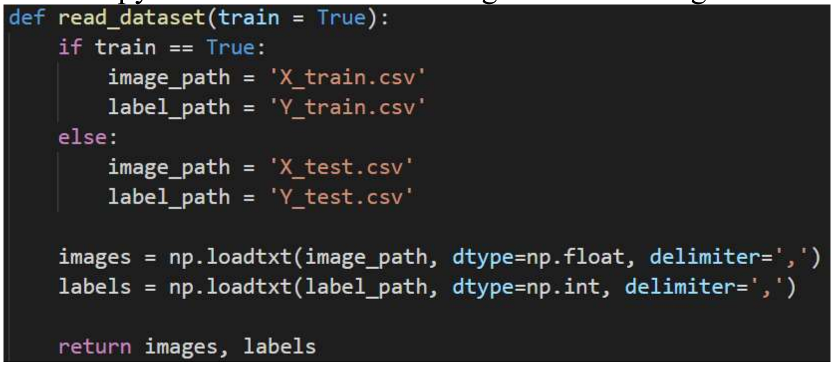
ML Homework 5 – 2 Report

I. SVM

1. Code with detailed explanations

- Read dataset

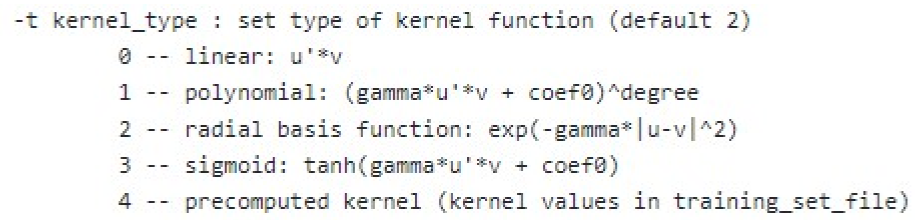
  Use numpy.loadtxt to read the training data and testing data.



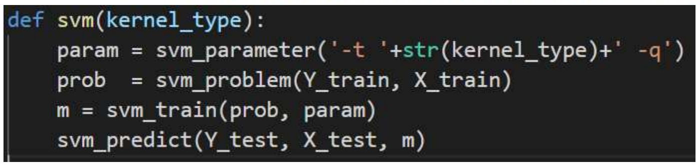
```
def read_dataset(train = True):
    if train == True:
        image_path = 'X_train.csv'
        label_path = 'Y_train.csv'
    else:
        image_path = 'X_test.csv'
        label_path = 'Y_test.csv'

    images = np.loadtxt(image_path, dtype=np.float, delimiter=',')
    labels = np.loadtxt(label_path, dtype=np.int, delimiter=',')

    return images, labels
```

- Part1

  Call the function in libsvm and use "-t" to select different kerenl function.



```
-t kernel_type : set type of kernel function (default 2)
        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)
        4 -- precomputed kernel (kernel values in training_set_file)
```

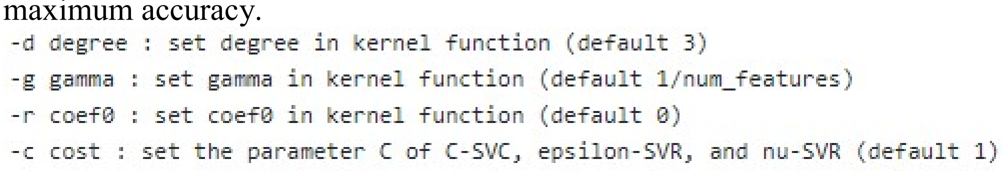


```
def svm(kernel_type):
    param = svm_parameter('-t '+str(kernel_type)+' -q')
    prob  = svm_problem(Y_train, X_train)
    m = svm_train(prob, param)
    svm_predict(Y_test, X_test, m)
```

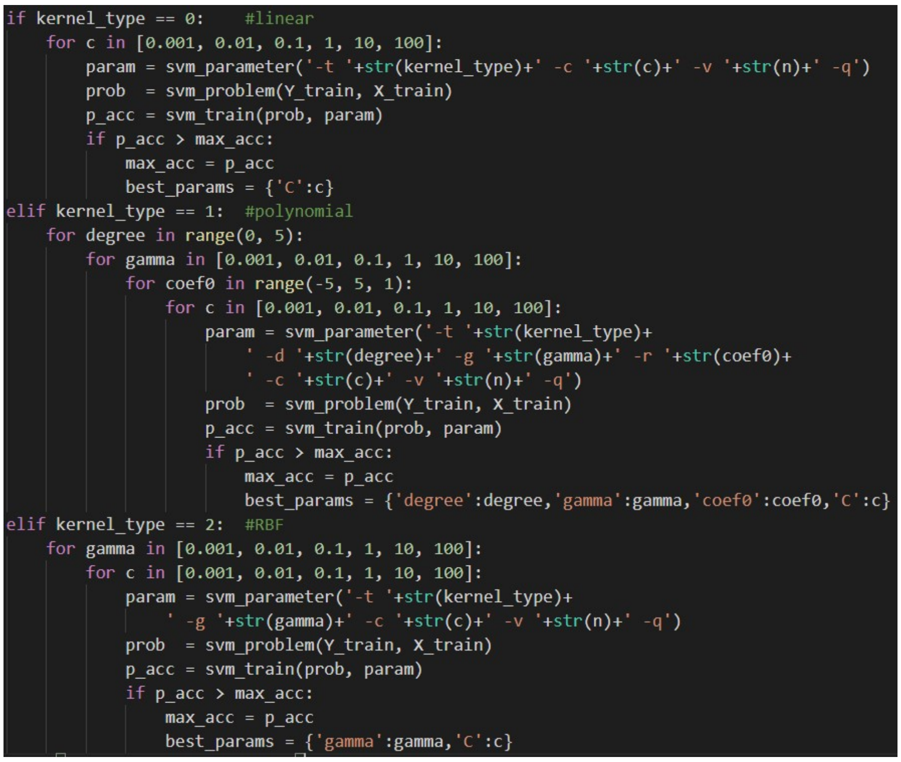
- Part2

  Hyper-parameters are not directly learnt within estimators. Therefore, it is recommended to search for the best cross-validation score in the hyper-parameter space.Grid search evaluates all possible combinations of parameter values through looping, and retains the best combination.

  For each kernel type, I use for loop to test all parameter combinations and do cross-validation. Then record the hyper-parameters with maximum accuracy.

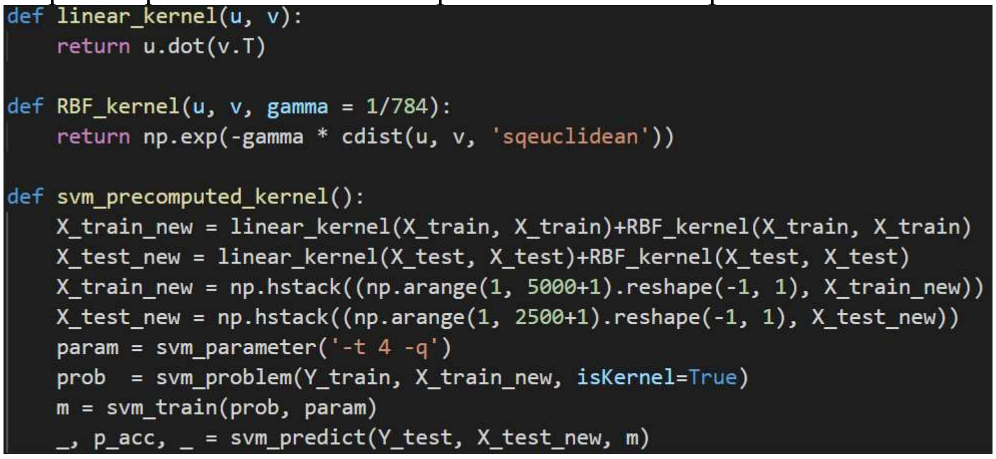


```
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
```



```python
if kernel_type == 0:    #linear
    for c in [0.001, 0.01, 0.1, 1, 10, 100]:
        param = svm_parameter('-t '+str(kernel_type)+' -c '+str(c)+' -v '+str(n)+' -q')
        prob  = svm_problem(Y_train, X_train)
        p_acc = svm_train(prob, param)
        if p_acc > max_acc:
            max_acc = p_acc
            best_params = {'C':c}
elif kernel_type == 1:  #polynomial
    for degree in range(0, 5):
        for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
            for coef0 in range(-5, 5, 1):
                for c in [0.001, 0.01, 0.1, 1, 10, 100]:
                    param = svm_parameter('-t '+str(kernel_type)+
                        ' -d '+str(degree)+' -g '+str(gamma)+' -r '+str(coef0)+
                        ' -c '+str(c)+' -v '+str(n)+' -q')
                    prob  = svm_problem(Y_train, X_train)
                    p_acc = svm_train(prob, param)
                    if p_acc > max_acc:
                        max_acc = p_acc
                        best_params = {'degree':degree,'gamma':gamma,'coef0':coef0,'C':c}
elif kernel_type == 2:  #RBF
    for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
        for c in [0.001, 0.01, 0.1, 1, 10, 100]:
            param = svm_parameter('-t '+str(kernel_type)+
               ' -g '+str(gamma)+' -c '+str(c)+' -v '+str(n)+' -q')
            prob  = svm_problem(Y_train, X_train)
            p_acc = svm_train(prob, param)
            if p_acc > max_acc:
                max_acc = p_acc
                best_params = {'gamma':gamma,'C':c}
```

- Part3
  First convert the image into kernel space by using linear_kernel and RBF_kernel. Then insert serial number to the first column.
  Use "-t 4" to set the type of kernel function and set "isKernel=True" for precomputed kernel. Other steps are the same as in part1.



```python
def linear_kernel(u, v):
    return u.dot(v.T)

def RBF_kernel(u, v, gamma = 1/784):
    return np.exp(-gamma * cdist(u, v, 'sqeuclidean'))

def svm_precomputed_kernel():
    X_train_new = linear_kernel(X_train, X_train)+RBF_kernel(X_train, X_train)
    X_test_new = linear_kernel(X_test, X_test)+RBF_kernel(X_test, X_test)
    X_train_new = np.hstack((np.arange(1, 5000+1).reshape(-1, 1), X_train_new))
    X_test_new = np.hstack((np.arange(1, 2500+1).reshape(-1, 1), X_test_new))
    param = svm_parameter('-t 4 -q')
    prob  = svm_problem(Y_train, X_train_new, isKernel=True)
    m = svm_train(prob, param)
    _, p_acc, _ = svm_predict(Y_test, X_test_new, m)
```

2. Experiments Settings and Results

- Part1:

Type of Kernel	Accuracy (%)	Time (s)
linear	95.08	3.55
polynomial	34.68	30.85
RBF	95.32	7.26

- Part2:

Type of Kernel	CV-Accuracy (%)	Testing Accuracy (%)	Parameters	Time (s)
linear	96.64	95.96	'C': 0.01	38.92

polynomial	98.22	97.64	'degree': 2, 'gamma': 1, 'coef0': -2, 'C': 100	33352.49
RBF	98.16	98.2	'gamma': 0.01, 'C': 10	1281.09

- Part3:

Type of Kernel	Accuracy (%)	Time (s)
Linear+RBF	21.4	36.52

3. Observations and Discussion

- Part1:

  1. RBF has the best accuracy than all the others on testing data. This is because RBF can map data into an infinite dimensional space so that everything can be linearly separated. Therefore, RBF allows svm to perform good classification.

  2. Linear has the best performance because it is the simplest kernel. And its accuracy is also relatively high Polynomial takes the most time but has the lowest accuracy.

  3. If time is more important, we should choose linear. If accuracy is more important, we should choose RBF.

- Part2:

  1. The higher the value of c, the smaller the tolerance of the model to errors. If it is too high, it will be overfitting for the training data.

  2. If we find the right parameters, we can also get better results in the polynomial kernel, but it takes a lot of time because of its many parameters.

  3. RBF still has the best accuracy than all the others on testing data. Part2 is more accurate than part1 but takes more time. We need to strike a balance between time and accuracy.

- Part3:

  1. Compared with Part 1, my linear + RBF kernel is less accurate and requires more time.

  2. My code will have OSError run on Windows and must be run on linux.