Support Vector Machine (with Python)

Tutorial 3  Yang
Through this tutorial, you will better know:

- What is Support Vector Machine
- The SVM in Scikit-learn – C-Support Vector Classification
- The method to train the SVM – SMO algorithm
- The parameters in SVC
- How to use the Scikit-learn.SVM
- Other SVMs in Scikit-learn
Linear model

Support vector machine:

- Margin: the smallest distance between the decision boundary and any of the samples
- maximizing the margin $\Rightarrow$ a particular decision boundary
- Location of boundary is determined by support vectors

- Canonical representation:
  \[
  \arg\min \frac{1}{2} \|w\|^2,
  \]
  \[
  \text{s.t. } t_n (w \cdot x_i + b) \geq 1, \ n = 1, 2, \ldots, N
  \]
- By Lagrangian, its dual form (QP problem)
  \[
  \min_{\tilde{a}} \psi(\tilde{a}) = \min_{\tilde{a}} \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} t_n t_m (x_n \cdot x_m) a_n a_m - \sum_{n=1}^{N} a_n,
  \]
  \[
  \text{s.t. } a_n \geq 0, \ n = 1, 2, \ldots, N,
  \]
  \[
  \sum_{n=1}^{N} a_n t_n = 0.
  \]
Nonlinear model

Soft margin:

- Slack variables $\xi_n \geq 0, n = 1, \ldots, N$
- Maximize the margin while softly penalizing incorrect points

$$\arg\min \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \xi_n,$$

subject to $t_n (w \ast x_i + b) \geq 1 - \xi_n, \ n = 1, \ldots, N.$

- The corresponding dual form by Lagrangian:

$$\min \limits_{\alpha} \psi(\alpha) = \min \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} t_n t_m k(x_n, x_m) a_n a_m - \sum_{n=1}^{N} a_n,$$

subject to $0 \leq a_n \leq C, \ n = 1, 2, \ldots, N,$

$$\sum_{n=1}^{N} a_n t_n = 0.$$

$C$ controls Trade-off between the slack variable penalty and the margin
Kernel Method

- The kernel trick (kernel substitution)
  - map the inputs into high-dimensional feature spaces properly
  - solve the problems of high complexity and computation caused by inner product

- Example: kernel function -- $k(X_i, X_j) = \langle \phi(X_i) \cdot \phi(X_j) \rangle$

Defined two vectors: $x = (x_1, x_2, x_3); y = (y_1, y_2, y_3)$

Defined the equations: $f(x) = (x_1 x_1, x_1 x_2, x_1 x_3, x_2 x_1, x_2 x_2, x_2 x_3, x_3 x_1, x_3 x_2, x_3 x_3)$,

$$K(x, y) = (\langle x, y \rangle)^2,$$

Assume $x = (1, 2, 3), y = (4, 5, 6)$

$f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9), f(y) = (16, 20, 24, 20, 25, 36, 24, 30, 36)$,

$\langle f(x), f(y) \rangle = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 = 1024$,

$K(x, y) = (4 + 10 + 18)^2 = 1024$.  **Kernel is much simpler**
C-Support Vector Classification:

- The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.
- The multiclass support is handled according to a one-vs-one scheme

LibSVM:

- LIBSVM implements the SMO algorithm for kernelized support vector machines (SVMs), supporting classification and regression.[1]

---

SMO Algorithm

- Sequential Minimal Optimization[2]:
  - A Fast Algorithm for Training Support Vector Machines
  - Quickly solve the SVM quadratic programming (QP) problem
  - The main steps:
    Repeat till convergence {
      1. Select some pair $a_i$ and $a_j$ to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
      2. Reoptimize $\Psi(\tilde{a})$ with respect to $a_i$ and $a_j$, while holding all the other $a_k$’s ($k \neq i, j$) fixed.
    }

Parameters of SVC

**Parameters**

- **C**: Penalty parameter $C$ of the error term, controls trade-off between the penalty and the margin, default=1.0
- **degree**: Degree of the polynomial kernel function
- **gamma**: Kernel coefficient(‘rbf’, ‘poly’ and ‘sigmoid’), gamma=auto means 1/n_features
- **coef0**: Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.
- **probability**: whether to enable probability estimates (true or false)
- **shrinking**: Whether to use the shrinking heuristic
- **tol**: Tolerance for stopping criterion
- **Cache_size**: Specify the size of the kernel cache
- **class_weight**: set different penalty for different data classes by the class weight values
- **verbose**: Enable verbose output, if enabled, may not work properly in a multithreaded context
- **max_iter**: Hard limit on iterations within solver, or -1 for no limit
- **decision_function_shape**: for multiple classifications, ovo for one-vs-one, ovr for one-vs-rest
- **random_state**: The seed of the pseudo random number generator to use when shuffling the data

[source]

Kernel selection

- Linear kernel: Choose based on the accuracy
  - Linear: \( u' \times v \)
    Mainly for linear classification, it has fewer parameters, computing fast

- Nonlinear kernel
  - Polynomial: \( (\gamma \times u' \times v + coef0)^{degree} \)
  - rbf: \( \exp(-\gamma \times |u - v|^2) \)
  - Sigmoid: \( \tanh(\gamma \times u' \times v + coef0) \)

\( C \): the penalty coefficient, low \( C \) makes the decision surface smooth, high \( C \) aims at classifying all training examples correctly

\( \gamma (\text{gamma}) \): defines how far the influence of a single training example reaches, low values mean ‘far’ and high values mean ‘close’.
Example-SVC

from sklearn.svm import SVC
import numpy as np
X=np.array([[[-1,-1],[-2,-1],[1,1],[2,1]])
y=np.array([1,1,2,2])
clf=SVC(kernel='linear')
clf.fit(X,y)
print(clf.fit(X,y))
print(clf.predict([[0,0]]))

# Only for linear kernel
w=clf.coef_[0]
xx=np.linspace(-2,2)
yy=-(w[0]*xx+clf.intercept_[0])/w[1]
plt.plot(xx, yy, 'k-', label='$hyperplane$')
plt.legend()
plt.savefig(path+"SVM.png")
plt.show()

import matplotlib.pyplot as plt

y1=y.copy()
a=np.hstack((X,y1.reshape(4,1)))
for i in range(len(a)):
    if a[i,2]==1:
        plt.plot(a[i,0],a[i,1], 'b*')
    else:
        plt.plot(a[i,0],a[i,1], 'r*')
plt.plot(-0.8, -1,'bo')
Exercise 1 - Linear model - Tasks

- First load the training data and testing data of a linear example
- Create a SVM by SVC
- Train the SVM model by the data in training file
- Classify the data in test file
- Plot the figure of data points and the hyperplane
- Pls change the parameter C and observe
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn import metrics
import matplotlib.pyplot as plt
import os

# load data
path=os.getcwd()
traindata=pd.read_csv(path+'\traindata.csv')
train_x=traindata.iloc[:, :-1]
train_y=traindata.iloc[:, -1]
testdata=pd.read_csv(path+'\testdata.csv')
test_x=testdata.iloc[:, :-1]
test_y=testdata.iloc[:, -1]

# introduce the SVC
clf=SVC(C=10, kernel='linear')
clf.fit(train_x, train_y)
Test_y=pd.Series(clf.predict(test_x), name='Y')
print('Classification report for classifier %s:
%s
' % (clf, metrics.classification_report(test_y, Test_y)))
print("Confusion matrix:%s" % metrics.confusion_matrix(test_y, Test_y))
Exercise 1-Linear model(2)

```python
# plot the training data
label=train_y.copy()
label[label<0]=0
label=label.astype(int)
label=label.values
colormap=np.array(['r','b'])
plt.scatter(train_x.iloc[:,0], train_x.iloc[:,1], zorder=3, marker='o', c=colormap[label], label='traindata')

# plot the hyperplane
w=clf.coef_[0]
xx=np.linspace(-2, 2)
yy=-(w[0]*xx+clf.intercept_[0])/w[1]
plt.axis([-2, 2, -2, 2])
plt.plot(xx, yy, 'k-', label='$hyperplane$')

# calculate the bias of margins
margin=1/np.sqrt(np.sum(clf.coef_**2))
yy_down=yy-np.sqrt(1+(w[0]/w[1])**2)*margin
yy_up=yy+np.sqrt(1+(w[0]/w[1])**2)*margin

# plot margins
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
```

# plot the support vectors
plt.scatter(clf.support_vectors_[:,0], clf.support_vectors_[:,1], zorder=2, facecolors = 'none', s=80, edgecolors='k', label='Support Vectors')
Exercise 1-Linear model(3)

#plot the test data set
labelt=test_y.copy()
labelt[labelt<0]=0
labelt=labelt.astype(int)
labelt=labelt.values
plt.scatter(test_x.iloc[:,0], test_x.iloc[:,1], zorder=3, marker='+', c=colormap[labelt], label='testdata')

plt.legend(loc=[0.26,0.01])
plt.savefig(path+'svc-linear.png')
plt.show()
Exercise 2-nonlinear model-Tasks

- First load the training data and testing data of a nonlinear example
- Create a SVM by SVC with three kernels
- Train the SVM model by the data in training file
- Classify the data in test file
- Plot the figure of data points and the hyperplane
- Please change the parameter
  - Change parameter C and observe
  - Change parameter $\gamma$ and observe
Exercise 2 - nonlinear model(1)

import numpy as np
import pandas as pd
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import os

# load data
path=os.getcwd()
traindata=pd.read_csv(path+'\traindata.csv')
train_x=traindata.iloc[:, :-1]
train_y=traindata.iloc[:, -1]
testdata=pd.read_csv(path+'\testdata.csv')
test_x=testdata.iloc[:, :-1]
test_y=testdata.iloc[:, -1]

# introduce the SVC and fit the model
for fig_n, kernel in enumerate(('linear', 'rbf', 'poly')):
    clf=SVC(C=1.0, kernel=kernel, gamma=10)
    clf.fit(train_x, train_y)
    print('Classification report: %s
Accuracy rate:%s
' % (clf, clf.score(test_x, test_y)))

#plot new window for figure
plt.figure(fig_n)
#clear the current figure
plt.clf()
Exercise 2-nonlinear model(2)

#plot the train data
plt.scatter(train_x.iloc[:,0], train_x.iloc[:,1], c=train_y.iloc[:], cmap=plt.cm.Paired,
            edgecolor='k', zorder=10, s=20)

#plot the support vectors
plt.scatter(clf.support_vectors_[:,0], clf.support_vectors_[:,1], s=80,
            facecolors='none', zorder=10, edgecolors='k')
plt.axis('tight')
x_min, x_max = train_x.iloc[:,0].min()-1, train_x.iloc[:,0].max()+1
y_min, y_max= train_x.iloc[:,1].min()-1, train_x.iloc[:,1].max()+1

# create a mesh to plot in
XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])
Exercise 2 - nonlinear model (3)

# Put the result into a color plot
Z = Z.reshape(XX.shape)
plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'], levels=[-.5, 0, .5])

# plot the test data set
plt.scatter(test_x.iloc[:, 0], test_x.iloc[:, 1], c=test_y.iloc[:], cmap=plt.cm.Paired, edgecolor='b', zorder=10, s=20)
plt.title(kernel)
plt.show()
Exercise 3-Multiclass classification-Tasks

- Pls import the digital dataset, divide the data into 5 parts
- Use 4 parts for training and others for prediction
- Tuning the parameters via cross validation
- Split the data to train and test subset
- Pls plot the first 4 images of training set
- Train the model of SVC by training data
- Print the classifier report and the score
- Plot the other 4 sub-figures in the end of prediction set
Exercise 3-Multiclass classification (1)

```
# Standard scientific Python imports
import matplotlib.pyplot as plt
import numpy as np
# Import datasets, classifiers and cross validation
from sklearn import datasets, svm
from sklearn.model_selection import cross_val_score

# The digits dataset
digits = datasets.load_digits()
print(digits.keys())
data=digits.data
target=digits.target
image=digits.images
print(data.shape)
```
The multiclass support is handled according to a one-vs-one scheme
# define the SVC and set its parameter
clf=svm.SVC(kernel='rbf')
gamma=np.logspace(-9,1,10)

# Calculate the Cross Validation scores for clf model to different gamma
s_mean=[]
s_std=[]
for x in gamma:
    clf.gamma=x
    scores = cross_val_score(clf, data, target, cv=5)
    s_mean.append(scores.mean())
    s_std.append(scores.std())

print (s_mean)
print (s_std)
Exercise 3-Multiclass classification(3)

# plot the figure to find the best setting for gamma
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.semilogx(gamma, s_mean)
plt.semilogx(gamma, np.array(s_mean) + np.array(s_std), 'b--')
plt.semilogx(gamma, np.array(s_mean) - np.array(s_std), 'b--')
locs, labels = plt.yticks()
plt.yticks(locs, list(map(lambda x: "%g" % x, locs)))
plt.ylabel('CV score')
plt.xlabel('Parameter Gamma')
plt ylim(0, 1.1)
plt.show()

#gamma=0.001 can get the best performance
Exercise 3-Multiclass classification(4)

# plot the first 4 images of training set
for index in range(4):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image[index], cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % target[index])

# split arrays into train and test subsets
from sklearn.model_selection import train_test_split as split
train_x, test_x, train_y, test_y = split(data, target, test_size=0.25, shuffle=False, random_state=0)

clf = svm.SVC(gamma=0.001)
clf.fit(train_x, train_y)
print("Classification report for classifier: %s
Accuracy: %s" % (clf, clf.score(test_x, test_y)))
Exercise 3-Multiclass classification (5)

for index in range(4):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(digits.images[index-4], cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % clf.predict(test_x)[index-4])

plt.show()

#subplot(numRows, numCols, plotNum)
Backup slices
Nu-Support Vector Classification:
- Similar to SVC but uses a parameter to control the number of support vectors
- The implementation is based on libsvm
- Parameter: nu—An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]

```python
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.svm import NuSVC
>>> clf = NuSVC()
>>> clf.fit(X, y)
>>> print(clf.predict([[-0.8, -1]]))
```
Nu-Support Vector Classification:

- Similar to SVC with parameter kernel='linear'
- implemented in terms of liblinear rather than libsvm
- it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples

Parameters:

- penalty: Specifies the norm used in the penalization;
- loss: Specifies the loss function;
- dual: Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.