Predictive Modeling

Series 7

Exercise 7.1

In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the `Auto` data set.

a) Load and inspect the data from `auto.cvs`. When needed, define categorical features with dummy variables. Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

b) Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of `gamma` and `degree` and `cost`. Comment on your results

Exercise 7.2

This problem involves the `OJ.csv` data set.

a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

b) Fit a support vector classifier to the training data using `cost=0.01`, with `Purchase` as the response and the other variables as predictors. Inspect the different attributes of the classifier, in particular the number of support vectors, and the classes to which they correspond.

c) What are the training and test error rates? Also give the corresponding confusion matrices.

d) Use the `GridSearchCV()` function to select an optimal cost. Consider 10 values in the range 0.01 to 10. If the computation takes too much time, you may want to increase the tolerance by using the *tol* setting, or setting a maximum number of iterations using the *max_iter* setting.

e) Compute the training and test error rates using this new value for `cost`.

f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for `gamma`.

g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set `degree=2`.

h) Overall, which approach seems to give the best results on this data?

Result Checker

Predictive Modeling

Solutions to Series 7

Solution 7.1

a) Python code:

[1]:
```
import numpy as np
import pandas as pd

# Load data
df = pd.read_csv('./data/auto.csv')

# Create new variable
df['mpg01'] = np.zeros(df.shape[0], dtype=int)
for i in range(df.shape[0]):
    if df.loc[i, 'mpg'] > df['mpg'].median():
        df.loc[i, 'mpg01'] = int(1)

# Redefine origin as a categorical variable
df = pd.get_dummies(data=df, drop_first=False,
                    columns=['origin'])
df = df.rename(columns={'origin_1': 'American',
                        'origin_2': 'European',
                        'origin_3': 'Japanese'})

# Drop and add columns
df = df.drop(['mpg', 'name'], axis=1)

# As a first inspection, print the first rows of the data:
print(df.head())
# As well as the dimensions of the set:
print('\nSize of Auto =\n', df.shape)
```

```
   cylinders  displacement  horsepower  weight  acceleration  year  mpg01    0 ␣
↪ 8        307.0          130     3504          12.0     70        0
1          8         350.0         165     3693          11.5     70        0
2          8         318.0         150     3436          11.0     70        0
3          8         304.0         150     3433          12.0     70        0
4          8         302.0         140     3449          10.5     70        0

   American  European  Japanese
0         1         0         0
1         1         0         0
2         1         0         0
3         1         0         0
4         1         0         0

Size of Auto =
 (392, 10)
```

b) The cross-validation errors can be produced with ease by the `GridsearchCV()` function from the `sklearn.model_selection` library. This is done by the code below.

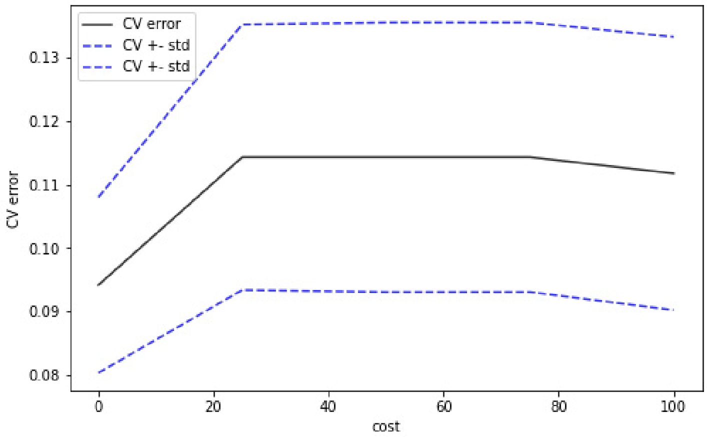
[2]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

n_folds = 10

# Define predictors and response:
y = df['mpg01']
x = df.drop('mpg01', axis=1)

# Set parameters to be tuned. Other options can be added
costs = np.linspace(0.05, 100, 5)
tune_parameters = {'C': costs}

# Tune SVM
clf_tune = GridSearchCV(svm.SVC(kernel='linear'),
                        tune_parameters,
                        cv=n_folds)
clf_tune.fit(x, y)

# Save Tune scores and corresponding standard deviation:
error_tune = 1 - clf_tune.cv_results_['mean_test_score']
error_std = clf_tune.cv_results_['std_test_score'] / np.sqrt(n_folds)

# Print some attributes of model:
print('Best parameter:\n', clf_tune.best_params_,
      '\nBest score:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter:
 'C': 0.05
Best score:
 0.0942
```

If we plot the values of the missclassification error, the minimum missclassification error occurs at cost 0.05.

[3]:
```
import matplotlib.pyplot as plt

# plot
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(1, 1, 1)

ax.plot(costs, error_tune,
        '-k', alpha=0.8, label='Cross validation error')
ax.plot(costs, error_tune + error_std, '--b',
        costs, error_tune - error_std, '--b',
```

```
        alpha=0.8, label='Cross validation error standard deviation')

ax.set_xlabel('cost')
ax.set_ylabel('error')
plt.legend()
plt.show()
```

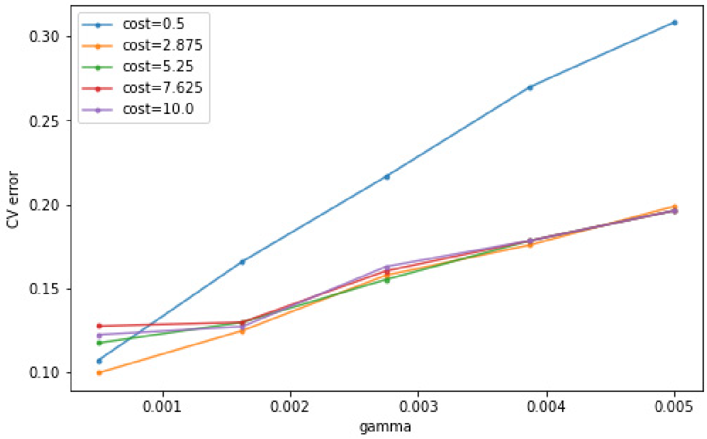


c) Radial Kernel:

[4]:
```
# Set parameters to be tuned. Other options can be added
costs = np.linspace(0.5, 10, 5)
gamma = np.linspace(0.0005, 0.005, 5)

tune_parameters = {'C': costs,
                   'gamma': gamma}

# Tune SVM
clf_tune = GridSearchCV(svm.SVC(kernel='rbf'),
                        tune_parameters,
                        cv=n_folds)
clf_tune.fit(x, y)

# Save Tune scores:
error_tune = 1 - clf_tune.cv_results_['mean_test_score']
error_tune = error_tune.reshape(len(costs), len(gamma))

# Print some attributes of model:
print('Best parameter Radial:\n', clf_tune.best_params_,
      '\nBest score Radial:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter Radial:
 'C': 2.875, 'gamma': 0.0005
Best score Radial:
 0.0998
```

[5]:
```
# plot
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(1, 1, 1)

# Plot error vs gamma for each value for cost:
for i in range(len(costs)):
    line, = ax.plot(gamma, error_tune[i, :],
            '.-', alpha=0.8)
    line.set_label(('cost=' + str(costs[i])))

ax.set_xlabel('gamma')
ax.set_ylabel('CV error')
plt.legend()
plt.show()
```

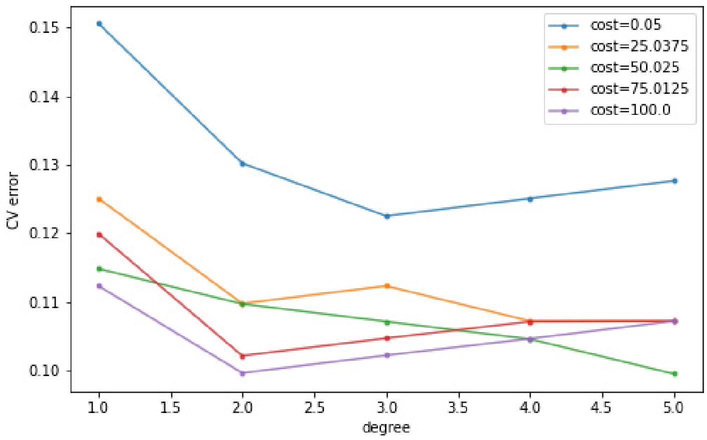


Polynomial Kernel:

[6]:
```
degree = [1, 2, 3, 4, 5]

tune_parameters = {'C': costs,
                   'degree': degree}

# Tune SVM
clf_tune = GridSearchCV(svm.SVC(kernel='poly'),
                        tune_parameters,
                        cv=n_folds)
clf_tune.fit(x, y)

# Save Tune scores:
error_tune = 1 - clf_tune.cv_results_['mean_test_score']
error_tune = error_tune.reshape(len(costs), len(degree))

# Print some attributes of model:
print('Best parameter Polynomial:\n', clf_tune.best_params_,
      '\nBest score Polynomial:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter Polynomial:
 'C': 100.0, 'degree': 2
Best score Polynomial:
 0.0996
```

[7]:
```
# plot
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(1, 1, 1)

# Plot error vs degree for each value of cost:
for i in range(len(costs)):
    line, = ax.plot(degree, error_tune[i, :],
                    '.-', alpha=0.8)
    line.set_label(('cost=' + str(costs[i])))

ax.set_xlabel('degree')
ax.set_ylabel('CV error')
plt.legend()
plt.show()
```



d)

Solution 7.2

a) Python code:

[1]:
```python
import numpy as np
import pandas as pd

# Load data
df = pd.read_csv('./data/OJ.csv')

# Redefine Store7 as a categorical variable
df = pd.get_dummies(data=df, drop_first=True,
                    columns=['Store7'])

# As a first inspection, print the first rows of the data:
print(df.head())
# As well as the dimensions of the set:
print('\nSize of OJ =\n', df.shape)
```

```
  Purchase  WeekofPurchase  StoreID  PriceCH  PriceMM  DiscCH  DiscMM   0        CH  ␣
↪        237         1      1.75     1.99     0.00     0.0
1       CH             239        1     1.75     1.99    0.00     0.3
2       CH             245        1     1.86     2.09    0.17     0.0
3       MM             227        1     1.69     1.69    0.00     0.0
4       CH             228        7     1.69     1.69    0.00     0.0

   SpecialCH  SpecialMM   LoyalCH  SalePriceMM  SalePriceCH  PriceDiff   0          0␣
↪          0  0.500000       1.99         1.75        0.24
1          0          1  0.600000         1.69         1.75      -0.06
2          0          0  0.680000         2.09         1.69       0.40
3          0          0  0.400000         1.69         1.69       0.00
4          0          0  0.956535         1.69         1.69       0.00

   PctDiscMM  PctDiscCH  ListPriceDiff  STORE  Store7_Yes
0   0.000000   0.000000           0.24      1           0
1   0.150754   0.000000           0.24      1           0
2   0.000000   0.091398           0.23      1           0
3   0.000000   0.000000           0.00      1           0
4   0.000000   0.000000           0.00      0           1

Size of OJ =
 (1070, 18)
```

[2]:
```python
# Set ramdom seed
np.random.seed(1)

i = df.index
# Index of train
i_train = np.random.choice(i, replace=False,
                           size=800)

# Save DataFrames
df_train = df.iloc[i_train]
```

```
df_test = df.drop(i_train)

# Check dimensions:
print('\nSize of Train =\n', df_train.shape,
      '\nSize of Test =\n', df_test.shape)
```

```
Size of Train =
 (800, 18)
Size of Test =
 (270, 18)
```

b) **Python** code:

[3]:
```
from sklearn import svm

# Define predictors and response:
y_train = df_train['Purchase']
x_train = df_train.drop('Purchase', axis=1)

# Fit SVM
cost = 0.01
clf = svm.SVC(kernel='linear', C=cost)
clf.fit(x_train, y_train)

# Print information on Support vectors:
print("Classes: ", clf.classes_,
      "\nNumber of Support Vectors: ", clf.n_support_)
```

```
Classes:  ['CH' 'MM']
Number of Support Vectors:  [312 313]
```

From the **Python** output we can see that 312 and 313 observations are used as support vector for the respective classes.

c) **Python** code:

[4]:
```
# Define predictors and response:
y_test = df_test['Purchase']
x_test = df_test.drop('Purchase', axis=1)

# Find error rates:
e_train = 1 - clf.score(x_train, y_train)
e_test = 1 - clf.score(x_test, y_test)

print('Train error:\n', np.round(e_train, 4),
      '\nTest error:\n', np.round(e_test, 4))
```

```
Train error:
 0.2175
Test error:
 0.2222
```

[5]:
```
def table_scores(ypredicted, ytrue):
    """ Return table showing predicted and true scores in n*n matrix
    Inputs:
    - Vector containing n predicted values
    - Vector containing n true values
    Returns:
    n*n Matrix with number of correct predictions on diagonal """
    # Empty Matrix:
    lables = np.unique(ytrue, return_inverse=False, axis=None)
    n = len(lables)
    scores = np.zeros((n, n))
    # Fill matrix with values:
    for i in range(len(ytrue)):
        true_class, pred_class = ytrue[i], ypredicted[i]
        scores[np.where(true_class == lables)[0][0]][
               np.where(pred_class == lables)[0][0]] += 1
    # Name rows and columns:
    r, c = [], []
    for i in range(len(lables)):
        r.append(("True " + str(lables[i])))
        c.append(("Pred " + str(lables[i])))
    scores = pd.DataFrame(scores, columns=c, index=r)

    return scores

print('Confusion Matrix train: \n',
      table_scores(clf.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
      table_scores(clf.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    432.0     49.0
True MM    125.0    194.0

Confusion Matrix test:
          Pred CH  Pred MM
True CH    151.0     21.0
True MM     39.0     59.0
```

d) **Python** code:

[6]:
```
from sklearn.model_selection import GridSearchCV

n_folds = 10

# Set parameters to be tuned. Other options can be added
costs = np.linspace(0.01, 10, 10)

tune_parameters = {'C': costs}
```

```
# Tune SVM
clf_tune = GridSearchCV(
    svm.SVC(kernel='linear', max_iter=1e6, tol=1e-1),
    tune_parameters, cv=n_folds)
clf_tune.fit(x_train, y_train)

# Print some attributes of model:
print('Best parameter:\n', clf_tune.best_params_,
      '\nBest score:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter:
 'C': 3.34
Best score:
 0.1637
```

e) **Python** code:

[7]:
```
print('Confusion Matrix train: \n',
      table_scores(clf_tune.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
      table_scores(clf_tune.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    427.0     54.0
True MM     71.0    248.0

Confusion Matrix test:
          Pred CH  Pred MM
True CH    148.0     24.0
True MM     24.0     74.0
```

f) **Python** code:

[8]:
```
# Fit SVM with radial kernel
cost = 0.01
clf = svm.SVC(kernel='rbf', C=cost)
clf.fit(x_train, y_train)

# Print information on Support vectors:
print("Classes: ", clf.classes_,
      "\nNumber of Support Vectors: ", clf.n_support_)
```

```
Classes:  ['CH' 'MM']
Number of Support Vectors:  [319 319]
```

[9]:
```
print('Confusion Matrix train: \n',
      table_scores(clf.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
```

```
      table_scores(clf.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    481.0      0.0
True MM    319.0      0.0

Confusion Matrix test:
          Pred CH  Pred MM
True CH    172.0      0.0
True MM     98.0      0.0
```

[10]:
```
# Tune SVM
clf_tune = GridSearchCV(svm.SVC(kernel='rbf'),
                        tune_parameters,
                        cv=n_folds)
clf_tune.fit(x_train, y_train)

# Print some attributes of model:
print('Best parameter:\n', clf_tune.best_params_,
      '\nBest score:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter:
 'C': 0.01
Best score:
 0.3988
```

[11]:
```
print('Confusion Matrix train: \n',
      table_scores(clf_tune.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
      table_scores(clf_tune.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    481.0      0.0
True MM    319.0      0.0

Confusion Matrix test:
          Pred CH  Pred MM
True CH    172.0      0.0
True MM     98.0      0.0
```

g) **Python** code:

[12]:
```
# Fit SVM with radial kernel
cost = 0.01
clf = svm.SVC(kernel='poly', C=cost, degree=2)
clf.fit(x_train, y_train)

# Print information on Support vectors:
```

```
print("Classes: ", clf.classes_,
      "\nNumber of Support Vectors: ", clf.n_support_)
```

```
Classes:  ['CH' 'MM']
Number of Support Vectors:  [319 319]
```

[13]:
```
# Print confusion Matrix
print('Confusion Matrix train: \n',
      table_scores(clf.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
      table_scores(clf.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    481.0      0.0
True MM    319.0      0.0

Confusion Matrix test:
          Pred CH  Pred MM
True CH    172.0      0.0
True MM     98.0      0.0
```

[14]:
```
# Tune SVM
clf_tune = GridSearchCV(
    svm.SVC(kernel='poly', degree=2),
    tune_parameters, cv=n_folds)
clf_tune.fit(x_train, y_train)

# Print some attributes of model:
print('Best parameter:\n', clf_tune.best_params_,
      '\nBest score:\n', np.round(1 - clf_tune.best_score_, 4))
```

```
Best parameter:
 'C': 0.01
Best score:
 0.3988
```

[15]:
```
print('Confusion Matrix train: \n',
      table_scores(clf_tune.predict(x_train), y_train.to_numpy()),
      '\n\nConfusion Matrix test: \n',
      table_scores(clf_tune.predict(x_test), y_test.to_numpy()))
```

```
Confusion Matrix train:
          Pred CH  Pred MM
True CH    481.0      0.0
True MM    319.0      0.0

Confusion Matrix test:
          Pred CH  Pred MM
```

```
True CH     172.0       0.0
True MM      98.0       0.0
```

h) The linear SVM performs best on this data. However, further investigating the optimal parameters for **gamma**, **degree** and **cost** could improve the behaviour of different classifiers.