Assignment 2

60-473/574

Dr. Rueda

Joel Rorseth

November 3, 2017

For this assignment, I have chosen to employ the *sklearn.svm.SVC* class from the Scikit-learn Python library. Scikit-learn is the most popular, open source machine learning library for the Python environment. The SVC class included supports a great variety of SVC classifiers, allowing many types of kernels including linear, polynomial, ref and sigmoid. The implementation of this functionality utilizes the popular libSVM software under the hood.

In the context of the assignment, the SVC classifier is created directly as an instance of the SVC class. To instantiate an SVM classifier with a specific kernel, it is simply specified in the constructor.

```python
from sklearn import svm
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

import numpy as np
import pandas as pd


# SVM Classifier using linear kernel
def svm_linear():
    return svm.SVC(kernel='linear')

# SVM Classifier using deg. 2 polynomial kernel
def svm_polynomial():
    return svm.SVC(kernel='poly', degree=2)

# SVM Classifier using RBF
def svm_rbf():
    return svm.SVC(kernel='rbf')
```

Figure 1.1

It is then used with a built in cross validation algorithm (specified as 10 fold), which then feeds the prediction statistics into a confusion matrix. The true/false positive/negative rates are returned by the confusion matrix Scikit algorithm, which is then used to manually calculate each of the five efficiency statistics by means of simple formulas.

```python
# Return 5-tuple corresponding to the 5 calculated measures of efficiency
def calculate_efficiency(classifier, samples, labels):

    # Use confusion matrix to extract statistics on cross validated prediction
    prediction = cross_val_predict(classifier, samples, labels, cv=10)
    tn, fp, fn, tp = confusion_matrix(labels, prediction).ravel()

    # Score the training fit compared against the test samples
    return accuracy(tn, fp, fn, tp), sensitivity(tp, fn),\
            specificity(tn, fp), ppv(tp, fp), npv(tn, fn)
```

Figure 1.2

To obtain an ROC curve for the RBF SVM classifier, a simpler approach had to be taken to generate efficiency statistics for each sample. The classifier is fitted according to its definition, yielding an array of *n* scores, where *n* is the number of samples in the test dataset. These are obtained by evaluating each test sample, and producing a score based on the accuracy of the prediction for each corresponding test sample. A true and false positive rate (TPR and FPR) is obtained from the Scikit

```python
# SVM Classifier using RBF
def svm_rbf(samples, labels):

    classifier = svm.SVC(kernel='rbf')

    # Use standard train/test split algorithm to obtain distinct sets
    X_train, X_test, y_train, y_test = train_test_split(samples, labels, test_size=.5, random_state=0)

    # Use classifier to fit model to training data, then score it with test data
    test_scores = classifier.fit(X_train, y_train).decision_function(X_test)

    # Find True and False Positive rate for ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, test_scores, pos_label=2)

    # Calculate AUC
    roc_auc = auc(fpr, tpr)
    return fpr, tpr, roc_auc
```

Figure 1.3

roc_curve function, which are graphed directly into the ROC curve displayed in Figure 3.1. Sequential pairs from the TPR and FPR arrays directly map to values on the Y and X axis respectively.

After creating three SVM classifiers with linear, polynomial (degree 2) and RBF kernels respectively, they are trained and tested using 10 fold cross validation, with the efficiency results being outputted as shown below in Figure 2.1.

```
SVM-L: Linear Kernel on twogaussians.csv
Accuracy:      0.9700
Sensitivity:   0.9691
Specificity:   0.9709
PPV:           0.9691
NPV:           0.9709

SVM-L: Linear Kernel on halfkernel.csv
Accuracy:      0.7330
Sensitivity:   0.5980
Specificity:   0.8680
PPV:           0.8192
NPV:           0.6835

SVM-L: Linear Kernel on twospirals.csv
Accuracy:      0.6420
Sensitivity:   0.6680
Specificity:   0.6160
PPV:           0.6350
NPV:           0.6498

SVM-L: Linear Kernel on clusterincluster.csv
Accuracy:      0.5420
Sensitivity:   0.3960
Specificity:   0.6880
PPV:           0.5593
NPV:           0.5325


SVM-P: Polynomial Kernel on twogaussians.csv
Accuracy:      0.9675
Sensitivity:   0.9691
Specificity:   0.9660
PPV:           0.9641
NPV:           0.9707

SVM-P: Polynomial Kernel on halfkernel.csv
Accuracy:      0.7920
Sensitivity:   0.5840
Specificity:   1.0000
PPV:           1.0000
NPV:           0.7062

SVM-P: Polynomial Kernel on twospirals.csv
Accuracy:      0.5560
Sensitivity:   0.8140
Specificity:   0.2980
PPV:           0.5369
NPV:           0.6157

SVM-P: Polynomial Kernel on clusterincluster.csv
Accuracy:      1.0000
Sensitivity:   1.0000
Specificity:   1.0000
PPV:           1.0000
NPV:           1.0000


SVM-R: RBF on twogaussians.csv
Accuracy:      0.9750
Sensitivity:   0.9794
Specificity:   0.9709
PPV:           0.9694
NPV:           0.9804

SVM-R: RBF on halfkernel.csv
Accuracy:      1.0000
Sensitivity:   1.0000
Specificity:   1.0000
PPV:           1.0000
NPV:           1.0000

SVM-R: RBF on twospirals.csv
Accuracy:      0.9480
Sensitivity:   0.9460
Specificity:   0.9500
PPV:           0.9498
NPV:           0.9462

SVM-R: RBF on clusterincluster.csv
Accuracy:      1.0000
Sensitivity:   1.0000
Specificity:   1.0000
PPV:           1.0000
NPV:           1.0000
```

Figure 2.1

Upon initial observation, it is apparent that the classifiers have mixed performance across the variety of kernels and datasets in the experiment.

Without a doubt, the most prominent pattern evident in the measures of efficiency is that of the performance of all classifiers on the *twogaussians.csv* dataset. In our experiment, *every* efficiency measure from *every* classifier's run on the dataset scored an outstanding 0.96 or higher. This makes sense however, as Support Vector Classifiers specifically perform well on smaller datasets. *twogaussians.csv* is relatively small, containing 400 samples as compared to the 1000 samples recorded in the other three datasets.

By definition, linear kernel SVM classifiers are an ideal pick for small numbers of samples, or a large number of features (due to the simplicity of its linear classification). In Figure 2.1, the efficiency scores of the linear kernel SVM are exceptional when trained and tested on *twogaussians.csv*, the only relatively small sample sized dataset. In contrast, it makes sense that it performs poorly on the remaining datasets because of their much larger sample size (1000 as opposed to 400 for *twogaussians.csv*). Most importantly, we know from the visualizations of the datasets in Figure 2.2 that the only *linearly separable* dataset is in fact *twogaussians.csv*, and thus the linear kernel SVM would have a very hard time making a linear classification for any other dataset.

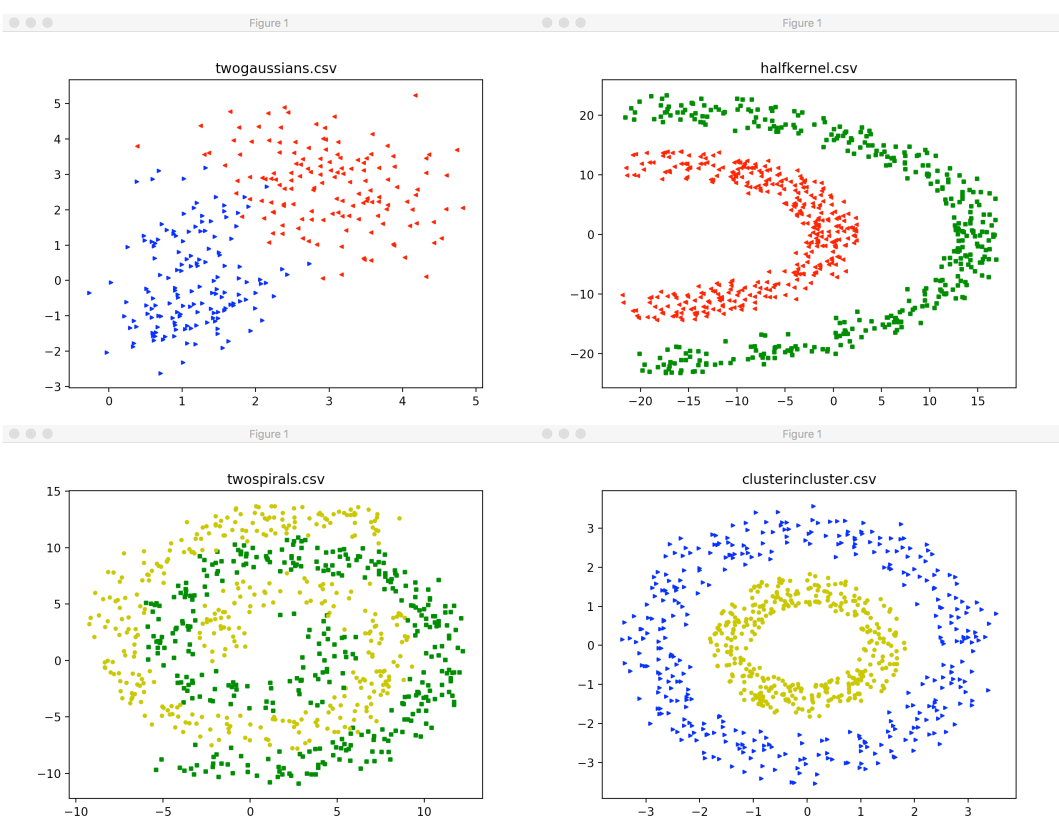

Figure 2.2

The polynomial kernel, degree 2 SVM classifier performs well in general, as it employs a polynomial classification technique that handles linearly inseparable datasets. It's classification performance is high on all datasets except *twospirals.csv*. This is because points from both classes are significantly interspersed, as seen in the visualization in Figure 2.2. Polynomial classification, by nature, will struggle with these interspersed samples, which make it difficult to define an adequate margin via support vectors. The efficiency of this classifier is noticeably worse than that of the other kernels. This is due to the fact that polynomial equations are usually *computationally expensive*, especially in comparison to the techniques used by other kernels.

Without a doubt, the radial basis function (RBF) kernel SVM classifier outperforms all other SVM classifiers in terms of prediction efficiency. Due to its Gaussian computational characteristics, it is particularly good at enforcing separability. Because it measures the Euclidean distance between samples during classification, it can discriminate samples regardless of linearly separability. This is distinctly evident in the efficiency scores (Figure 1) for *halfkernel.csv* and *clusterincluster.csv*, which both score 1.0 for all measures. As clearly depicted in the dataset visualizations of Figure 2.2, both of these datasets have a noticeably large margin that will be easy to maximize with support vectors. Perfect classification is visually sensible for these datasets, however the slightly lower scores for the remaining datasets is due to that lack of potential (and separating gap) to maximize the margin between classes.

In summary, the optimal kernel choices for each dataset are obvious by comparing their classification nature (in the context of SVM) to the dataset's visual representation. RBF kernel SVM classifier should be used to classify all of them for optimal classification performance, especially *halfkernel.csv* and *clusterincluster.csv.* However, linear kernel would be a computationally inexpensive alternative for *twogaussians.csv*, while maintaining a near perfect classification efficiency score. *twospirals.csv* should be classified by RBF, as there is no distinct linear or polynomial separation, clearly indicated by its underwhelming score on other kernels.

```
Joels-iMac:Assignment2 joelrorseth$ python3 question3.py
Now running SVM Classifier (Linear Kernel)

Evaluating twogaussians.csv...
Evaluating halfkernel.csv...
Evaluating twospirals.csv...
Evaluating clusterincluster.csv...

Summary
--------------------------------
Accuracy:        0.72175
Sensitivity:     0.657768041237
Specificity:     0.785718446602
PPV:             0.745638317408
NPV:             0.709158780703


Now running SVM Classifier (Polynomial Kernel, Degree = 2)

Evaluating twogaussians.csv...
Evaluating halfkernel.csv...
Evaluating twospirals.csv...
Evaluating clusterincluster.csv...

Summary
--------------------------------
Accuracy:        0.828875
Sensitivity:     0.841768041237
Specificity:     0.816004854369
PPV:             0.875260469522
NPV:             0.82316221898


Now running SVM Classifier (RBF)

Evaluating twogaussians.csv...
Evaluating halfkernel.csv...
Evaluating twospirals.csv...
Evaluating clusterincluster.csv...

Summary
--------------------------------
Accuracy:        0.98075
Sensitivity:     0.981345360825
Specificity:     0.980218446602
PPV:             0.979796737972
NPV:             0.981651824076


Joels-iMac:Assignment2 joelrorseth$
```

Figure 2.3

If we take an alternate perspective on the efficiency of each classifier averaged across all datasets (Figure 2.3), there is a distinct performance difference between each. When run over every dataset and its resulting efficiency measures are averaged, SVM with linear kernel is the worst performing, with most measures falling in the mid 0.65 - 0.75 range. It is bested by SVM with polynomial kernel, with measures in the 0.8 - 0.9 range. The RBF SVM outperforms both others on average, scoring an average in the high 0.97 - 1.0 range.

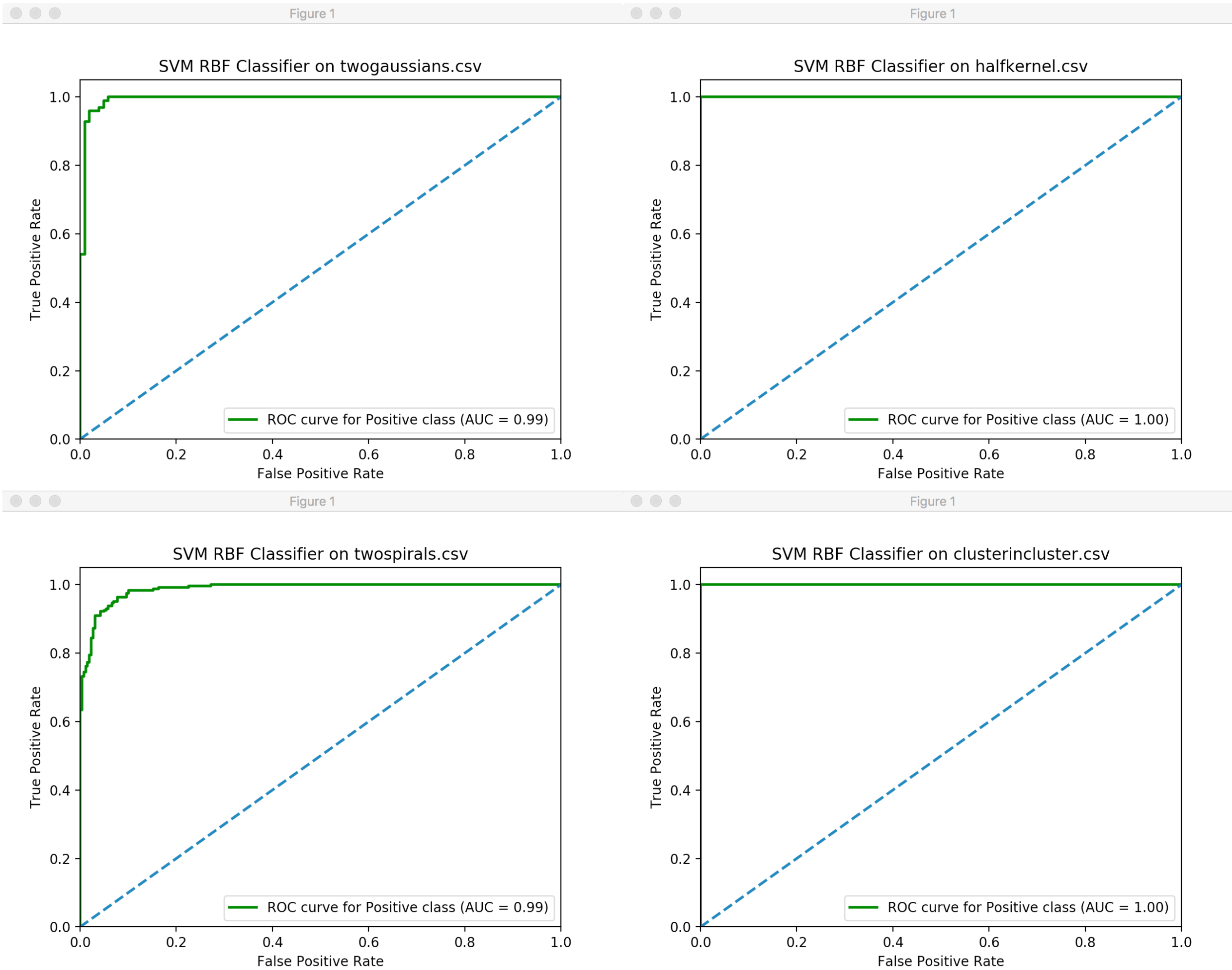

Figure 3.1

When graphed in the form of an ROC curve, the classification efficiency of the RBF kernel becomes apparent. Most noticeably, the straight, horizontal line TPR = 1.0 is the graphical representation of a perfect classification. We know from the five efficiency measures in Figure 2.1 that *twogaussians.csv* and *twospirals.csv* scored excellent efficiency ratings on all five measures, coming in around the mid 0.90 range. The distinct curve in the upper left corner of these graphs represents this high score, with both being very close in proximity to the optimal positioning (being in the top left corner).

We can observe that the area under the curve (AUC) is the maximum value of 1.0 for the perfectly classified datasets, and an extremely impressive 0.99 for the remaining datasets. We know that AUC is a sound measure of performance for our classifiers, with maximum values being preferable to minimal. Therefore, we can conclude that based on the shape and AUC, all four datasets have been classified *extremely well* under the SVM classifier with the RBF kernel function.