A2 - How to do Machine Learning

September 19, 2022

[ ]:
```
# import packages
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from ReliefF import ReliefF
from sklearn.model_selection  import train_test_split
from sklearn.model_selection import cross_val_score
from autorank import autorank
```

As the missing values scatter across the data set, so we can't simply delete them. And data itself covers a range of different values so we rule out imputation and instead fill in a global constant not to undermine the variabiliy while remain validility of the data.

[ ]:
```
# read data
X = pd.read_csv('data_A2.csv').to_numpy()
y = pd.read_csv('labels_A2.csv').to_numpy().flatten()
# Fill the missing value with -1
X = np.nan_to_num(X, copy=True, nan=-1)
X.shape
```

[ ]:
```
(999, 100)
```

Majority of learning methods won't behave well in the data set due to curse of dimensionality (100 features!). Thus it is important that before a model is trained, a feature selection technique is applied.

We can 1. calculate the correlation between each feature and the class label 2. use a classifier to test out the different combianations of features (random forest will be a good option as it can randomly impute the variables in the OOB cases, and then compare them to those not in the OOB) 3. rely on a dedicated feature selection method (PCR or Relief)

In our case, we opt for the dedicated feature selection method RelieF. This is because we don't want to assume conditional independence upon the class lable betweeen features (rule out method number one) and any classifier won't function properly with these many features, so no point in testing feature importance with an underperformed classifier in the first place (rule out method number two).

[ ]:
```python
# set a random state to keep a reproduciable output
RANDOM_STATE = 1234
np.random.seed(RANDOM_STATE)
fs = ReliefF(n_neighbors=1, n_features_to_keep=10)
X = fs.fit_transform(X, y)
# X_clean = X_fill[:,np.argsort(fi)[:10]]
```

We use tree-based classifiers (decistion stump, unprunned tree, prunned tree, random forest) to establish a baseline result before we play around the data set (additive noice, multiplicative noice, etc.)

[ ]:
```python
def TreebasedClassifiers(X, y):
    scores = pd.DataFrame()

    ds = DecisionTreeClassifier(max_depth=1, random_state= RANDOM_STATE)
    scores["Decistion Stump"] = cross_val_score(ds, X, y, cv=10)

    # we set the max-depth equal to the number of features, so it will grow to␣
 ↪the maximum depth (unprunned)
    dt = DecisionTreeClassifier(max_depth=10, random_state= RANDOM_STATE)
    scores["Decistion Tree"] = cross_val_score(dt, X, y, cv=10)

    # we set minimal_cost_complexity_pruning close enough to 0, so it will␣
 ↪differ from decistion stumps, but slight larger than 0, so it will differ␣
 ↪from an unpruned decistion tree (pruned)
    pt = DecisionTreeClassifier(random_state=RANDOM_STATE, ccp_alpha=0.005)
    scores["Decistion Tree (Prunned)"] = cross_val_score(pt, X, y, cv=10)

    rf = RandomForestClassifier(max_depth=10, random_state= RANDOM_STATE)
    scores["Random Forest"] = cross_val_score(rf, X, y, cv=10)

    result = autorank(scores, alpha=0.05, verbose=False)
    print(result)

TreebasedClassifiers(X, y)
```

```
RankResult(rankdf=
                          meanrank      mean       std  ci_lower  ci_upper  \
Decistion Stump               3.10  0.558525  0.065985  0.526063  0.590987
Decistion Tree (Prunned)      2.85  0.576525  0.050196  0.544063  0.608987
Decistion Tree                2.70  0.575525  0.056072  0.543063  0.607987
Random Forest                 1.35  0.633606  0.040060  0.601144  0.666068

                          effect_size   magnitude
Decistion Stump                   0.0  negligible
Decistion Tree (Prunned)    -0.307041       small
Decistion Tree              -0.277644       small
Random Forest               -1.375519       large
```

```
pvalue=0.00023423524796781616
cd=None
omnibus=anova
posthoc=tukeyhsd
all_normal=True
pvals_shapiro=[0.31625089049339294, 0.8351843953132629, 0.02381780743598938,
0.7750453352928162]
homoscedastic=True
pval_homogeneity=0.5368611191270642
homogeneity_test=bartlett
alpha=0.05
alpha_normality=0.0125
num_samples=10
posterior_matrix=
None
decision_matrix=
None
rope=None
rope_mode=None
effect_size=cohen_d)
```

The decision stump performed understandably the worst as it only threshhold on one feature which leads to underfitting.

[ ]:
```python
# additive normal noise
noise = np.random.normal(0, 0.2, np.shape(X))
X_addictive_noice = X + np.multiply(noise, np.average(X, axis=0))
TreebasedClassifiers(X_addictive_noice, y)
```

```
RankResult(rankdf=
                          meanrank      mean       std  ci_lower  ci_upper  \
Decistion Stump               2.95  0.548525  0.056658  0.517434  0.579616
Decistion Tree (Prunned)      2.95  0.550505  0.053456  0.519414  0.581596
Decistion Tree                2.90  0.550505  0.043612  0.519414  0.581596
Random Forest                 1.20  0.624626  0.051881  0.593535  0.655717

                         effect_size   magnitude
Decistion Stump                  0.0  negligible
Decistion Tree (Prunned)   -0.035944  negligible
Decistion Tree              -0.03916  negligible
Random Forest              -1.400924       large
pvalue=3.436984291080978e-05
cd=None
omnibus=anova
posthoc=tukeyhsd
all_normal=True
pvals_shapiro=[0.6545984148979187, 0.23763686418533325, 0.04130295291543007,
0.27765145897865295]
```

```
homoscedastic=True
pval_homogeneity=0.8920393834889994
homogeneity_test=bartlett
alpha=0.05
alpha_normality=0.0125
num_samples=10
posterior_matrix=
None
decision_matrix=
None
rope=None
rope_mode=None
effect_size=cohen_d)
```

All the classifiers remain stable, as the noise can be learned.

```
[ ]: # multiplicative normal noise
noise = np.random.normal(0, 0.2, np.shape(X))
X_multiplicative_noise = np.multiply(X, noise)
TreebasedClassifiers(X_multiplicative_noise, y)
```

```
RankResult(rankdf=
                          meanrank  median    mad ci_lower  ci_upper  \
Decistion Stump               2.95   0.490   0.01     0.45  0.505051
Random Forest                 2.60   0.495  0.025     0.38      0.52
Decistion Tree (Prunned)      2.50   0.500    0.0     0.43  0.505051
Decistion Tree                1.95   0.505  0.015     0.42  0.535354

                         effect_size   magnitude
Decistion Stump                  0.0  negligible
Random Forest               -0.17713  negligible
Decistion Tree (Prunned)   -0.953874       large
Decistion Tree             -0.793671      medium
pvalue=0.3088962111457305
cd=1.483221853685529
omnibus=friedman
posthoc=nemenyi
all_normal=False
pvals_shapiro=[0.01277504488825798, 0.05152953788638115, 3.401902404220891e-07,
0.04103650152683258]
homoscedastic=False
pval_homogeneity=0.03401818284279286
homogeneity_test=levene
alpha=0.05
alpha_normality=0.0125
num_samples=10
posterior_matrix=
None
```

```
decision_matrix=
None
rope=None
rope_mode=None
effect_size=akinshin_gamma)
```

All the classifiers are worse as the noise can't be corrected due to randomised scalar.

[ ]:
```
## class noise
mask = np.random.binomial(1, 0.05, y.shape[0])
y_class_noise = abs(y - mask)
TreebasedClassifiers(X, y_class_noise)
```

```
RankResult(rankdf=
                          meanrank      mean       std  ci_lower  ci_upper  \
Decistion Stump               3.55  0.564566  0.041128  0.541299  0.587832
Decistion Tree                2.80  0.577636  0.038182   0.55437  0.600903
Decistion Tree (Prunned)      2.15  0.594616  0.023738   0.57135  0.617882
Random Forest                 1.50  0.631667  0.047512    0.6084  0.654933

                         effect_size   magnitude
Decistion Stump                  0.0  negligible
Decistion Tree             -0.329383       small
Decistion Tree (Prunned)   -0.894942       large
Random Forest              -1.510099       large
pvalue=0.00010545055797056721
cd=None
omnibus=anova
posthoc=tukeyhsd
all_normal=True
pvals_shapiro=[0.5788647532463074, 0.20331569015979767, 0.3019199073314667,
0.868999183177948]
homoscedastic=True
pval_homogeneity=0.2673054416861556
homogeneity_test=bartlett
alpha=0.05
alpha_normality=0.0125
num_samples=10
posterior_matrix=
None
decision_matrix=
None
rope=None
rope_mode=None
effect_size=cohen_d)
```

1 Summary

The multiplicative noise will have a big impact on the performance, while the other kinds of noise can be coped well.