CST Part III/MPhil in Advanced Computer Science 2016-2017

Machine Learning and Algorithms for Data Mining

Practical 1: Support Vector Machines

Demonstrators: Duo Wang, Petar Veličković
Lecturers: Mateja Jamnik, Pietro Lio', Thomas Sauerwald
University of Cambridge Computer Laboratory
{mateja.jamnik,pietro.lio,thomas.sauerwald}@cl.cam.ac.uk

1 Introduction

In this practical you will learn how to apply Support Vector Machine (SVM) to real-world data mining problems. We will use the SVM module from scikit-learn, a commonly used machine learning package for Python. Since its invention in 90s, SVM has been a very popular machine learning classifier for high-dimensional dataset with relatively small sample size. In this practical session we will first apply SVM on the iris flower dataset. You will be able to visualize the decision boundary between different classes. You will learn how to tune the hyper-parameters using grid search. At the end, you will have a chance to explore a bigger, more complex dataset of your choice.

2 Environment setup

We will need the following packages for this tutorial:

- NumPy
- SciPy
- matplotlib
- scikit-learn

These packages can be installed via pip install (or pip3 if you are using Python 3). You can install them via:

```
$ pip install numpy
$ pip install scipy
$ pip install matplotlib
$ pip install –U scikit-learn
```

For those of you using a lab machine, these packages are already installed, so you can import them directly in your python script. If you are using your own machine, or you would like to install other packages on a lab machine, we suggest you install these packages in a Python Virtual Environment. A Virtual Environment is a tool which keeps the dependencies required by different projects in separate places, by creating virtual Python environments for them. 'virtualenv' can be installed via pip:

```
$ pip install virtualenv
```

To create a virtual environment with name 'venv':

```
$ virtualenv –p /usr/bin/python{2,3} ~/venv
```

Then you can activate and deactivate the environment:

```
$ source ~/venv/bin/activate
$ deactivate
```

3 Training an SVM classifier

3.1 Iris dataset

In this section we will train a linear classifier on the Iris flower dataset. Iris dataset is a multivariate dataset of three different types of Iris flower. Your job is to classify the data into 3 different classes (Setosa, Versicolor and Virginica) based on 4 features including Sepal length, Sepal width, Pedal length and Pedal width. All 4 features are positive real values. There are in total 150 data instances. Iris dataset is contained in the scikit-learn package. You can import the dataset by:

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
```

3.2 SVM in sklearn

To use SVM module in scikit-learn package, you need to first import it via:

```
from sklearn import svm
```

An SVM Classifier object can be created by:

```
SVM_linear = svm.SVC(kernel='linear', C=1)
```

We will use a linear kernel first. Here 'C' is the tuning parameter C for regulating the permitted margin violation budget. This parameter controls the trade-off between weights and margins in the cost function for maximising the margin:

(1) $\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1}\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{maximize}\ M$

(2) subject to $\sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2 = 1$

(3) $y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_{j1}x_{ij} + \sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right) \geq M(1-\epsilon_i)$

(4) $\epsilon_i \geq 0, \sum_{i=1}^{n}\epsilon_i \leq C$

In order to visualize SMV's decision boundaries in a 2-dimensional plot, we will only use the first two features in our classification. Also we can split our dataset into training set and test set to evaluate the performance of our SVM classifiers. It is also a good idea to randomize the dataset before splitting, so that our test set is sampled uniformly from the dataset.

```
X = iris.data[:,:2]
y = iris.target
indices = np.random.permutation(len(X))
test_size=15
X_train = X[indices[:-test_size]]
y_train = y[indices[:-test_size]]
X_test  = X[indices[-test_size:]]
y_test  = y[indices[-test_size:]]
```

We can train the model with training dataset by 'fit' method:

```
SVM_linear.fit(X_train,y_train)
```

The model object now contains a classifier optimized on the training dataset. Next, we will test the model on the test set to see if the model does capture some inherent structure in the data, rather than just overfit to the training data. We can first predict the label for the test data by 'predict' method. We can then use 'metrics' module from sklearn to automatically generate classification accuracy reports.

```
from sklearn import metrics
y_pred=SVM_linear.predict(X_test)
print (metrics.classification_report(y_test, y_pred))
print ("Overall Accuracy:", round(metrics.accuracy_score(y_test, y_pred),2))
```

'classification_report' produces different statistical measures including precision, recall, F1-score and the number of support vectors. 'accuracy_score' computes the classification accuracy for each class. The size of our test dataset is very small since there are only around 150 samples in total. Therefore, the test accuracies can vary quite a lot as the test data is drawn randomly from the dataset.
So what is the classification report and accuracy produced by your first SVM classifier? Please write it here:

3.3 Visualizing the decision boundaries

It is helpful if we can visualize the decision boundaries of the SVM for each class. This helps us to both visualize the quality of the classifier and to observe how do changes in hyperparameters affect the decision boundaries. On the course website on Moodle, we have provided you with a sampling plotting function. You can just plug in your model and data to generate the decision boundary plot. However, we do encourage you to write your own plotting function that best fits your style. For this dataset, we can visualize the decision boundary and all data points in the plots by calling the function with our trained model and the whole dataset.

```
from svmPlot import svmPlot
svmPlot(X,y,SVM_linear)
```

You can change the C value of your SVM classifier to visualize how does it affect the decision boundary. So what did you observe? Please write it down:

3.4 The kernel trick

In the lectures you have learnt about the kernel trick and different types of kernels. So far, we have only used a linear kernel on the iris dataset. In this section we will try out two more kernels, namely, the Polynomial kernel and the Radial Basis Function (RBF) kernel. To apply Polynomial or RBF kernel, you need to change the kernel parameter of the svm model:

```
SVM_poly = svm.svc(kernel='poly', C=1,degree=2) #For Polynomial Kernel
SVM_rbf = svm.svc(kernel='rbf', C=1,gamma=1) #For RBF Kernel
```

Notice that there are additional parameters 'degree' and 'gamma' here. They are the kernel coefficients. For Polynomial kernel, degree defines the polynomial degree that we referred to in the lectures as dimensions d:

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

For RBF kernel, gamma is the free variable coefficient γ of the Euclidean distance term:

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

Now you can try out these kernels on the iris dataset with different 'gamma' and 'C' values. You can observe their effect on decision boundaries by generating a series of plot with increasing 'gamma' and 'C' values. For example, you can start with gamma and C equal to 1 at the beginning, and increase it gradually to 100. So what have you found by comparing these plots?:

3.5 Grid search for the best parameters

To find the best combination of gamma and C values, we can use grid search. Grid search is simply an exhaustive search over the range of hyper-parameters with uniform spacing between each sampling point in the grid. Grid search can be applied to almost all types of machine learning algorithms. In scikit-learn there is a built-in package named 'GridSearchCV', which does k-fold cross validation along with grid search. Below is a code sample of how to do cross validation in the range of $[-10, 10]$ for C and gamma values with step size equal to 1.

```
from sklearn.grid_search import GridSearchCV
g_range = 2. ** np.arange(-10, 10, step=1)
C_range = 2. ** np.arange(-10, 10, step=1)
parameters = [{'gamma': g_range, 'C': C_range, 'kernel': ['rbf']}]
grid = GridSearchCV(svm.SVC(), parameters, cv= 10, n_jobs=4)
grid.fit(X_train, y_train)
bestG = grid.best_params_['gamma']
bestC = grid.best_params_['C']
print ("The best parameters are: gamma=", np.log2(bestG), " and Cost=", np.
    log2(bestC))
```

Try to adjust ranges and step size for gamma and C values to find the best combination. So which pair did you find that it gives the best accuracy?

4 DIY for a real-world problem

Now it is your chance to apply what you have learnt on a new real-world classification problem. Iris dataset is a simple educational dataset for which it is easy to find a good classifier even in low dimensions. For real-world problems the decision boundaries are usually hyperplanes in high dimensional feature space. We suggest two datasets: digits dataset and spam email dataset. However, you are not limited to these two

datasets. There are plenty classification datasets online for you to try out. You can take a look at the UCI machine learning repository:

https://archive.ics.uci.edu/ml/datasets.

4.1 The digits dataset

The digits dataset contains 1797 8×8 images of hand written digits from 0 to 9. Your task is to design a classifier that determines which digit is in the image. You can look for more details here:

http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html.

This dataset is contained in the scikit-learn package, so you can just use it straight away by importing the dataset.

```
from sklearn import datasets
iris = datasets.load_digits()
```

4.2 The spam email dataset

The spam email dataset contains emails that can be classified into either spam or non-spam emails. Each instance has attributes including word count frequencies, character count frequencies, average, maximum and total length of capital letter sequence. This dataset is hosted online at:

https://archive.ics.uci.edu/ml/datasets/Spambase.