Optimizing the SVM Classifier

Machine learning models are parameterized so that their behavior can be tuned for a given problem. Models can have many parameters and finding the best combination of parameters can be treated as a search problem. In this notebook, I aim to tune parameters of the SVM Classification model using scikit-learn.

Load Libraries and Data

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
import time
```

In [2]:

```
data = pd.read_csv('C:\data\clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)
```

In [3]:

```
print(data.groupby('diagnosis').size())
```

```
diagnosis
B    357
M    212
dtype: int64
```

In [4]:

```
Y = data['diagnosis'].values
X = data.drop('diagnosis', axis=1).values

X_train, X_test, Y_train, Y_test = train_test_split (X, Y, test_size = 0.20, random_state=2
```

Importance of optimizing a classifier

We can tune two key parameters of the SVM algorithm:

- the value of C (how much to relax the margin)

- and the type of kernel.

The default for SVM (the SVC class) is to use the Radial Basis Function (RBF) kernel with a C value set to 1.0. We will try a number of simpler kernel types and C values with less bias and more bias (less than and more than 1.0 respectively).

Python scikit-learn provides two simple methods for algorithm parameter tuning:

- Grid Search Parameter Tuning.
- Random Search Parameter Tuning.

Grid Search Parameter Tuning

In [5]:

```
models_list = []
models_list.append(('SVM', SVC()))
num_folds = 10
results = []
names = []

for name, model in models_list:
    kfold = KFold(n_splits=num_folds, random_state=123)
    start = time.time()
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    end = time.time()
    results.append(cv_results)
    names.append(name)
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
kfold = KFold(n_splits=num_folds, random_state=21)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=kfold)
grid_result = grid.fit(rescaledX, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.969231 using {'C': 2.0, 'kernel': 'rbf'}
0.964835 (0.026196) with: {'C': 0.1, 'kernel': 'linear'}
0.826374 (0.058723) with: {'C': 0.1, 'kernel': 'poly'}
0.940659 (0.038201) with: {'C': 0.1, 'kernel': 'rbf'}
0.949451 (0.032769) with: {'C': 0.1, 'kernel': 'sigmoid'}
0.962637 (0.029474) with: {'C': 0.3, 'kernel': 'linear'}
0.868132 (0.051148) with: {'C': 0.3, 'kernel': 'poly'}
0.958242 (0.031970) with: {'C': 0.3, 'kernel': 'rbf'}
0.958242 (0.033368) with: {'C': 0.3, 'kernel': 'sigmoid'}
0.956044 (0.030933) with: {'C': 0.5, 'kernel': 'linear'}
0.881319 (0.050677) with: {'C': 0.5, 'kernel': 'poly'}
0.964835 (0.029906) with: {'C': 0.5, 'kernel': 'rbf'}
0.953846 (0.026785) with: {'C': 0.5, 'kernel': 'sigmoid'}
0.953846 (0.031587) with: {'C': 0.7, 'kernel': 'linear'}
0.885714 (0.038199) with: {'C': 0.7, 'kernel': 'poly'}
0.967033 (0.037271) with: {'C': 0.7, 'kernel': 'rbf'}
0.953846 (0.028513) with: {'C': 0.7, 'kernel': 'sigmoid'}
0.951648 (0.028834) with: {'C': 0.9, 'kernel': 'linear'}
0.887912 (0.038950) with: {'C': 0.9, 'kernel': 'poly'}
0.967033 (0.037271) with: {'C': 0.9, 'kernel': 'rbf'}
0.949451 (0.034009) with: {'C': 0.9, 'kernel': 'sigmoid'}
0.953846 (0.026546) with: {'C': 1.0, 'kernel': 'linear'}
0.890110 (0.038311) with: {'C': 1.0, 'kernel': 'poly'}
0.967033 (0.033027) with: {'C': 1.0, 'kernel': 'rbf'}
0.947253 (0.032755) with: {'C': 1.0, 'kernel': 'sigmoid'}
0.956044 (0.025765) with: {'C': 1.3, 'kernel': 'linear'}
0.894505 (0.039427) with: {'C': 1.3, 'kernel': 'poly'}
0.967033 (0.028188) with: {'C': 1.3, 'kernel': 'rbf'}
0.942857 (0.031144) with: {'C': 1.3, 'kernel': 'sigmoid'}
```

```
0.958242 (0.024765) with: {'C': 1.5, 'kernel': 'linear'}
0.896703 (0.039791) with: {'C': 1.5, 'kernel': 'poly'}
0.967033 (0.028188) with: {'C': 1.5, 'kernel': 'rbf'}
0.940659 (0.035237) with: {'C': 1.5, 'kernel': 'sigmoid'}
0.956044 (0.021766) with: {'C': 1.7, 'kernel': 'linear'}
0.903297 (0.033409) with: {'C': 1.7, 'kernel': 'poly'}
0.967033 (0.024479) with: {'C': 1.7, 'kernel': 'rbf'}
0.945055 (0.035539) with: {'C': 1.7, 'kernel': 'sigmoid'}
0.956044 (0.021766) with: {'C': 2.0, 'kernel': 'linear'}
0.909890 (0.033680) with: {'C': 2.0, 'kernel': 'poly'}
0.969231 (0.022370) with: {'C': 2.0, 'kernel': 'rbf'}
0.931868 (0.028237) with: {'C': 2.0, 'kernel': 'sigmoid'}
```

We can see the most accurate configuration was SVM with an RBF kernel with the accuracy of 96.92%.

In [6]:

```python
import warnings
# prepare the model
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
model = SVC(C=2.0, kernel='rbf')
start = time.time()
model.fit(X_train_scaled, Y_train)
end = time.time()
print( "Run Time: %f" % (end-start))
```

```
Run Time: 0.006002
```

In [7]:

```python


# estimate accuracy on test dataset
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    X_test_scaled = scaler.transform(X_test)
predictions = model.predict(X_test_scaled)
print("Accuracy score %f" % accuracy_score(Y_test, predictions))
print(classification_report(Y_test, predictions))
```

```
Accuracy score 0.991228
             precision    recall  f1-score   support

          B       1.00      0.99      0.99        75
          M       0.97      1.00      0.99        39

avg / total       0.99      0.99      0.99       114
```

In [9]:

```python
print(confusion_matrix(Y_test, predictions))
```

```
[[74  1]
 [ 0 39]]
```