# Xi Liu

methodology:
libraries:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

details: use read_csv to store data into a pandas data frame df. last column in spambase.data is the label (y) for training the model, read last column from df. before train the model, remove label from data, remove last column from df. transfer df to array. split dataset into train and test dataset paired with its label. set up kernel and hyperparameter (C, gamma) for the model. train the model. inputs are train dataset and train label. inputs are test dataset and test label (see code).

experimental results:
table can be seen at following page

intuition for results observed for different kernels and C:
linear: $< x, x' >$
polynomial: $(\gamma < x, x' > +r)^d$, $d$ is specified by degree
radial basis function: $\exp(-\gamma||x - x'||^2)$
soft margin SVM objective function:

$$\frac{1}{2}\sum_{k=1}^{n} w_k^2 + C\sum_{i=1}^{l}\xi_i \rightarrow \min_{w,b,\xi}$$

$$\text{s.t.}$$

$$y_i\left(\sum_{k=1}^{n} w_k x_{ik} + b\right) \geq 1 - \xi_i, i = 1, 2, ..., l$$

$$\xi_i \geq 0, i = 1, 2, ..., l$$

parameter C trades off between goals of maximizing margin and minimizing errors. when C is small, sum of error terms become small in objective function, so optimization goal is maximize margin, makes the decision surface smooth, as a result, margin can become so large that includes all points. when C is large, sum of error terms become large in objective function, so optimization goal is minimize sum of error terms, it aims at classifying all training examples correctly. as a result, margin can become so small that include no points.
large $C$ has lower bias and high variance, small $C$ has higher bias and low variance

for this svm:
$O(\text{linear}) = O(\text{quadratic}) = O(\text{radial basis function})$, for kernels that have higher asymptotic complexity, more regularization is needed to avoid overfit (good performance on training data, poor generalization to test data), since $C$ is inversely proportional to regularization strength, for high test accuracy, radial basis function kernel's $C <$ quadratic function kernel's $C <$ linear function kernel's $C$
so, radial basis function has high test accuracy when $C$ is $1 = 10^{-0}$, quadratic kernel has

high test accuracy when $C$ is $10^1$, linear kernel has high test accuracy when $C$ is $10^4$

| Kernel | C | 10 ^ -2 | 10 ^ -1 | 10 ^ -0 | 10 ^ 1 | 10 ^ 2 | 10 ^ 3 | 10 ^ 4 |
|---|---|---|---|---|---|---|---|---|
| Linear | Train Accuracy | 0.9198757764 | 0.9257763975 | 0.9310559006 | 0.9319875776 | 0.9335403727 | 0.9335403727 | 0.9335403727 |
| | Test Accuracy | 0.9217391304 | 0.9326086957 | 0.9326086957 | 0.9326086957 | 0.9326086957 | 0.934057971 | **0.9347826087** |
| Quadratic | Train Accuracy | 0.6372670807 | 0.7329192547 | 0.852484472 | 0.950931677 | 0.9720496894 | 0.9878881988 | 0.9919254658 |
| | Test Accuracy | 0.6195652174 | 0.7224637681 | 0.8405797101 | **0.9101449275** | 0.9 | 0.8905797101 | 0.8739130435 |
| RBF | Train Accuracy | 0.6928571429 | 0.9074534161 | 0.947826087 | 0.9661490683 | 0.9869565217 | 0.9937888199 | 0.9953416149 |
| | Test Accuracy | 0.6746376812 | 0.915942029 | **0.9420289855** | 0.9362318841 | 0.9260869565 | 0.9195652174 | 0.9123188406 |

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('spambase/spambase.data')
df
```

```
         0  0.64  0.64.1  0.1  0.32   0.2   0.3   0.4   0.5   0.6  ...
0.41  \
0     0.21  0.28    0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...
0.000
1     0.06  0.00    0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...
0.010
2     0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...
0.000
3     0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...
0.000
4     0.00  0.00    0.00  0.0  1.85  0.00  0.00  1.85  0.00  0.00  ...
0.000
...    ...   ...     ...  ...   ...   ...   ...   ...   ...   ...  ...
...
4595  0.31  0.00    0.62  0.0  0.00  0.31  0.00  0.00  0.00  0.00  ...
0.000
4596  0.00  0.00    0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...
0.000
4597  0.30  0.00    0.30  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...
0.102
4598  0.96  0.00    0.00  0.0  0.32  0.00  0.00  0.00  0.00  0.00  ...
0.000
4599  0.00  0.00    0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...
0.000

       0.42  0.43  0.778   0.44   0.45  3.756   61   278  1
0     0.132   0.0  0.372  0.180  0.048  5.114  101  1028  1
1     0.143   0.0  0.276  0.184  0.010  9.821  485  2259  1
2     0.137   0.0  0.137  0.000  0.000  3.537   40   191  1
3     0.135   0.0  0.135  0.000  0.000  3.537   40   191  1
4     0.223   0.0  0.000  0.000  0.000  3.000   15    54  1
...     ...   ...    ...    ...    ...    ...  ...   ... ..
4595  0.232   0.0  0.000  0.000  0.000  1.142    3    88  0
4596  0.000   0.0  0.353  0.000  0.000  1.555    4    14  0
4597  0.718   0.0  0.000  0.000  0.000  1.404    6   118  0
4598  0.057   0.0  0.000  0.000  0.000  1.147    5    78  0
4599  0.000   0.0  0.125  0.000  0.000  1.250    5    40  0

[4600 rows x 58 columns]
```

```
y = df.iloc[:, -1:] # last column in spambase.data is the label (y)
for training the model, read last column from df
y = np.array(y).ravel()
x = df.iloc[: , :-1] # before train the model, remove label from data,
```

```
remove last column from df
x = x.copy()
x = np.array(x) # transfer df to array

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.3, random_state = 0) # split dataset into train and test dataset
paired with its label
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3220, 57)
(1380, 57)
(3220,)
(1380,)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)


from sklearn.svm import SVC

def svm(C, kernel):
    classifier = SVC(C = C, kernel = kernel, degree = 2, gamma =
'auto') # set up kernel and hyper-parameter (C, gamma) for the model
    classifier.fit(x_train, y_train) # train the model
    train_score = classifier.score(x_train, y_train) # inputs are
train dataset and train label
    test_score = classifier.score(x_test, y_test) # inputs are test
dataset and test label
    return train_score, test_score

kernel_list = ['linear', 'rbf', 'poly']
C_list = [0.01, 0.1, 1, 10, 100, 1000, 10000]

for kernel in kernel_list:
    for C in C_list:
        train_score, test_score = svm(C, kernel)
        print('C:', C)
        print('kernel:', kernel)
        print('train accuracy: ', train_score)
        print('test accuracy: ', test_score, '\n')
```

```
C: 0.01
kernel: linear
train accuracy:  0.9198757763975155
```

```
test accuracy:  0.9217391304347826

C: 0.1
kernel: linear
train accuracy:  0.925776397515528
test accuracy:  0.9326086956521739

C: 1
kernel: linear
train accuracy:  0.931055900621118
test accuracy:  0.9326086956521739

C: 10
kernel: linear
train accuracy:  0.9319875776397516
test accuracy:  0.9326086956521739

C: 100
kernel: linear
train accuracy:  0.9335403726708075
test accuracy:  0.9326086956521739

C: 1000
kernel: linear
train accuracy:  0.9335403726708075
test accuracy:  0.9340579710144927

C: 10000
kernel: linear
train accuracy:  0.9335403726708075
test accuracy:  0.9347826086956522

C: 0.01
kernel: rbf
train accuracy:  0.6928571428571428
test accuracy:  0.6746376811594202

C: 0.1
kernel: rbf
train accuracy:  0.9074534161490683
test accuracy:  0.9159420289855073

C: 1
kernel: rbf
train accuracy:  0.9478260869565217
test accuracy:  0.9420289855072463

C: 10
kernel: rbf
```

```
train accuracy:  0.9661490683229814
test accuracy:  0.936231884057971

C: 100
kernel: rbf
train accuracy:  0.9869565217391304
test accuracy:  0.9260869565217391

C: 1000
kernel: rbf
train accuracy:  0.9937888198757764
test accuracy:  0.9195652173913044

C: 10000
kernel: rbf
train accuracy:  0.9953416149068323
test accuracy:  0.9123188405797101

C: 0.01
kernel: poly
train accuracy:  0.6372670807453417
test accuracy:  0.6195652173913043

C: 0.1
kernel: poly
train accuracy:  0.7329192546583851
test accuracy:  0.722463768115942

C: 1
kernel: poly
train accuracy:  0.8524844720496895
test accuracy:  0.8405797101449275

C: 10
kernel: poly
train accuracy:  0.9509316770186336
test accuracy:  0.9101449275362319

C: 100
kernel: poly
train accuracy:  0.9720496894409938
test accuracy:  0.9

C: 1000
kernel: poly
train accuracy:  0.987888198757764
test accuracy:  0.8905797101449275

C: 10000
```

```
kernel: poly
train accuracy:  0.9919254658385093
test accuracy:  0.8739130434782608
```