5   Basic machine learning

Machine Learning (ML) or Statistical Learning (SL) refers to the techniques ued to model, describe and understand complex data sets. This emcompasses basic methods from linear regressions to more complex classifying techniques.

The kind of questions that machine learning wants to address are the following:

- Can we predict how a viewer would rate a movie? [1]

- Can we predict a possible rise of criminality in a city?[2]

- Can we guess whether a certain dot in a picture is a distant galaxy or a star?

- What are the common traits between different genome sequences?

What do these questions have in common? First of all, we believe that a pattern exists that can lead us to the answer. Secondly, we cannot make a mathematical model that describe the pattern, i.e. we cannot build a model from basic physics that describe how a viewer would rate a movie. The third *very important* commonality is that we have data on it; we have a measurement or series of measurements that can help us to discover the pattern.

With these three components you are ready to apply machine learning techniques.

In this chapter we talk about three different machine learning techniques: linear regresion, principal component analysis (PCA) and k-means clustering. Before making a good description of each one of these techniques we will make a short linear algebra review, which is needed for the linear regresion discussion.

The last two chapters on PCA and k-means clustering can be studied in great detail from Chapter 10 in the book "An Introduction to Statistical Learning, with applications in R" (Springer, 2013). You can get a pdf from

    `http://www-bcf.usc.edu/~gareth/ISL/data.html`.

5.1   Linear Algebra

5.1.1   Matrices

We will consider a matrix as a rectangular array of real (or complex) numbers arranged in sets of m rows with n entries. The set of matrices with real coefficients with size $m \times n$ is called $\mathbb{R}^{m \times n}$. In the context of this chapter vectors will be understood as a column vectors, that is as a matrix of one column belonging to the set $\mathbb{R}^m$.

We will refer to the a_{ij} element of matrix $\mathbf{A}$ to indicate the element located in row i and column j. In the case of a vector $\mathbf{x}$ we will refer to its components as x_i.

---

[1]See: `http://www.youtube.com/watch?v=mbyG85GZ0PI`
[2]See: `https://github.com/finiterank/homicidios`

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \dots & a_{mn} \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

5.1.2 Matrices as pointers in C

Matrices in C are best stored as one dimensional arrays. This implies that there is convention to translate the position i, j in the matrix into the index p in the one dimensional array.

This convention in C is called **raw major order**, where the second index moves continuously in memory. It means that the following 3×3 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

will be stored in memory as the following one dimensional array

$$\mathbf{a} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

.

In FORTRAN the convention is the opposite, the first index moves continuously in memory. This convention is known as **column major order**.

In the following example we define a matrix 5×5 and initialize it to be the identity.

```
/*
  Creates an identity matrix following the
  row major order convention in C.
*/
#include <stdio.h>
#include <stdlib.h>
int main(void){
  float *I;
  int n_x=5;
  int n_y=5;
  int pos;
  int i,j;

  if(!(I = malloc(sizeof(float)*n_x *n_y))){
      fprintf(stderr, "Problem with memory allocation");
      exit(1);
  }
```

```
  /*Initialization*/
  for(i=0;i<n_x;i++){
    for(j=0;j<n_y;j++){
      pos = j + (n_y * i) ;/*position in the array*/
      if(i==j){
I[pos]=1.0;
      }else{
I[pos]=0.0;
      }
    }
  }

  /*Prints to screen*/
  for(i=0;i<n_x;i++){
    for(j=0;j<n_y;j++){
      pos = j + (n_y * i);/*position in the array*/
fprintf(stdout, " %f ",I[pos]);
    }
    fprintf(stdout, "\n");
  }

  return 0;
}
```

5.1.3   Functions taking pointers in C

We go into the details of dealing with a array/matrix operation in C. Most of the operations we make onto a matrix will be done using functions. Functions in C work with variables passed as reference, not as values. That means that if you want to have a variable modified, you have to give the address where the variable lives in memory. In practice, that means that you should work only with pointers.

In the next example we define a matrix 3×3 an take its transpose.

```
/*
  Creates an identity matrix following the
  row major order convention in C.
*/
#include <stdio.h>
#include <stdlib.h>
void transpose(float * m, int n_x, int n_y);
void print_matrix(float * m, int n_x, int n_y);

int main(void){
  float *I;
  int n_x=5;
```

```c
    int n_y=5;
    int pos;
    int i,j;

    if(!(I = malloc(sizeof(float)*n_x *n_y))){
        fprintf(stderr, "Problem with memory allocation");
        exit(1);
    }

    /*Initialization*/
    for(i=0;i<n_x;i++){
      for(j=0;j<n_y;j++){
        pos = j + (n_y * i) ;/*position in the array*/
        I[pos] = pos;
      }
    }

    print_matrix(I, n_x, n_y);

    transpose(I, n_x, n_y);

    print_matrix(I, n_x, n_y);

    return 0;
}



void print_matrix(float * m, int n_x, int n_y){
  int i,j,pos;

  fprintf(stdout, "\n");
  /*Prints to screen*/
  for(i=0;i<n_x;i++){
    for(j=0;j<n_y;j++){
       pos = j + (n_x * i);/*position in the array*/
       fprintf(stdout, " %f ",m[pos]);
    }
    fprintf(stdout, "\n");
  }
  fprintf(stdout, "\n");
}

void transpose(float * m, int n_x, int n_y){
  int i,j,pos_ij, pos_ji;
  float tmp;
```

```
  /*Prints to screen*/
  for(i=0;i<n_x;i++){
    for(j=i;j<n_y;j++){
      pos_ij = i + (n_x * j);
      pos_ji = j + (n_x * i);

      tmp = m[pos_ij];
      m[pos_ij] = m[pos_ji];
      m[pos_ji] = tmp;
    }
  }

}
```

What would happen if you try to modify the variable `n_x` inside a function?

5.1.4   GNU Scientific Library

Most of the computations that we will perform in C will be done using external libraries. We will tap on the work done by other people. However simple it may seem it is important that you understand how external libraries can be used in C.

```
sudo apt-get install libgsl0-dev
sudo apt-get install liblapack-dev
```

Let's start by considering the following example that uses the gsl library to compute the the Bessel function $J_0(x)$ for $x = 5$:

```
#include <stdio.h>
#include <gsl/gsl_sf_bessel.h>

int main (void){
      double x = 5.0;
      double y = gsl_sf_bessel_J0 (x);
      printf ("J0(%g) = %.18e\n", x, y);
      return 0;
}
```

If the setup for the library are the standard ones, the program can be compiled as

```
cc -lgsl -lgslcblas -lm test.c
```

If you are under UBUNTU probably the compilation order matters and you should instead write

```
cc test.c -lgsl -lgslcblas -lm
```

Executing the program will produce the following results:

```
J0(5) = -1.775967713143382642e-01
```

GSL also has special libraries to deal with linear algebra operations. A detailed webpage including the manual and useful examples can be found here:

```
http://www.gnu.org/software/gsl/manual/html_node/
```

Now try to follow the structure in the following program that takes a square matrix and computes its eigenvalues[3]):

```
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>

int main (void){
  double data[] = { 1.0  , 1/2.0, 1/3.0, 1/4.0,
    1/2.0, 1/3.0, 1/4.0, 1/5.0,
    1/3.0, 1/4.0, 1/5.0, 1/6.0,
    1/4.0, 1/5.0, 1/6.0, 1/7.0 };

  gsl_matrix_view m
    = gsl_matrix_view_array (data, 4, 4);

  gsl_vector *eval = gsl_vector_alloc (4);
  gsl_matrix *evec = gsl_matrix_alloc (4, 4);

  gsl_eigen_symmv_workspace * w = gsl_eigen_symmv_alloc (4);

  gsl_eigen_symmv (&m.matrix, eval, evec, w);

  gsl_eigen_symmv_free (w);

  gsl_eigen_symmv_sort (eval, evec,
GSL_EIGEN_SORT_ABS_ASC);

  {
    int i;

    for (i = 0; i < 4; i++)
      {
double eval_i
  = gsl_vector_get (eval, i);
gsl_vector_view evec_i
```

---

[3]Example taken from the example manual of the GSL webpage.

```
    = gsl_matrix_column (evec, i);

printf ("eigenvalue = %g\n", eval_i);
printf ("eigenvector = \n");
gsl_vector_fprintf (stdout,
    &evec_i.vector, "%g");
      }
  }

  gsl_vector_free (eval);
  gsl_matrix_free (evec);

  return 0;
}
```

5.1.5   Linear Algebra and Regressions: Mathematical Aspects

We are now interested in giving a mathematical background to the problem of regressions in physics[4]. The following situation is very common in physics, we the measurement of some quantity, say the spectra from a distant galaxy and we want infer other physical properties in that galaxy (mass and age, for instance).

In general the methods to infer the properties of a model from the measurements is known as inverse theory. The methods that allows us to pick the most probable model from a series of measurements will be called statistical inference. Both are related and have many similarities.

In general mathematical terms, if we have a model for a physical phenomenon described by a set of parameter and we have a theory that can relate those parameters to produce an observable, we can write

$$d_i = G_i(\mathbf{m}), \tag{1}$$

where $\mathbf{m}$ contains all the instances of the physical parameters in my model (i.e time, masses, etc), d is the observed data and G is the is the theory that produces the observed data d from the parameters in the model m. The index i goes over the number of measurements that you have.

Why is this problem hard? Because you might have a model described by infinite number of parameters and a finite number of measurements. In that case we talk about and undetermined problem. You might decide to oversimplify the problem and reduce the number of parameters, if you have more measurements than parameters in the model, you are in the extreme of an overdetermined problem.

This problem can be written as matrix multiplications:

$$\mathbf{d} = \mathbf{Gm} \tag{2}$$

_____

[4]This section is based on German Prieto notes on inversion theory in geosciences.

where $\mathbf{d}$ represents the vector with the data. $\mathbf{G}$ is the theory that we use (electrodynamics, gravitation, etc) for predicting data and $\mathbf{m}$ is the model that we wish to obtain.

5.1.6 Linear regressions

The measurements that you have might be the positions as a function of time of ball flying through the air. You decided that the physics that best apply in that case are the kinematics of a movement with uniform acceleration. Under that theory the model that describes the position as a function of time is the following:

$$y(t) = m_1 + m_2 t + (1/2)m_3 t^2 \tag{3}$$

where $y(t)$ represents the altitude of the object at time t, and the three ($N = 3$) model parameters m_i are associated with a constant, slope and quadratic terms. Note that even if the function is quadratic, the problem in question is linear for the three parameters.

If we have M discrete measurements y_i at times t_i, the linear regression problem or inverse problem can be written in the form

$$
\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix}
=
\begin{bmatrix}
1 & t_1 & \frac{1}{2}t_1^2 \\
1 & t_2 & \frac{1}{2}t_2^2 \\
\vdots & \vdots & \vdots \\
1 & t_M & \frac{1}{2}t_M^2
\end{bmatrix}
\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}
\tag{4}
$$

When the regression model is linear in the unknown parameters, then we call this a linear regression or linear inverse problem.

Solving this problem means finding a model $\hat{\mathbf{m}}$ that best reproduced the observed values. *Best reproducing* means seeing how different are the observations $\mathbf{d}$ when compared to the results of the theory computed over the model: $\mathbf{G}\hat{\mathbf{m}}$.

Suppose we are given a collection of M measurements of a property to form a vector $\mathbf{d} \in \mathbb{R}^M$. We know forward problem such that we can predict the data from a known model $\mathbf{m} \in \mathbb{R}^N$. That is, we know the N vectors $\mathbf{g}_k \in \mathbb{R}^M$ such that

$$\mathbf{d} = \sum_{k=1}^{N} \mathbf{g}_k m_k = \mathbf{Gm} \tag{5}$$

where

$$\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N] \tag{6}$$

We are looking for a model $\hat{\mathbf{m}}$ that minimizes the size of the residual vector defined

$$\mathbf{r} = \mathbf{d} - \sum_{k=1}^{N} \mathbf{g}_k \hat{m}_k \tag{7}$$

We do not expect to have an exact fit so there will be some error, and we use a norm to measure the size of the residual

$$\|\mathbf{r}\| = \|\mathbf{d} - \mathbf{Gm}\| \tag{8}$$

For the least squares problem we use the $L2$ or Euclidean norm

$$\|\mathbf{r}\|_2 = \sqrt{\sum_{k=1}^{M} r_k^2} \tag{9}$$

Minimizing this quantity will give you the best parameters of your model.

We won't derive the results here, but assuming the inverse of $(\mathbf{G}^T\mathbf{G})$ exists, we can isolate $\hat{\mathbf{m}}$ to end up with

$$\hat{\mathbf{m}} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\mathbf{d} \tag{10}$$

Note that the matrix $(\mathbf{G}^T\mathbf{G})$ is a square $N \times N$ matrix and $\mathbf{G}^T\mathbf{d}$ is an N column vector.

5.2   Principal Component Analysis

Principal Component Analysis is used when you have a set of measurements or individuals described by a large number of variables and you want (hope to be able) to reduce that dimensionality.

This time we start from M different measurements of points in N dimensions. The dimensions might be different physical quantities (velocities, masses), or might be the points in wavelength where you measure the intensity of a spectrum.

We start by measuring the covariance matrix, $\boldsymbol{\Sigma}$ of different dimension of all the data points. This covariance matrix will be a $N \times N$ matrix where each element is computed as:

$$\sigma_{i,j} = \frac{\sum_{k=1}^{M}(d_i^k - \bar{d}_i)(d_j^k - \bar{d}_j)}{M-1} \tag{11}$$

In this notation there are M vector of measurements $\mathbf{d}^1,\ldots,\mathbf{d}^M$. Where each vector has N components, for instance the k-th measurement will be $\mathbf{d}^k = (d_1^k, \ldots, d_N^k)$.

Some properties of the covariance matrix are the following:

1. The matrix $\boldsymbol{\Sigma}$ is symmetric; $\sigma_{ij} = \sigma_{ji}$.

2. Diagonal elements correspond to the the variance of the corresponding variables $\sigma_{ii} = \sigma_i^2$.

3. The covariance matrix is defined as non-negative ($\sigma_{ij} \geq 0$).

The value of each entry in the covariance matrix depends on the units defined to make the measurements. An alternative is to use the correlation matrix,$\mathbf{R}$, which normalizes each measuremente by the square root of the variance along that dimension. As a result the diagonal elements of $\mathbf{R}$ are unity $\rho_{ii} = 1$.

The PCA technique consists in calculating the covariance matrix (or the correlation matrix to avoid the arbitrary unit scaling) and calculating its eigenvalues and eigenvectors. If you rank the eigenvalues in descending order the associated eigenvectors will tell the directions (in N dimensional space) where the variance decreases. With this you will be able to reduce the dimensionality of your problem and express each point as the sum of the eigenvectors of this matrix.

As a more concrete example of the application of Principal Component Analysis let's take some data from the context of neurosciences. We have a patient and we are measuring her brain activity through encephalograms taken with 24 electrodes at different points on its head. The encephalogram measures electric activity as a function of time. In this case we have now 24 different signals taken at 400 different times. In the figure we show two different signals.

5.3   K-means clustering

Clustering means to the ways of finding groups in a set of data. This means that within a set of observations there will be groups that are similar and others are different. A clear application can be though in a group of students that is measured by two numbers: the amount of hours spent studyng every week and the final grade in a course. In this case there might be four clusters; students that

1. Spend a lot of time studying and have good grades

2. Don't spend time studying and have good grades

3. Spend a lot of time studying and have bad grades

4. Don't spend time studying and have bad grades

The question is. How can we design an algorithm to find these clusters?

K-means clustering is an easy way to partition a data set into K clusters. We denote by $S_1, \cdots, C_K$ the K sets containing all the observations. These sets satisfy two properties: (1) the sum of all the sets is equal to the original observation set and (2) the clusters are disjoint.

A possible way to define the partitioning that splits the observations into K clusters is to minimize the sum of the variance of each cluster.

$$\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{m,m' \in C_k} \sum_{n=1}^{N} (d_n^m - d_n^{m'})^2$$

meaning that the variance is the average of all the euclidian interparticle distances between all the pairs in the cluster.

An heuristic algorithm to minimize this variance is the following

- Initialize by asigning a number from 1 to K to each observation.

- Iterate until the number assigned to each observation stops changing.

    - For each of set of points wih the label k, compute the centroid.
    - Assign each measurement to the cluster number k whose centroid is closest.