Income Prediction via Support Vector Machine

丁勇铭 116039910001
顾家威 116039910070

1. Abstract

Support vector machine methods are employed to generate and evaluate income prediction data extracted by Barry Becker from the 1994 Census database. In this paper, RBF kernel is used to realize the SVM algorithm. Grid search for parameter C and gamma with cross validation is used to save time and reduce computational complexity. After the experiments, there is a conclusion that our SVM algorithm, using RBF kernel with refined parameters, can retain high accuracy rate up to 84.53%.

2. Introduction

Supervised learning methods based on statistical learning theory, for classification and regression, provide good generalization and classification accuracy on real data. However, their inherent trade-off is their computational expense. Recently, support vector machines (SVM) [1]-[4] have become a popular tool for learning methods since they translate the input data into a larger feature space where the instances are linear separable, thus increasing efficiency. In SVM methods a kernel which can be considered a similarity measure is used to recode the input data. The kernel is used accompanied by a map function $\emptyset$.

Even if the mathematics behind the SVM is straight forward, finding the best choices for the kernel function and parameters can be challenging, when applied to real data sets. Usually, the recommended kernel function for nonlinear problems is the Gaussian radial basis function, because it resembles the sigmoid kernel for certain parameters and it requires less parameters than a polynomial kernel. The kernel function parameter γ and the parameter C, which control the complexity of the decision function versus the training error minimization, can be determined by running a two-dimension grid search, which means that the values for pairs of parameters (C, γ) are generated in a predefined interval with a fixed step. The performance of each combination is computed and used to determine the best pair of parameters. However, due to memory limitations and the quadratical grow of the kernel with the number of training examples, it is not practical to grid search for SVM's parameters by using datasets with more than 10^3 data instances.

In this paper, adult data set is used to predict whether income exceeds $50K/yr based on census data. It is also known as "Census Income" dataset.

3. Dataset

One income dataset was previously extracted from the CPS data and posted on the University of California Irvine (UCI) repository. The dataset, posted on the UCI machine learning repository and named “adult dataset”, was extracted from the 1994 CPS data. Records with unreal values were eliminated and finally the 48,842 instances were divided into two files: a training file and a testing one. Fourteen attributes, eight categorical and six continuous were chosen. They were age, work class, weight, education, education number, marital status, occupation, relationship, race, sex, capital gain, capital loss, hours per week and native country.

4. Data Preprocessing

Six attributes are continuous and eight attributes are categorical. Those eight attributes should be processed to numerical form before applying any SVM algorithms. Each element in one category is mapped to an integer in order. After the mapping, all kinds of attributes are numerical and they are mapped to the Gaussian distribution with mean equaling zero and variance equaling one.

5. Model Establishment

To choose the kernel function used in the SVM, we select RBF (Radial basis functions), Linear, Poly and Sigmoid four kernel functions for test. The experiment results of different kernel functions are shown in Table. 1. As we can see that RBF kernel got the highest accuracy when all kernel functions use default parameters or settings.

Table. 1. Results of different kernel function

Kernel function	Number of train set/test set	Accuracy
RBF	30000/15000	84.45%
Linear	30000/15000	83.67%
Sigmoid	30000/15000	77.91%
Poly	30000/15000	81.45%

So, in this paper, we choose RBF kernel to realize the SVM algorithm. Radial basis functions are typically used to build up function approximations of the form

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \varphi(\|\boldsymbol{x} - \boldsymbol{x_i}\|)$$

where the approximating function y(x) is represented as a sum of N radial basis functions, each associated with a different center x_i, and weighted by an appropriate coefficient w_i. The weights w_i can be estimated using the matrix methods of linear least squares, because the approximating function is linear in the weights.

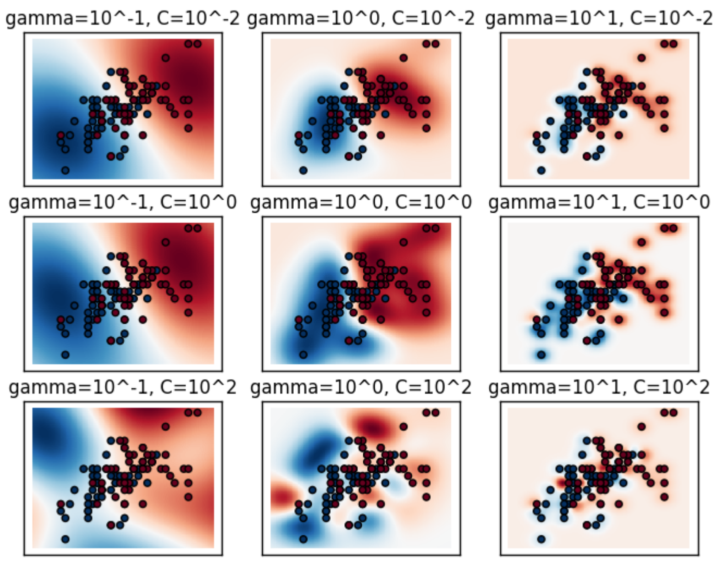


Fig. 1. visualization of the decision function

The figure 1 is a visualization of the decision function for a variety of parameter values on a simplified classification problem involving only 2 input features and 2 possible target classes (binary classification).

Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma parameters can be considered as the inverse of the radius of influence of samples selected by the model as support vectors.

The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.

6. Model Solution

As mentioned before in the SVM section, in order to use the SVM learning algorithm with the Gaussian (RBF, Radial Basis Function) kernel to train it efficiently on the input data sets, values for two parameters (C, γ) are needed. There are no universal best values for all the problems and very often the two parameters are determined empirically. A straightforward method is to use cross-validation on the training dataset and perform a grid search. Cross-validation means that the training dataset is divided into n subsets of equal size. The classifier obtained after training on n-1 subsets is tested on the remaining subset. In order to find the best training, the parameters, (C, γ) are generated in the $[2^{-5}, 2^{30}]\times[2^{-30}, 2^{5}]$ interval with a 1 step for C and -1 step for γ, and the pair with the best cross validation classification rate is chosen. Because the grid-search with cross validation is time consuming it is hard to apply on data sets with the magnitude order larger than 10^3. Since the adult training dataset has over 30000 instances, a subset of cases is randomly selected.

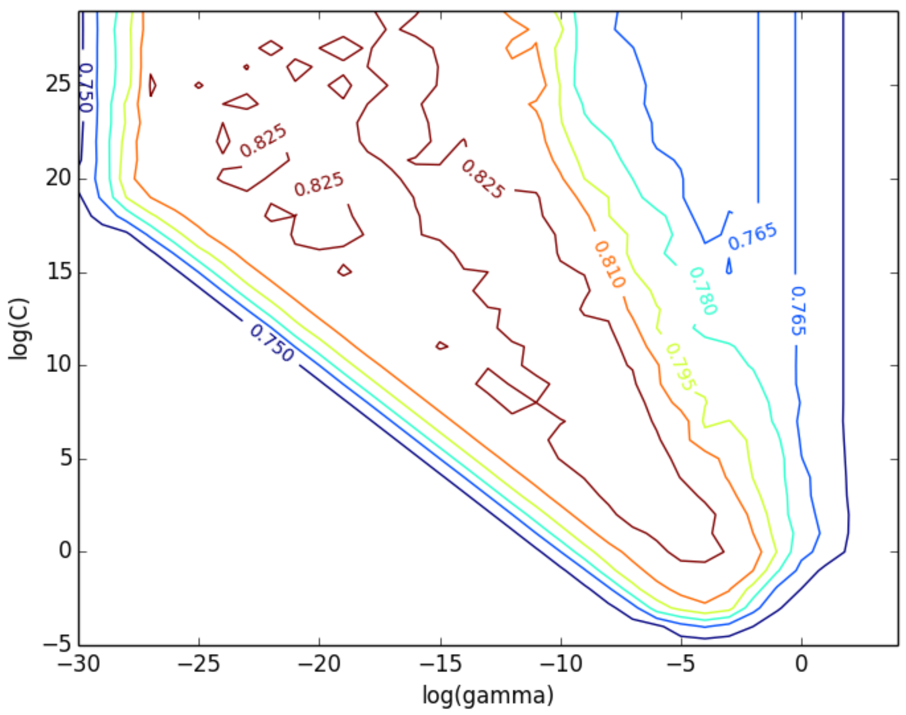


Fig. 2. grid search

The grid for a 10 % randomly chosen train data set, about 3000 instances, is shown in figure 2 with the best (C, γ) pair (2^7, 2^{-7}), resulting in a cross-validation rate of 84.53%.

7. Code Implementation

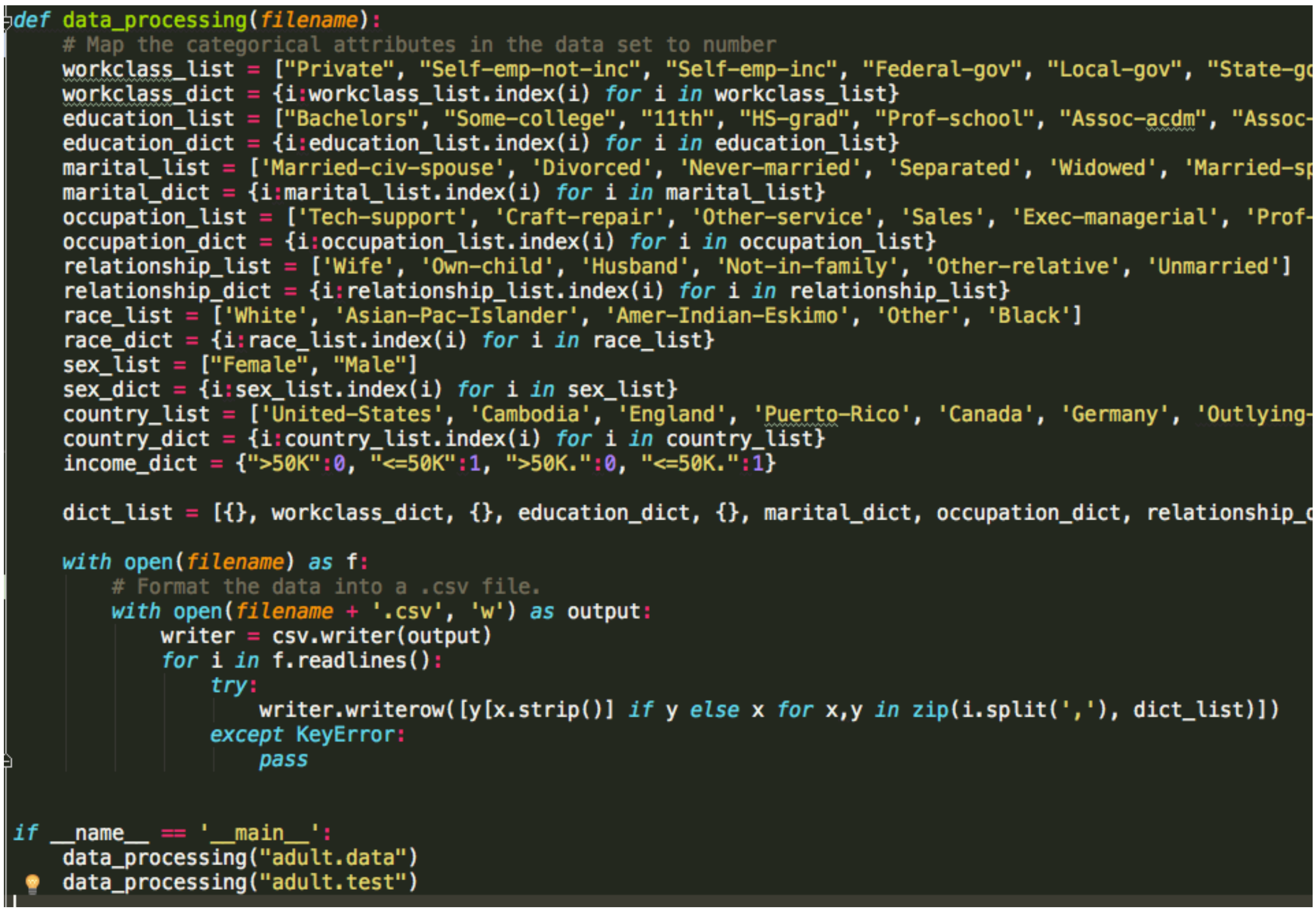
We use Python to implement this SVM method with scikit-learn[5] and matplotlib[6]. We push the whole project repository on Github.com at https://github.com/dreamtalen/adult_income. The project structure is shown in Table. 2.

Table. 2. Project structure

Function Name	Job of Function
data_processing	Map the categorical attributes in the data set to numbers and format the data into a .csv file.
grid_search	Perform a grid search method with cross validation to adjust the parameter C and gamma.
draw_contour	Get the result of grid search method, draw a contour looks like Fig. 2 to find the trend intuitively.
svm	Perform the SVM method with RBF kernel using parameters got in grid search. Output the experiment result.

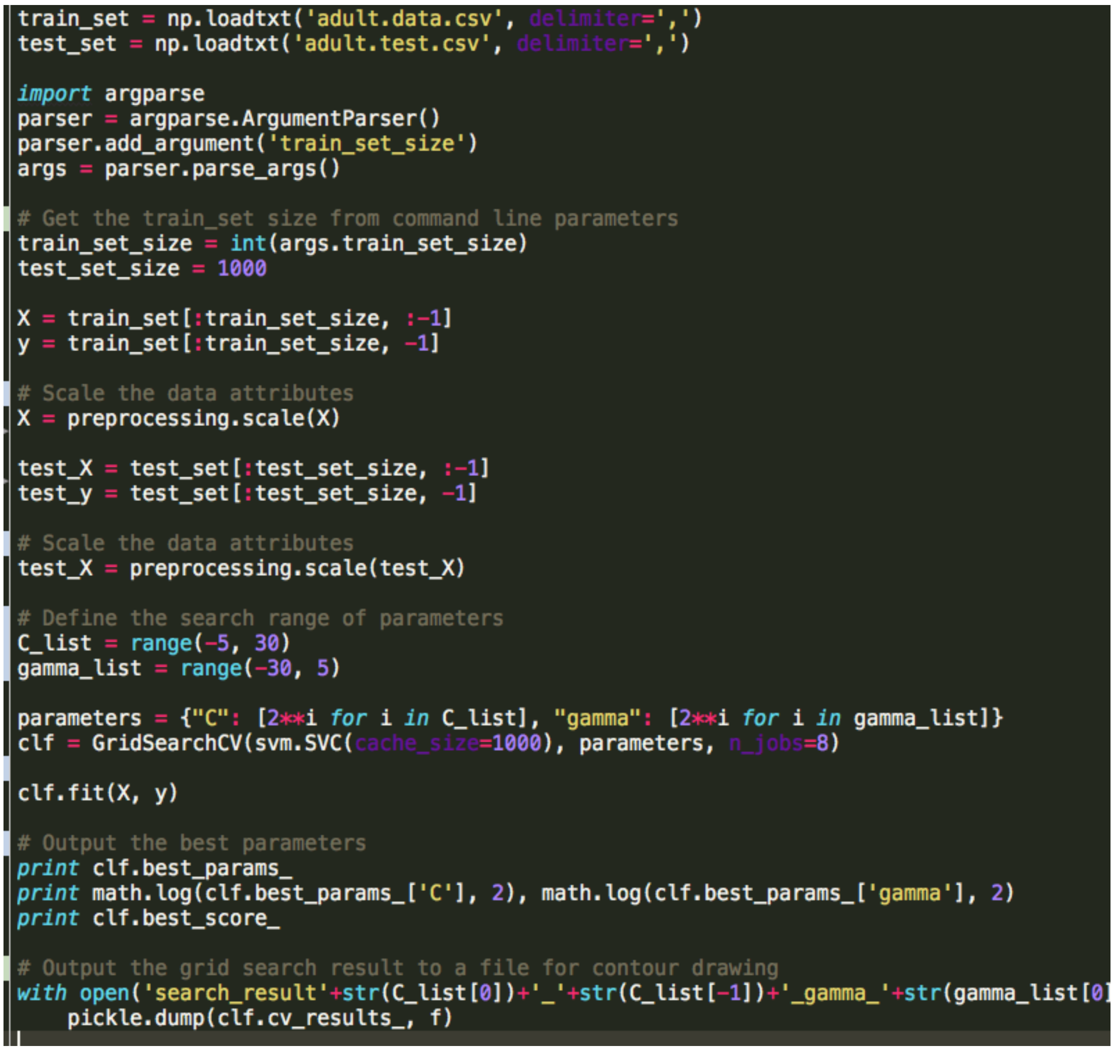
Several important code snippets (Please check the complete code in files):

Data_processing.py



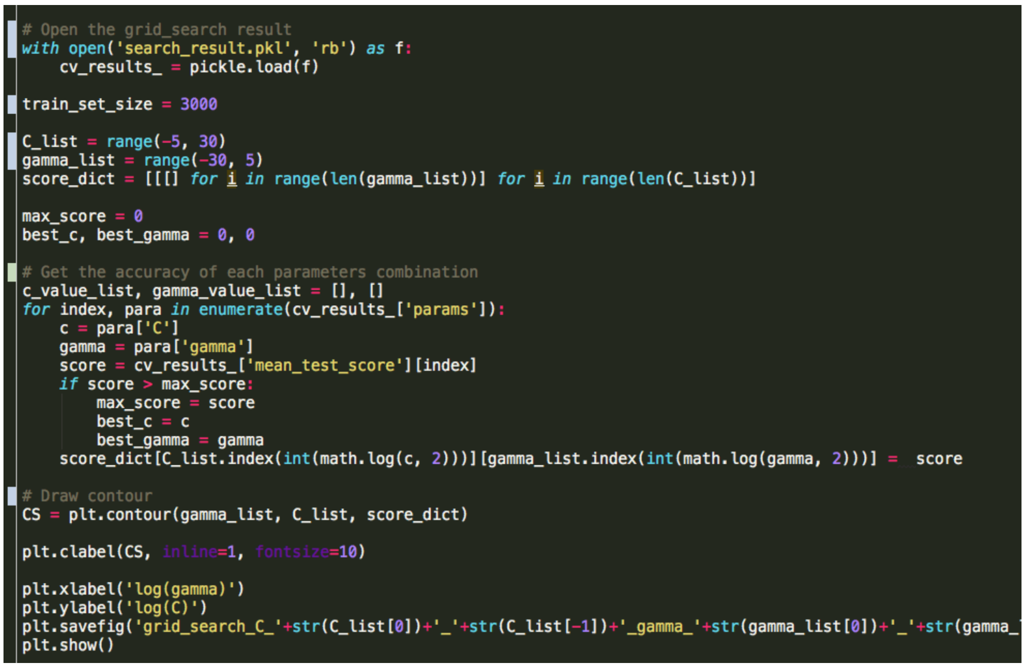
```
def data_processing(filename):
    # Map the categorical attributes in the data set to number
    workclass_list = ["Private", "Self-emp-not-inc", "Self-emp-inc", "Federal-gov", "Local-gov", "State-go
    workclass_dict = {i:workclass_list.index(i) for i in workclass_list}
    education_list = ["Bachelors", "Some-college", "11th", "HS-grad", "Prof-school", "Assoc-acdm", "Assoc-
    education_dict = {i:education_list.index(i) for i in education_list}
    marital_list = ['Married-civ-spouse', 'Divorced', 'Never-married', 'Separated', 'Widowed', 'Married-sp
    marital_dict = {i:marital_list.index(i) for i in marital_list}
    occupation_list = ['Tech-support', 'Craft-repair', 'Other-service', 'Sales', 'Exec-managerial', 'Prof-
    occupation_dict = {i:occupation_list.index(i) for i in occupation_list}
    relationship_list = ['Wife', 'Own-child', 'Husband', 'Not-in-family', 'Other-relative', 'Unmarried']
    relationship_dict = {i:relationship_list.index(i) for i in relationship_list}
    race_list = ['White', 'Asian-Pac-Islander', 'Amer-Indian-Eskimo', 'Other', 'Black']
    race_dict = {i:race_list.index(i) for i in race_list}
    sex_list = ["Female", "Male"]
    sex_dict = {i:sex_list.index(i) for i in sex_list}
    country_list = ['United-States', 'Cambodia', 'England', 'Puerto-Rico', 'Canada', 'Germany', 'Outlying-
    country_dict = {i:country_list.index(i) for i in country_list}
    income_dict = {">50K":0, "<=50K":1, ">50K.":0, "<=50K.":1}

    dict_list = [{}, workclass_dict, {}, education_dict, {}, marital_dict, occupation_dict, relationship_d

    with open(filename) as f:
        # Format the data into a .csv file.
        with open(filename + '.csv', 'w') as output:
            writer = csv.writer(output)
            for i in f.readlines():
                try:
                    writer.writerow([y[x.strip()] if y else x for x,y in zip(i.split(','), dict_list)])
                except KeyError:
                    pass


if __name__ == '__main__':
    data_processing("adult.data")
    data_processing("adult.test")
```

grid_search.py



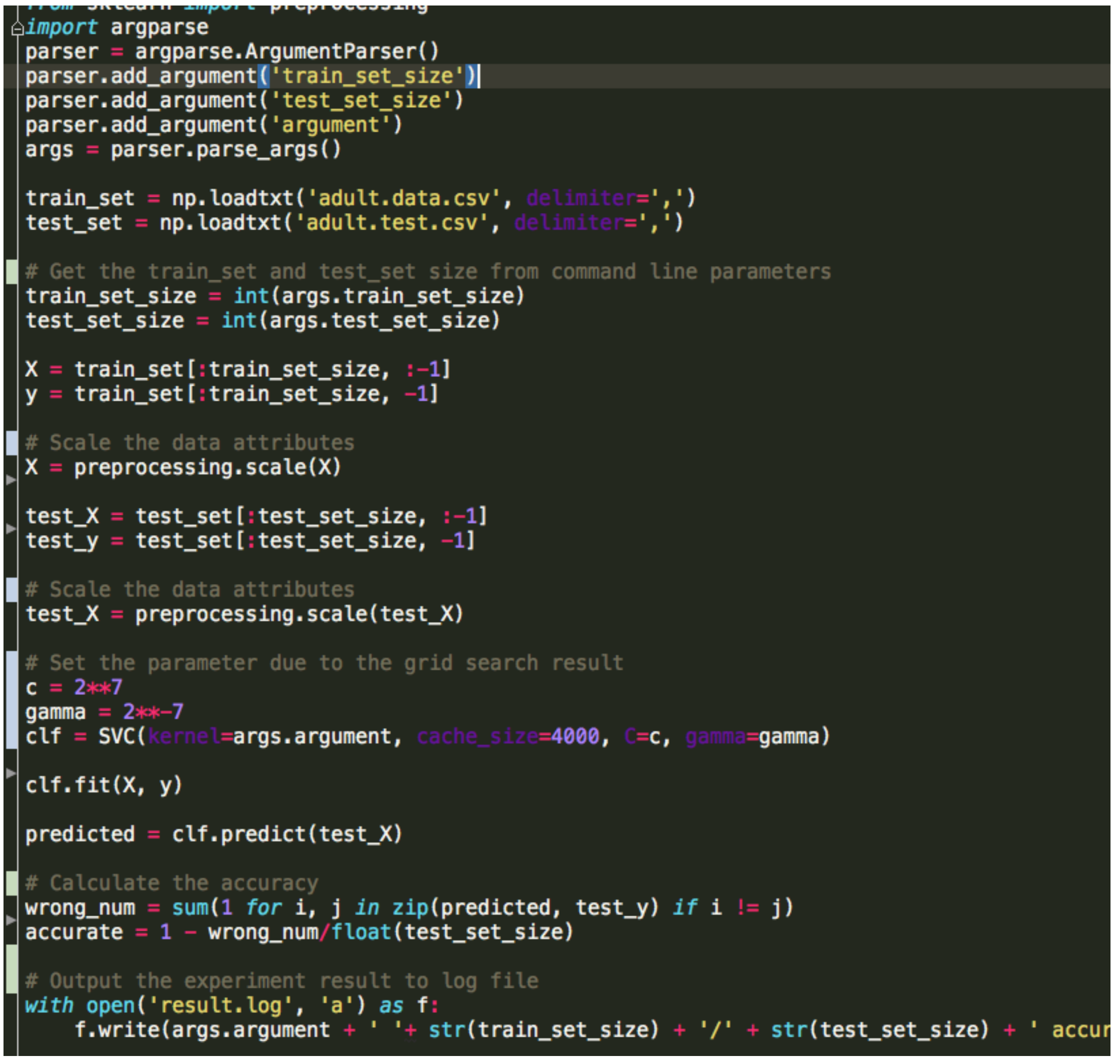
```python
train_set = np.loadtxt('adult.data.csv', delimiter=',')
test_set = np.loadtxt('adult.test.csv', delimiter=',')

import argparse
parser = argparse.ArgumentParser()
parser.add_argument('train_set_size')
args = parser.parse_args()

# Get the train_set size from command line parameters
train_set_size = int(args.train_set_size)
test_set_size = 1000

X = train_set[:train_set_size, :-1]
y = train_set[:train_set_size, -1]

# Scale the data attributes
X = preprocessing.scale(X)

test_X = test_set[:test_set_size, :-1]
test_y = test_set[:test_set_size, -1]

# Scale the data attributes
test_X = preprocessing.scale(test_X)

# Define the search range of parameters
C_list = range(-5, 30)
gamma_list = range(-30, 5)

parameters = {"C": [2**i for i in C_list], "gamma": [2**i for i in gamma_list]}
clf = GridSearchCV(svm.SVC(cache_size=1000), parameters, n_jobs=8)

clf.fit(X, y)

# Output the best parameters
print clf.best_params_
print math.log(clf.best_params_['C'], 2), math.log(clf.best_params_['gamma'], 2)
print clf.best_score_

# Output the grid search result to a file for contour drawing
with open('search_result'+str(C_list[0])+'_'+str(C_list[-1])+'_gamma_'+str(gamma_list[0]
    pickle.dump(clf.cv_results_, f)
```

draw_contour.py



```python
# Open the grid_search result
with open('search_result.pkl', 'rb') as f:
    cv_results_ = pickle.load(f)

train_set_size = 3000

C_list = range(-5, 30)
gamma_list = range(-30, 5)
score_dict = [[[] for i in range(len(gamma_list))] for i in range(len(C_list))]

max_score = 0
best_c, best_gamma = 0, 0

# Get the accuracy of each parameters combination
c_value_list, gamma_value_list = [], []
for index, para in enumerate(cv_results_['params']):
    c = para['C']
    gamma = para['gamma']
    score = cv_results_['mean_test_score'][index]
    if score > max_score:
        max_score = score
        best_c = c
        best_gamma = gamma
    score_dict[C_list.index(int(math.log(c, 2)))][gamma_list.index(int(math.log(gamma, 2)))] =  score

# Draw contour
CS = plt.contour(gamma_list, C_list, score_dict)

plt.clabel(CS, inline=1, fontsize=10)

plt.xlabel('log(gamma)')
plt.ylabel('log(C)')
plt.savefig('grid_search_C_'+str(C_list[0])+'_'+str(C_list[-1])+'_gamma_'+str(gamma_list[0])+'_'+str(gamma_
plt.show()
```

svm.py



```
from sklearn import preprocessing
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('train_set_size')
parser.add_argument('test_set_size')
parser.add_argument('argument')
args = parser.parse_args()

train_set = np.loadtxt('adult.data.csv', delimiter=',')
test_set = np.loadtxt('adult.test.csv', delimiter=',')

# Get the train_set and test_set size from command line parameters
train_set_size = int(args.train_set_size)
test_set_size = int(args.test_set_size)

X = train_set[:train_set_size, :-1]
y = train_set[:train_set_size, -1]

# Scale the data attributes
X = preprocessing.scale(X)

test_X = test_set[:test_set_size, :-1]
test_y = test_set[:test_set_size, -1]

# Scale the data attributes
test_X = preprocessing.scale(test_X)

# Set the parameter due to the grid search result
c = 2**7
gamma = 2**-7
clf = SVC(kernel=args.argument, cache_size=4000, C=c, gamma=gamma)

clf.fit(X, y)

predicted = clf.predict(test_X)

# Calculate the accuracy
wrong_num = sum(1 for i, j in zip(predicted, test_y) if i != j)
accurate = 1 - wrong_num/float(test_set_size)

# Output the experiment result to log file
with open('result.log', 'a') as f:
    f.write(args.argument + ' '+ str(train_set_size) + '/' + str(test_set_size) + ' accur
```

8. Results and Analysis

The Experiments were run with 30000 instances for training and 15000 instances for testing. In Table 3, We can find that our best (C, γ) pair (2^7, 2^{-7}) found in the grid search method can retain highest accuracy rate up to 84.53%.

Table. 3. Results of different C and γ

C	γ	Accuracy
default	default	84.47%
128	0.0078125	84.53%
32	0.0078125	84.49%
32	0.125	83.51%

9. References

[1] Trafalis, T.B., Santosa B., Richman, M.B.: Tornado Detection with Kernel-Based Classifiers From WSR-D88 Radar. Submitted to: Darema, F. (ed.) Dynamic Data Driven Application Systems, Kluwer, 2004.

[2] H. Nuncz, C. Angulo and A. Catala, ”Rule Extraction from Support Vector Machine”, Proccedings of ESANN’2002 – European Symposium on Artificial Neural Networks, Bruges (Belgium), pp. 107-112, 24-26 April 2002.

[3] G. M. Fung,, O. L. Mangasarian, and J. W. Shavlik, “Knowledge-based Nonlinear Kernel Classifiers”, Data Mining Institute Technical Report 03-02. Computer Sciences Department, University of Wisconsin, 2003.

[4] J. Wang and C. Zhang, “Support Vector Machines Based on Set Covering”, Proc. Of the 2nd International Conference on Information Technology for Application (ICITA), Harbin, China, January 2004.

[5] http://scikit-learn.org/stable/index.html

[6] https://matplotlib.org/