280295	0.634147	56475	686	3415	1330	61906	
20	0.626910	0.268072	0.627414	56404	757	3473	1272	61906	
3	0.668828	0.260906	0.625091	56548	613	3507	1238	61906	
29	0.672808	0.253952	0.621850	56575	586	3540	1205	61906	
23	0.650907	0.249526	0.619208	56526	635	3561	1184	61906	
2	0.686797	0.244468	0.617607	56632	529	3585	1160	61906	
9	0.559924	0.249104	0.616426	56232	929	3563	1182	61906	
24	0.606366	0.240885	0.613952	56419	742	3602	1143	61906	
21	0.612108	0.230137	0.609015	56469	692	3653	1092	61906	
1	0.730208	0.200211	0.597035	56810	351	3795	950	61906	
6	0.650680	0.191570	0.591516	56673	488	3836	909	61906	
0	0.767936	0.182719	0.589068	56899	262	3878	867	61906	
16	0.736285	0.161222	0.578214	56887	274	3980	765	61906	
14	0.709330	0.124974	0.560361	56918	243	4152	593	61906	
19	0.786271	0.106217	0.551910	57024	137	4241	504	61906	
26	0.709010	0.101159	0.548856	56964	197	4265	480	61906	
13	0.745600	0.098209	0.547714	57002	159	4279	466	61906	
15	0.702032	0.065543	0.531617	57029	132	4434	311	61906	
7	0.868020	0.036038	0.517792	57135	26	4574	171	61906	
11	0.822485	0.029294	0.514385	57131	30	4606	139	61906	
27	0.816327	0.016860	0.508272	57143	18	4665	80	61906	
8	0.627273	0.014542	0.506912	57120	41	4676	69	61906	

	project	training_size
5	cloudstack	52390
28	zeppelin	10953
22	sqoop	3080
18	oodt	6933
17	nutch	3321

10	helix	6790
12	jmeter	8599
4	bookkeeper	12711
25	tez	8947
20	openmeetings	4839
3	archiva	5995
29	zookeeper	5279
23	storm	24208
2	ambari	21997
9	giraph	8039
24	syncope	14915
21	reef	6150
1	accumulo	25458
6	commons-beanutils	1176
0	apache-all	388095
16	myfaces-tobago	3866
14	lens	6231
19	oozie	8821
26	thrift	1797
13	knox	6821
15	metamodel	4122
7	cxf	33589
11	ignite	65181
27	tomcat	23793
8	fluo	2094