

Paper Evaluation

August 18, 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 = 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("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]:

```

	acc	prec	recall	tn	fp	fn	tp	total
model								
lr	0.678597	0.656053	0.373446	56232	929	2973	1772	61906
ab	0.712096	0.645349	0.444468	56002	1159	2636	2109	61906
rf	0.803219	0.814496	0.618124	56493	668	1812	2933	61906
dt	0.746284	0.585751	0.523288	55405	1756	2262	2483	61906
et	0.777034	0.740093	0.570706	56210	951	2037	2708	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]:
      acc      prec      recall      tn      fp      fn      tp
model balancing
lr      smote      0.886724  0.385021  0.891675  50403  6758  514  4231
      rus      0.887904  0.382456  0.895890  50297  6864  494  4251
ab      smote      0.888355  0.314637  0.948156  47361  9800  246  4499
      rus      0.900482  0.371991  0.931507  49699  7462  325  4420
rf      smote      0.909537  0.504170  0.891886  52999  4162  513  4232
      rus      0.923860  0.412009  0.961644  50649  6512  182  4563
dt      smote      0.879321  0.389187  0.872287  50665  6496  606  4139
      rus      0.893832  0.323624  0.953003  47710  9451  223  4522
et      smote      0.909087  0.457025  0.907692  52044  5117  438  4307
      rus      0.918232  0.408215  0.950896  50620  6541  233  4512

```

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]:

```

		acc	prec	recall	tn	fp	fn	tp
model balancing								
lr	smote	0.208127	-0.271032	0.518230	-5829	5829	-2459	2459
	rus	0.209308	-0.273597	0.522445	-5935	5935	-2479	2479
ab	smote	0.176259	-0.330711	0.503688	-8641	8641	-2390	2390
	rus	0.188386	-0.273358	0.487039	-6303	6303	-2311	2311
rf	smote	0.106318	-0.310326	0.273762	-3494	3494	-1299	1299
	rus	0.120641	-0.402487	0.343519	-5844	5844	-1630	1630
dt	smote	0.133038	-0.196565	0.348999	-4740	4740	-1656	1656
	rus	0.147548	-0.262127	0.429715	-7695	7695	-2039	2039
et	smote	0.132052	-0.283068	0.336986	-4166	4166	-1599	1599
	rus	0.141198	-0.331878	0.380190	-5590	5590	-1804	1804

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
3	maxNestedBlocks	15	11	1	0	2	1
23	loc_method	9	1	2	2	2	2
10	uniqueWordsQty_method	7	0	2	2	2	1
49	methodsInvokedQty	6	0	0	1	1	4
11	cbo_method	5	0	2	2	0	1
29	wmc_method	4	0	4	0	0	0
16	cbo_class	4	0	1	0	3	0
0	constructor_True	3	1	0	1	1	0
1	type_interface	2	1	1	0	0	0
4	constructor_False	2	0	1	0	0	1
5	loopQty_method	2	0	0	2	0	0
40	variablesQty_method	2	0	0	1	0	1
12	type_anonymous	2	0	0	0	1	1
22	comparisonsQty_method	1	1	0	0	0	0
8	maxNestedBlocksQty	1	0	1	0	0	0
2	type_enum	1	0	0	1	0	0
7	publicMethodsQty	1	0	0	1	0	0
9	abstractMethodsQty	1	0	0	1	0	0
48	rfc_method	1	0	0	1	0	0
24	returnsQty	1	0	0	0	1	0
26	rfc_class	1	0	0	0	1	0
46	publicFieldsQty	1	0	0	0	1	0
27	stringLiteralQty_class	1	0	0	0	0	1
37	type_class	1	0	0	0	0	1
39	assignmentsQty_method	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)
cxfs	(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	0.621649	0.513172	0.743622	55679	1482	2310	2435	61906	
28	0.566470	0.464278	0.717391	55475	1686	2542	2203	61906	
18	0.537485	0.463857	0.715361	55267	1894	2544	2201	61906	
3	0.512434	0.429926	0.697985	55220	1941	2705	2040	61906	
17	0.548102	0.404636	0.688471	55578	1583	2825	1920	61906	
10	0.464710	0.403793	0.682592	54954	2207	2829	1916	61906	
22	0.609648	0.375553	0.677796	56020	1141	2963	1782	61906	
25	0.539005	0.369863	0.671802	55660	1501	2990	1755	61906	
23	0.556629	0.352160	0.664438	55830	1331	3074	1671	61906	
9	0.433325	0.356797	0.659032	54947	2214	3052	1693	61906	
29	0.628972	0.321180	0.652726	56262	899	3221	1524	61906	
20	0.482411	0.332350	0.651375	55469	1692	3168	1577	61906	
12	0.515404	0.299684	0.638147	55824	1337	3323	1422	61906	
21	0.481291	0.295469	0.634517	55650	1511	3343	1402	61906	
24	0.518350	0.288725	0.633227	55888	1273	3375	1370	61906	
0	0.649789	0.259642	0.624013	56497	664	3513	1232	61906	
1	0.576710	0.263014	0.623494	56245	916	3497	1248	61906	
6	0.643316	0.232244	0.610778	56550	611	3643	1102	61906	
16	0.381102	0.255005	0.610314	55196	1965	3535	1210	61906	
2	0.501761	0.240253	0.610225	56029	1132	3605	1140	61906	
4	0.482438	0.240253	0.609429	55938	1223	3605	1140	61906	
15	0.570383	0.213488	0.600070	56398	763	3732	1013	61906	
19	0.600608	0.208219	0.598363	56504	657	3757	988	61906	
13	0.507295	0.190516	0.587578	56283	878	3841	904	61906	
14	0.463646	0.185458	0.583825	56143	1018	3865	880	61906	
26	0.242255	0.201054	0.574425	54177	2984	3791	954	61906	
8	0.466184	0.122023	0.555212	56498	663	4166	579	61906	
7	0.502825	0.093783	0.543043	56721	440	4300	445	61906	

11	0.565574	0.072708	0.534036	56896	265	4400	345	61906
27	0.559633	0.012856	0.506008	57113	48	4684	61	61906

	mean_fit_time	std_fit_time	mean_test_score	std_test_score	\
5	34.136095	0.453911	0.701172	0.049714	
28	5.086968	0.294842	0.608371	0.052231	
18	2.874667	0.015300	0.687929	0.051630	
3	2.429385	0.068277	0.575410	0.027376	
17	1.505548	0.049739	0.743647	0.036559	
10	3.089493	0.042276	0.610663	0.069676	
22	0.308523	0.002308	0.637366	0.061589	
25	4.123706	0.057962	0.689929	0.028932	
23	1.305398	0.049080	0.614461	0.034391	
9	0.345735	0.018126	0.696688	0.048262	
29	2.819376	0.067457	0.658597	0.047932	
20	0.249204	0.020481	0.581521	0.046188	
12	4.105839	0.051512	0.682666	0.026414	
21	2.284006	0.040719	0.745460	0.040013	
24	6.378376	0.092332	0.689769	0.046570	
0	58.438388	0.992319	0.631243	0.019912	
1	1.687905	0.068308	0.659687	0.063352	
6	0.197350	0.005305	0.645367	0.092532	
16	0.139963	0.002156	0.652011	0.051416	
2	10.269593	0.025725	0.618623	0.044476	
4	5.891956	0.094264	0.652809	0.040228	
15	0.137413	0.003269	0.580599	0.052718	
19	0.498774	0.021204	0.587284	0.042232	
13	2.822734	0.042975	0.581728	0.040740	
14	0.312104	0.014874	0.596098	0.041634	
26	0.638799	0.071764	0.587256	0.100076	
8	0.880590	0.060118	0.498227	0.113801	
7	2.057116	0.082908	0.555690	0.021273	
11	4.942617	0.136505	0.604817	0.045741	
27	11.775489	0.064326	0.578226	0.042667	

	project	training_size
5	cloudstack	52390
28	zeppelin	10953
18	oodt	6933
3	archiva	5995
17	nutch	3321
10	helix	6790
22	sqoop	3080
25	tez	8947
23	storm	24208
9	giraph	8039
29	zookeeper	5279

20	openmeetings	4839
12	jmeter	8599
21	reef	6150
24	syncope	14915
0	apache-all	388095
1	accumulo	25458
6	commons-beanutils	1176
16	myfaces-tobago	3866
2	ambari	21997
4	bookkeeper	12711
15	metamodel	4122
19	oozie	8821
13	knox	6821
14	lens	6231
26	thrift	1797
8	fluo	2094
7	cxfr	33589
11	ignite	65181
27	tomcat	23793