COVID 19 30 Days Mortality Prediction
CS4602 NTHU

KUO, KUAN-TING 郭冠廷
SID 107062274
Date: 12/ Nov/ 2020

Table of Contents

w

Introduction

This project is the second assignment for CS4602. Using a dataset of 1834 patients and their information, a SVM model is built to predict the patients' mortality after 30 days.

Dataset Information

Training dataset

- Number of patients: There are 1834 labelled patients in the training datasets.
- Number of attributes: Each patient has 47 columns attached after their patient ID, which serves as the attributes for training the model.

Testing dataset

The performance of this model would later be evaluated using a dataset of 612 patients. The recall rate and the precision rate would be used to measure the result.

Data Preprocess

Missing information

I use two different methods to handle the missing data.

- Most frequent value
    - For attributes with categorical features, I used the most frequent value to fill up the missing part.
- Median
    - For the numerical features, the median is use to fill up the missing part.

One hot transformation

There are also same attributes, such as gender, which needed to be transform into One-hot before training the model.
The command used is listed below.

```
df = pd.read_csv(path + df_name)
```

```
data_dum = pd.get_dummies(df, prefix=['s', 'd'], columns=['sex',
'ed_diagnosis'])
df = pd.DataFrame(data_dum)
```

MinMaxScaler

Originally I also used MinMaxScaler to normalize the data in the preprocess. But after observing the results, I found MinMaxScaler is redundant and removed it. More explaination would be listed in the observation section.
The command used is listed below.

```
  x = df.values #returns a numpy array
  min_max_scaler = preprocessing.MinMaxScaler()
  x_scaled = min_max_scaler.fit_transform(x)
  df = pd.DataFrame(x_scaled)
```

Model

Feature selection

Because I assume not all the features may have an important impact on the classification, I used a function to make the feature selection in my model.

from sklearn.feature_selection import SelectPercentile, chi2

SVM

I used the sklearn library of SVM for this classification problem. One thing needed to be careful with this dataset is that the positive and negative labels are inbalance. The SVC function can be adjusted to overcome this issue.

```
  svc = SVC(class_weight='balanced')
```

Aside from the setting above, I also used StandardScaler() to scale the data sent to my model and the feature selection function mentioned above to select features.

```
 model = Pipeline([('anova', SelectPercentile(chi2,
percentile=feature_percentage)), ('scaler', StandardScaler()), ('svc', svc)])
```

Observation

MinMaxScaler vs StandardScaler

MinMaxScaler normalizes attributes into [0, 1] range and StandardScaler normalizes them to [-1, 1] range. While the two normalization methods return different values, by running the model with different preprocessed datasets, I noticed that f1 score is higher when StandardScaler is applied.
On the other hand, if MinMaxScaler is applied first and StandardScaler is applied later, the result remains the same. Therefore, I only used StandardScaler for normalizing the data.

Kernal

Among the SVM models that I tried, including linear, poly, sigmoid, and rbf, sigmoid models are usually found with the most stable f1 score.

Parameters Tuning

C and gamma

Two parameters, C and gamma, are tuned using cross-validation.

```
param_grid = [{'svc__kernel': ['poly']}, {'svc__kernel': ['sigmoid']},
{'svc__kernel': ['rbf']}, {'svc__kernel': ['linear']}]
scores = ['recall', 'precision', 'f1']
grid = GridSearchCV(model, param_grid, scoring=scores, refit='f1', cv=skf)
```

Feature selection percentage

- By using a for-loop and the feature selection method SelectPercentile(), I tried features ranging from top 30% to 100%. To my surprise, the best f1 score is calculated after using all the features. Therefore, I decided to use all the features to predict the testing dataset.

Comparison of other models

Before digging into SVM, I also tried out some other classifiers. However, none of them performs as good as SVM model. Below are some of the models that I tried and the result I obtain.

SGDClassifier

```
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss="log", penalty="l2", max_iter=5)
%time scores = cross_validate(clf, features, label, scoring=['recall', 'precision', 'f1'], n_jobs=-1)
print(scores['test_recall'].mean(), scores['test_precision'].mean(), scores['test_f1'].mean())
```

```
CPU times: user 36.2 ms, sys: 2.01 ms, total: 38.2 ms
Wall time: 79.6 ms
0.12549019607843137 0.1124839124839125 0.1138047138047138
```

AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
bdt = AdaBoostClassifier(DecisionTreeClassifier(class_weight='balanced', max_depth=30), algorithm="SAMME", n_estimators=200)
%time scores = cross_validate(bdt, features, label, scoring=['recall', 'precision', 'f1'], n_jobs=-1)
print(scores['test_recall'].mean(), scores['test_precision'].mean(), scores['test_f1'].mean())
```

```
CPU times: user 41.2 ms, sys: 4.01 ms, total: 45.3 ms
Wall time: 523 ms
0.30708898944193064 0.3799887852213434 0.33811374083114504
```

RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=100, class_weight='balanced', random_state=0)
%time scores = cross_validate(clf, features, label, scoring=['recall', 'precision', 'f1'], n_jobs=-1)
print(scores['test_recall'].mean(), scores['test_precision'].mean(), scores['test_f1'].mean())
```

```
CPU times: user 46.7 ms, sys: 16 ms, total: 62.7 ms
Wall time: 2.94 s
0.17895927601809955 0.6908793820558526 0.2824389538424065
```

BaggingClassifier

```
[ ]  from sklearn.model_selection import cross_validate
     clf = BaggingClassifier(base_estimator=SVC(class_weight='balanced'), n_estimators=100, random_state=0)
     %time scores = cross_validate(clf, features, label, scoring=['recall', 'precision', 'f1'], n_jobs=-1) # n_job=-1 means use every core
     print(scores['test_recall'].mean(), scores['test_precision'].mean(), scores['test_f1'].mean())
```

```
CPU times: user 36.6 ms, sys: 6.28 ms, total: 42.9 ms
Wall time: 37.9 s
0.40067873303167423 0.3030175762934384 0.3441294868530117
```

My SVM model

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold, KFold

features = pd.read_csv(path + 'train/Train_Onehot.csv')
skf = StratifiedKFold(n_splits=10)
scaler = StandardScaler()
features = scaler.fit_transform(features)
svc = SVC(class_weight='balanced', C=7, gamma=0.0001, kernel='sigmoid')
model = Pipeline([('scaler', StandardScaler()), ('svc', svc)])
%time scores = cross_validate(svc, features, label, cv=skf, scoring=['recall', 'precision', 'f1'], n_jobs=-1) # n_job=-1 means use every core
print(scores['test_recall'].mean(), scores['test_precision'].mean(), scores['test_f1'].mean())
xfinal = pd.read_csv(path + 'fixed_test.csv')
Patient_id = preprocess(xfinal, 'fixed_test')
svc.fit(features, label)

xfinal_final = pd.read_csv(path + 'fixed_test_Onehot.csv')
xfinal_final = scaler.transform(xfinal_final)
yfinal = svc.predict(xfinal_final)
df= pd.DataFrame(columns = ['PATIENT ID', 'hospital_outcome'])
df['PATIENT ID'] = Patient_id
df['hospital_outcome'] = yfinal
df.to_csv(path + 'test_output_final.csv', index = False)
```

```
CPU times: user 50.4 ms, sys: 1.76 ms, total: 52.1 ms
Wall time: 1.49 s
0.7241538461538461 0.4192983781606968 0.5294968503079205
```

Result

The mortality prediction based on the testing data is written to a csv file (107062274.csv), which would be used to evaluate later.