Minh Tran – HW6: Support Vector Machine

1. Part 1: Implementation:

a. Linear SVM:

- Libraries:

Pycharm IDE is used with Python 3.6 as the programming language. Basic libraries such as numpy, matplotlib and sys are used to facilitate the program. CVXOPT (convex optimization) library is used for constrained optimization solver.

- Linear SVM details:

✓ Using linear kernel (dot product)

- Data structure:

✓ The given txt files are read by np.loadtxt, then separates into X and y

✓ Train_test_split is used to separate the given dataset to train and test sets with random seed embedded in the function to ensure controlled randomization

✓

- Files included:

✓ svmMT.py - Contains the implementation of SVM from scratch

- Class SupportVectorMachine:
  - initialization (kernel type, kernel parameters including power for polynomial and gamma for rbf, penalty term C, intercept, weights, support vector sv)
  - fit (with embedded cvxopt optimizer and kernel calculation)
  - predict: classify a new point
  - predict_X2: strictly for linear kernel plotting
  - plot_linear_SVM, plot_nonlinear_SVM, display: for plotting purposes
- main: prepare data, build svm engine, train, test and plot.

✓ utils.py:

- Different types of kernel functions: linear, polynomial, rbf
- Accuracy_score function

- Data preparation functions: shuffle_data, train_test_split, normalize, standardize

- Code-level optimization:

✓ Only the Lagrange multipler greater than certain threshold is selected to make the solution scalable.

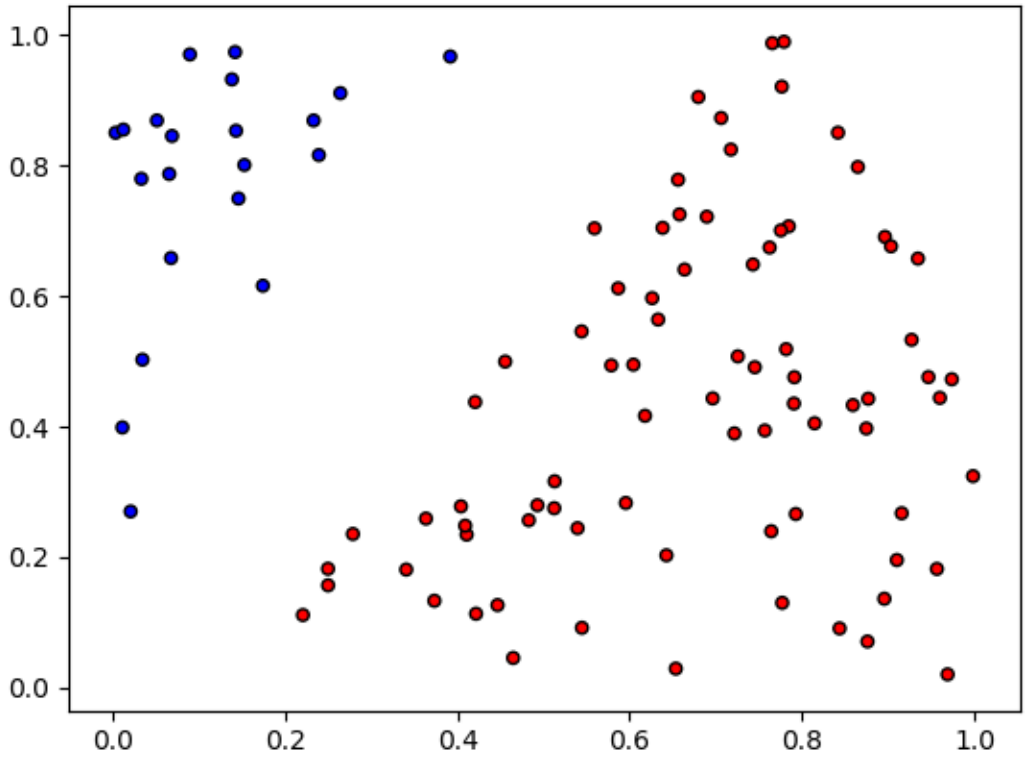
✓ Penalty terms are added to further constrain the lagrange multiplers if needed

✓ Different test sizes are tested to observe the robustness of the svm engine

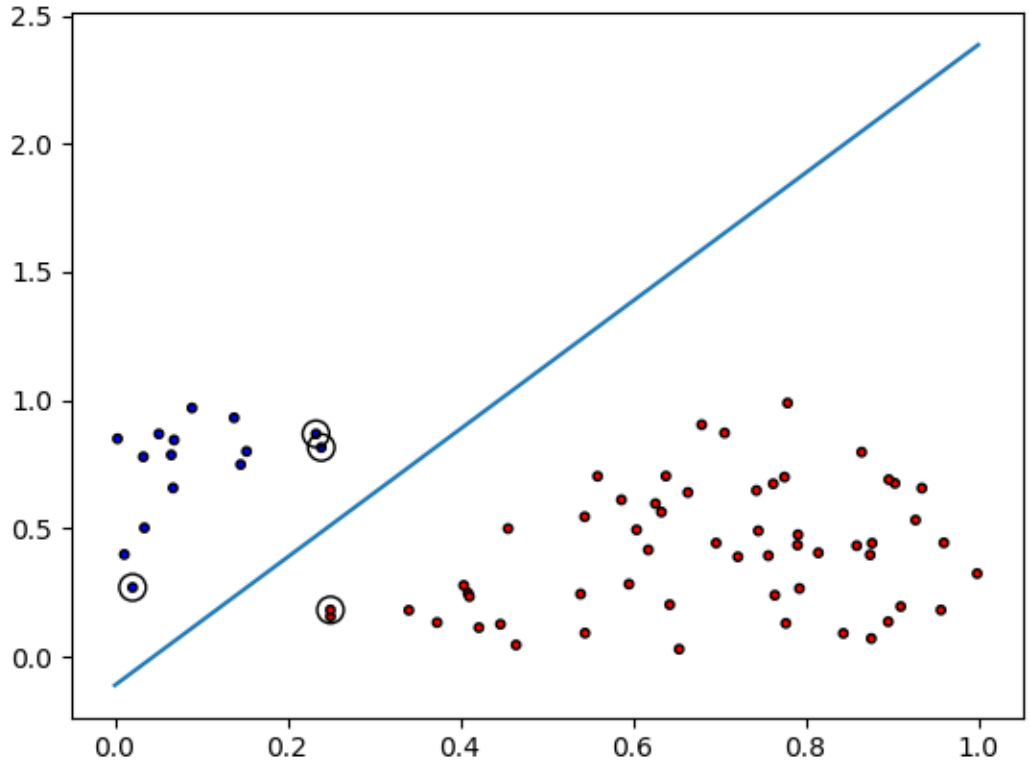
- Challenges:

✓ Dataset is quite small so if we trained only a subset of data (say 30%), then testing error may be high.

- Results:
  - ✓ Original dataset is plotted:



- ✓ Setting the test size to be 30%, then the linear SVM plane is plotted with the support vectors circled



- ✓ Testing accuracy: 100% (because the data is well linearly separated)
- ✓ Training accuracy: 100%
- ✓ Support vectors:

```
[[0.23918196 0.81585285]
 [0.24979414 0.18230306]
 [0.23307747 0.86884518]
 [0.02066458 0.27003158]]
```

With corresponding labels {-1, 1, -1, -1}, so 3 points on the one side and the other point on the other side.

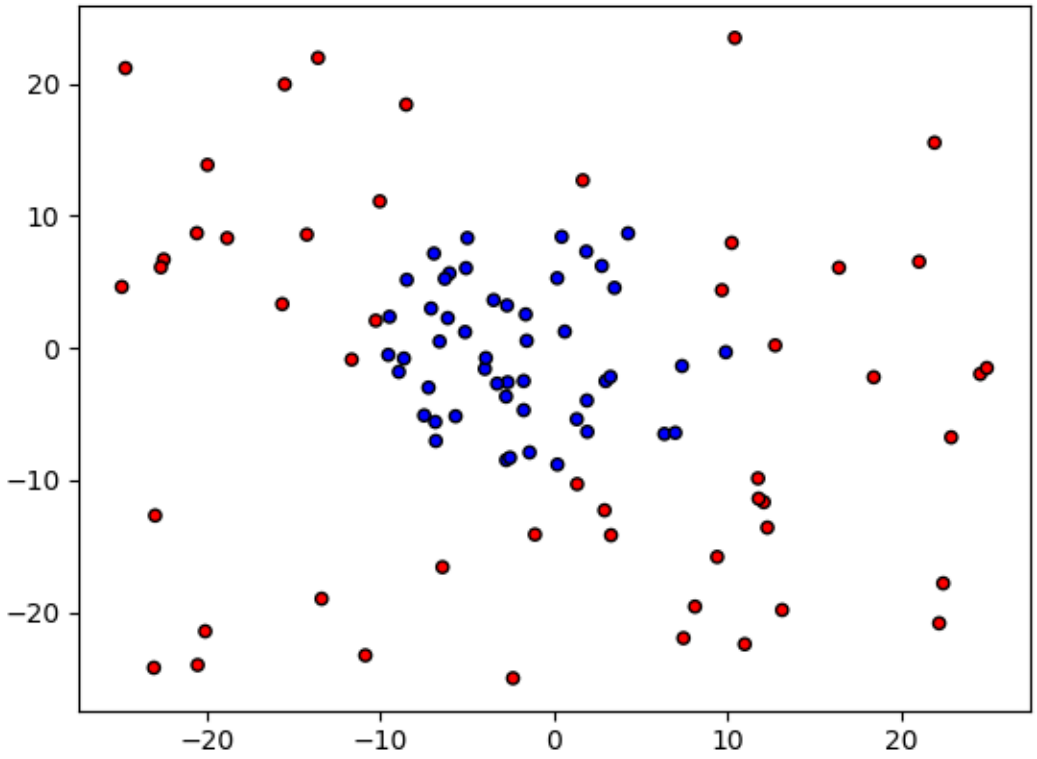
b. Nonlinear SVM:

- Nonlinear SVM details:

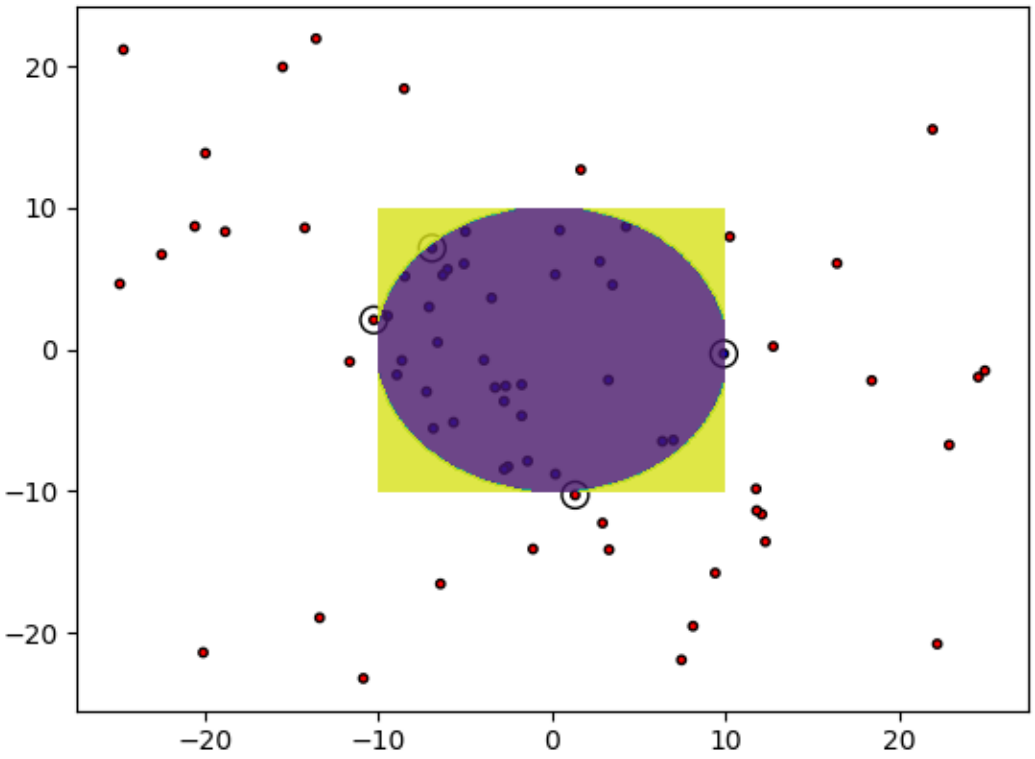
✓ Using polynomial kernel (power=2, coefficient=0) and rbf kernel (gamma=0.4)

- Results:

✓ Original data:



✓ Setting the test size to be 30%, then the nonlinear polynomial SVM plane is plotted with the support vectors circled



- ✓ Testing accuracy: 100% (because the data is well linearly separated)
- ✓ Training accuracy: 100%
- ✓ Support vectors:

```
[[0.23918196 0.81585285]
 [0.24979414 0.18230306]
 [0.23307747 0.86884518]
 [0.02066458 0.27003158]]
```

With corresponding labels {1, 1, -1, -1}, so 2 points on the one side and the other 2 points on the other side.

2. Part 2: Software familiarization

a. Sklearn for SVM:

✓ Scikit-Learn offers useful built-in packages to implement SVM with the ability to tune different parameters

✓ 3 simple steps are needed:

- Import data
- Create model using the module svm.SVC

```
# Build svm engine using sklearn
"""
penalty: Specifies the norm used in the penalization
loss: specify the loss function
dual: Select the algorithm to either solve the dual or primal optimization problem.
Prefer dual=False when n_samples > n_features
tol: tolerance for stopping criteria
multi_class: Determines the multi-class strategy if y contains more than two classes.
ovr: n_classes one vs rest classifiers
fit_intercept: calculate the intercept for the model
class_weight: Set the parameter C of class i to class_weight[i]*C for SVC
verbose: Enable verbose output
C: Regularization parameter
random_state: e seed of the pseudo random number generator to use when shuffling the data for the dual coordinate descent
"""
```

Linear model

```
clf = svm.SVC(C=10.0, kernel='linear', degree=3, gamma='scale',
              coef0=0.0, shrinking=True, probability=False, tol=0.00001,
              cache_size=200, class_weight=None, verbose=False, max_iter=-1,
              decision_function_shape='ovr', random_state=None)
```

Nonlinear model

```
clf = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale',
              coef0=0.0, shrinking=True, probability=False, tol=0.001,
              cache_size=200, class_weight=None, verbose=False, max_iter=-1,
              decision_function_shape='ovr', random_state=None)
```

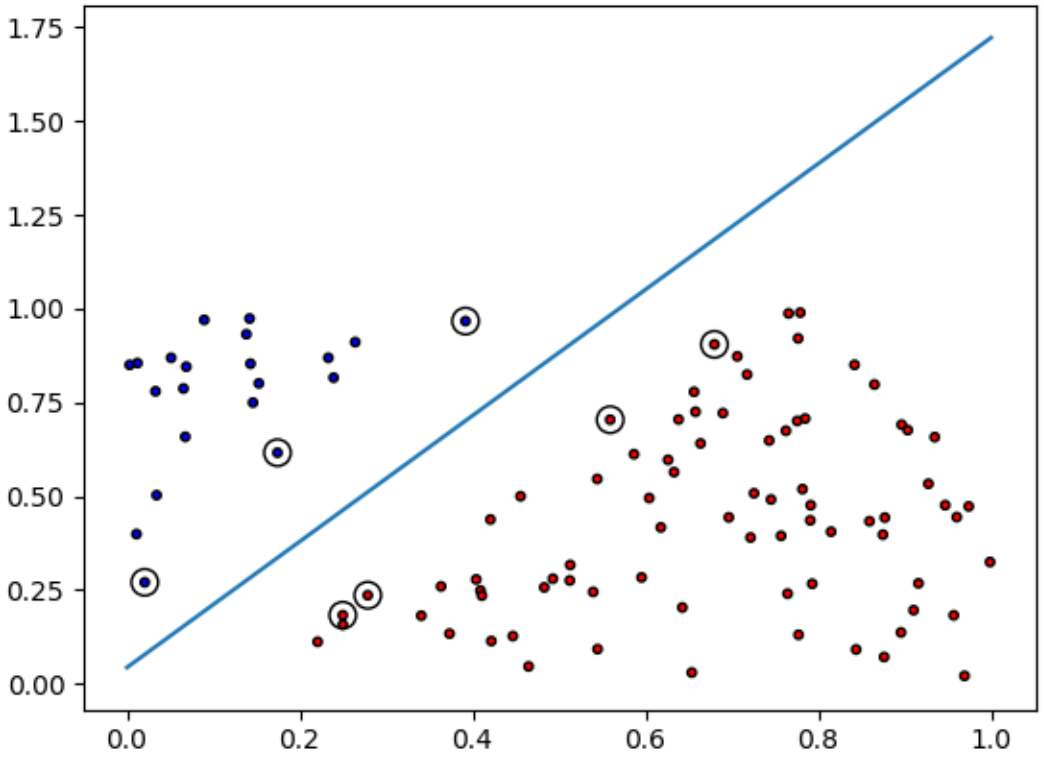
- Fit the model
- Calculate the accuracy rate on test data

✓ Linear SVM Results

- Run file svm_MT_SK.py

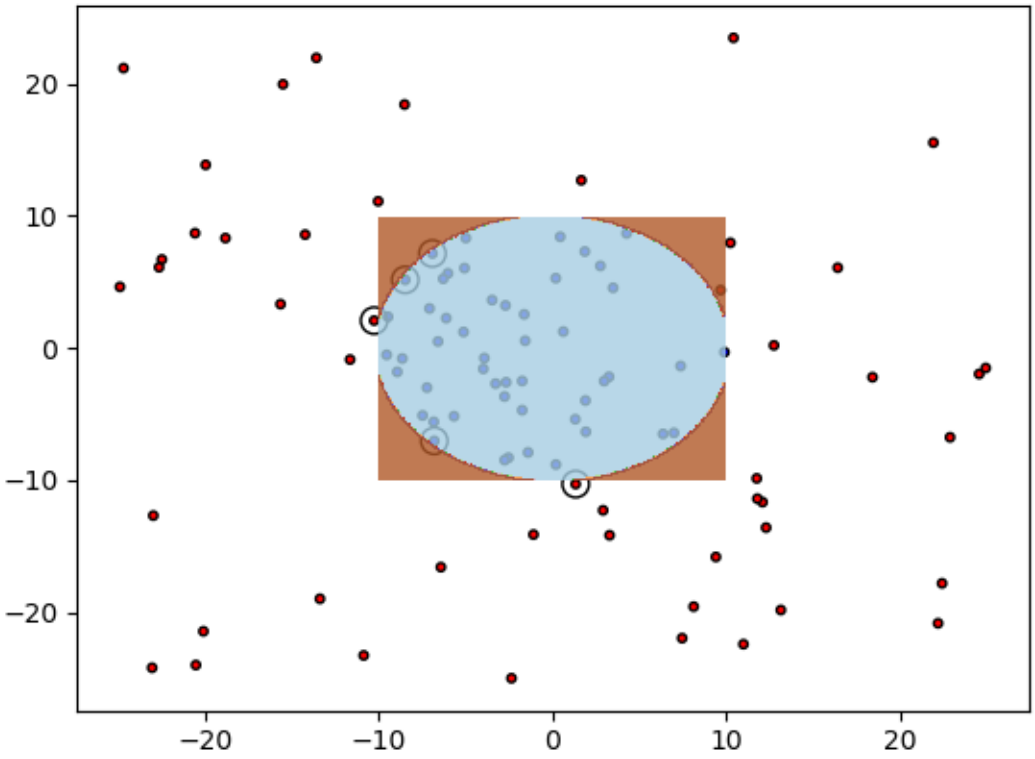
Support vectors for linear SVM

```
support vectors:  [[0.02066458 0.27003158]
 [0.17422964 0.6157447 ]
 [0.3917889  0.96675591]
 [0.24979414 0.18230306]
 [0.55919837 0.70372314]
 [0.6798148  0.90468041]
 [0.27872572 0.23552777]]
```



Support vectors for nonlinear SVM

```
support vectors:  [[ -8.47422847   5.15621613]
 [ -6.90647562   7.14833849]
 [ -6.80002274  -7.02384335]
 [  1.3393313  -10.29098822]
 [-10.260969     2.07391791]]
```



3. Applications

The aim of using SVM is to correctly classify unseen data. SVMs have a number of applications in several fields. Some common applications of SVM are:

- ✓ Face detection – SVM classify parts of the image as a face and non-face and create a square boundary around the face.
- ✓ Text and hypertext categorization.
- ✓ Classification of images – Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
- ✓ Bioinformatics – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.
- ✓ Protein fold and remote homology detection – Apply SVM algorithms for protein remote homology detection.

- ✓ Handwriting recognition – We use SVMs to recognize handwritten characters used widely.
- ✓ Generalized predictive control(GPC) – Use SVM based GPC to control chaotic dynamics with useful parameters.