Exercise Notes

Fundamentals of Big Data Analytics
Programming Exercises

Dr. Arash Behboodi
Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Theoretische Informationstechnik
Kopernikusstrasse 16
52074 Aachen

Contents

1 Introduction

The terms *Data Science* and *Big Data* are used interchangeably, as reference to a certain contemporary trend. There is no widely accepted definition of either term that distinguishes one from another. Sometimes Data Science is taken as an all encompassing term involving the technology of extracting information from huge unstructured data and it has Big Data as one of its building blocks. Sometimes Big Data is the general term that includes Data Science as the block providing a set of tools for information extraction. There is no agreed upon definition of these terms, nevertheless one can find recurring themes in many related discussions.

The starting point, the object of study, the material of data science are datasets that posses some, if not all, of the following properties. They are huge, heterogeneous, distributed, unstructured, unreliable and with limited time accessibility. Hundreds of petabytes of data from the large Hadron collider at CERN, brain imaging repositories, genotyping and sequencing, telecommunication call detail records, social media contents and flow field data for investigation of wake flow of rockets are some examples of Big Data. Some of the questions in this context are about the way the data is gathered as well as political issues regarding how to make various data publicly available for research with proper regularization to guarantee privacy concerns.

The next issue is *Data Management.* The tremendous volume of data is prohibitive for utilizing off-the-shelf storage and processing mechanisms. Data Management is concerned with compression, storage, streaming and computing as well as abstraction and integration. It provides the efficient implementation of algorithms of Data Analytics. Complex algorithms usually run over the data with the goal of extracting information, and these difficult computational tasks cannot be performed in the conventional way given the huge volume of data, stored in parallel storage devices. For instance, in the Read-Write stream model, sorting data requires an internal memory space, increasing at least logarithmically with data size. One way to perform these algorithms is to exploit the structure of the data, for example exploiting symmetries and regularities in the data, or to decompose data into more or less independent parts. Data Management relies on massively parallel architectures. Theoretical limits of the system as well as its practical developments are the focus of the research community.

Data Analytics provides the algorithms, developed mainly to extract quantifiable information from data sets as described above. Data Analytics is concerned with the development and analysis of information extraction algorithms and relies on tools from a variety of disciplines such as, to name a few, statistics, machine learning, classification theory, detection theory and compressed sensing. It is very difficult to describe and classify the inventory of tools in Data Analytics in a comprehensive way. One can attempt to classify these algorithms with respect to the specific problem they want to solve such as classification, regression, etc. Alternatively one can put these algorithms in different categories based on the mathematical field they draw from, such as graph-based and probabilistic methods. The theories such as optimization theory appear ubiquitously in many different algorithms and therefore have importance of their own for Data Analytics. In the next part, the methods used in Data Analytics are reviewed in a similar way. The challenge is to come up with tools that provide satisfactory performance with respect to certain metrics in addition to being computationally efficient.

Another less discussed issue is *mathematical modeling*, i.e. the development of mathematical models for a specific problem to provide an adequate benchmark for both Data Analytics and data scientists. The availability of theoretical benchmarks is important for performance evaluation in Data Analytics and also for deriving guidelines about the best algorithms. It sets the path for developing algorithms that realize the benchmark. The challenge is to devise a tractable model for a complex system acting as a data source.

1.1 On Methodology of Data Analytics

In this part, a survey of the algorithms of Data Analytics is provided. A comprehensive survey of all algorithms used in Data Analytics, even by rough classification, is very difficult, if not impossible.

There are some objectives that frequently appear as the goal of Data Analytics. Some of the more well-known ones are classification, clustering, regression, dimensionality reduction and visualization.

The purpose of dimensionality reduction and data visualization is to embed the high dimensional data into a lower dimensional space with the goal of efficiently providing the information about the data as well as using tools suitable to the target space for extracting information. For instance, the data can be embedded into a proper continuous framework to exploit natural problem metrics. Some examples of such tools are Multidimensional Scaling (MDS) and Principal component analysis (PCA).

Classification is concerned with separation of data into already available categories. Some algorithms that are used for this purpose are perceptron learning algorithms, Support Vector Machines (SVMs), Decision Trees & Ensemble Methods, Artificial Neural Networks (ANNs) and Naive Bayes Classifiers.

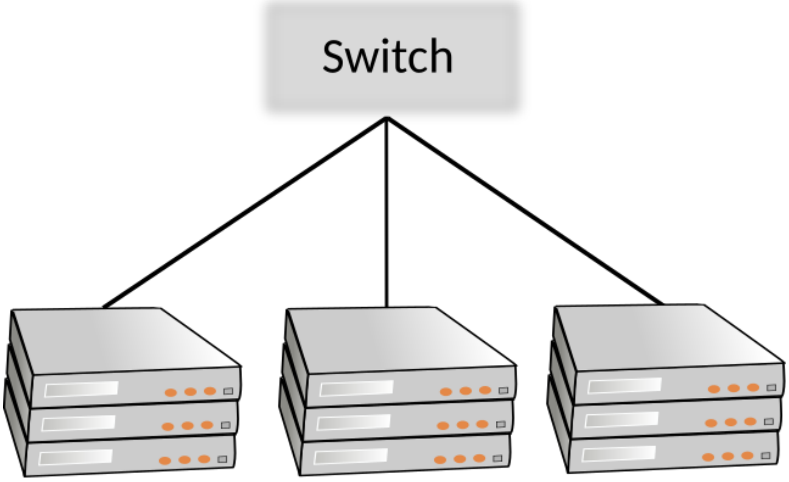
Clustering on the other hand does not rely on any predisposed categorization and one goal is indeed to find the categorization of data as well. A topic, closely related to clustering, is the community detection problem where the goal is to find clusters of nodes based on the similarity of their attributes. Clustering methods include for example K-Means Clustering, K-Medoids Clustering, Sequential Agglomerative Hierarchic Nonoverlapping (SAHN) clustering, Clustering Using Representatives (CURE) and Density-Based Spatial Clustering of Applications plus Noise (DBSCAN).

Regression problems aim at revealing the relation between variables in data-sets. Its main application is prediction which is central to fault detection applications. An example of regression algorithms is Gaussian process learning.

There are general theories concerned with information extraction and recovery. Compressive sensing, machine learning, pattern recognition, statistics and signal processing can be used to extract information from noisy data-sets. It is hard to make a definite distinction between these theories since their scope overlaps to a large extent and include some of the problems we discussed before. Denoising algorithms such as block-matching and 3-D filtering (BM3D) are also used for information recovery. The choice of model for each problem is an important step as it can naturally lead to usage of one of existing algorithms. There are theories such as harmonic analysis that have direct application in information recovery by providing a structured representation of data, for instance, by finding a sparsifying basis or frame for the data. Optimization theory is ubiquitous in Data Analytics. Performing many of these algorithms boils down to solving an optimization problem which may turn out to be nonconvex and NP-hard and therefore computationally efficient solvers are desirable for many algorithms. The formulation of the optimization problem plays a significant role in the computational efficiency of the solving algorithm.

Previously mentioned algorithms are based on tools from different mathematical fields. A method can be a combination of many tools from different fields. Although the categorization of algorithms of Data Analytics according to their used mathematical tools is not exhaustive, we can nevertheless recognize a few important classes. Many algorithms utilize the graph-based representation of problems and their solutions. These methods are labled as Graph based methods and include, for example, neural networks, deep learning, random forests, Bayesian networks and decision tree algorithms. Probability based methods are also prevalent in Data Analytics, for instance, by modeling the data as random processes in time and space leading to possibly highly correlated highdimensional data or by using tools of estimation and detection theory for extracting information. Some examples include Gaussian processes learning, large sample approximations, Bayesian network and Hidden Markov Models. Finally, another set of tools exploit mainly the geometric structure of the data, for example, when it forms a vector space equipped with an inner product, or when the data can be embedded into a lower dimensional structure, such as a manifold. Some examples of these geometry-based methods are manifold learning, Principal Component Analysis (PCA), compressed sensing, and SVM.

1.2 Distributed Data Storage



Racks of compute nodes

When the computation is to be performed on very large data sets, it is not efficient to fit the whole data in a data-base and perform the computations sequentially. The key idea is to use parallelism from “computing clusters”, not a super computer, built of commodity hardware, connected by Ethernet or inexpensive switches.

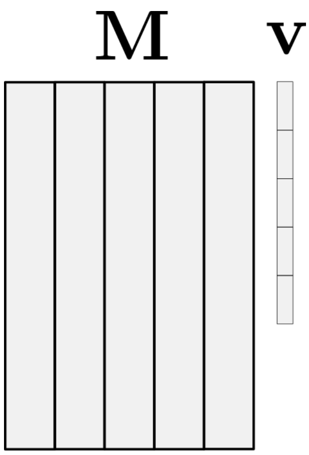
The software stack consists of distributed file systems (DFS) and MapReduce. In a distributed file system Files are divided into chunks (typically 64 MB) and chunks are replicated, typically 3 times on different racks. There exists a file master mode or name mode with information where to find copies of files. Some of the implementations of DFS are GFS (Google file system), HDFS (Hadoop Distributed File System, Apache) and Cloud Store (open source DFS).

On the other hand MapReduce is the computing paradigm. In MapReduce, the system manages parallel execution and coordination of tasks. Two functions are written by users namely Map and Reduce. The advantage of this system is its robustness to hardware failures and it is able to handle large datasets. MapReduce is implemented internally by Google.

The architecture of this system is such that compute nodes are stored on racks, each with its own processor and storage device. Many racks are connected by a switch as presented in Figure 1.1. They are connected by some fast network, interconnection by Gigabit Internet. The principles of this system are as follows. First, files must be stored redundantly to protect against failure of nodes. Second, computations must be divided into independent tasks. If one fails it can be restored without affecting others.

We discuss an example of implementation matrix-vector multiplication using MapReduce.

Example (Matrix-Vector Multiplication by MapReduce)**.** *Suppose that the matrix* $\mathbf{M} \in \mathbb{R}^{m \times n}$



Matrix-Vector Multiplication

and the vector $\mathbf{v} \in \mathbb{R}^n$ *are given and the goal is to compute their multiplication* $\mathbf{x} = \mathbf{M}\mathbf{v}$*:*

$$x_i = \sum_{j=1}^{n} m_{ij} v_j.$$

When n *is large, say* 10^7 *then the direct computation requires the storage of the whole matrix in the storage which might not be efficient. Particularly in practice the matrix* $\mathbf{M}$ *can be sparse with say 10 or 15 non-zeros per row.*
First the matrix and the vector is stored as the pairs $(i, j, m_{ij}$ *and the vector is stored as* (i, v_i)*. MapReduce consists of two main functions, Map function and Reduce function. To implement the multiplication using MapReduce, Map function produces a key-value pair to each entries of the matrix and the vector. To the entry* m_{ij} *the pair* $(i, m_{ij}v_j)$ *is associated where* i *is the key and* $m_{ij}v_j$ *is the pair. Note that it is assumed here that* m *is small enough to store the vector* $\mathbf{v}$ *in its entirety in the memory. The Reduce function receives all the key-value pairs, lists all pairs with key* i *and sum their values to get* $(i, \sum_{j=1}^{n} m_{ij}x_j)$ *which gives the ith entry of the product. If the vector* $\mathbf{v}$ *cannot fit into the memory then the matrix* $\mathbf{M}$ *is divided into horizontal strips with certain width and the vector* $\mathbf{v}$ *is divided into vertical stripes with the same size as the matrix stripes' width. Accordingly the multiplication can be divided into sub-tasks, each feasible using the MapReduce.*

Example (Matrix-Matrix Multiplication by MapReduce)**.** *Given two matrices* $\mathbf{M} \in \mathbb{R}^{n\times m}$ *and* $\mathbf{N} \in \mathbb{R}^{m\times r}$*, the goal is to compute* $\mathbf{MN}$*. Map function generates the following key-value pairs:*

- *For each element* m_{ij} *of* $\mathbf{M}$ *produce* r *key-value pairs* $((i,k),(\mathbf{M}, j, m_{ij}))$ *for* $k = 1, \ldots, r$*.*
- *For each element* n_{jk} *of* $\mathbf{N}$ *produce* n *key-value pairs* $((i,k),(\mathbf{N}, j, n_{jk}))$ *for* $i = 1, \ldots, n$*.*

The Reduce function computes the multiplication as follows:

- *For each key* (i,k)*, find the values with the same* j*.*
- *Multiply* m_{ij} *and* n_{jk} *to get* $m_{ij}n_{jk}$*.*
- *Sum up all* $m_{ij}n_{jk}$ *over* j *to get* $\sum_{j=1}^{m} m_{ij}n_{jk}$*.*

1.3 Training, Validation and Test Datasets

A central problem in machine learning is overfitting. It occurs when the learning algorithm gives a very good performance on the training data but it performs badly when it is tested after training. This is because the learning algorithm utilize a model which is more complex and therefore learns those features of particular training set which is non-essential to the task. Thereby it becomes sensitive to variation of those features. Often this is due to the high number of free parameters in the model. Regularization is an important technique in statistics particularly in the context of inverse problems used for adding more constraints on the desired solution. By limiting the solutions as such, it can be used to prevent overfitting.

However to avoid overfitting, the available dataset for learning is divided into two datasets. Only one of them is used for training called the training set. The other one is called test set and is used to examine the overfitting phenomena. After each update in the model, the training error and the test error are observed. A low training error and high test error indicates overfitting. In some cases, the dataset is divided into three new datasets namely training, validation and test dataset. The validation set is not used for training but rather for choosing the hyperparameters of a particular model used for training.

2 MNIST Dataset

2.1 MNIST Dataset Description

The Modified National Institute of Standards and Technology dataset, shortly called MNIST dataset is commonly used for benchmarking purpose in machine learning research. It contains images of handwritten digits from 0 to 9. The dataset consists of training and test dataset each containing these gray-scale images of hand-drawn digits. The training set contains data that should be used for designing (or training) a model, while the second dataset is used for testing the obtained solution.
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as $x = i \times 28 + j$, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero). For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below. Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

```
000  001  002  003  ...  026  027
028  029  030  031  ...  054  055
056  057  058  059  ...  082  083
 |    |    |    |   ...   |    |
728  729  730  731  ...  754  755
756  757  758  759  ...  782  783
```

MNIST dataset is included in software packages like Tensorflow and PyTorch. There are many works on MNIST dataset. One can check the following website for detailed information and downloading the dataset as well:

http://yann.lecun.com/exdb/mnist/

Four files can be found in the above website:

- Training set: *train-images-idx3-ubyte*
- Training labels: *train-labels-idx1-ubyte*
- Test set: *t10k-images-idx3-ubyte*
- Test labels: *t10k-labels-idx1-ubyte*

As it is explained in the webpage, the size of training set is 60000 while the size of test set is 10000.

2.2 Loading MNIST dataset

After these datasets have been downloaded, we load them and try to play around with them. We define the file names corresponding to the dataset as well as the directory where they are stored.

```
In [1]: FILES_DIR = 'MNIST_torch/raw/'
        TRAIN_FILE = 'train-images-idx3-ubyte'
        TRAIN_LABEL = 'train-labels-idx1-ubyte'
        TEST_FILE = 't10k-images-idx3-ubyte'
        TEST_LABEL = 't10k-labels-idx1-ubyte'
```

Note that there are additional information attached to the top of each file, for example in the training set:

offset	type	value	description
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
⋮	⋮	⋮	⋮
xxxx	unsigned byte	??	label

We now open the provided dataset files and output their sizes. The size of training and test set is the same as before. Note that the validation set should be manually constructed from the training set. Moreover the additional header information should be removed.

```
In [2]: import numpy as np
        with open(FILES_DIR + TRAIN_FILE,'rb') as ftemp:
              datatemp = np.fromfile(ftemp, dtype = np.ubyte)
              dataRTraining=datatemp[16::].reshape(60000,784)
              print('Size of the training set: ',dataRTraining.shape)
        with open(FILES_DIR + TRAIN_LABEL,'rb') as ftemp:
              datatemp = np.fromfile(ftemp, dtype = np.ubyte)
              labelRTraining=datatemp[8::]
              print('Size of the training labels: ',labelRTraining.shape)
        with  open(FILES_DIR + TEST_FILE) as ftemp:
              datatemp = np.fromfile(ftemp, dtype = np.ubyte)
              dataRTest=datatemp[16::].reshape(10000,784)
              print('Size of the test set: ',dataRTest.shape)
        with open(FILES_DIR + TEST_LABEL,'rb') as ftemp:
              datatemp = np.fromfile(ftemp, dtype = np.ubyte)
              labelRTest=datatemp[8::]
              print('Size of the test labels: ',labelRTest.shape)
```

```
Size of the training set:  (60000, 784)
Size of the training labels:  (60000,)
Size of the test set:  (10000, 784)
Size of the test labels:  (10000,)
```

Now let’s visualize some of the data.

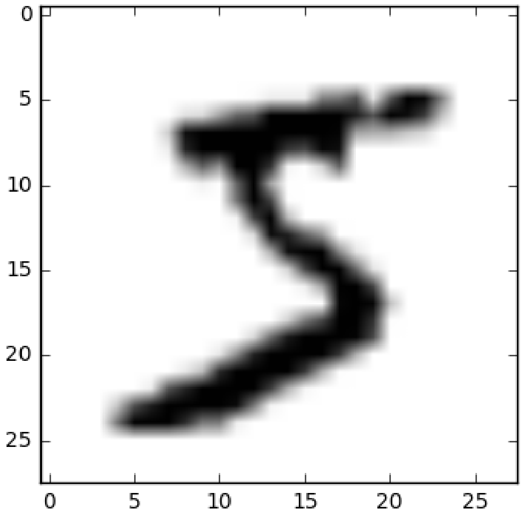
```
In [3]: print(dataRTraining[0])
        print('The label is:',labelRTraining[0])
```

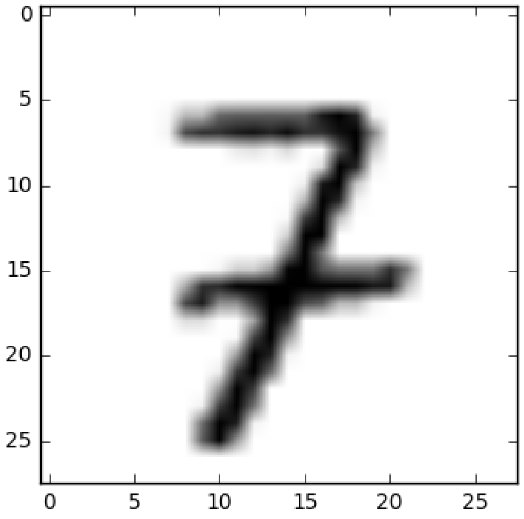
```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
The label is: 5
```

We can visualize it using another python function as follows. Note that the python the libraries pyplot and numpy are used for plotting and computing numerical operations.

```
In [4]: # %matplotlib inline
        from matplotlib.pyplot import imshow
        import matplotlib.pyplot as plt
        imshow(dataRTraining[0].reshape(28,28) ,cmap='binary')
        plt.show()
```

```
imshow(dataRTraining[2018].reshape(28,28) ,cmap='binary')
plt.show()
```





2.3 Loading MNIST using Tensorflow

Now we use the tensorflow library provided by Google for loading the MNIST dataset. When using tensorflow there is a built-in solution for importing the MNIST dataset within the tensorflow library. This spares us the work of loading and formating the data contained in the aforementioned CSV files. Instead, we may import a data structure from the tensorflow library as:

```
In [5]: import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data
        data = input_data.read_data_sets("tensorboard_MNIST/MNIST_data/", one_hot=True)
```

```
Extracting tensorboard_MNIST/MNIST_data/train-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/train-labels-idx1-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-labels-idx1-ubyte.gz
```

Note that, when loading the data, we enabled the one_hot option. *One-hot* refers to the formatting style of the label vector(s) provided. For this dataset, there are 10 possible labels corresponding to handwritten digits between 0 and 9. Then, for labeling an image, it is sufficient to assign an integer number between 0 and 9 corresponding to its label. Nevertheless, when using the aforementioned one-hot representation of the labels, every image is labeled using a 10-dimensional vector with the value 1 in the entry corresponding to its assigned label (i.e., the "hot" entry) and zero elsewhere. Later in this course we will discuss the role of the one-hot format in the context supervised learning.
Up to this point, we have extracted the **MNIST** datset, which is composed of 70.000 images and associated labels.
When loading this dataset from the examples provided by the tensorflow library, these 70.000 images and lables are already partioned into 3 datasets:

```
In [6]: print("Size of:")
        print("- Training-set:\t\t{}".format(len(data.train.labels)))
        print("- Test-set:\t\t{}".format(len(data.test.labels)))
        print("- Validation-set:\t{}".format(len(data.validation.labels)))
```

```
Size of:
- Training-set:55000
- Test-set:10000
- Validation-set:5000
```

Each element of the dataset is a vector of dimension $784 = 28 \times 28$.

```
In [7]: print("shape of first entry:",data.train.images[0,:].shape)
        print("shape of second entry:",data.train.images[1,:].shape)
        print("shape of third  entry:",data.train.images[2,:].shape)
```

```
shape of first entry: (784,)
shape of second entry: (784,)
shape of third  entry: (784,)
```

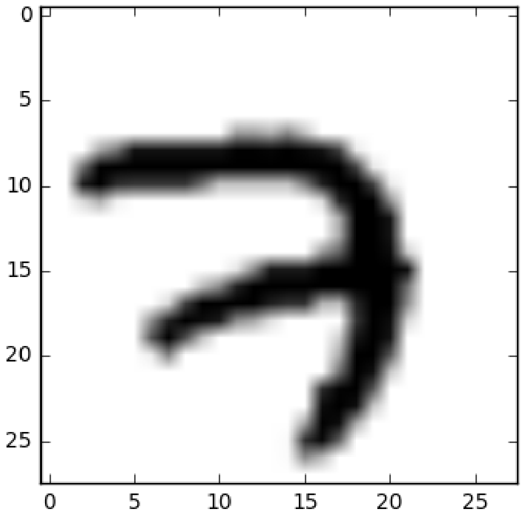
As discussed, the label vectors inside "data.train.labels" are in a one-hot format. We can extract the label number (between 0 and 9) by seaching for the entry with maximum value within the one-hot vector:

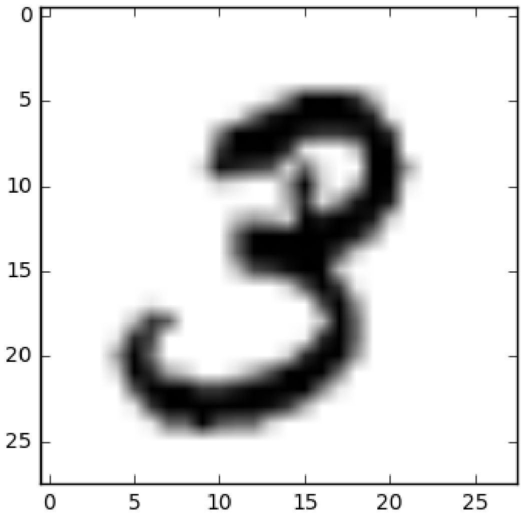
```
In [8]: print("label of first entry:",data.train.labels[0,:])
        print("label of first entry:",data.train.labels[0,:].argmax())
        print("label of second entry:",data.train.labels[1,:])
        print("label of second entry:",data.train.labels[1,:].argmax())
```

```
label of first entry: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
label of first entry: 7
label of second entry: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
label of second entry: 3
```

We now show that this images correspond to the handritten digits '7' and '3' as expected. Note that, in order to display these digits we must reshape the 784-dimensional image vectors back into 28 × 28 images.

```
In [9]: plt.imshow(data.train.images[0,:].reshape([28,28]),cmap='binary')
        plt.show()
        plt.imshow(data.train.images[1,:].reshape([28,28]),cmap='binary')
        plt.show()
```





2.4 MNIST dataset in PyTorch

PyTorch is a python library developed by Facebook particularly for deep learning using GPU and CPU. In this tutorial, we load the MNIST dataset. We import two libraries torch and torchvision, the later for datasets.

```
In [10]: import torch
         from torchvision import datasets
```

MNIST dataset is loaded using the following command. There is a download flag which is set to True if the dataset is to be downloaded from the internet. If MNIST dataset for PyTorch has been downloaded, two files training.pt and test.pt are found in the folder “processed”. In this case, put the flag to False so that it is not downloaded again. The flag “train” determines if you intend to load training or test set.

```
In [11]: DNLD=True
         trainingMNIST = datasets.MNIST('./MNIST_torch', train=True, download=DNLD)
         testMNIST = datasets.MNIST('./MNIST_torch', train=False, download=DNLD)
```

The size of training and test set is the same as before. Note that the validation set should be manually constructed from the training set.

```
In [12]: print("Size of:")
         print("- Training-set:\t\t{}".format(len(trainingMNIST)))
         print("- Test-set:\t\t{}".format(len(testMNIST)))
```

```
Size of:
- Training-set:60000
- Test-set:10000
```

Each entry of the dataset consists of PIL image module and its label. In the following, we explore similarly the content of entries.

```
In [13]: trainingMNIST[0]
```

```
Out[13]: (<PIL.Image.Image image mode=L size=28x28 at 0x7FEE0431D748>, 5)
```

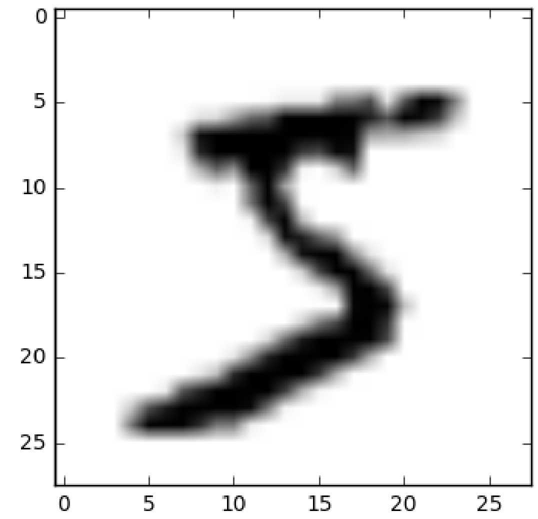
```
In [14]: print("shape of first entry:",trainingMNIST[0][0].size)
         print("shape of second entry:",trainingMNIST[1][0].size)
         print("shape of third entry:",trainingMNIST[2][0].size)
```

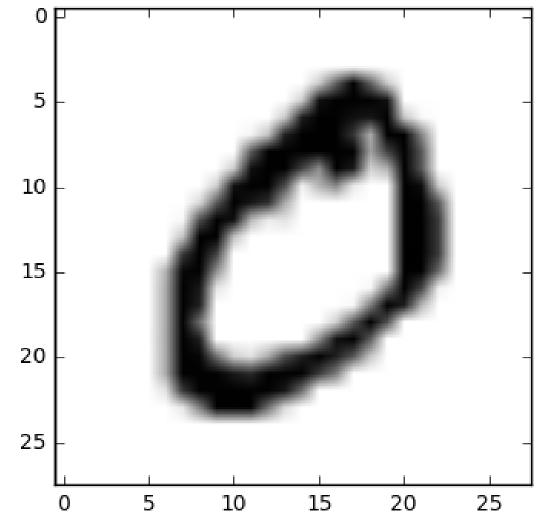
```
shape of first entry: (28, 28)
shape of second entry: (28, 28)
shape of third entry: (28, 28)
```

```
In [15]: print("label of first entry:",trainingMNIST[0][1])
         print("label of first entry:",trainingMNIST[1][1])
         print("label of first entry:",trainingMNIST[2][1])
```

```
label of first entry: 5
label of first entry: 0
label of first entry: 4
```

```
In [16]: imshow(np.asarray(trainingMNIST[0][0]),cmap='binary')
         plt.show()
         imshow(np.asarray(trainingMNIST[1][0]),cmap='binary')
         plt.show()
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

3 Dimensionality Reduction

In this part, the goal is to find a suitable low-dimensional representation of a dataset. Our focus will be on the following tools:

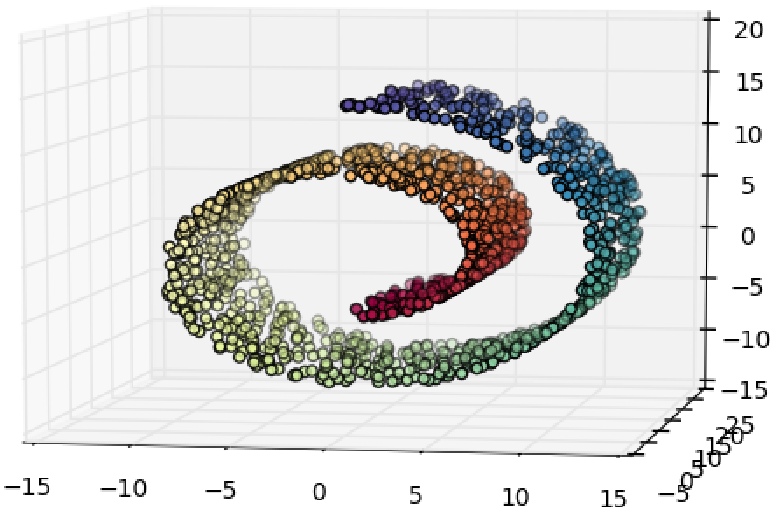
- Principal component analysis
- Isomap
- Diffusion Maps

We will also use Tensorboards for data visualization at the end. We use two dataset, MNIST and Swiss Roll. We have introduced MNIST previously. We now introduce Swiss Roll. We use sklearn package for that purpose.

```
In [1]: from sklearn.datasets.samples_generator import make_swiss_roll as sroll
        n_samples = 1500
        noise = 0
        X, color = sroll(n_samples, noise)
```

Let's plot the data and see how it looks like.

```
In [2]: %matplotlib inline
        import matplotlib.pyplot as plt
        import mpl_toolkits.mplot3d.axes3d as p3
        fig = plt.figure()
        ax = p3.Axes3D(fig)
        ax.view_init(7, -80)
        ax.scatter(X[:, 0], X[:, 1], X[:, 2],'o',c=color, cmap=plt.cm.Spectral)
        plt.show()
```



This dataset is an example of data model with non-linear structure. More precisely the data lives on a manifold and the goal is to exactly learn that.

3.1 Dimensionality Reduction for Swiss Roll

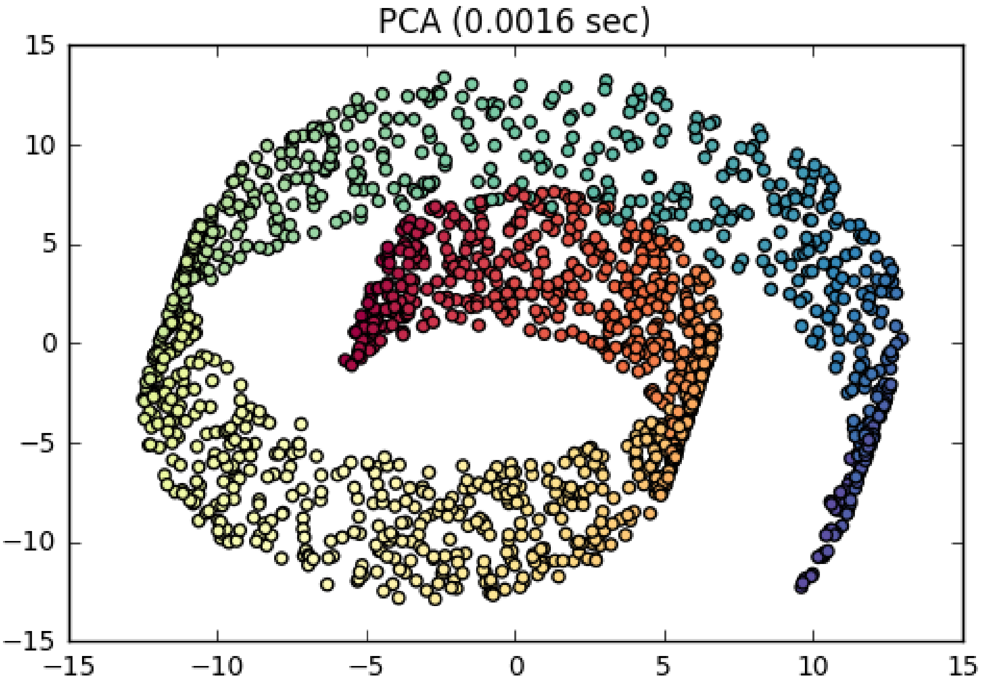
We now try to apply our algorithms to find a suitable representation of Swiss Roll in two-dimensional space. We measure the run time of each algorithm as well.

```
In [3]: from time import time
```

3.1.1 PCA

We first run the PCA algorithm from sklearn package and then compare it with our own implementation.

```
In [4]: from sklearn.decomposition import PCA
        t0pca = time()
        XPCA = PCA(n_components=2).fit_transform(X)
        t1pca = time()
        plt.scatter(XPCA[:, 0], XPCA[:, 1], c=color, cmap=plt.cm.Spectral)
        plt.title("PCA (%.2g sec)" % (t1pca - t0pca))
        plt.show()
```



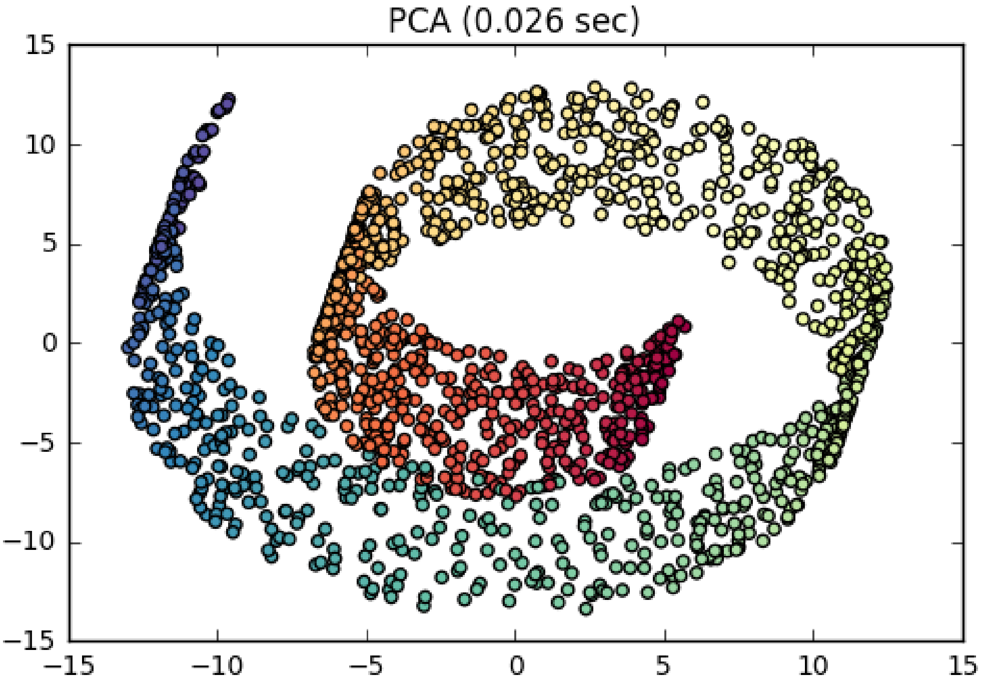
To implement PCA the sample covariance matix of the data should be formed. We first import linear algebra package from python.

```
In [5]: import numpy as np
        from numpy import linalg as la
```

Here are the main steps:

- The sample covariance matrix is found using a simple command.
- Spectral decomposition of the sample covariance matrix is found.
- The top eigenvectors are chosen and the projection matrix is formed.

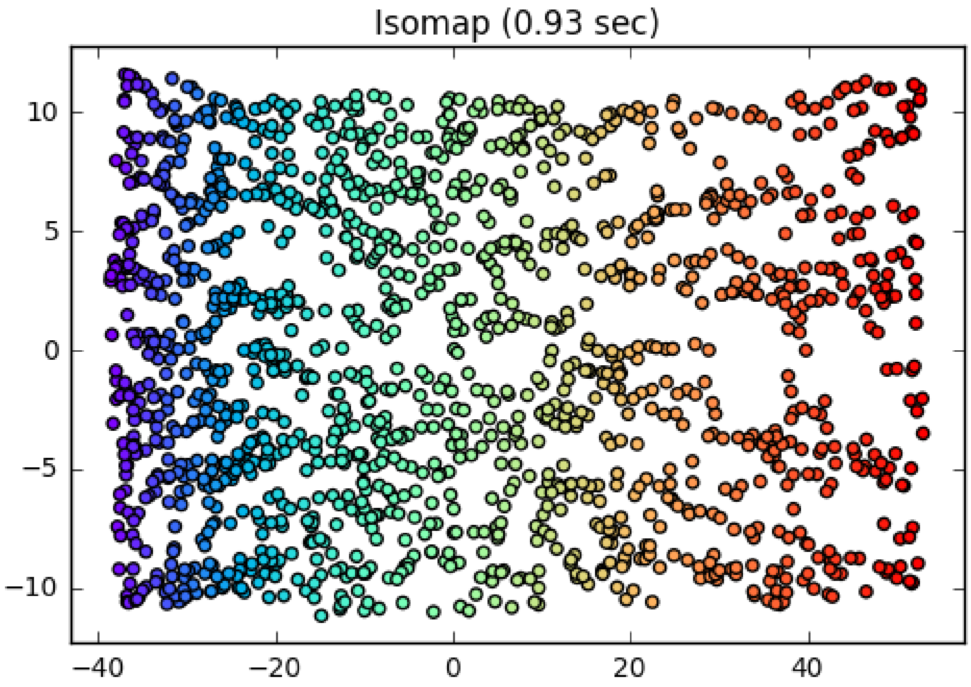
```
In [6]: t0pca = time()
        sampleCov=np.cov(X.T)
        eigvalCov, eigvecCov = la.eig(sampleCov)
        idx = eigvalCov.argsort()[::-1]
        eigvalCov = eigvalCov[idx]
        eigvecCov = eigvecCov[:,idx]

        ## Finding the projection matrx
        n=n_samples
        Qproj=eigvecCov[:,[0,1]]
        En=np.eye(n)-np.ones((n,n))/n # Centering matrix

        ## This is one way of finding PCA
        XPCA=(Qproj.T@X.T@En).T
        t1pca = time()
        plt.scatter(XPCA[:, 0], XPCA[:, 1], c=color, cmap=plt.cm.Spectral)
        plt.title("PCA (%.2g sec)" % (t1pca - t0pca))
        plt.show()
```



3.1.2 Isomap

Isomap is a manifold learning algorithm. In this example we construct the underlying graph using 10 nearest neighbors.

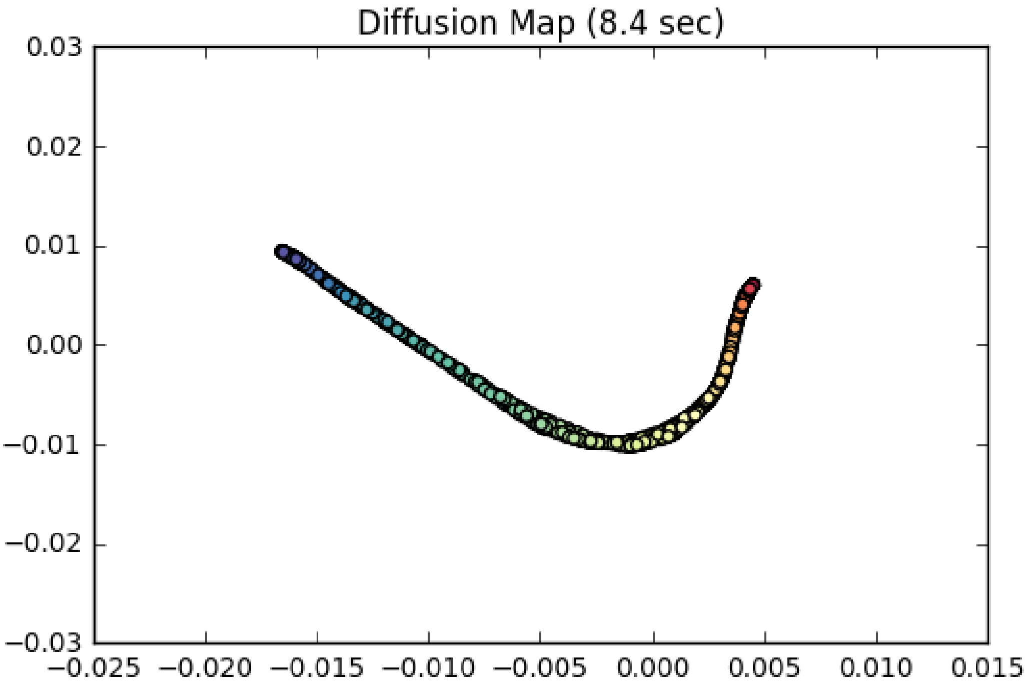
```
In [7]: from sklearn import manifold
        n_neighbors = 10
        t0iso = time()
        XISO= manifold.Isomap(n_neighbors, n_components=2).fit_transform(X).T
        t1iso = time()


        plt.scatter(XISO[0], XISO[1], c=color, cmap=plt.cm.rainbow)
        plt.title("%s (%.2g sec)" % ('Isomap', t1iso - t0iso))
        plt.axis('tight')
        plt.show()
```



3.1.3 Diffusion Maps

Diffusion maps are another example of non-linear dimensionality reduction algorithm. They do not have an embeded implementation inside python so we use our own development here.

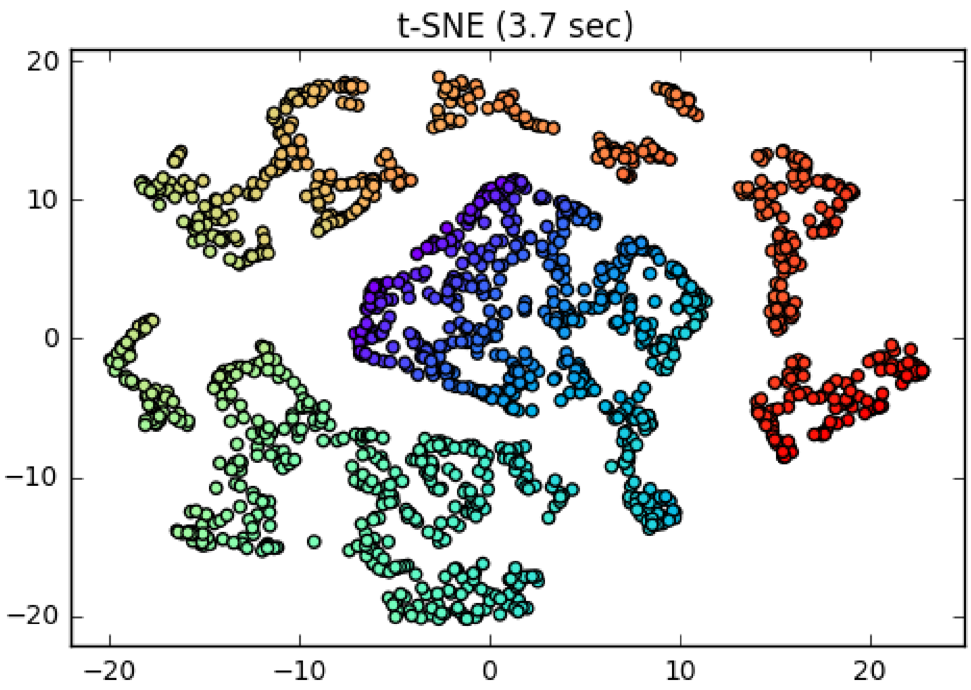
```
In [8]: from diffusionmapmod import diffusionmap as Diff

        eps= 6
        t=10
        k=2
        t0diff = time()
        XDM=Diff(X,n_samples,eps,t,k).T
        t1diff = time()

        plt.scatter(XDM[:, 0], XDM[:, 1], c=color, cmap=plt.cm.Spectral)
        plt.title("Diffusion Map (%.2g sec)" % (t1diff - t0diff))
        plt.show()
```



3.1.4 t-SNE

The t-distributed stochastic neighbor embedding (t-SNE) is another dimensionality reduction algorithm which is used for non-linear models.

```
In [9]: t0sne = time()
        tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
        XSNE = tsne.fit_transform(X).T
        t1sne = time()

        plt.scatter(XSNE[0], XSNE[1], c=color, cmap=plt.cm.rainbow)
        plt.title("t-SNE (%.2g sec)" % (t1sne - t0sne))
        plt.axis('tight')

        plt.show()
```



3.2 Spike Models

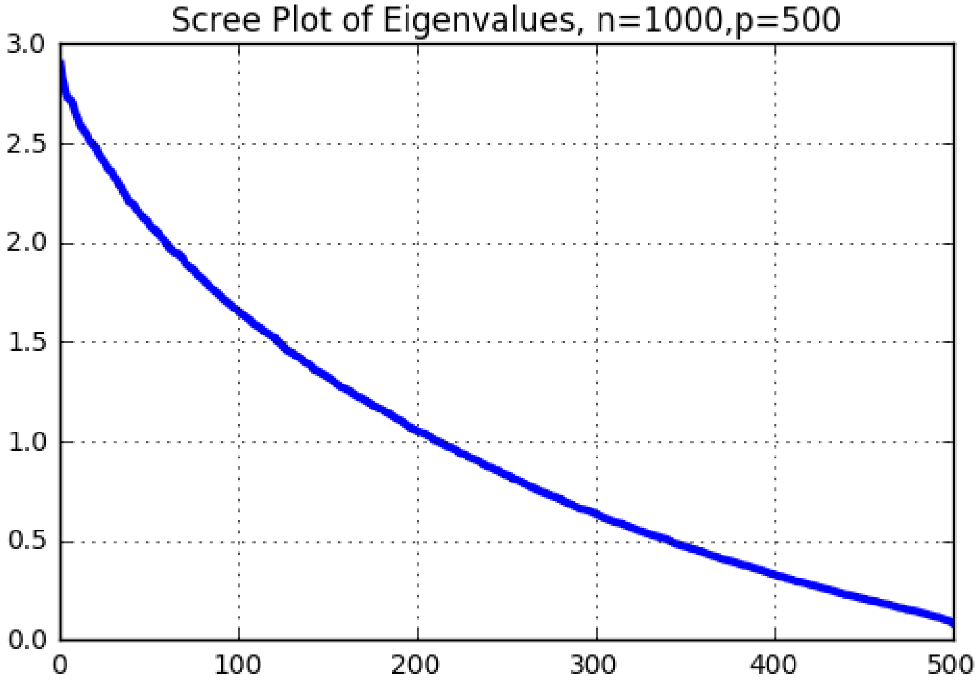
One interesting question that we can ask is whether PCA is capable of finding low-dimensional structures in presence of noise.
First assume that only noise is observed. We draw i.i.d Gaussian vector in dimension p and observe n samples from it.

```
In [10]: ## Number of vectors
         n=1000
         #######################
         # Dimension
         from scipy.stats import norm
         p=500
         X=norm.rvs(0,1,size=(p,n))
```

We see what is the spectral decomposition of the sample covariance matrix. The histogram of eigenvalues is plotted.

```
In [11]: Sn=(X@X.transpose())/n
         v, u=la.eig(Sn)
         idx = v.argsort()[::-1]    # Sorting eigenvalues
         vsorted=v[idx]

         basex=range(1,p+1)
         fig = plt.figure()
         baseline = plt.plot(basex,vsorted)
         plt.setp(baseline, 'color','b', 'linewidth', 3)
         plt.grid(True)
         plt.title('Scree Plot of Eigenvalues, n={},p={}'.format(n,p))
```

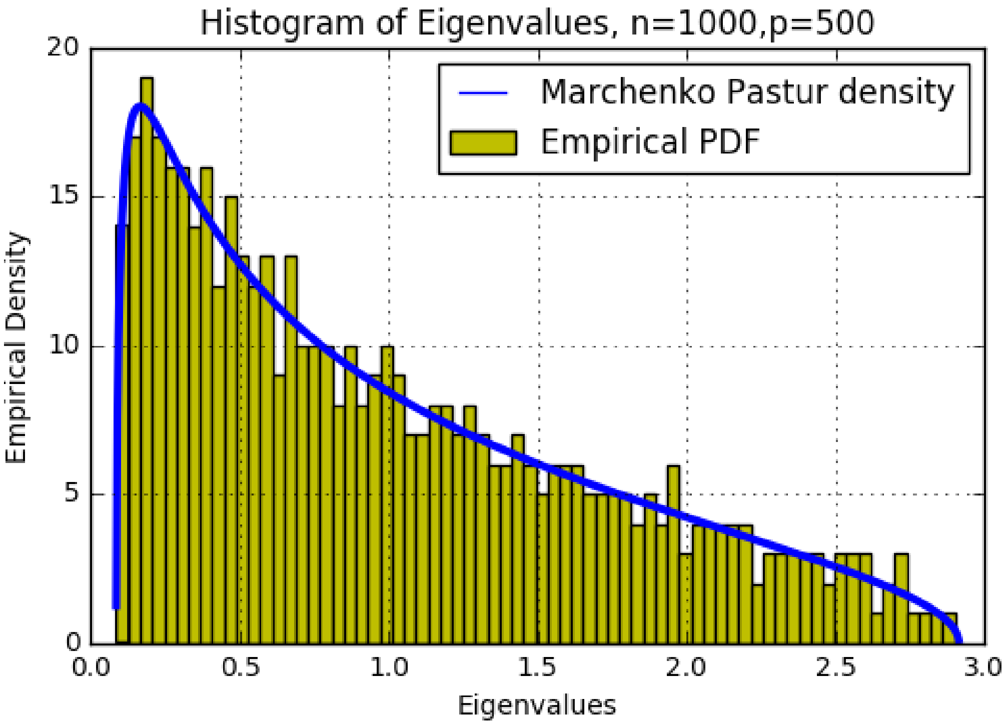
```
Out[11]: <matplotlib.text.Text at 0x7f832a62ba58>
```



Interestingly, it seems that from some eigenvalues are small and it seems that there is a lower dimensional structure here. The reason is that the number of samples is small for detecting the geometry of data. Indeed the density of eigenvalues is given by the Marchenko Pastur density function.

```
In [12]: ##################################################################
         # the Marchenko Pastur density
         ##################################################################
         def MPden(x,p,n):
             ###    Definition of the Marchenko Pastur density
             q = p/n
             lplus = 1+q+2*np.sqrt(q)
             lminus = 1+q-2*np.sqrt(q)
             return 1/(2*np.pi*x*q)*np.sqrt((lplus-x)*(x-lminus))
         ##################################################################
         binsize=20
         binnum=(p/binsize)
         binmax=int((vsorted[0]-vsorted[p-1])*binnum)
         xaxis=np.linspace(0.01,4,1000)

         plt2=plt.plot(xaxis,[MPden(x,p,n)*binsize for x in xaxis],
                  label="Marchenko Pastur density")
         ##################################################################
         # Histogram
         h = plt.hist(vsorted, bins=binmax, color='w', facecolor='y', label="Empirical PDF")
         plt.legend()
         plt.xlabel('Eigenvalues')
```

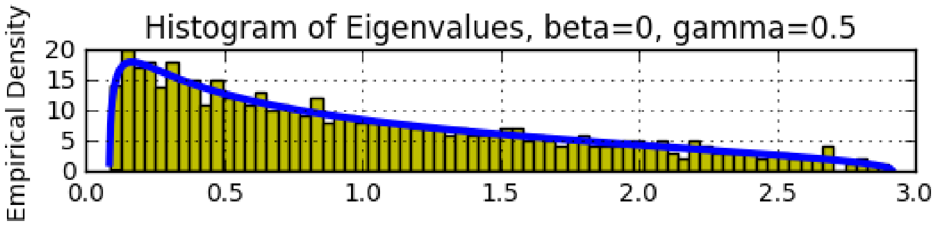
```
        plt.ylabel('Empirical Density')
        plt.title('Histogram of Eigenvalues, n={},p={}'.format(n,p))
        plt.setp(plt2, 'color','b', 'linewidth', 3)
        plt.grid(True)
        plt.show()
```

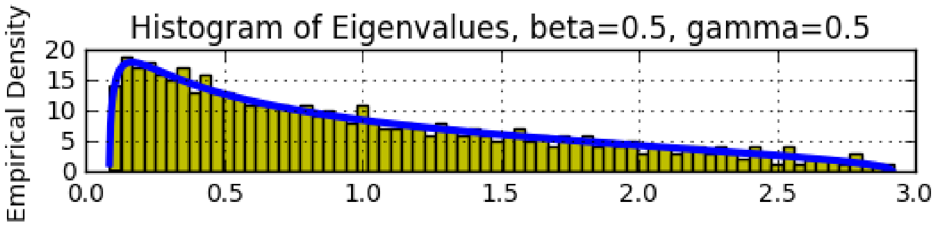


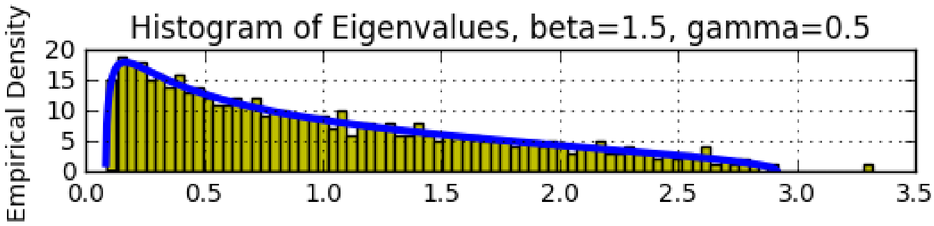
What happens when we have a lower dimensional structure here? We assume that a one-dimensional data is added to the noise. This is called the spike model. As it can be seen below, the *power* of this one-dimensional data should be high enough so that it is detected properly.

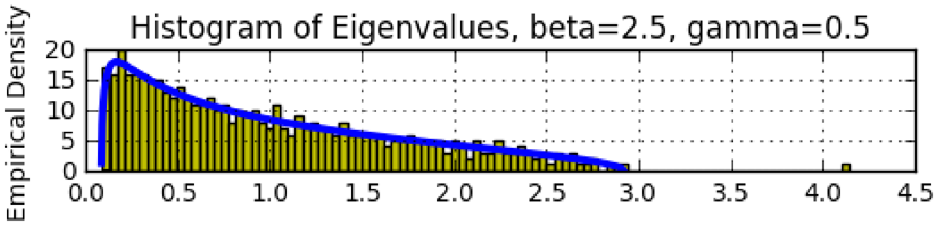
```
In [13]: from scipy.stats import multivariate_normal as mvnorm
         ######################
         # Covariance Matrix
         gamma=p/n
         Mu=np.zeros(p)
         Covect=np.array(np.append([1], np.zeros(p-1)),ndmin=2)
         Covect2=np.array(np.append([0, 1], np.zeros(p-2)),ndmin=2)
         beta2=0.5
         betavector=[0, 0.5, 1.5, 2.5]
         xaxis=np.linspace(0.01,4,1000)
         m=len(betavector)
         for beta in betavector:
             Sigma=np.eye(p)+beta*Covect.transpose()@Covect
                      +beta2*Covect2.transpose()@Covect2
             ######################
             X=mvnorm.rvs(Mu,Sigma,size=(1,n))
             Sn=(X.transpose()@X)/n
             v, u=la.eig(Sn)
```

```
idx = v.argsort()[::-1]    # Sorting eigenvalues
vsorted=v[idx]
binmax=int((vsorted[0]-vsorted[p-1])*binnum)
######################
## Plotting Densities
#######################
# Marchenko Pastur density
plt.subplot(m,1,betavector.index(beta)+1)
plt2=plt.plot(xaxis,[binsize*MPden(x,p,n) for x in xaxis],
         label="Marchenko Pastur density")
#######################
# Histogram
h = plt.hist(vsorted, bins=binmax, color='w', facecolor='y',
         label="Empirical PDF")
plt.ylabel('Empirical Density')
plt.title(r'Histogram of Eigenvalues, beta={}, gamma={}'.format(beta,gamma))
plt.setp(plt2, 'color','b', 'linewidth', 3)
plt.grid(True)
plt.show()
```









3.3 Tensorboard MNIST dataset

We first load a number of examples from MNIST dataset.

```
In [14]: import tensorflow as tf
         from tensorflow.contrib.tensorboard.plugins import projector
         from tensorflow.examples.tutorials.mnist import input_data
         TO_EMBED_COUNT = 100
         mnist = input_data.read_data_sets("tensorboard_MNIST/MNIST_data/", one_hot=False)
         batch_xs = mnist.train.images[:TO_EMBED_COUNT, :]
         batch_ys = mnist.train.labels[:TO_EMBED_COUNT]
         print(np.array(batch_xs).shape)
```

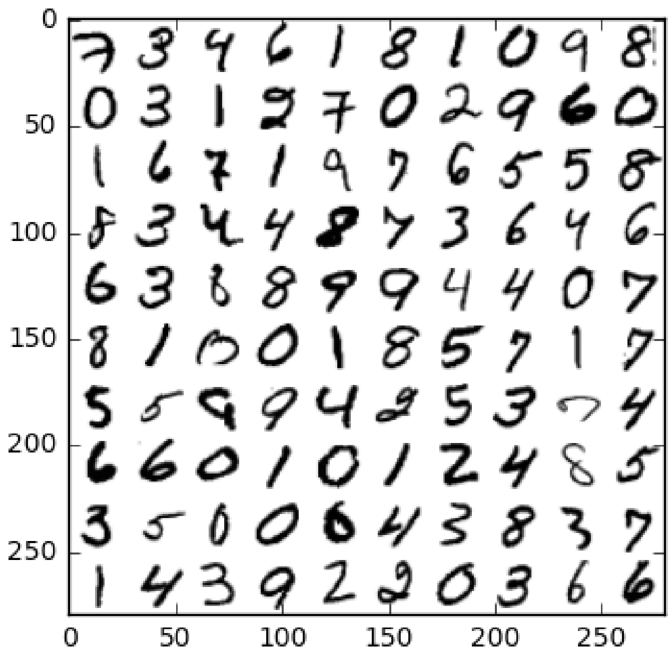
```
Extracting tensorboard_MNIST/MNIST_data/train-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/train-labels-idx1-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-labels-idx1-ubyte.gz
(100, 784)
```

These selected images can be presented as follows. This is called *sprite image*. It is a single image that contains the small image of each data point.

```
In [15]: import os
         LOG_DIR = 'tensorboard_MNIST/pca_sample'
         path_for_mnist_sprites =  os.path.join(LOG_DIR,'mnistdigits.png')

         def create_sprite_image(images):
             """Returns a sprite image consisting of images passed as argument."""
             """Images should be count x width x height."""

             if isinstance(images, list):
                 images = np.array(images)
             img_h = images.shape[1]
             img_w = images.shape[2]
             n_plots = int(np.ceil(np.sqrt(images.shape[0])))


             spriteimage = np.ones((img_h * n_plots ,img_w * n_plots ))

             for i in range(n_plots):
```

```
        for j in range(n_plots):
            this_filter = i * n_plots + j
            if this_filter < images.shape[0]:
                this_img = images[this_filter]
                spriteimage[i * img_h:(i + 1) * img_h,
                  j * img_w:(j + 1) * img_w] = this_img

    return spriteimage

def vector_to_matrix_mnist(mnist_digits):
    """Reshapes normal mnist digit (batch,28*28) to matrix (batch,28,28)"""
    return np.reshape(mnist_digits,(-1,28,28))

def invert_grayscale(mnist_digits):
    """ Makes black white, and white black """
    return 1-mnist_digits

to_visualise = batch_xs
to_visualise = vector_to_matrix_mnist(to_visualise)
to_visualise = invert_grayscale(to_visualise)

sprite_image = create_sprite_image(to_visualise)

plt.imsave(path_for_mnist_sprites,sprite_image,cmap='gray')
plt.imshow(sprite_image,cmap='gray')
```

```
Out[15]: <matplotlib.image.AxesImage at 0x7f831c3199e8>
```



3.3.1 PCA in Tensorboard

To carry out PCA, tensorboard has already a built-in option for PCA. Therefore, for the case of PCA visualization we can leave the samples within the batch as they are.

```
In [16]: Xout = np.array(batch_xs) # tensorflow does the embedding (PCA)
         ########################################
         ## visualization is prepared in PCA sample
         NAME_TO_VISUALISE_VARIABLE = "mnistembedding"
         path_for_mnist_metadata =  os.path.join(LOG_DIR,'metadata.tsv')
```

We provide the variable to visualize and the visualization directory to tensorboard.

```
In [17]: embedding_var = tf.Variable(Xout, name=NAME_TO_VISUALISE_VARIABLE)
         summary_writer = tf.summary.FileWriter(LOG_DIR)
```

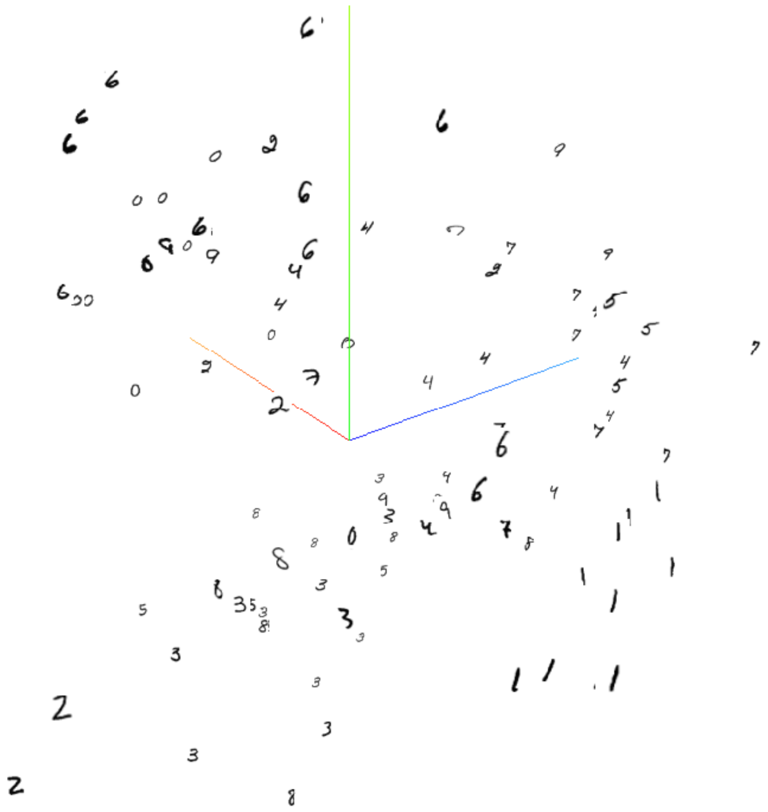
Finally, we setup the visualization files.

```
In [18]: config = projector.ProjectorConfig()
         embedding = config.embeddings.add()
         embedding.tensor_name = embedding_var.name

         # Specify where you find the metadata
         embedding.metadata_path = 'metadata.tsv'

         # Specify where you find the sprite
         embedding.sprite.image_path = 'mnistdigits.png'
         embedding.sprite.single_image_dim.extend([28,28])

         # Say that you want to visualise the embeddings
         projector.visualize_embeddings(summary_writer, config)

         sess = tf.InteractiveSession()
         sess.run(tf.global_variables_initializer())
         saver = tf.train.Saver()
         saver.save(sess, os.path.join(LOG_DIR, "model_mds.ckpt"), 1)


         with open(path_for_mnist_metadata,'w') as f:
             f.write("Index\tLabel\n")
             for index,label in enumerate(batch_ys):
                 f.write("%d\t%d\n" % (index,label))
```

Now we can go the console and start tensorboard. This is done by specifying tensorboard the directory where the visualization files are located. In linux/ubuntu systems the following command is used

```
$ tensorboard --logdir=tensorboard_MNIST/pca_sample
```

The output of the tensorboard would be in this case the following image.



4 Classification and Clustering

4.1 A Toy Example

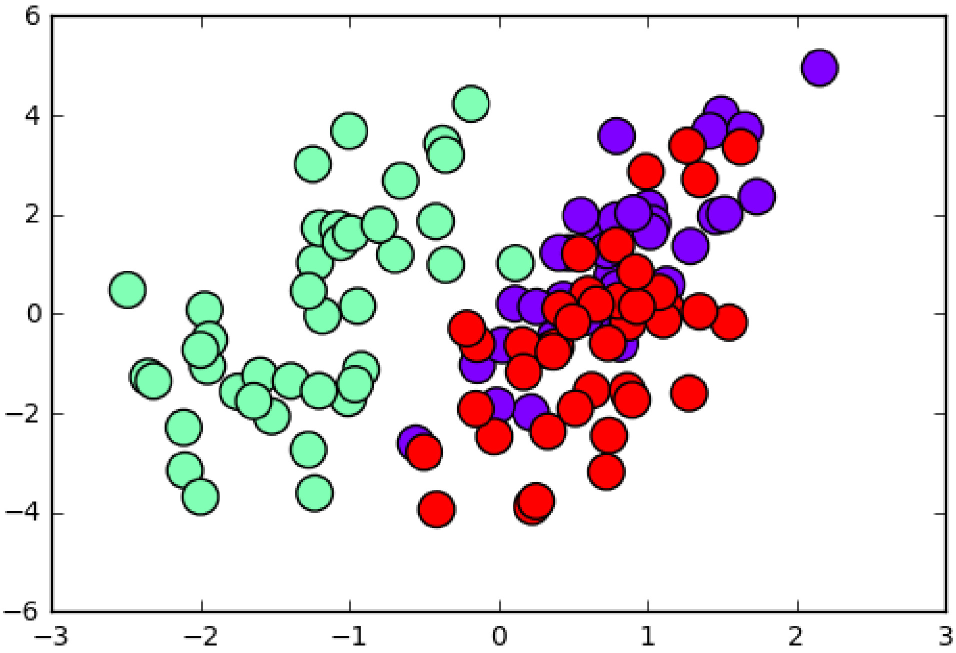
We assume that dataset is generated from different Gaussian multivariate distributions.

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

        ### Plots
        %matplotlib inline
        from matplotlib.pyplot import imshow
        import matplotlib.pyplot as plt
        import matplotlib.colors as colors
        import numpy as np
        bounds = np.linspace(-1, 3, 10)
        normcolor = colors.BoundaryNorm(boundaries=bounds, ncolors=256)
        ##########
        from scipy.stats import multivariate_normal as mvnorm
        from scipy.stats import norm
```

We first fix mean values and covariance matrices corresponding to each class as well as their labels. Three mean values are selected currently for the dataset.

```
In [2]: # Ambient dimension
        p=2
        # Number of elements in each class
        n1=40
        n2=40
        n3=40
        #### Labels
        color=np.concatenate((0*np.ones(n1,),1*np.ones(n2,),2*np.ones(n3,)),
                             axis=0)[np.newaxis]
        #### Mean values
        # Either random
        m1=norm.rvs(0,1,size=(p,))
        m2=norm.rvs(0,1,size=(p,))
        m3=norm.rvs(0,1,size=(p,))
        # Or fixed
        #m1= [1,2]
        #m2=[-2,-1]
        #m3=[-2,3]
        #### Covariance matrix
        temp= norm.rvs(0, 0.5, size=(p, p))
        Sigma1 =  temp.T@temp
        Sigma2 = Sigma1
        Sigma3 = Sigma1
```

```
#### Generate data
X1=mvnorm.rvs(m1, Sigma1, size=(n1, 1))
X2=mvnorm.rvs(m2, Sigma2, size=(n2, 1))
X3=mvnorm.rvs(m3, Sigma3, size=(n3, 1))
#### Build the dataset
X=np.concatenate((X1,X2,X3),axis=0)
Xlabeled=np.concatenate((X,color.T),axis=1)
xbar1=np.mean(X1,0)
xbar2=np.mean(X2,0)
xbar3=np.mean(X3,0)
xmean1=(xbar1+xbar2)/2
xmean2=(xbar1+xbar3)/2
xmean3=(xbar2+xbar3)/2
## Plot
fig = plt.figure()
plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
plt.show()
```



Now the data is ready for further processing.

4.2 Two-class classification

First, two classes are selected for classification. In that regard, Fisher's linear discriminant analysis (Fisher LDA) provides the same classifier as the maximum likelihood (ML) classifier. We first prepare the dataset.

```
In [3]: X12=np.concatenate((X1,X2),axis=0)
        y12=np.concatenate((0*np.ones(n1,),1*np.ones(n2,)),axis=0)[np.newaxis]
```

Before continuing, the centering matrix $\mathbf{E}_n$ will be used to simplify the whole process.

```
In [4]: #### Centering Matrix
        def centering(n):
            return np.eye(n)-np.ones((n,n))/n
```

Furthermore the following function provides the Fisher LDA function.

```
In [5]: def FisherLDA(a,mean_vector,Xinput):
            n_class=mean_vector.shape[0]
            Xtemp=np.tile(Xinput@a,(n_class,1))
            return np.argmin(np.abs(Xtemp.T-mean_vector@a),axis=1)
```

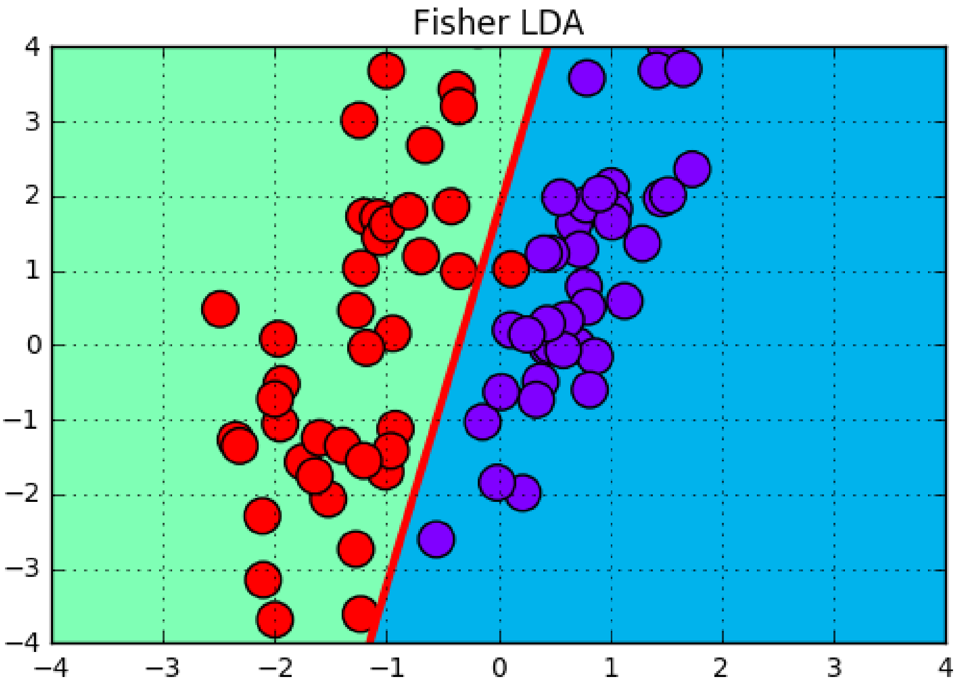
The following building block implement the Fisher's linear discriminant analysis.

```
In [6]: from numpy import linalg as la

        ########################################
        #### Fisher Linear Discriminant Analysis
        ########################################
        E=centering(n1+n2)
        E1=centering(n1)
        E2=centering(n2)
        W=X1.T@E1@X1+X2.T@E2@X2
        S=X12.T@E@X12
        FLDAmat=la.inv(W)@S
        #### Eigenvalue analysis
        ## Find the eigenvalues and eigenvectors
        eigval, eigvec =la.eig(FLDAmat)
        ## Sort them in decreasing order
        idx = eigval.argsort()[::-1]
        eigval = eigval[idx]
        eigvec = eigvec[:,idx]
        a=eigvec[:,0]
```

The output of the classifier is presented as follows.

```
In [7]: fig=plt.figure()
        ###################################
        # Meshgrid of the classifier:
        x_min = -4
        x_max = 4
        y_min=-4
        y_max=4
        nx, ny = 400, 200
        xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                             np.linspace(y_min, y_max, ny))
        Xmesh=np.array(np.c_[xx.ravel(), yy.ravel()])
        Ymesh=FisherLDA(a,np.array([xbar1,xbar2]),Xmesh).reshape(xx.shape)
        plt.pcolormesh(xx, yy, Ymesh, norm=normcolor, cmap=plt.cm.rainbow)
        ###################################
        plt.scatter(X12[:,0], X12[:,1], s=180, c=y12, cmap=plt.cm.rainbow)
        w,z=np.ogrid[-4:4:100j,-4:4:100j]
        xmean1
        g=a[1]*w+a[0]*z-a.T@xmean1
```

```
plt.contour(w.ravel(),z.ravel(),g,[0],linewidths=(3,), colors=('r',))
plt.title('Fisher LDA')
plt.grid(True)
plt.legend()
plt.axis([-4, 4, -4, 4])
plt.show()
```



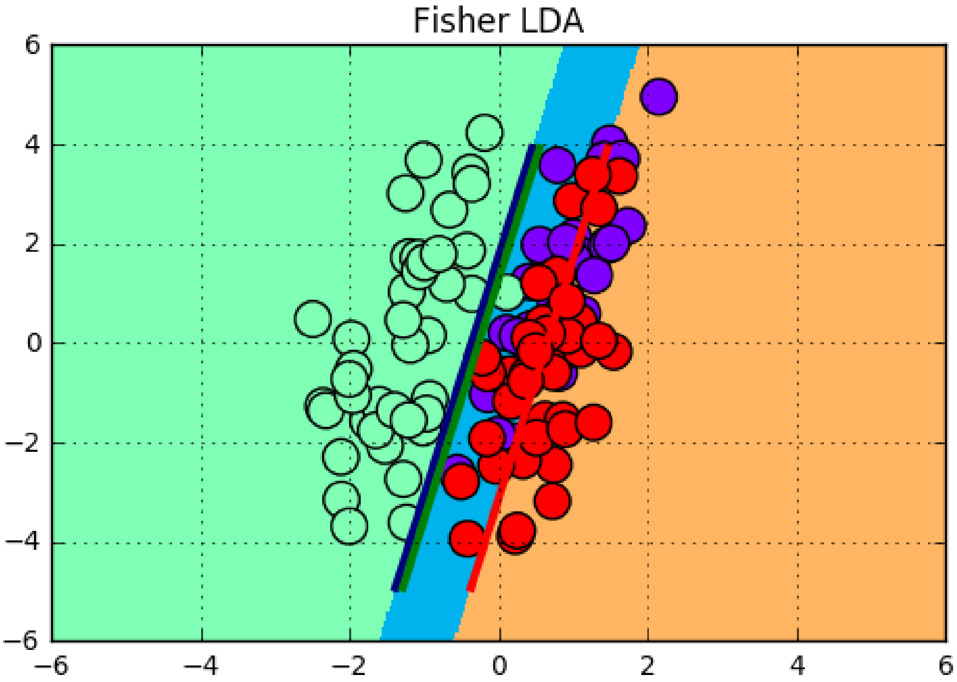
4.3 Fisher's LDA versus Maximum Likelihood

Now consider three classes for classification. Note that ML and Fisher LDA will have different outputs here. We start by training Fisher LDA model.

```
In [8]: ##########################################
        #### Fisher Linear Discriminant Analysis
        n=n1+n2+n3
        E=centering(n)
        E1=centering(n1)
        E2=centering(n2)
        E3=centering(n3)
        W=X1.T@E1@X1+X2.T@E2@X2+X3.T@E3@X3
        S=X.T@E@X
        FLDAmat=la.inv(W)@S
        #### Eigenvalue analysis
        ## Find the eigenvalues and eigenvectors
        eigval, eigvec =la.eig(FLDAmat)
        ## Sort them in decreasing order
        idx = eigval.argsort()[::-1]
        eigval = eigval[idx]
        eigvec = eigvec[:,idx]
        a=eigvec[:,0]
```

The output of the classifier is plotted in the following block.

```
In [9]: fig = plt.figure()
        ###################################
        # Meshgrid of the classifier:
        x_min = -6
        x_max = 6
        y_min=-6
        y_max=6
        nx, ny = 400, 200
        xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                             np.linspace(y_min, y_max, ny))
        Xmesh=np.array(np.c_[xx.ravel(), yy.ravel()])
        Ymesh=FisherLDA(a,np.array([xbar1,xbar2,xbar3]),Xmesh).reshape(xx.shape)

        plt.pcolormesh(xx, yy, Ymesh, norm=normcolor, cmap=plt.cm.rainbow)
        #######################################
        plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
        w,z=np.ogrid[-5:4:100j,-5:4:100j]
        g=a[1]*w+a[0]*z
        baseline=plt.contour(w.ravel(),z.ravel(),g-a.T@xmean1,[0],linewidths=(3,))
        plt.contour(w.ravel(),z.ravel(),g-a.T@xmean2,[0],linewidths=(3,),
                    colors=('r',))
        plt.contour(w.ravel(),z.ravel(),g-a.T@xmean3,[0],linewidths=(3,),
                    colors=('g',))
        plt.title('Fisher LDA')
        plt.grid(True)
        plt.legend()
        plt.axis([-6, 6, -6, 6])

        plt.show()
```

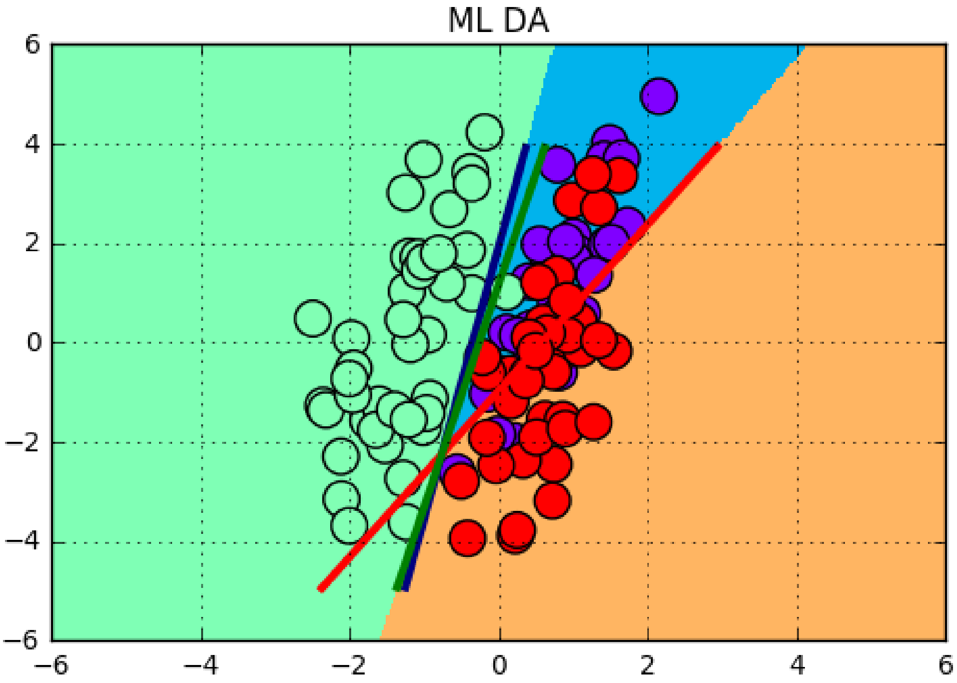


Next step is to train ML classifier. The following block implements the ML discriminant analysis.

```
In [10]: ##############################################
         #### ML Discriminant Analysis
         Sigma =W/n
         d1=xbar1-xbar2
         d2=xbar1-xbar3
         d3=xbar2-xbar3
         a1=d1@la.inv(Sigma)
         a2=d2@la.inv(Sigma)
         a3=d3@la.inv(Sigma)
         ##############################################
         ## ML classifier
         import scipy.linalg as sla

         def MLDA(S,mean_vector,Xinput):
             n_class=mean_vector.shape[0]
             n_dimension=mean_vector.shape[1]
             n_instance=Xinput.shape[0]
             Xtemp0=np.tile(Xinput,(1,n_class))-mean_vector.ravel()
             Xtemp1=Xtemp0.reshape((n_instance*n_class,n_dimension)).T
             Xtemp2=sla.sqrtm(la.inv(S))@Xtemp1
             Xtemp3= la.norm(Xtemp2,axis=0)
             Xtemp4=Xtemp3.reshape((n_instance,n_class))
             return np.argmin(Xtemp4,axis=1)
         ##############################################
```

The output of ML classifier is ploted below.

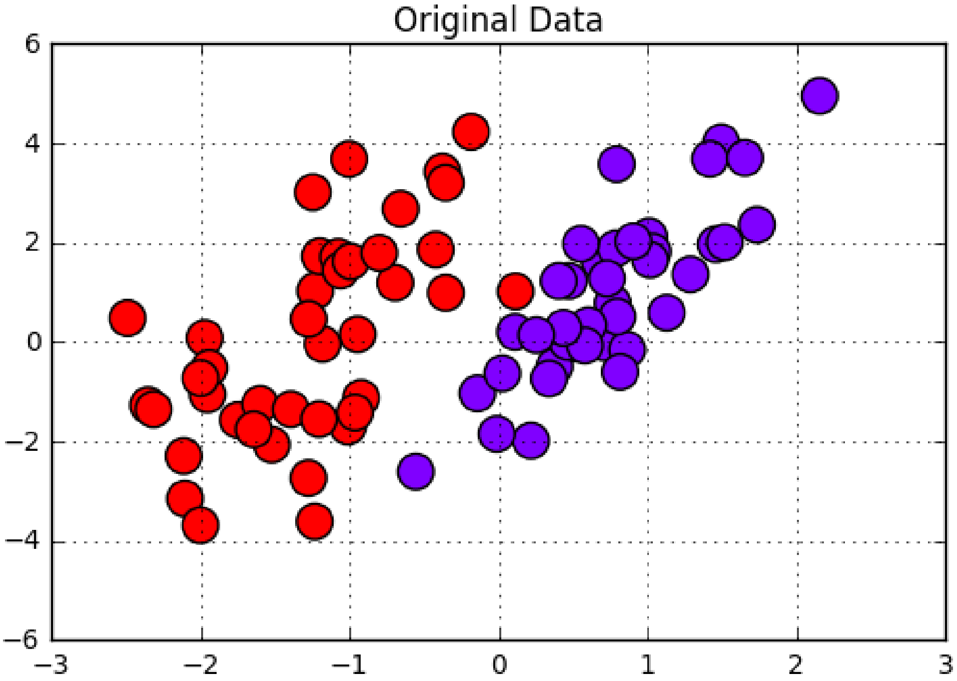
```
In [11]: fig=plt.figure()
         ####################################
         # Meshgrid of the classifier:
         x_min = -6
         x_max = 6
         y_min=-6
         y_max=6
         nx, ny = 400, 200
         xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                              np.linspace(y_min, y_max, ny))
         Xmesh=np.array(np.c_[xx.ravel(), yy.ravel()])
         Ymesh=MLDA(Sigma,np.array([xbar1,xbar2,xbar3]),Xmesh).reshape(xx.shape)
         plt.pcolormesh(xx, yy, Ymesh, norm=normcolor, cmap=plt.cm.rainbow)
         #######################################
         plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
         w,z=np.ogrid[-5:4:100j,-5:4:100j]
         g=a1[1]*w+a1[0]*z
         plt.contour(w.ravel(),z.ravel(),a1[1]*w+a1[0]*z-a1.T@xmean1,[0]
                     ,linewidths=(3,))
         plt.contour(w.ravel(),z.ravel(),a2[1]*w+a2[0]*z-a2.T@xmean2,[0]
                     ,linewidths=(3,), colors=('r',))
         plt.contour(w.ravel(),z.ravel(),a3[1]*w+a3[0]*z-a3.T@xmean3,[0]
                     ,linewidths=(3,), colors=('g',))
         plt.title('ML DA')
         plt.grid(True)
         plt.legend()
         plt.axis([-6, 6, -6, 6])
         plt.show()
```

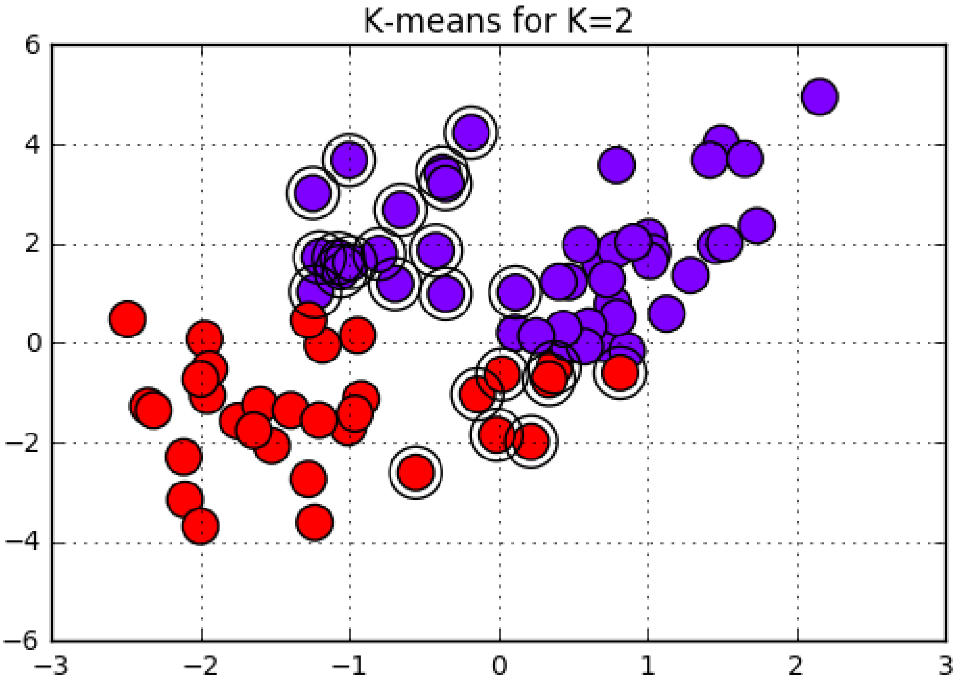


4.4 K-Means Clustering

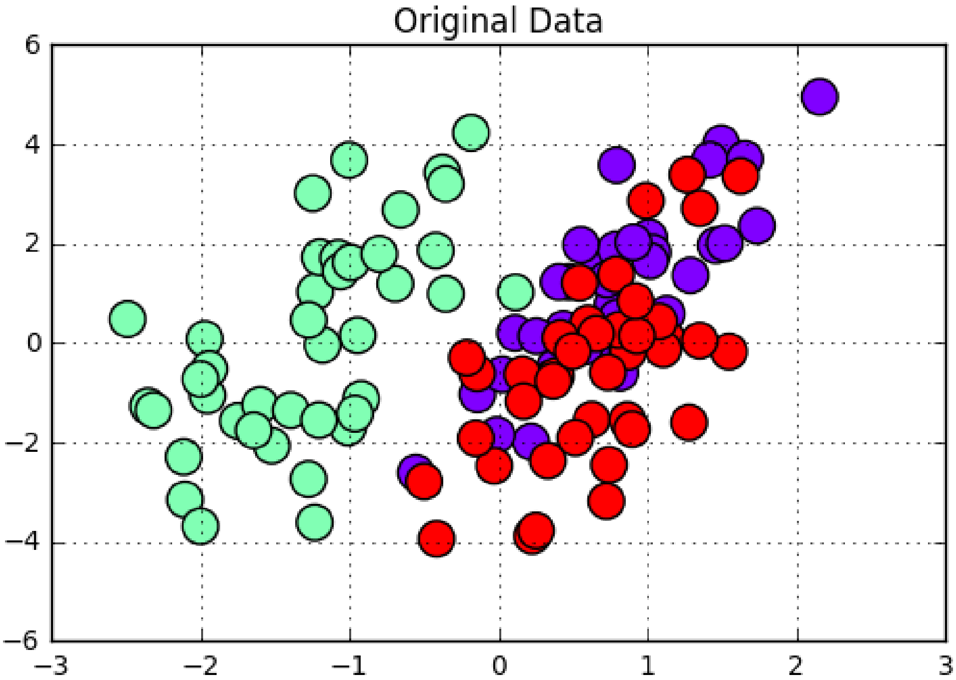
We simply run K-meanse clustering algorithms for two and three clusters.

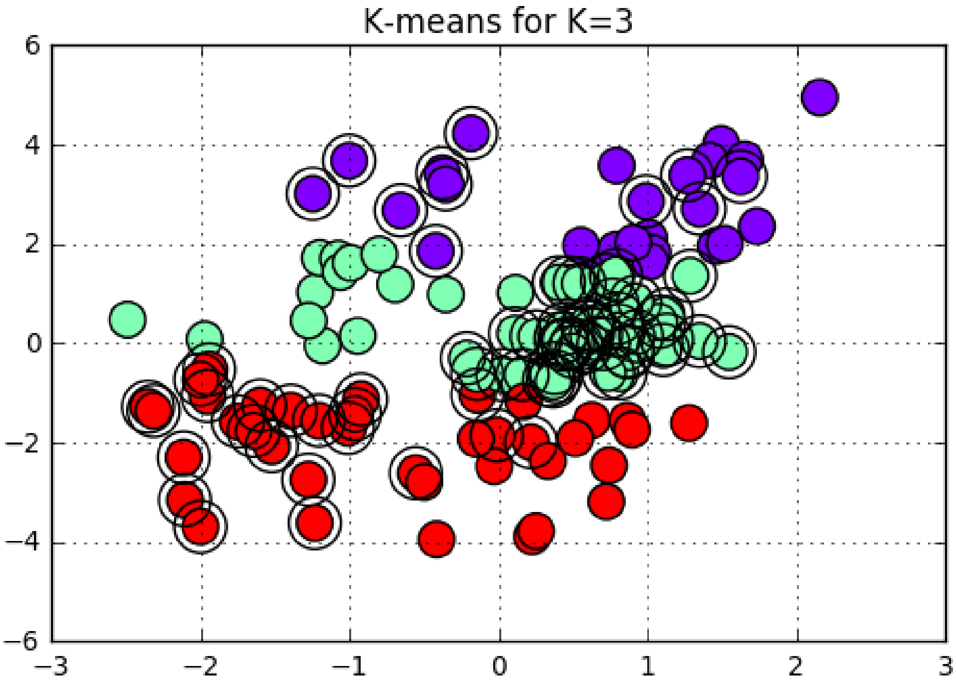
```
In [12]: from sklearn.cluster import KMeans
         kmeans=KMeans(n_clusters=2, random_state=0).fit(X12)
         #################################################
         ## Detecting Correct Labels
         ykmeans=kmeans.labels_
         I=[i for i in range(n1+n2) if (y12[0,i]!=ykmeans[i])]
         Xfalse=X12[I]
         #################################################
         ## Plot
         fig = plt.figure()
         plt.scatter(X12[:,0], X12[:,1], s=180, c=y12, cmap=plt.cm.rainbow)
         plt.title('Original Data')
         plt.grid(True)
         plt.legend()
         plt.show()
         ###########
         fig = plt.figure()
         plt.scatter(X12[:,0], X12[:,1], s=180, c=ykmeans,
                     cmap=plt.cm.rainbow)
         plt.scatter(Xfalse[:,0], Xfalse[:,1], s=380, facecolors='none')

         plt.title('K-means for K=2')
         plt.grid(True)
         plt.legend()
         plt.show()
```





```
In [13]: from sklearn.cluster import KMeans
         kmeans=KMeans(n_clusters=3, random_state=0).fit(X)
         #################################################
         ## Detecting correct labels
         ytemp=kmeans.labels_-1
         ykmeans=[2 if ytemp[i]==-1 else ytemp[i] for i in range(n)]
         I=[i for i in range(n) if (color[0,i]!=ykmeans[i])]
         Xfalse=X[I]
         #################################################
         ## Plot
         fig = plt.figure()
         plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
         plt.title('Original Data')
         plt.grid(True)
         plt.legend()
         plt.show()
         ###########
         fig = plt.figure()
         plt.scatter(X[:,0], X[:,1], s=180, c=ykmeans, cmap=plt.cm.rainbow)
         plt.scatter(Xfalse[:,0], Xfalse[:,1], s=380, facecolors='none')

         plt.title('K-means for K=3')
         plt.grid(True)
         plt.legend()

         plt.show()
```





4.5 Discriminant Analysis for MNIST dataset

In this script, we apply Fisher's linear discriminant analysis for classification of MNIST dataset. The dataset is first loaded.

```
In [14]: ## Loading the data
         import tensorflow as tf
         from tensorflow.examples.tutorials.mnist import input_data
         data = input_data.read_data_sets("tensorboard_MNIST/MNIST_data/"
                                          , one_hot=True)
```

```
Extracting tensorboard_MNIST/MNIST_data/train-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/train-labels-idx1-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-labels-idx1-ubyte.gz
```

After loading the dataset, training and test sets, we choose N entry for the training and Ntest for the evaluation part. We would like to evaluate the effect of training set size and test set size on the performance.

```
In [15]: ## Training set
         N=100
         Xtraining = data.train.images[0:N]
         Ytraining = data.train.labels[0:N].argmax(axis=1)


         ## Test set
         Ntest=100
         Xtest=data.test.images[0:Ntest]
         Ytest=data.test.labels[0:Ntest].argmax(axis=1)
```

Here only the binary classification problem is considered. Two classes are chosen accordingly with labels C_i and C_j.

```
In [16]: Ci=2
         Cj=5

         ## Choosing respective classes from the training set
         Ind12=np.array([ind for ind in range(N) if ((Ytraining[ind]==Ci)
                                                     or (Ytraining[ind]==Cj))])
         X12=Xtraining[Ind12]
         Y12=Ytraining[Ind12]
         print("The size of the trainging set with two classes is given by:", len(Ind12))
         N12=len(Ind12)
         ## Choosing respective classes from the test set
         Ind12=np.array([ind for ind in range(Ntest) if ((Ytest[ind]==Ci)
                                                         or (Ytest[ind]==Cj))])
         Xtest12=Xtest[Ind12]
         Ytest12=Ytest[Ind12]
         print("The size of the test set with two classes is given by:", len(Ind12))
         N12test=len(Ind12)
```

```
The size of the trainging set with two classes is given by: 14
The size of the test set with two classes is given by: 15
```

Now we fit the linear discriminant analysis (LDA) to this model. First, LDA is applied to the training set. The training error reperesents how well LDA can separate two classes. The performance of LDA is evaluated on the test set. The test error shows the generalization property of LDA.

```
In [17]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
         model = LDA()
         model.fit(X12, Y12)
         TrainingError=np.count_nonzero(np.array(model.predict(X12))-Y12)/N12
         print("The misclassification error for the training set is given by:"
               ,TrainingError)
         TestError=np.count_nonzero(np.array(model.predict(Xtest12))-Ytest12)/N12test
         print("The misclassification error for the test set is given by:"
               ,TestError)
```

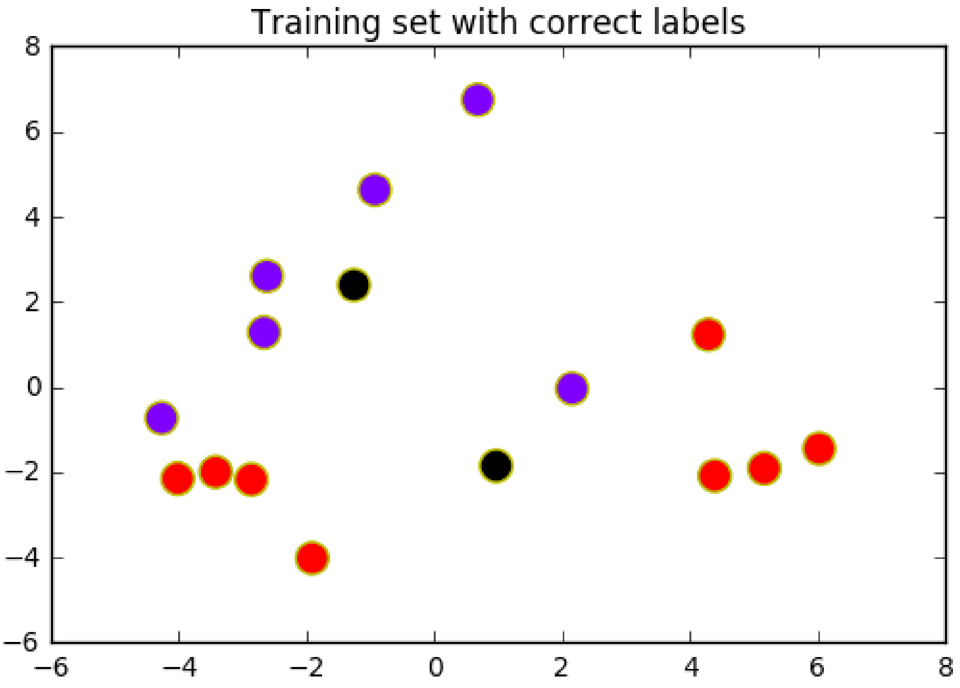
```
The misclassification error for the training set is given by: 0.07142857142857142
The misclassification error for the test set is given by: 0.3333333333333333
```

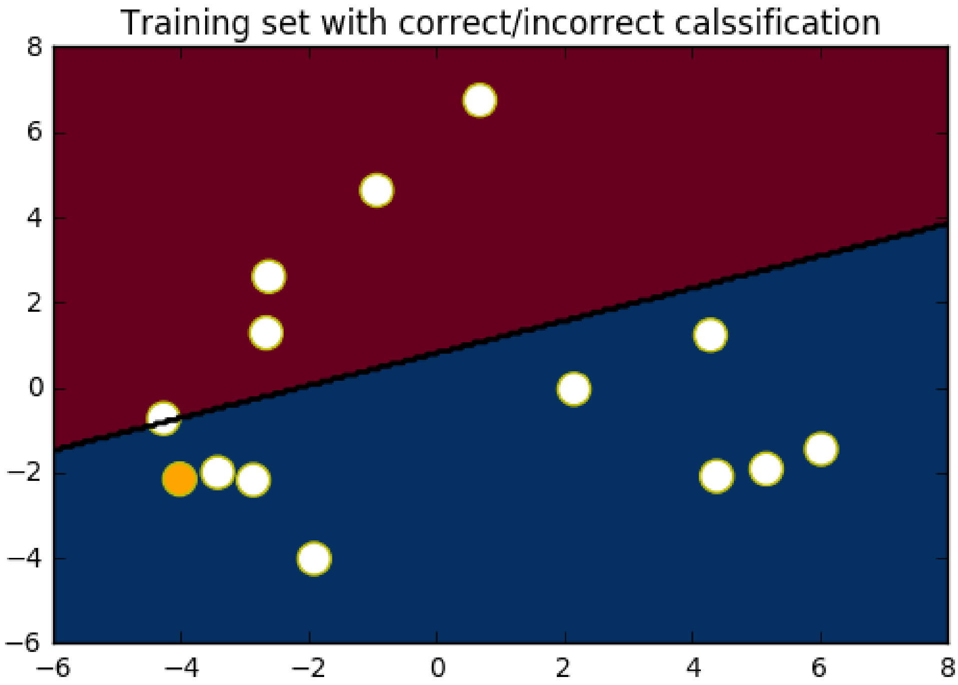
4.6 Visualizing LDA

Using PCA, we project the data in two dimensional space and then visualize the performance of LDA. Since the classifier is linear the output of PCA in two dimensional space is a line.

```
In [18]: ############################################
         #### Loading PCA
         from sklearn.decomposition import PCA
         ## Fitting the model to the data
         Xpca = PCA(n_components=2).fit_transform(X12)
         pca=PCA(n_components=2)
         Xpca=pca.fit_transform(X12)
         #############################################
         #### Plotting the output
         import matplotlib.pyplot as plt
         fig = plt.figure()
         ## Plotting the training data with the correct labels
         plt.scatter(Xpca[:, 0], Xpca[:, 1],  s=150, c=Y12,
                     cmap=plt.cm.rainbow, edgecolor='y')
         ## Plotting the mean values per each class
         Xmean=pca.transform(model.means_)
         plt.scatter(Xmean[:, 0], Xmean[:, 1],  s=150, c='black',
                     cmap=plt.cm.rainbow, edgecolor='y')
         plt.title("Training set with correct labels")
         x_min, x_max = plt.xlim()
         y_min, y_max = plt.ylim()
         plt.show()
         ##############################################
         #### Classification partition of the space
         nx, ny = 400, 200
         xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                              np.linspace(y_min, y_max, ny))
         Xmesh=pca.inverse_transform(np.c_[xx.ravel(), yy.ravel()])
```

```
Ymesh=model.predict(Xmesh).reshape(xx.shape)
plt.pcolormesh(xx, yy, Ymesh, cmap=plt.cm.RdBu)
plt.contour(xx, yy, Ymesh, [0.5*Ci+0.5*Cj], linewidths=2.
            , colors='k')
###############################################
#### Error for the training set
ErrorColor=['white' if (i==0)  else 'orange'
            for i in np.absolute(model.predict(X12)-Y12)]
plt.scatter(Xpca[:, 0], Xpca[:, 1],  s=150, c=ErrorColor,
            cmap=plt.cm.rainbow, edgecolor='y')

plt.title("Training set with correct/incorrect calssification")
# plt.axis('tight')
plt.xlim(x_min,x_max)
plt.ylim(y_min,y_max)

plt.show()
```





4.7 Three-class classification

In this part, we repeat the experiments for three different classes. The linear discriminant analysis should naturally extend to multi-class cases.

```
In [19]: Ck=7
         ## Choosing respective classes from the training set
         Ind12=np.array([ind for ind in range(N) if ((Ytraining[ind]==Ci)
                                                     or (Ytraining[ind]==Cj)
                                                     or (Ytraining[ind]==Ck))])
         X12=Xtraining[Ind12]
         Y12=Ytraining[Ind12]
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("The size of the trainging set with three classes is given by:", len(Ind12))
         N12=len(Ind12)
         ## Choosing respective classes from the test set
         Ind12=np.array([ind for ind in range(Ntest) if ((Ytest[ind]==Ci)
                                                         or (Ytest[ind]==Cj)
                                                         or (Ytest[ind]==Ck))])
         Xtest12=Xtest[Ind12]
         Ytest12=Ytest[Ind12]
         print("The size of the test set with three classes is given by:", len(Ind12))
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         N12test=len(Ind12)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
The size of the trainging set with three classes is given by: 24
The size of the test set with three classes is given by: 30
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

We load LDA model once again and fit it to the new data. The performance is measured afterward.

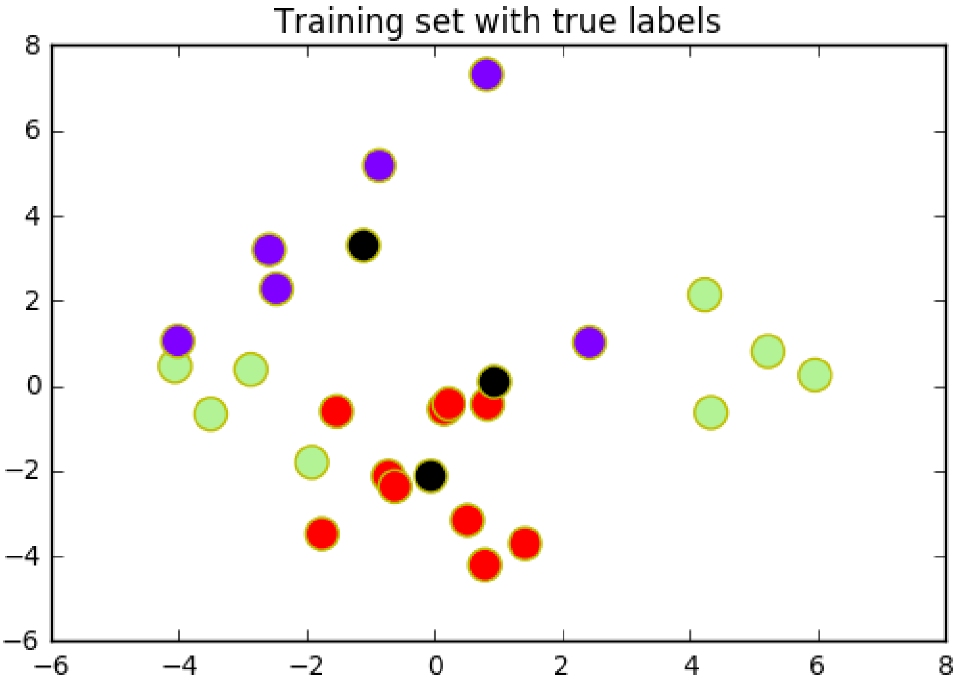
```
In [20]: modelII = LDA()
         modelII.fit(X12, Y12)
         TrainingError=np.count_nonzero(np.array(modelII.predict(X12))-Y12)/N12
         TestError=np.count_nonzero(np.array(modelII.predict(Xtest12))-Ytest12)/N12test
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("The misclassification error for the training set is given by:",TrainingError)
         print("The misclassification error for the test set is given by:",TestError)
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
```

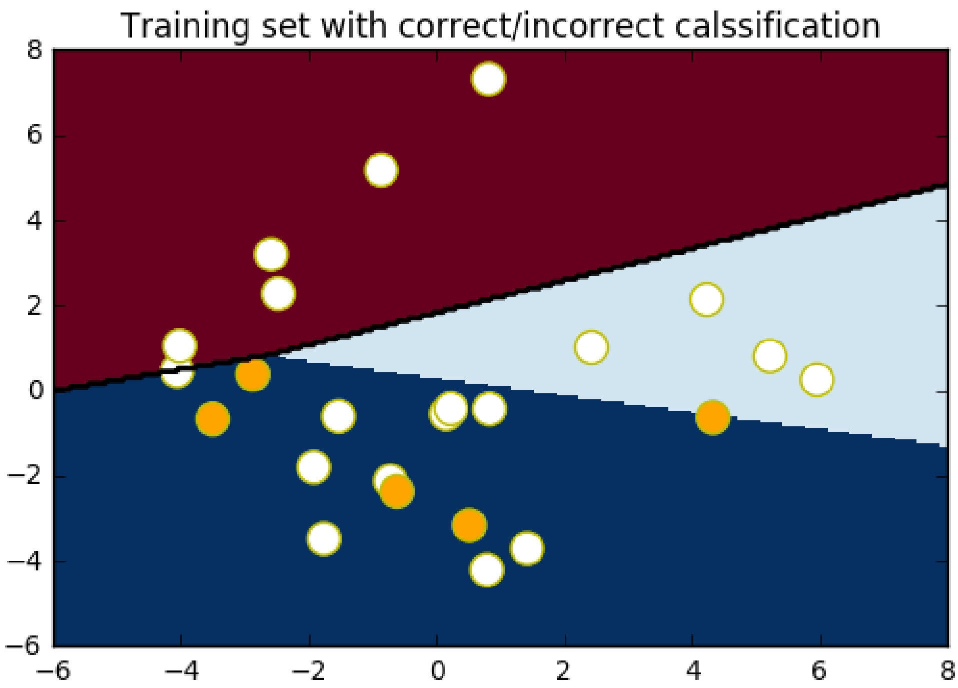
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The misclassification error for the training set is given by: 0.20833333333333334
The misclassification error for the test set is given by: 0.36666666666666664
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Finally the output is again represented in two-dimensional space using PCA.

```
In [21]: Xpca = PCA(n_components=2).fit_transform(X12)
         pca=PCA(n_components=2)
         Xpca=pca.fit_transform(X12)
         ###############################################################
         ###############################################################
         ### Plotting
         import matplotlib.pyplot as plt
         fig = plt.figure()
         ##############################
         # Plot 1
         ##############################
         plt.scatter(Xpca[:, 0], Xpca[:, 1],  s=150, c=Y12,
                     cmap=plt.cm.rainbow, edgecolor='y')
         plt.title("Training set with true labels")
         Xmean=pca.transform(modelII.means_)
         plt.scatter(Xmean[:, 0], Xmean[:, 1],  s=150, c='black',
                     cmap=plt.cm.rainbow, edgecolor='y')
         x_min, x_max = plt.xlim()
         y_min, y_max = plt.ylim()
         plt.show()
         ##############################
         # Plot 2
```

```
##############################
###################
## Creating the mesh
nx, ny = 400, 200
xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                     np.linspace(y_min, y_max, ny))
Xmesh=pca.inverse_transform(np.c_[xx.ravel(), yy.ravel()])
Ymesh=modelII.predict(Xmesh).reshape(xx.shape)
plt.pcolormesh(xx, yy, Ymesh, cmap=plt.cm.RdBu)
plt.contour(xx, yy, Ymesh, [0.5*Ci+0.5*Cj], linewidths=2., colors='k')
###################
ErrorColor=['white' if (i==0)  else 'orange'
            for i in np.absolute(modelII.predict(X12)-Y12)]
plt.scatter(Xpca[:, 0], Xpca[:, 1],  s=150, c=ErrorColor,
            cmap=plt.cm.rainbow, edgecolor='y')
plt.title("Training set with correct/incorrect calssification")
plt.xlim(x_min,x_max)
plt.ylim(y_min,y_max)
#######################################################################
#######################################################################
plt.show()
```





4.8 Multi-class classification

In this part, we just apply Fisher's linear discriminant analysis to multi-class classification. We choose the whole training set for training phase and the algorithm is evaluated on the whole MNIST test set. One can achieve 12.86% error on the training set and 12.7% error on the test set.

```
In [22]: ##############################################################
         ## Training set
         N=55000
         Xtraining = data.train.images
         Ytraining = data.train.labels.argmax(axis=1)


         ## Test set
         Ntest=10000
         Xtest=data.test.images
         Ytest=data.test.labels.argmax(axis=1)



         ##############################################################
         ##############################################################
         modelIII = LDA()
         modelIII.fit(Xtraining, Ytraining)
         TrainingError=np.count_nonzero(np.array(modelIII.predict(Xtraining))-Ytraining)/N
         TestError=np.count_nonzero(np.array(modelIII.predict(Xtest))-Ytest)/Ntest
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
```

```
        print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
        print("The misclassification error for the training set is given by:",TrainingError)
        print("The misclassification error for the test set is given by:",TestError)
        print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
        print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The misclassification error for the training set is given by: 0.12818181818181817
The misclassification error for the test set is given by: 0.1264
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5 Support Vector Machines

5.1 Primal Problem - Linearly Separable Data

We again generate our toy example using multivariate Gaussian distribution.

```
In [1]: import warnings
        warnings.filterwarnings('ignore')
        import numpy as np
        from scipy.stats import multivariate_normal as mvnorm
        # Number of classes
        c=2
        # Number of elements in each class
        n1=300
        n2=300
        n=n1+n2
        color=np.array(np.r_[[0]*n1,[1]*n2])[np.newaxis]
        # Ambient dimension
        p=2
        ########################################################
        # Generate randomly two centers and a covariance matrix
        datamean=10
        m1=np.array([datamean,datamean])
        m2=np.array([-datamean,-datamean])
        Sigma1 =10*np.eye(p)
        Sigma2 = Sigma1
        X1=mvnorm.rvs(m1, Sigma1, size=(n1, 1))
        X2=mvnorm.rvs(m2, Sigma2, size=(n2, 1))
        ########################################################
        X=np.r_[X1,X2]
        Xlabeled=np.r_['1',X,color.T]
```

The next step is to formulate the quadratic optimization problem. We use cvxopt package to solve the optimization problem.

```
In [2]: from cvxopt import matrix
        import cvxopt.solvers as solvers
        from time import time
        ########################################################
        ### Solving the quadratic program
        QQP=matrix(np.diag([1]*p+[0]),tc='d')
        pQP=matrix([0]*(p+1),tc='d')
        yi=2*color-1
        GQP=matrix(-1*np.r_['1',np.diag(yi.reshape(n,))@X,yi.T],tc='d')
        hQP=matrix([-1]*(n),tc='d')
        t0svm = time()
        sol=solvers.qp(QQP, pQP, GQP, hQP)
        t1svm = time()
        w=np.array(sol['x'])
```

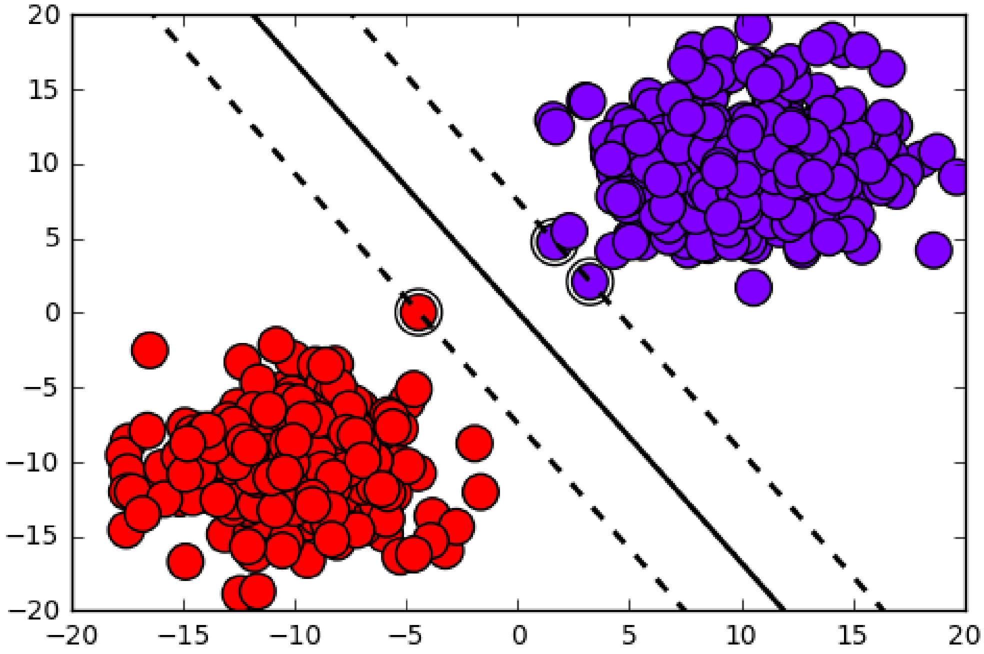
```
     pcost       dcost       gap    pres   dres
 0:  2.2356e-03  7.2186e+01  2e+03  2e+00  1e+04
 1:  1.3404e-02 -3.4628e+02  5e+02  5e-01  4e+03
 2:  3.3328e-02 -2.6787e+02  3e+02  2e-01  2e+03
 3:  4.3287e-02 -8.3140e+01  9e+01  6e-02  5e+02
 4:  4.7137e-02 -1.6724e+00  2e+00  1e-03  1e+01
 5:  4.3253e-02 -1.5803e-01  2e-01  1e-04  1e+00
 6:  4.0408e-02 -8.5850e-02  1e-01  8e-05  6e-01
 7:  4.1248e-02 -6.2281e-02  1e-01  5e-05  3e-01
 8:  3.5315e-02  1.3429e-02  2e-02  6e-16  3e-16
 9:  3.8186e-02  2.3041e-02  2e-02  6e-16  1e-16
10:  3.5065e-02  3.0906e-02  4e-03  7e-16  5e-16
11:  3.4212e-02  3.4113e-02  1e-04  6e-16  2e-15
12:  3.4187e-02  3.4186e-02  1e-06  6e-16  9e-15
13:  3.4186e-02  3.4186e-02  1e-08  7e-16  5e-15
Optimal solution found.
```

The next step is to find the parameters of the classifier.

```
In [3]: ###############################
        ## Ploting the hyperplane
        aVec=np.array([w[0],w[1]])
        a = -w[0] / w[1]
        xx = np.linspace(-20, 20)
        yy = a * xx - (w[2]) / w[1]
        ## Margins
        yy_up = a * xx - (w[2]) / w[1]+1/w[1]
        yy_down = a * xx - (w[2]) / w[1]-1/w[1]
        ## Finding the support vectors
        upperSV=np.array([X[i] for i in range(n)
                          if (np.abs(aVec.T@X[i]+w[2]-yi.T[i])<=0.001)])
```

The output of the classifier is plotted as follows. The support vectors are encircled.

```
In [4]: import matplotlib.pyplot as plt
        fig = plt.figure()

        # plot the line, the points, and the nearest vectors to the plane
        plt.plot(xx, yy, 'k-', linewidth=2)
        plt.plot(xx, yy_down, 'k--', linewidth=2)
        plt.plot(xx, yy_up, 'k--', linewidth=2)
        plt.scatter(upperSV[:, 0], upperSV[:, 1], s=300, facecolors='none')
        plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
        plt.axis([-20, 20, -20, 20])
        plt.show()
```



Note that the data is linearly separable. Otherwise the optimization problem would not have any solution.

5.2 Dual Problem- Linearly Separable Data

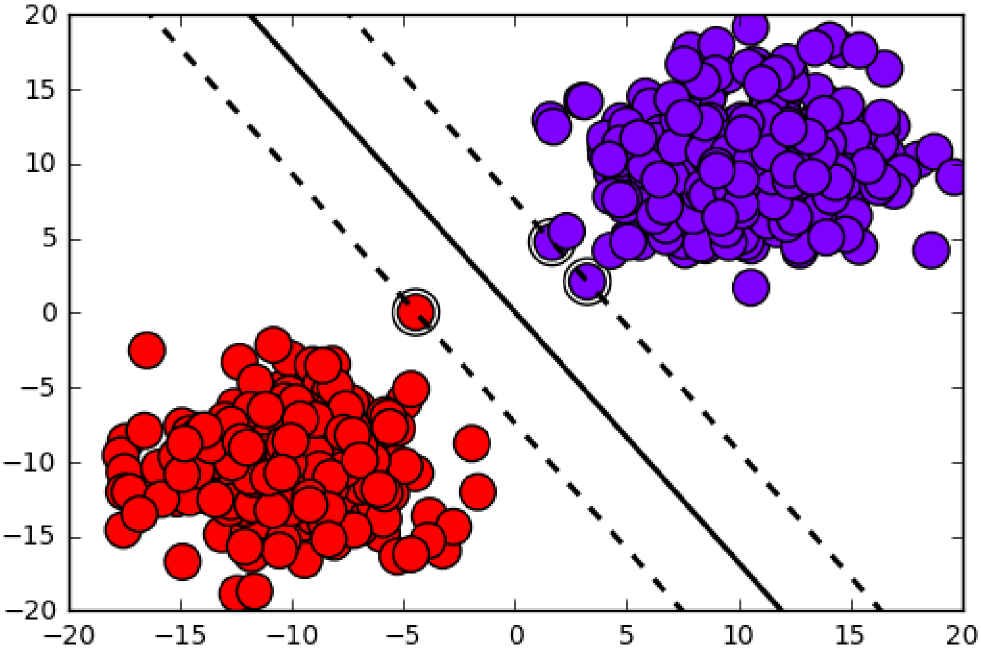
One can instead solve the dual problem.

```
In [5]: ##########################################################
        ### Solving the dual program
        yi=2*color-1
        Qtemp=(yi.T*X)@(yi.T*X).T
        QQP=matrix(Qtemp,tc='d')
        pQP=matrix([-1]*n,tc='d')
        GQP=matrix( -np.eye(n),tc='d')
        hQP=matrix([0]*n,tc='d')
        AQP=matrix(yi,tc='d')
        bQP=matrix([0],tc='d')
        t0svmdual=time()
        sol=solvers.qp(QQP, pQP, GQP, hQP, AQP, bQP)
        t1svmdual=time()
        lam=np.array(sol['x'])
```

```
     pcost       dcost       gap    pres   dres
 0: -2.8148e+01 -5.0693e+01  2e+03  4e+01  2e+00
 1: -2.0731e+01 -1.0951e+01  5e+02  1e+01  5e-01
 2: -4.3666e+01 -1.0630e+01  3e+02  5e+00  2e-01
 3: -1.5298e+01 -1.0373e+00  9e+01  1e+00  6e-02
 4: -3.8901e-01 -4.7700e-02  2e+00  3e-02  1e-03
 5: -5.9805e-02 -4.3267e-02  2e-01  3e-03  1e-04
 6: -4.0984e-02 -4.0415e-02  1e-01  2e-03  8e-05
 7: -9.7017e-03 -4.1251e-02  1e-01  1e-03  5e-05
 8: -1.3429e-02 -3.5315e-02  2e-02  5e-18  2e-15
 9: -2.3041e-02 -3.8186e-02  2e-02  3e-17  3e-15
10: -3.0906e-02 -3.5065e-02  4e-03  8e-18  3e-15
11: -3.4113e-02 -3.4212e-02  1e-04  7e-18  3e-15
12: -3.4186e-02 -3.4187e-02  1e-06  4e-18  3e-15
13: -3.4186e-02 -3.4186e-02  1e-08  4e-17  3e-15
Optimal solution found.
```

```
In [6]: ####################################
        ## Finding the paramateres
        ## supporting normal vector
        w=lam.T@(np.diag(yi.reshape(n,))@X)
        a = -w[0,0] / w[0,1]
        # ## Finding the support vectors
        SVs=np.array([Xlabeled[i] for i in range(n)
                      if np.abs(lam[i])>0.001])
        nSV=len(SVs) # Their numbers
        for i in range(1,nSV):
            if SVs[i,-1] != SVs[0,-1]:
                svtemp=(SVs[i]+SVs[0])[0:-1]
                bw=w@svtemp/2
                break
        b=bw/w[0,1]
        ###############################
        ## Ploting the hyperplane
        xx = np.linspace(-20, 20)
        yy = a* xx + b
        # ## Margins
        yy_up = a * xx +b+1/w[0,1]
        yy_down = a * xx +b-1/w[0,1]
```

```
In [7]: ### Just a plot
        fig = plt.figure()
        # plot the line, the points, and the nearest vectors to the plane
        plt.plot(xx, yy, 'k-', linewidth=2)
        plt.plot(xx, yy_down, 'k--', linewidth=2)
        plt.plot(xx, yy_up, 'k--', linewidth=2)
        plt.scatter(SVs[:, 0], SVs[:, 1], s=300, facecolors='none')
        plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
        plt.axis([-20, 20, -20, 20])
        plt.show()
```



5.2.1 Computational Time

When the data is linearly separable, the dimension of search space for the primal problem is the dimension of data plus one. However the dimension of search space for the primal problem is equal to the number of training samples. Therefore when the dimension of data is much higher than the number of training samples, the dual problem is more efficient to solve. Otherwise the primal problem is the good choice. In this problem, obviously the primal problem is more efficient.

```
In [8]: print("Solving time of the primal problem:",t1svm-t0svm)
        print("Solving time of the dual problem:",t1svmdual-t0svmdual)

Solving time of the primal problem: 0.006913185119628906
Solving time of the dual problem: 0.5178413391113281
```

5.3 Primal Problem - Linearly non-separable case

When the data is not linearly separable, penalty terms are added. We first generate a new dataset which is not linearly separable.

```
In [9]: ########################################################
        # Generate randomly two centers and a covariance matrix
        datamean=4
        m1=np.array([datamean,datamean])
        m2=np.array([-datamean,-datamean])
        Sigma1 =10*np.eye(p)
        Sigma2 = Sigma1
        X1=mvnorm.rvs(m1, Sigma1, size=(n1, 1))
```

```
    X2=mvnorm.rvs(m2, Sigma2, size=(n2, 1))
    #########################################################
    X=np.r_[X1,X2]
    Xlabeled=np.r_['1',X,color.T]
```

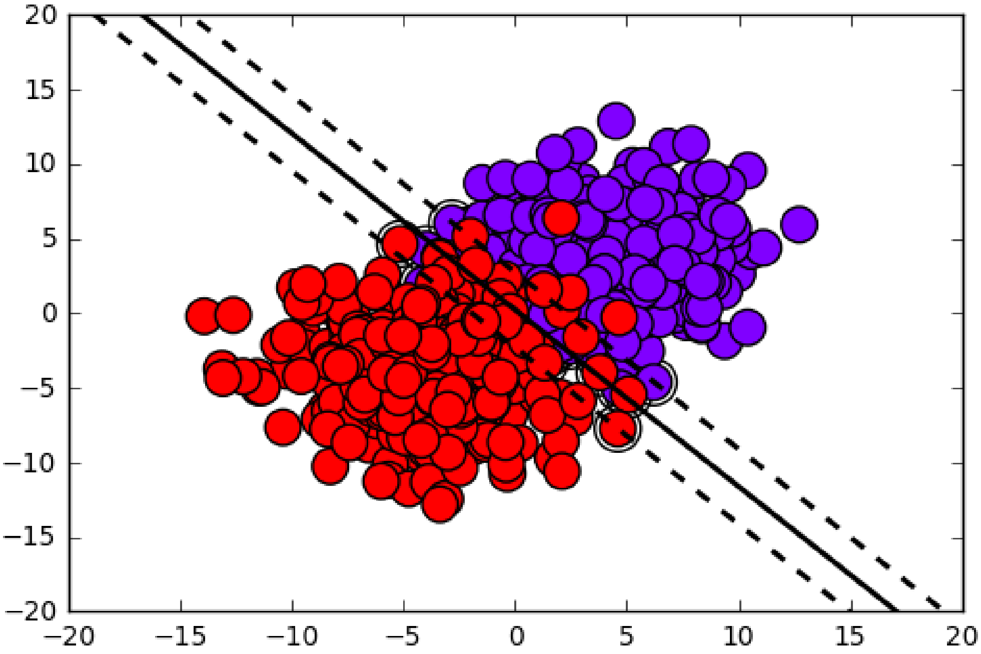
The primal problem is then solved similar to the previous step.

```
In [10]: C=6
         QQP=matrix(np.diag([1]*p+[0]*(n+1)),tc='d')
         pQP=matrix([0]*(p+1)+[C]*n,tc='d')
         yi=2*color-1
         Gtemp=np.r_[-1*np.r_['1',np.diag(yi.reshape(n,))@X,yi.T,np.eye(n)],
                     np.r_['1',np.zeros((n,p+1)),-np.eye(n)]]
         GQP=matrix(Gtemp,tc='d')
         hQP=matrix([-1]*(n)+[0]*n,tc='d')
         t0svmnsp=time()
         sol=solvers.qp(QQP, pQP, GQP, hQP)
         t1svmnsp=time()
         w=np.array(sol['x'])
```

```
     pcost       dcost       gap    pres   dres
 0: -1.8621e+04  1.5633e+04  5e+04  1e+01  1e+02
 1:  2.3596e+03 -1.7466e+03  8e+03  1e+00  9e+00
 2:  6.9557e+02  2.8717e+01  1e+03  2e-01  1e+00
 3:  5.0647e+02  2.5298e+02  4e+02  5e-02  3e-01
 4:  4.8480e+02  2.9077e+02  3e+02  3e-02  2e-01
 5:  4.7754e+02  3.0257e+02  2e+02  3e-02  2e-01
 6:  4.4923e+02  3.3256e+02  1e+02  7e-03  5e-02
 7:  4.4410e+02  3.3736e+02  1e+02  6e-03  4e-02
 8:  4.3701e+02  3.4836e+02  1e+02  3e-03  2e-02
 9:  4.2618e+02  3.4957e+02  8e+01  1e-03  1e-02
10:  4.2473e+02  3.5117e+02  8e+01  1e-03  8e-03
11:  4.0484e+02  3.6586e+02  4e+01  1e-04  7e-04
12:  3.9040e+02  3.7434e+02  2e+01  3e-05  2e-04
13:  3.8340e+02  3.7840e+02  5e+00  3e-06  2e-05
14:  3.8191e+02  3.7954e+02  2e+00  1e-06  7e-06
15:  3.8108e+02  3.8019e+02  9e-01  1e-07  8e-07
16:  3.8075e+02  3.8050e+02  2e-01  3e-08  2e-07
17:  3.8063e+02  3.8061e+02  2e-02  1e-09  6e-09
18:  3.8062e+02  3.8062e+02  3e-04  1e-11  9e-11
Optimal solution found.
```

```
In [11]: ## Ploting the hyperplane
         aVec=np.array([w[0],w[1]])
         a = -w[0] / w[1]
         xx = np.linspace(-20, 20)
         yy = a * xx - (w[2]) / w[1]
         ## Margins
         yy_up = a * xx - (w[2]) / w[1]+1/w[1]
         yy_down = a * xx - (w[2]) / w[1]-1/w[1]
         ## Finding the support vectors
```

```
upperSV=np.array([X[i] for i in range(n)
                  if (np.abs(aVec.T@X[i]+w[2]-yi.T[i]*(1-w[i+3]))<=0.001)])
########################################
## Plots
fig = plt.figure()
# plot the line, the points, and the nearest vectors to the plane
plt.plot(xx, yy, 'k-', linewidth=2)
plt.plot(xx, yy_down, 'k--', linewidth=2)
plt.plot(xx, yy_up, 'k--', linewidth=2)
plt.scatter(upperSV[:, 0], upperSV[:, 1], s=300, facecolors='none')
plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
plt.axis([-20, 20, -20, 20])

plt.show()
```



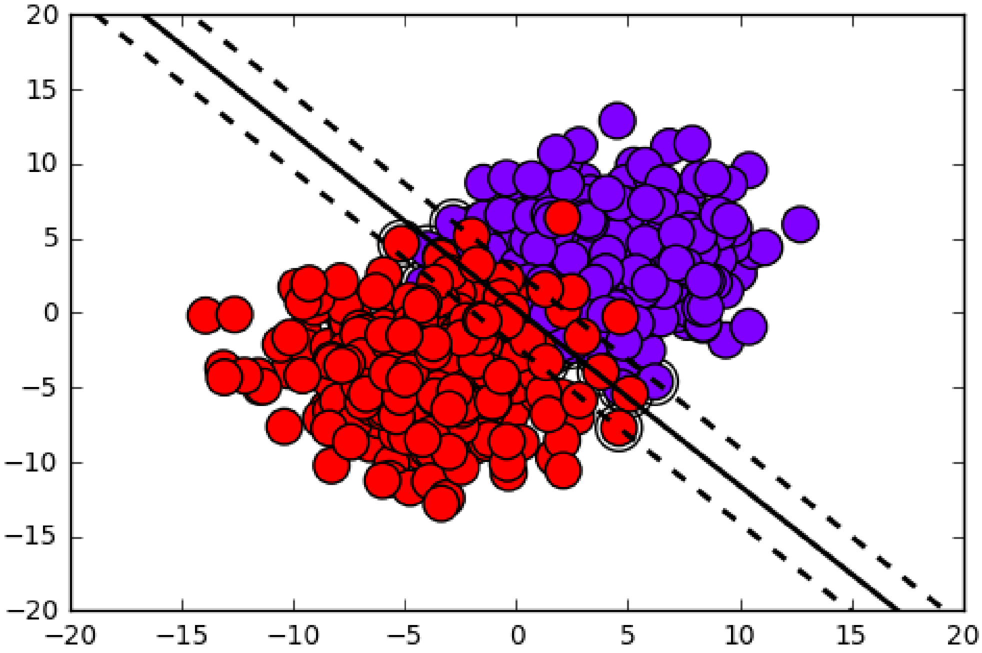
5.4 Dual Problem - Linearly non-separable case

The steps are similar to above. We just run the dual problem accordingly.

```
In [12]: C=6
         yi=2*color-1
         Qtemp=(yi.T*X)@(yi.T*X).T
         QQP=matrix(Qtemp,tc='d')
         pQP=matrix([-1]*n,tc='d')
         GQP=matrix( np.r_[np.eye(n),-np.eye(n)],tc='d')
         hQP=matrix([C]*n+[0]*n,tc='d')
         AQP=matrix(yi,tc='d')
         bQP=matrix([0],tc='d')
         t0svmdualnsp=time()
         sol=solvers.qp(QQP, pQP, GQP, hQP, AQP, bQP)
         t1svmdualnsp=time()
         lam=np.array(sol['x'])

     pcost       dcost       gap    pres   dres
 0: -4.9598e+02 -1.8116e+04  5e+04  1e+00  9e-13
 1: -3.5261e+02 -5.6804e+03  8e+03  9e-02  7e-13
 2: -2.7351e+02 -1.0954e+03  1e+03  1e-02  4e-13
 3: -3.0380e+02 -6.3870e+02  4e+02  4e-03  3e-13
 4: -3.1975e+02 -5.7076e+02  3e+02  2e-03  3e-13
 5: -3.2442e+02 -5.4653e+02  2e+02  2e-03  3e-13
 6: -3.3679e+02 -4.6654e+02  1e+02  5e-04  3e-13
 7: -3.4079e+02 -4.5839e+02  1e+02  4e-04  3e-13
 8: -3.4992e+02 -4.4359e+02  1e+02  2e-04  3e-13
 9: -3.5033e+02 -4.2988e+02  8e+01  1e-04  3e-13
10: -3.5180e+02 -4.2778e+02  8e+01  8e-05  3e-13
11: -3.6591e+02 -4.0511e+02  4e+01  7e-06  4e-13
12: -3.7435e+02 -3.9047e+02  2e+01  2e-06  4e-13
13: -3.7840e+02 -3.8341e+02  5e+00  2e-07  4e-13
14: -3.7954e+02 -3.8191e+02  2e+00  7e-08  3e-13
15: -3.8019e+02 -3.8108e+02  9e-01  8e-09  3e-13
16: -3.8050e+02 -3.8075e+02  2e-01  2e-09  3e-13
17: -3.8061e+02 -3.8063e+02  2e-02  7e-11  3e-13
18: -3.8062e+02 -3.8062e+02  3e-04  9e-13  4e-13
Optimal solution found.
```

```
In [13]: ####################################
         ## Finding the paramateres
         ## supporting normal vector
         w=lam.T@(np.diag(yi.reshape(n,))@X)
         a = -w[0,0] / w[0,1]
         # ## Finding the support vectors
         SVs=np.array([Xlabeled[i] for i in range(n) if np.abs(lam[i])>0.001])
         SVmargin=np.array([Xlabeled[i] for i in range(n)
                            if (C-0.001>np.abs(lam[i])>0.001)])
         bnonscaled=-w@SVmargin[0,0:-1]+2*SVmargin[0,-1]-1
         b=-bnonscaled/w[0,1]

         xx = np.linspace(-20, 20)
         yy = a* xx + b
         # ## Margins
         yy_up = a * xx +b+1/w[0,1]
         yy_down = a * xx +b-1/w[0,1]
         #####################################################################
         #####################################################################
         # Plot
         #####################################################################
         #####################################################################
         ### Just a plot
         fig = plt.figure()
```

```
# plot the line, the points, and the nearest vectors to the plane
plt.plot(xx, yy, 'k-', linewidth=2)
plt.plot(xx, yy_down, 'k--', linewidth=2)
plt.plot(xx, yy_up, 'k--', linewidth=2)
plt.scatter(SVs[:, 0], SVs[:, 1], s=300, facecolors='none')
plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
plt.axis([-20, 20, -20, 20])
plt.show()
```



5.4.1 Computational Time

In case of non-separable data, the dimension of search space is equal to $n+p+1$ where n is the number of samples and p is the data dimension. For the dual problem however, the dimension of search space remains unchanged equal to n. Therefore it is in general more efficient to solve the dual problem in this case.

```
In [14]: print("Solving time of the primal problem:",t1svmnsp-t0svmnsp)
         print("Solving time of the dual problem:",t1svmdualnsp-t0svmdualnsp)
```

```
Solving time of the primal problem: 1.1849477291107178
Solving time of the dual problem: 1.1714956760406494
```

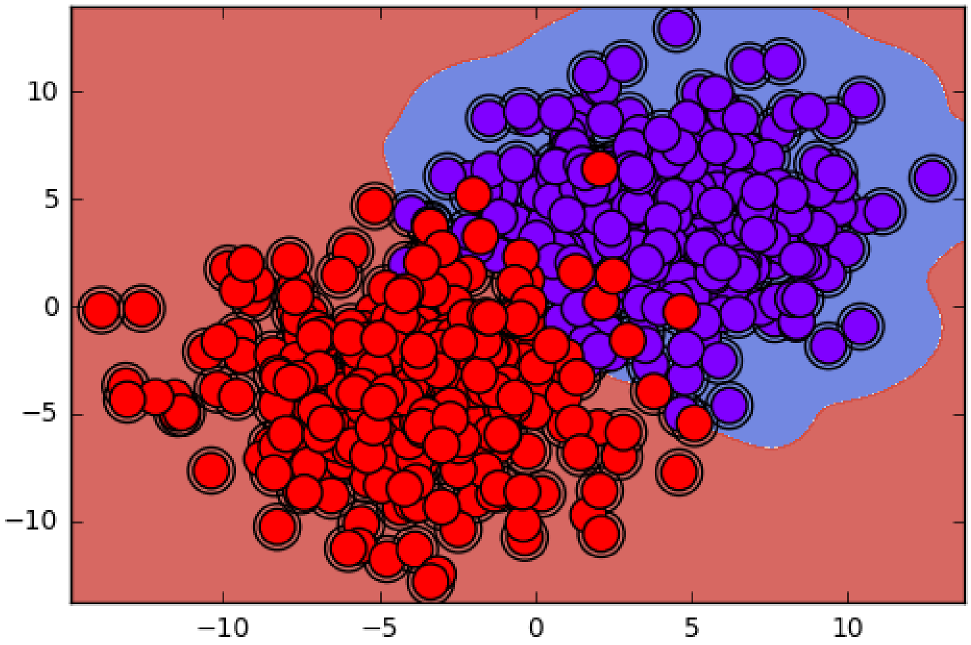
5.5 Kernel-Based Methods

We use the built-in kernel implementation of python. We first load the model and fit it to the data.

```
In [15]: from sklearn import svm
         kern='rbf'
         t0svmkernel=time()
         if (kern=='poly'):
                 kernelsvm =svm.SVC(kernel='poly', degree=2,
                               gamma=0.2, coef0=0.1).fit(X, color.T)
         elif (kern=='rbf'):
                 kernelsvm =svm.SVC(kernel='rbf', gamma=0.4).fit(X, color.T)
         t1svmkernel=time()
```

Afterwards, we plot the figures.

```
In [16]: fig = plt.figure()
         h = 0.02
         x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
         xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                              np.arange(y_min, y_max, h))
         Z = kernelsvm.predict(np.c_[xx.ravel(), yy.ravel()])

         Z = Z.reshape(xx.shape)
         plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
         plt.scatter(kernelsvm.support_vectors_[:, 0],
                     kernelsvm.support_vectors_[:, 1], s=300, facecolors='none')
         #####################################################################
         plt.scatter(X[:,0], X[:,1], s=180, c=color, cmap=plt.cm.rainbow)
         plt.axis([x_min, x_max, y_min, y_max])
         plt.show()
```



Note that the computational complexity of the kernel based methods is similar to the dual problem.

5.6 Kernel-Based Method for MNIST classification

We first load the dataset using the tensorflow backend.

```
In [17]: import tensorflow as tf
         from tensorflow.examples.tutorials.mnist import input_data
         data = input_data.read_data_sets("tensorboard_MNIST/MNIST_data/"
                                          , one_hot=False)
         ## Training set
         N=55000
         Xtraining = data.train.images[0:N]
         #Ytraining = data.train.labels[0:N].argmax(axis=1)
         Ytraining = data.train.labels[0:N]
         ## Test set
         Ntest=10000
         Xtest=data.test.images[0:Ntest]
         #Ytest=data.test.labels[0:Ntest].argmax(axis=1)
         Ytest=data.test.labels[0:Ntest]
```

```
Extracting tensorboard_MNIST/MNIST_data/train-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/train-labels-idx1-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting tensorboard_MNIST/MNIST_data/t10k-labels-idx1-ubyte.gz
```

Two classification problem is considered:

- Two-class classification problem
- Classification of the whole MNIST dataset

We first start by linear SVM and using two-class classifciation problem.

```
In [18]: ################################################################
         ################################################################
         #### Two classes are chosen accordingly from the training set
         Ci=2
         Cj=9
         ## Choosing respective classes from the training set
         Ind12=np.array([ind for ind in range(N) if ((Ytraining[ind]==Ci)
                                                 or (Ytraining[ind]==Cj))])
         X12=Xtraining[Ind12]
         Y12=Ytraining[Ind12]
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("The size of the trainging set with two classes is given by:", len(Ind12))
         N12=len(Ind12)
         ## Choosing respective classes from the test set
         Ind12=np.array([ind for ind in range(Ntest) if ((Ytest[ind]==Ci)
                                                     or (Ytest[ind]==Cj))])
         Xtest12=Xtest[Ind12]
         Ytest12=Ytest[Ind12]
         print("The size of the test set with two classes is given by:", len(Ind12))
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         N12test=len(Ind12)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The size of the trainging set with two classes is given by: 10924
The size of the test set with two classes is given by: 2041
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
In [19]: model = svm.LinearSVC()
         model.fit(X12, Y12)
         TrainingError=np.count_nonzero(np.array(model.predict(X12))-Y12)/N12
         TestError=np.count_nonzero(np.array(model.predict(Xtest12))-Ytest12)/N12test
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("The misclassification error for the training set is given by:",TrainingError)
         print("The misclassification error for the test set is given by:",TestError)
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The misclassification error for the training set is given by: 0.0009154155986818016
The misclassification error for the test set is given by: 0.014698677119059285
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
In [20]: model = svm.LinearSVC()
         t0svmmnist=time()
         model.fit(Xtraining, Ytraining)
         t1svmmnist=time()
         ### Training error and Test error
         TrainingError=np.count_nonzero(np.array(model.predict(Xtraining))-Ytraining)/N
         TestError=np.count_nonzero(np.array(model.predict(Xtest))-Ytest)/Ntest
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("The misclassification error for the training set is given by:",TrainingError)
         print("The misclassification error for the test set is given by:",TestError)
         print("The training time is:",t1svmmnist-t0svmmnist)
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
         print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The misclassification error for the training set is given by: 0.0730909090909091
The misclassification error for the test set is given by: 0.082
The training time is: 71.01174092292786
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Linear SVM provides a strictly better classification error compared to Fisher LDA.