Report

Code with Detailed Explanation

- Part 1
  - Linear kernel

    $$K(x, x') = x^T x' + c$$, where c is a coefficient.

  - Polynomial kernel w/ degree-d

    $$K(x, x') = (x^T x' + c)^d$$, where c is a coefficient.

    But notice that in libsvm, there is a parameter γ in the polynomial function, so it will be like this $K(x, x') = (\gamma x^T x' + c)^d$

  - RBF kernel

    $$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$, where $\|x - x'\|^2$ is squared Euclidean distance b/w two feature vectors. σ is a free parameter. We can also rewrite it as following.

    Let $\gamma = \frac{1}{2\sigma^2} \Rightarrow K(x, x') = \exp(-\gamma\|x - x'\|^2)$

  - For soft-margin SVM (a.k.a C-SVC),

    by solving $L(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)$ subject to $0 \leq a_n \leq C$ and $\sum_{n=1}^{N} a_n t_n = 0$

    (C is the regularization parameter)
    using SMO (Sequential Minimal Optimization) for QP (Quadratic Programming) to solve coefficients a, we can have prediction

    $$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b$$, where $0 \leq a_n \leq C$

    if $a_n = 0$, x_n will not used in prediction
    if $a_n < C$, x_n is on the margin
    if $a_n \leq C$, x_n is inside the margin

    In addition, libsvm use <u>one-against-one</u> instead of one-against-rest to deal with multi-class classification.

```
'''
-s 0: C-SVC
-t 0: Linear kernel w/ default C=1
-t 1: Polynomial kernel w/ default C=1, degree=3 and default gamma=1/k
-t 2: RBF kernel w/ default C=1 and gamma=1/k
'''
kernels = {'Linear': 0, 'Polynomial': 1, 'RBF': 2}
options = np.array(['-q -s 0 -t 0', '-q -s 0 -t 1', '-q -s 0 -t 2'])

for kernel_type, idx in kernels.items():
    model =  svmutil.svm_train(Y_train, X_train, options[idx])
    p_label, p_acc, p_val = svmutil.svm_predict(Y_test, X_test, model,'-q')
    print('{}: Accuracy = {:.2f}%'.format(kernel_type, p_acc[0]))
```

- Part 2

  Use C-SVC w/ grid search.

  Grid search: Set proper parameters range and make combinations for them. Then, Iterate through every combination, and find the optimal parameters with highest 3-fold cross validation accuracy for each kernel.

```
def grid_search(kernel, opt, Y_train, X_train):
    '''
    Parameters
    ----------
    C:      [1e-3, 1e-2, 1e-1, 1e+0, 1e+1, 1e+2, 1e+3]
    gamma:  [1e-3, 1e-2, 1e-1, 1e+0, 1e+1, 1e+2, 1e+3]
    degree: [2, 3, 4]

    Options
    -------
    -v 3: set cross validation to 3-fold
    -c:   set cost
    -g:   set gamma
    -d:   set degree
    '''
    param = {'C': np.logspace(-3, 3, 7),
             'gamma': np.logspace(-3, 3, 7),
             'degree': range(2, 5)}
    opt += ' -v 3'

    if kernel == 'Linear':
        df = pd.DataFrame(np.zeros((7,2)), columns=['C', 'ACC'])
        for i, C in enumerate(param['C']):
            acc = svmutil.svm_train(Y_train, X_train, '{} -c {}'.format(opt, C))
            df.iloc[i] = [C, acc]
        return df

    elif kernel == 'Polynomial':
        i = 0
        df = pd.DataFrame(np.zeros((7*7*3,4)), columns=['C', 'gamma', 'degree', 'ACC'])
        for C in param['C']:
            for gamma in param['gamma']:
                for degree in param['degree']:
                    acc = svmutil.svm_train(Y_train, X_train, '{} -c {} -g {} -d {}'.format(opt, C, gamma, degree))
                    df.iloc[i] = [C, gamma, degree, acc]
                    i += 1
        return df

    elif kernel == 'RBF':
        i = 0
        df = pd.DataFrame(np.zeros((7*7,3)), columns=['C', 'gamma', 'ACC'])
        for C in param['C']:
            for gamma in param['gamma']:
                acc = svmutil.svm_train(Y_train, X_train, '{} -c {} -g {}'.format(opt, C, gamma))
                df.iloc[i] = [C, gamma, acc]
                i += 1
        return df
```

```
dfs = []
for kernel_type, idx in kernels.items():
    df = grid_search(kernel_type, options[idx], Y_train, X_train)
    best_param, best_acc = tuple(df.loc[df['ACC'].argmax()][:-1]), df.loc[df['ACC'].argmax()][-1]
    print('{}: {}, Cross Validation Accuracy = {:.2f}%'.format(kernel_type, best_param, best_acc))
    dfs.append(df)
```

Plot heatmaps

```
# plot heatmaps
fig, axes = plt.subplots(4, 1, figsize=(9, 32))
for i in range(3):
    df = dfs[1].sort_values(by=['degree']).iloc[i*49:(i+1)*49]
    plot(axes[i], df, "Polynomial degree {}".format(i+2))
plot(axes[3], dfs[2], "RBF")
plt.show()
```

- Part 3

  Construct a customized kernel that adds linear kernel and RBF kernel together as below. Compute the kernel for training and testing, pass them into the libsvm training and testing functions. (-t 4: precomputed kernel)

$$K(x, x') = x^T x' + \exp(-\gamma \|x - x'\|^2)$$

```
def customized_kernel(X, X_, gamma):
    '''
    Combine linear and RBF kernel
    '''
    kernel_linear = X @ X_.T
    dist          = np.sum(X**2, 1).reshape(-1, 1) + np.sum(X_**2, 1) - 2 * X @ X_.T
    kernel_rbf    = np.exp(-gamma*dist)
    kernel = np.hstack((np.arange(1,len(X)+1).reshape(-1,1), kernel_linear+kernel_rbf))
    return kernel
K_train = customized_kernel(X_train, X_train, 1e-2)
K_test = customized_kernel(X_test, X_train, 1e-2)
model = svmutil.svm_train(Y_train, K_train, '-q -t 4')
p_label, p_acc, p_val = svmutil.svm_predict(Y_test, K_test, model,'-q')
print('Linear + RBF Kernel:  Accuracy = {:.2f}%'.format(p_acc[0]))
```

Experiments Settings and Results/Observations and Discussion

- Part 1

  k = number of attributes in the input data

  We can find that for the default setting in libsvm, the performance of RBF kernel is slightly better than the performance of linear kernel, and the performance of linear kernel is much better than the performance of polynomial kernel.

  I speculated that the reason why polynomial kernel performs quite bad is due to the degree. That is, the performance of polynomial kernel is vulnerable to the degree. If we set the degree to 2, we can obtain the corresponding accuracy 88.24% which is much higher than the default setting.

	Linear	Polynomial	RBF
Accuracy	95.08%	34.68%	95.32%
Options	`‘-q -s 0 -t 0’` `(default -c 1)`	`‘-q -s 0 -t 1’` `(default -c 1 -d 3 -g 1/k)`	`‘-q -s 0 -t 2’` `(default -c 1 -g 1/k)`
	search=C-SVC, kernel=linear cost=1	search=C-SVC, kernel=polynomial cost=1 degree=3 gamma=1/k	search=C-SVC, kernel=RBF cost=1 gamma=1/k

- Part 2

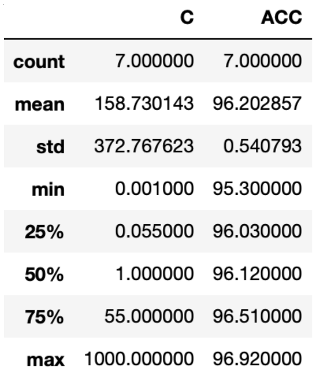
  By grid-search across C = [1e−3, 1e−2, 1e−1, 1e+0, 1e+1, 1e+2, 1e+3]

  γ = [1e−3, 1e−2, 1e−1, 1e+0, 1e+1, 1e+2, 1e+3]

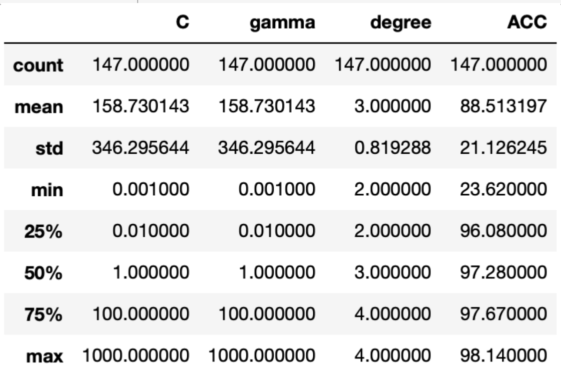
  degree = [2, 3, 4]

  The corresponding optimal parameters and 3-fold cross validation accuracies are as follows. From observation of CV accuracies, the performances of these three kernels are approximately the same. However, linear kernel has higher mean and lower standard deviation (mean: 96.2 > 88.51、56.75/std: 0.54 < 21.13、31.30), which may mean that linear kernel is more stable if we try to tune the parameters. On the other hand, although polynomial and RBF kernel is not that stable when tuning the parameters, their optimal CV accuracies outperform linear kernel. (98.2、98.14 >96.92)

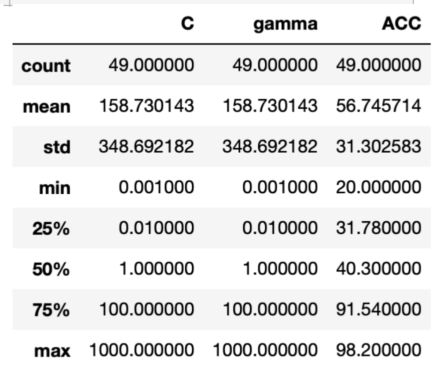
	Linear	Polynomial	RBF
CV Accuracy	96.92%	98.14%	98.2%
Options	`‘-q -s 0 -t 0` `-c 0.01’`	`‘-q -s 0 -t 1` `-c 0.001` `-g 1` `-d 2’`	`‘-q -s 0 -t 2` `-c 100` `-g 0.01’`
Params	(C=0.01)	(C=0.001, γ=1, degree=2)	(C=100, γ=0.01)



	C	ACC
count	7.000000	7.000000
mean	158.730143	96.202857
std	372.767623	0.540793
min	0.001000	95.300000
25%	0.055000	96.030000
50%	1.000000	96.120000
75%	55.000000	96.510000
max	1000.000000	96.920000



	C	gamma	degree	ACC
count	147.000000	147.000000	147.000000	147.000000
mean	158.730143	158.730143	3.000000	88.513197
std	346.295644	346.295644	0.819288	21.126245
min	0.001000	0.001000	2.000000	23.620000
25%	0.010000	0.010000	2.000000	96.080000
50%	1.000000	1.000000	3.000000	97.280000
75%	100.000000	100.000000	4.000000	97.670000
max	1000.000000	1000.000000	4.000000	98.140000



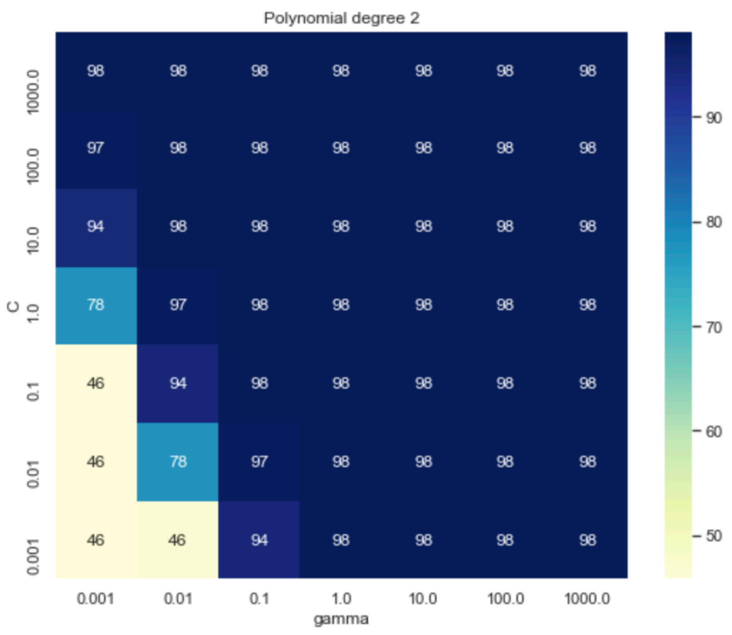
	C	gamma	ACC
count	49.000000	49.000000	49.000000
mean	158.730143	158.730143	56.745714
std	348.692182	348.692182	31.302583
min	0.001000	0.001000	20.000000
25%	0.010000	0.010000	31.780000
50%	1.000000	1.000000	40.300000
75%	100.000000	100.000000	91.540000
max	1000.000000	1000.000000	98.200000

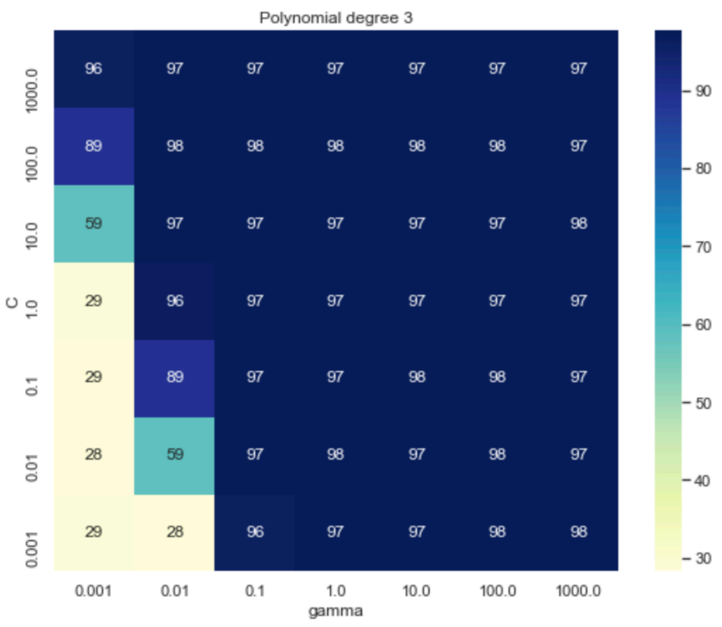
Visualizations

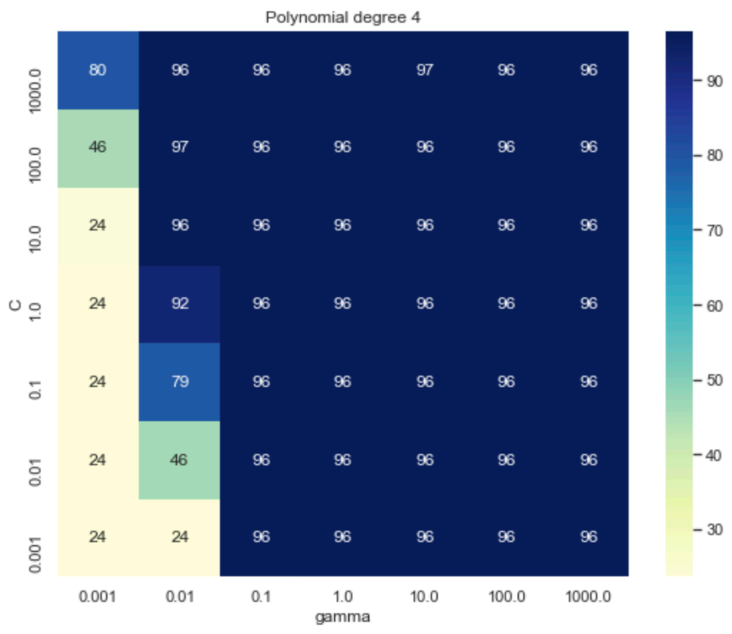
1. Linear: no matter what C value is, the cross validation accuracies are approximately the same.

C	ACC
0.001	95.30
0.010	96.92
0.100	96.80
1.000	96.06
10.000	96.12
100.000	96.00
1000.000	96.22

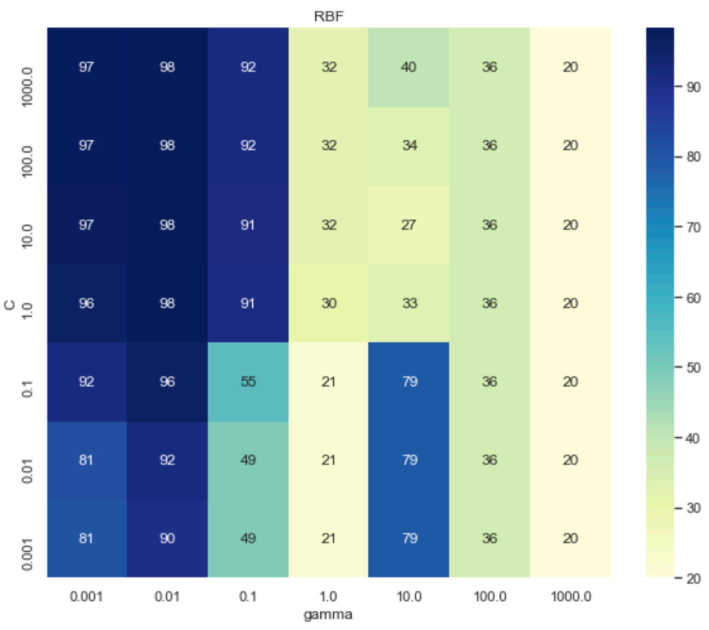
2. For polynomial kernel, plot heatmap for each degree that x-axis is gamma and y-axis is C. From rough observation, degree 2 outperforms others. Besides, the accuracies are high except those have lower C and gamma.







3. For RBF kernel, the CV accuracy seems higher when gamma is lower, and we can observe that the performance of RBF kernel is more susceptible to gamma than C. However, when gamma is 10 and C is in [0.001, 0.1], the accuracies are higher than their neighbors in grid search.



- Part 3

  Set γ=0.01 as the results in part 2, and we can obtain

  Linear + RBF Kernel: Accuracy = 95.32%

  If we set the cost to 100 as the best param in part 2, we still obtain accuracy=95.32%.

  ```
  model = svmutil.svm_train(Y_train, K_train, '-q -t 4 -c 100')
  ```

  For these two results, the accuracies of Linear+RBF are the same as the accuracy of default setting for RBF.

  Even though I only use the accuracy to compare their performance, we can also consider the running time.