Machine Learning – CS 6375.001
Assignment – 3

Ameya S. Gamre
asg160330

LIBSVM is an integrated software for support vector classification, regression and distribution estimation. It supports multi-class classification.
LIBSVM provides a simple interface where users can easily link it with their own programs. **Main features of LIBSVM include** - Different SVM formulations, Efficient multi-class classification, Cross validation for model selection, Probability estimates, Various kernels (including precomputed kernel matrix), Weighted SVM for unbalanced data and interfaces with other languages such as Python, R, MATLAB, etc.

A Kernel Function is a function that obeys certain mathematical properties.
Suppose that we have a two-class dataset, and we wish to train a classifier to predict the class labels of future data points. This is known as a **"binary classification"** problem, and can be cast as "Yes"/"No" questions such as:

Medicine
    Given a patient's vital data, does the patient have a cold?
Computer Vision
    Does this image contain a person?

Many problems inherently have more than two possible outcomes. For instance, you may want to train a face verification system that can detect the identity of a photograph from a pool of N people (where $N > 2$). This sort of problem is known as a "**multiclass**" classification problem.

A popular classifier is the **Support Vector Machine (SVM)**, so we will use this as our classification algorithm.
Most off-the-shelf classifiers allow the user to specify one of three popular kernels: the polynomial, radial basis function, and sigmoid kernel.
The resulting SVMs are able to learn high-quality decision boundaries through the application of kernels.
I tried these different kernels ranging from 0-3,
0 -- linear: u'*v
1 -- polynomial: (gamma*u'*v + coef0)^degree
2 -- radial basis function: exp(-gamma*|u-v|^2)
3 -- sigmoid: tanh(gamma*u'*v + coef0)

out of which default is 2 -- radial basis function

The other parameters such as degree, gamma, cost, nu, epsilon,etc. are all kept to their default values.

Following are the observed results on trying various kernels:

OUTPUTS:

```
>> ./svm-train -t 0 training.new
....*....*
optimization finished, #iter = 579
nu = 0.017662
obj = -0.627017, rho = 1.172955
nSV = 40, nBSV = 0
Total nSV = 40

>> ./svm-predict validation.new training.new.model output-0.txt
Accuracy = 85.7143% (30/35) (classification)
```

```
>> ./svm-train -t 1 training.new
.*.*
optimization finished, #iter = 162
nu = 0.022567
obj = -0.801149, rho = 0.404372
nSV = 57, nBSV = 0
Total nSV = 57

>> ./svm-predict validation.new training.new.model output-1.txt
Accuracy = 74.2857% (26/35) (classification)
```

```
>> ./svm-train -t 2 training.new
.*
optimization finished, #iter = 99
nu = 0.801753
obj = -30.091940, rho = -0.076980
nSV = 71, nBSV = 22
Total nSV = 71

>> ./svm-predict validation.new training.new.model output-2.txt
Accuracy = 77.1429% (27/35) (classification)
```

```
>> ./svm-train -t 3 training.new
*
optimization finished, #iter = 37
nu = 0.957746
```

obj = -65.367107, rho = -0.492870
nSV = 68, nBSV = 68
Total nSV = 68

>> ./svm-predict validation.new training.new.model output-3.txt
Accuracy = 45.7143% (16/35) (classification)

The **linear kernel** works the best with an accuracy of **85%** whereas the **sigmoid kernel** is poor with an accuracy of 45%.

What I learned:
By replacing all dot products with a kernel function, we can implicitly work in a higher-dimensional space, without explicitly building the higher-dimensional representation. Thus, the SVM can learn a nonlinear decision boundary in the original space, which corresponds to a linear decision boundary in a higher dimensional space.
This is the "trick" in "Kernel trick".

Conclusion: Successfully evaluated SVMs with various kernels empirically and observed the outputs.