Statistical Learning weekly assignment 5

Gaspard Gaches (s4645251)

March 11, 2025

[32]:
```
import feather
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score, confusion_matrix
```

[ ]:
```
train = feather.read_dataframe("masq_train.feather").replace('NA', np.nan).
 ↪dropna()
test = feather.read_dataframe("masq_test.feather").replace('NA', np.nan).dropna()

train['GENDER.f'] = train.apply(lambda r: 0 if r['GENDER'] == 'm' else 1, axis=1)
test['GENDER.f'] = test.apply(lambda r: 0 if r['GENDER'] == 'm' else 1, axis=1)
```

We create a new gender column with binary values since we cannot run predictive models on string variables.

[7]:
```
masq_cols = [col for col in train.columns if col.startswith("MASQ")]
df_masq = train[masq_cols]

df_vif = df_masq.copy()
df_vif["intercept"] = 1

vif_data = pd.DataFrame()
vif_data['parameter'] = df_vif.columns
vif_data['VIF'] = [variance_inflation_factor(df_vif.values, i) for i in␣
 ↪range(df_vif.shape[1])]

vif_data = vif_data[vif_data["parameter"] != "intercept"]
print(vif_data.sort_values(by="VIF", ascending=False))
```

```
   parameter       VIF
20    MASQ22  4.037796
77    MASQ79  3.917666
14    MASQ16  3.864820
45    MASQ47  3.691897
75    MASQ77  3.642254
..       ...       ...
```

```
6     MASQ07  1.488875
83    MASQ85  1.367481
10    MASQ12  1.331534
35    MASQ37  1.320590
30    MASQ32  1.196895

[89 rows x 2 columns]
```

We sorted the VIF values in descending order in order to spot the parameters with the highest multicollinearity, but it looks like none of them really show bad multicollinearity.

[10]:
```
outcome_counts = train['D_DEPDYS'].value_counts()
outcome_prob = outcome_counts/outcome_counts.sum()
outcome_prob
```

[10]:
```
D_DEPDYS
0    0.52614
1    0.47386
Name: count, dtype: float64
```

After checking the proportions of our response variable, it looks like our data set is pretty well balanced so we can just use a regular KFold:

[1]:
```
def predictRidge(X_train, y_train, X_test, alpha):
    ridge = Ridge(alpha)
    ridge.fit(X_train, y_train)
    y_hat = ridge.predict(X_test)
    return y_hat


def predictLasso(X_train, y_train, X_test, alpha):
    lasso = Lasso(alpha)
    lasso.fit(X_train, y_train)
    y_hat = lasso.predict(X_test)
    return y_hat


def predictNet(X_train, y_train, X_test, alpha):
    elastic = ElasticNet(alpha)
    elastic.fit(X_train, y_train)
    y_hat = elastic.predict(X_test)
    return y_hat
```

[31]:
```
X_train, y_train = train.drop(['D_DEPDYS', 'GENDER'], axis=1), train['D_DEPDYS']

kf = KFold(n_splits=10, shuffle=True, random_state=0)
df_accuracies = pd.DataFrame(columns=['ridge', 'lasso', 'elastic net'])

for train_ix, val_ix in kf.split(X_train, y_train):
```

```
    fold_X_train, fold_X_val = X_train.iloc[train_ix], X_train.iloc[val_ix]
    fold_y_train, fold_y_val = y_train.iloc[train_ix], y_train.iloc[val_ix]

    # we need to convert the y preds into binary variables:
    y_pred_ridge = (predictRidge(fold_X_train, fold_y_train, fold_X_val, alpha=0.
↪1) > 0.5).astype(int)
    y_pred_lasso = (predictLasso(fold_X_train, fold_y_train, fold_X_val, alpha=0.
↪1) > 0.5).astype(int)
    y_pred_net = (predictNet(fold_X_train, fold_y_train, fold_X_val, alpha=0.1)␣
↪> 0.5).astype(int)

    df_accuracies = pd.concat([
        df_accuracies
        , pd.DataFrame({
            'ridge': [accuracy_score(y_true=fold_y_val, y_pred=y_pred_ridge)]
            , 'lasso': [accuracy_score(y_true=fold_y_val, y_pred=y_pred_lasso)]
            , 'elastic net': [accuracy_score(y_true=fold_y_val,␣
↪y_pred=y_pred_net)]
        })
    ])

df_accuracies = df_accuracies.mean(axis=0)
df_accuracies
```

```
C:\Windows\Temp\ipykernel_26400\1077609598.py:15: FutureWarning: The behavior of
DataFrame concatenation with empty or all-NA entries is deprecated. In a future
version, this will no longer exclude empty or all-NA columns when determining
the result dtypes. To retain the old behavior, exclude the relevant entries
before the concat operation.
  df_accuracies = pd.concat([
```

[31]:

```
ridge          0.741684
lasso          0.746607
elastic net    0.754575
dtype: float64
```

Elastic net stands out with the highest accuracy. We will now test it on the holdout set we've kept aside until now:

[41]:

```
X_test, y_test = test.drop(['D_DEPDYS', 'GENDER'], axis=1), test['D_DEPDYS']

elastic = ElasticNet(alpha=0.1)
elastic.fit(X_test, y_test)
y_hat = (elastic.predict(X_test) > 0.5).astype(int)
```

[42]:

```
cf_mat = confusion_matrix(y_true=y_test, y_pred=y_hat)
MCR = round((cf_mat[0,1] + cf_mat[1,0]) / len(y_test), 4)
MCR
```

[42]: np.float64(0.2286)

We obtain an MCR of 0.229 on the test set. Let's extract the coefficient from the elastic net model:

[50]:
```
coefs = pd.Series(elastic.coef_, index=X_test.columns)
coefs = coefs[coefs !=0]
coefs
```

[50]:
```
Leeftijd    0.001300
MASQ01      0.013209
MASQ16      0.080116
MASQ22      0.004418
MASQ29      0.002592
MASQ30      0.055978
MASQ33      0.017074
MASQ43      0.003388
MASQ46      0.000583
MASQ51      0.012450
MASQ53      0.009676
MASQ56      0.013934
MASQ60      0.000414
MASQ62      0.016268
MASQ83      0.003872
MASQ89      0.007496
dtype: float64
```

[46]:
```
masq_subscales = {
    "Anhedonic Depression": [1, 14, 18, 21, 23, 26, 27, 30, 33, 35, 36, 39, 40,␣
↪44, 49, 53, 58, 66, 72, 78, 86, 89],
    "Anxious Arousal": [3, 19, 25, 45, 48, 52, 55, 57, 61, 67, 69, 73, 75, 79,␣
↪85, 87, 88],
    "General Distress Depression": [6, 8, 10, 13, 16, 22, 24, 42, 47, 56, 64,␣
↪74],
    "General Distress Anxiety": [2, 9, 12, 15, 20, 59, 63, 65, 77, 81, 82],
    "General Distress Mixed": [4, 5, 17, 29, 31, 34, 37, 50, 51, 70, 76, 80, 83,␣
↪84, 90]
}
```

[47]:
```
coefs.index
```

[47]:
```
Index(['Leeftijd', 'MASQ01', 'MASQ16', 'MASQ22', 'MASQ29', 'MASQ30', 'MASQ33',
       'MASQ43', 'MASQ46', 'MASQ51', 'MASQ53', 'MASQ56', 'MASQ60', 'MASQ62',
       'MASQ83', 'MASQ89'],
      dtype='object')
```

[51]:
```
coef_ids = list(map(lambda item: int(item.split('MASQ')[1]), coefs.index[1:]))
coef_ids
```

[51]: [1, 16, 22, 29, 30, 33, 43, 46, 51, 53, 56, 60, 62, 83, 89]

[52]:
```
subscales = {}
for key, value in masq_subscales.items():
    counter = 0
    for i in coef_ids:
        if i in value:
            counter += 1
    subscales[key] = counter
subscales
```

[52]:
```
{'Anhedonic Depression': 5,
 'Anxious Arousal': 0,
 'General Distress Depression': 3,
 'General Distress Anxiety': 0,
 'General Distress Mixed': 3}
```

It looks like the subscale with the most selected items is anhedonic depression.