Artificial Intelligence. Practice 3.

Logic Programming.

March-April 2022

Introduction

This practice will help you to master logic programming. At the same time, guided exercises of machine learning are proposed for you to learn while programming and to make you the next practice easier. Please solve the following exercises. Do not forget to code tests to check whether the exercises are ok or not. Remember that the command `trace/0` of Prolog helps you to debug your Prolog code in an easy way.

Training We strongly recommend you to have a look to the 99 Prolog problems located at `https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/`. There you can find a big collection of problems with code that can help you as traning to solve the proposed exercises of this practice if at the beggining they are too complicated for you.

Delivery You must send a single compressed file whose name, all in lowercase and without accents or special characters, must have the following structure:
p3_[gggg]_[mm]_[surname1]_[surname2].zip.
Where **[gggg]** is the group identifier and **[mm]** is the pair identifier. For example, if Alejandro Bellogin and Alberto Suárez are pair 2 of practice group 2351, their submission file would be `p3_2351_02_bellogin_suarez.zip`. Please respect this format strictly.

This .zip file will contain the code **code_[gggg]_[mm]_[surname1]_[surname2].pl** and a report related with the exercises 1 and 9.5 (reading, analysis and discussion of the results). **The submission deadline is on April 20th-22nd at 09:00 AM, depending on whether you have class on Wednesday, Thursday or Friday**. You must not change the headers that are provided in the file `cabecera.pl` because an automatic checker will assume that the predicates have the names and parameters that are provided in the header `cabecera.pl`.

Managing errors The automatic checker will also verify the accurate managing of the errors that are requested in each exercise. The error logs that your files print must coincide with the ones that are given to you. To write in the log file you are given in the code of `cabecera.pl` the predicate `escribir_log/1` that prints in the file `error_logs.txt` the messages that you give it as parameter.

1 Reading exercise (0.5 points)

Write the declarative (for the general case) and procedural (for the query `slice([1, 2, 3 ,4],2,3,L2)` reading of the predicate `slice/4`, available at `https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/p18.pl`. See `https://www.metalevel.at/prolog/reading` for an example.

2 Sum of the products powered to a number. (0.4 points)

Let `X` and `Y` be two vectors whose elements are real number. Code a Prolog predicate `sum_pot_prod/4` of parameters `sum_pot_prod(X, Y, Potencia, Resultado)` that returns the sum `Resultado` of the product of the elements of the vectors `X` and `Y` powered to a number `Potencia`. You must compute:

$$z = \sum_{i=1}^{l} (x_i y_i)^p \, .$$

Where p is the power, z is the final result, l is the length of the vectors $\mathbf{x}$ and $\mathbf{y}$. x_i and y_i are the elements that are referenced by the index i of the vectors $\mathbf{x}$ and $\mathbf{y}$. This power must be positive. The vectors must have the same length. If the power is 1, this operation is equivalent to the scalar product. Examples:

```
sum_pot_prod([1,2,3],[3,4,5], 1, X). X = 26.
sum_pot_prod([1,2,3],[3,4,5], 2, X). X = 298.
sum_pot_prod([1,2,3],[3,4,5], -1, _).  false.   Prints: ERROR 1.1 Potencia.
sum_pot_prod([1,2,3],[3,4,5,6], 3, _).  false.   Prints: ERROR 1.2 Longitud.
```

3 Second and penultimate (0.4 points)

Code a predicate `segundo_penultimo/3` of parameters `segundo_penultimo(L, X, Y)` that returns the second element `X` and the penultimate element `Y` of a list `L`. The list must have a length greater than 1. The interpreter must return a single solution. Examples:

```
segundo_penultimo([1,2], X, Y). X = 2.  Y = 1.
segundo_penultimo([1,2,3], X, Y). X = 2.  Y = 2.
segundo_penultimo([1,2,3,4], X, Y). X = 2.  Y = 3.
segundo_penultimo([1], X, Y). false. Prints: ERROR 2.1 Longitud.
```

4 Sublist that contains an element. (0.5 points)

Code a predicate `sublista/5` of parameters `sublista(L, Menor, Mayor, E, Sublista)` that returns a sublist `Sublista` of the list `L` given by the indexes `Menor` y `Mayor`. This sublist must also contain the element `E` given by the fourth parameter. If the list does not contain this element, the predicate will fail. The index starts by 1. The sublist must include the two elements referenced by the indexes `Menor` and `Mayor`. Check that the indexes are correct. That is, `Menor <= Mayor` y `Mayor <= longitud_lista(L)`. Examples:

```
sublista(['a','b','c','d','e'], 2, 4, 'b', X). X = ['b','c','d'].
sublista(['a','b','c','d','e'], 2, 4, 'f', X). false. Prints: ERROR 3.1 Elemento.
sublista(['a','b','c','d','e'], 5, 4, 'b', X). false. Prints: ERROR 3.2 Indices.
```

5 Linear space. (0.5 points)

Code a predicate `espacio_lineal/4` of parameters `espacio_lineal(Menor, Mayor, Numero_elementos, Rejilla)` that generates the linear grid (list of real numbers) `Rejilla` with the number of points `Numero_elementos` of the interval given by `[Menor, Mayor]`. The linear grid is a vector of elements between the numbers `Menor` y `Mayor` (included) such that the distance between each element is equal and each element is greater than the previous one. `Menor` y `Mayor` must lie in the linear grid `Rejilla`. Check that the indexes are correct. The examples illustrate with better clarity the functionality that is requested. Examples:

```
espacio_lineal(0, 1, 5, L).
L = [0, 0.25, 0.5, 0.75, 1.0].

espacio_lineal(0, 1, 100, L).
L=[0, 0.01010101, 0.02020202, 0.03030303, 0.04040404, 0.05050505, 0.06060606, 0.07070707,
0.08080808, 0.09090909, 0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414, 0.15151515,
0.16161616, 0.17171717, 0.18181818, 0.19191919, 0.2020202 , 0.21212121, 0.22222222, 0.23232323,
0.24242424, 0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929, 0.3030303 , 0.31313131,
0.32323232, 0.33333333, 0.34343434, 0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444, 0.45454545, 0.46464646, 0.47474747,
0.48484848, 0.49494949, 0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455, 0.55555556,
0.56565657, 0.57575758, 0.58585859, 0.5959596 , 0.60606061, 0.61616162, 0.62626263, 0.63636364,
0.64646465, 0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 , 0.70707071, 0.71717172,
0.72727273, 0.73737374, 0.74747475, 0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798
, 0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485, 0.85858586, 0.86868687, 0.87878788,
0.88888889, 0.8989899 , 0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495, 0.95959596,
0.96969697, 0.97979798, 0.98989899, 1]

espacio_lineal(2,0,3,L). false. Prints: ERROR 4.1 Indices.
```

6 Normalize probability distributions. (0.4 points)

Code a predicate `normalizar/2` of parameters `normalizar(Distribucion_sin_normalizar, Distribucion)` that normalize the unnormalized univariate distribution `Distribucion_sin_normalizar` in the parameter `Distribucion`. To normalize an univariate distribution you must obtain the normalization constant z. This constant is given by the integral of the distribution in its support (sum of all the elements of the univariate distribution).

$$z = \int p(x)dx\,.$$

Where the integral is computed in its support (space range) where the distribution is defined. x represents each element of the integral. Do not fear by this! Remember that, in essence, the integral is a simple sum. If we have, as it is the case, the univariate distribution represented by a list $p(\mathbf{x})$, the previous integral can be approximated by the sum of the elements of that list.

$$z \approx \sum_{i=1}^{l} p(x_i)\,.$$

Where $p(x_i)$ is the element of the list referenced by the index i and l is the length of the list. In other words, to estimate the normalization constant of a distribution represented by a list it is enough to sum the elements of that list. All the elements of the unnormalized distribution must be positive.

To normalize the distribution given by the parameter `Distribucion_sin_normalizar` you must divide each of the elements of `Distribucion_sin_normalizar` by the normalization constant z. A good way to test if you have done this exercise correctly is to verify that the output vector of your program represents a probability density function. In other words, its integral must be equal or almost equal (it is approximated)

to 1. Examples:

```
normalizar([3,4,5], X).
X = [0.25, 0.3333333333333333, 0.4166666666666667].
normalizar([1,2,3,4,5], X)
X = [0.06666666666666667, 0.13333333333333333, 0.2, 0.26666666666666666, 0.3333333333333333].
normalizar([-1,2,3,4,5], X). false. Prints: ERROR 5.1.  Negativos.
```

7 Kullback-Leibler divergence. (0.5 points)

As we know by the absolute value of the substraction the distance between two points or we use the Euclidean distance to represent the distance between two points that belong to $\mathbb{R}^d$ where d is a number greater than 1 we can also compute a notion of distance or a divergence (not exactly a distance) between two probability distributions. This divergence shows us how similar are two probability distributions. For instance, two identical distributions have divergence 0. As different distance metrics exist, as Manhattan or Euclidean, different divergencies exist between distributions. An example of divergence is Kullback Leibler divergence.

Code a predicate `divergencia_kl/3` of parameters `divergencia_kl(D1, D2, KL)` that compute the Kullback Leibler divergence `KL` of two distributions `D1` and `D2`. The KL divergence between two discrete univariate probability distributions $p(x)$ y $q(x)$ is computed according to the following expression (using the trick of approximating an integral by a sum that is described in the previous exercise):

$$D_{KL}(p(x)||q(x)) = \int p(x)\ln\frac{p(x)}{q(x)}dx \approx \sum_{x=1}^{N} p(x_i)\ln\frac{p(x_i)}{q(x_i)}\,.$$

In other words, for each element of the two vectors $p(\mathbf{x})$ and $q(\mathbf{x})$ you must compute $p(x_i)\ln\frac{p(x_i)}{q(x_i)}$ and keep summing those quantities. Specify whether any element of the distributions has value 0 or lower than 0. The divergence is not defined in these cases. You must also specify whether any of the parameters that represent distributions are not distributions. In order to do so, you can reuse code of the previous exercise. Examples:

```
divergencia_kl([0.2, 0.3, 0.5], [0.2, 0.3, 0.5], D). D = 0.0
Distributions are equal. KL divergence is 0.
divergencia_kl([0.5, 0.3, 0.2], [0.2, 0.3, 0.5], D). D = 0.27488721956224654.
divergencia_kl([0.98, 0.01, 0.01], [0.01, 0.01, 0.98], D). D = 4.447418454310455.
Very different distributions have high KL divergence.
divergencia_kl([0.99, 0.0, 0.01], [0.01, 0.01, 0.98], D). false.
Prints: ERROR 6.1.  Divergencia KL no definida.
divergencia_kl([0.2,0.3,0.6], [0.1,0.5,0.4], D). false.
Prints: ERROR 6.2.  Divergencia KL definida solo para distribuciones.
```

8 Block matrix given by the Kronecker product. (1.6 points)

As several distances are defined between elements of the same space, they are also defined several product between, in this case, matrices. Imagine that you have two squared matrices 2x2 of real numbers $\mathbf{A}$ y $\mathbf{B}$:

$$\mathbf{A} = \begin{pmatrix} 2 & 5 \\ 3 & 4 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 7 \\ 9 & 3 \end{pmatrix}.$$

Imagine that you want to obtain as a result of this matrices a matrix $\mathbf{C}$ that contains so many elements C_{ij} as possible multiplications we can compute over the elements A_{ij} y B_{ij} of the matrices $\mathbf{A}$ y $\mathbf{B}$. That is, the block matrix:

$$\mathbf{C} = \begin{pmatrix} \begin{pmatrix} 2*1 & 2*7 \\ 2*9 & 2*3 \end{pmatrix} & \begin{pmatrix} 5*1 & 5*7 \\ 5*9 & 5*3 \end{pmatrix} \\ \begin{pmatrix} 3*1 & 3*7 \\ 3*9 & 3*3 \end{pmatrix} & \begin{pmatrix} 4*1 & 4*7 \\ 4*9 & 4*3 \end{pmatrix} \end{pmatrix}.$$

This product can be useful to not lose information of any element and to gain information of the product of the elements of the two matrices. Before describing the generalization of this product, let us define this operation as the Kronecker product of two matrices $\mathbf{A}$ and $\mathbf{B}$ and let us define it as $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$. The Kronecker product must not require squared matrices, as in the previous case. It can be computed between matrices of different dimensions as in this example:

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \end{pmatrix}.$$

$$\mathbf{Z} = \begin{pmatrix} \begin{pmatrix} x_{11}y_{11} & x_{11}y_{12} & x_{11}y_{13} \\ x_{11}y_{21} & x_{11}y_{22} & x_{11}y_{23} \end{pmatrix} & \begin{pmatrix} x_{12}y_{11} & x_{12}y_{12} & x_{12}y_{13} \\ x_{12}y_{21} & x_{12}y_{22} & x_{12}y_{23} \end{pmatrix} \\ \begin{pmatrix} x_{21}y_{11} & x_{21}y_{12} & x_{21}y_{13} \\ x_{21}y_{21} & x_{21}y_{22} & x_{21}y_{23} \end{pmatrix} & \begin{pmatrix} x_{22}y_{11} & x_{22}y_{12} & x_{22}y_{13} \\ x_{22}y_{21} & x_{22}y_{22} & x_{22}y_{23} \end{pmatrix} \\ \begin{pmatrix} x_{31}y_{11} & x_{31}y_{12} & x_{31}y_{13} \\ x_{31}y_{21} & x_{31}y_{22} & x_{31}y_{23} \end{pmatrix} & \begin{pmatrix} x_{32}y_{11} & x_{32}y_{12} & x_{32}y_{13} \\ x_{32}y_{21} & x_{32}y_{22} & x_{32}y_{23} \end{pmatrix} \end{pmatrix}.$$

Code a predicate `producto_kronecker/3` of arguments `producto_kronecker(Matriz_A, Matriz_B, Matriz_bloques)` that computes the Kronecker product `Matriz_bloques` of two matrices `Matriz_A` and `Matriz_B`. If any of the elements of the matrix is lower than 0, then, you must interrupt the computation and alert the user of the failure. You must obtain a block matrix where, if you observe the examples and as a tip, each block is the multiplication of an element of the first matrix by the other matrix. Another tip: Apply a bottom-up methodology to solve this exercise. That is, start with the easiest functionality and end coding the most complex functionality. Do not hesitate to use several recursion levels to solve this exercise. That is precisely what you are supposed to do. Examples:

```
producto_kronecker([[1,2],[3,4]], [[0,5], [6,7]], R).
```

$$\mathbf{R} = \begin{pmatrix} \begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix} & \begin{pmatrix} 0 & 10 \\ 12 & 14 \end{pmatrix} \\ \begin{pmatrix} 0 & 15 \\ 18 & 21 \end{pmatrix} & \begin{pmatrix} 0 & 20 \\ 24 & 28 \end{pmatrix} \end{pmatrix}.$$

```
R = [[[[0, 5], [6, 7]], [[0, 10], [12, 14]]], [[[0, 15], [18, 21]], [[0, 20], [24, 28]]]].
producto_kronecker([[-1,2],[3,4]], [[0,5], [6,7]], R). false
```

Prints: `ERROR 7.1.  Elemento menor que cero.`

9 K nearest neighbours. (3.8 points)

The K nearest neighbours algorithm is a machine learning algorithm. Nevertheless, you are not required to know anything about machine learning to solve this exercise, since it is completely guided. Although, coding this exercise in Prolog will make you to master logic programming and, at the same time, studying the results given by this algorithm will be an excellent prologue for you to understand machine learning. Let us first describe briefly, for those of you interested, what is machine learning about in order to give you the intuition of what problems does machine learning solve.

The machine learning algorithm K nearest neighbours is able to, by a simple mechanism, predict to which label between a set of labels does some data belong. This algorithm is able to predict this information because it has been trained over a data set. This data set is a collection of labelled data. For instance, some data that we will define as instance, contains the qualifications of a student in its last high school course. The label here is the career degree that the student chooses. For example, an instance contains that a student has obtained a high degree in physics and mathematics and has decided to study Software Engineering at UAM. If lots of instances share this pattern, the K nearest neighbours algorithm will predict that a student will choose to study Software Engineering at UAM if he has obtained a high degree in physics and mathematics. Machine learning algorithms can be used to classify an instance according to one label based on other data. Most of them work in 2 stages. Training, where they are presented a data set called

training set where each instance is labelled and the algorithm computes a configuration to minimize some prediction error. The other stage is to predict, where the algorithm takes an unlabelled instances and must determine which label is associated to that instance.

Code a predicate `k_vecinos_proximos/5` of parameters `k_vecinos_proximos(X_entrenamiento, Y_entrenamiento, K, X_test, Y_test)` that implements the K nearest neigbhours algorithm. We are going to solve the implementation of this algorithm step by step. K nearest neighbour will use the Euclidean distance. We will have as an input a matrix `X_entrenamiento`. This matrix will have N instances, where each instance is a list of real numbers. Each instance has M features, that is the list has length M. You will also have as an input a list `Y_entrenamiento` of N elements or labels that represent a categorical variable. Each element of `Y_entrenamiento` is respectively associated to an instance of `X_entrenamiento`. That is, the first label of `Y_entrenamiento` is associated to the first instance of `X_entrenamiento` and so on. The algorithm `k_vecinos_proximos/5` will determine the label of each instance of the test set `X_test`, that as `X_entrenamiento` is a matrix. Although, in the case of `X_test` we do not know its labels. Precisely, `k_vecinos_proximos/5` will predict the labels of `X_test` and store them on `Y_test`. As we will further see, the label of each element will be the determined by most of the labels stored in a vector of K positions. This vector of K positions associated to each instance of the test set `X_test` is computed as the labels of the instances of the training set `X_entrenamiento` that have the lowest Euclidean distances with respect to each instance of `X_test`. In this exercise we will no longer check errors and assume that the input parameters are OK. To implement this classifier, follow the next steps:

9.1 Euclidean distance.

Code a predicate `distancia_euclidea/3` of parameters `distancia_euclidea(X1, X2, D)` that returns the Euclidean distance `D` between the instances `X1` and `X2`. Remember that if we have two vectors $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$ where K is a number greater than 0, then, the Euclidean distance $d(\mathbf{x}, \mathbf{y})$ between both vectors is equal to the next expression.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_i - y_i)^2 + ... + (x_k - y_k)^2}\,.$$

Examples:

```
distancia_euclidea([1,4,3], [1,2,1], D).
D = 2.8284271247461903.
distancia_euclidea([1,2,1], [1,2,1], D).
D = 0.0.
```

9.2 Computing distances between points.

Code a predicate `calcular_distancias/3` of parameters `calcular_distancias(X_entrenamiento, X_test, Matriz_resultados)` that obtains a matrix of distances `Matriz_resultados` where each row is the distance of one test point that is on `X_test` with respect to all the other points of the training set `X_entrenamiento`. Tip: think that you will need to use a double recursion procedure to build this matrix. Examples:

```
calcular_distancias([[1,2,3],[1,2,4],[2,3,1]], [[1,2,1], [1,2,5]], D).
D = [[2.0, 3.0, 1.4142135623730951], [2.0, 1.0, 4.242640687119285]].
calcular_distancias([[1,2,3,4],[1,2,4,5],[2,3,1,5]], [[1,2,1,2], [1,2,5,2], [1,2,3,4], [5,6,7,8],
[9,10,11,12]], D).
D = [[2.8284271247461903, 4.242640687119285, 3.3166247903554], [2.8284271247461903,
3.1622776601683795, 5.196152422706632], [0.0, 1.4142135623730951, 2.6457513110645907], [8.0,
7.0710678118654755, 7.937253933193772], [16.0, 15.033296378372908, 15.716233645501712]].
```

9.3 Predicting the labels of test points.

Code a predicate `predecir_etiquetas/4` of parameters `predecir_etiquetas(Y_entrenamiento, K, Matriz_resultados, Y_test)` that computes the predictions `Y_test` with respect to the matrix of distances `Matriz_resultados`

computed in the previous exercise. Please observe that here we will no longer need the training instances nor the test ones, as the useful information of them is coded on `Matriz_resultados`. Remember that in each row of `Matriz_resultados` we have the distance of a test point with respect to all the training set points. The label of each test set point corresponds to its row on `Matriz_resultados`. Examples:

```
   predecir_etiquetas(['a','b','a','a','a','a','d'], 3, [[12,32,11,3,44,32,0], [1,2,3,4,5,6,7]],
Etiquetas).
   Etiquetas = ['a','a'].
   predecir_etiquetas(['c','b','a','a','a','a','d'], 1, [[12,32,11,3,44,32,0], [1,2,3,4,5,6,7]],
Etiquetas).
   Etiquetas = ['d','c']
```

It will be easier for you to code the following exercise before completing this exercise.

9.3.1 Predicting the label of each test point.

Code a predicate `predecir_etiqueta/4` of parameters `predecir_etiqueta(Y_entrenamiento, K, Vec_distancias, Etiqueta)` that computes in `Etiqueta` the most closest label according to `Vec_distancias` of each test set point whose useful information is coded on `Vec_distancias`. The labels of each point are on `Y_entrenamiento`. `Vec_distancias` is contained on each row of `Matriz_resultados`. Examples:

```
   predecir_etiqueta(['a','b','a','a','a','a','d'], 3, [12,32,11,3,44,32,0], Etiqueta).
   Etiqueta = 'a'
   predecir_etiqueta(['a','b','a','a','a','a','d'], 1, [12,32,11,3,44,32,0], Etiqueta).
   Etiqueta = 'd'
```

It will be easier for you to code the following exercise before completing this exercise.

Computation of the most relevant labels. Code a predicate `calcular_K_etiquetas_mas_relevantes/4` of parameters `calcular_K_etiquetas_mas_relevantes(Y_entrenamiento, K, Vec_distancias, K_etiquetas)` that builds the label vector `K_etiquetas` of length `K` of the test point ommited in code but whose useful information for the implementation of this predicate is found in the vector of distances `Vec_distancias`. Tip: There are several ways to solve this exercise. Tip: In order to solve this exercise, we suggest to implement support functions as insert in order in a list or deleting the first element of a list. Another tip: It is possible that you may have to code a list of tuples where every tuple consists on the distance of a point and its label. Examples:

```
   calcular_K_etiquetas_mas_relevantes(['a','b','a','b','c','a','d'], 1, [12,32,11,3,1,2,0],
Etiquetas).
   Etiquetas = ['d'].
   calcular_K_etiquetas_mas_relevantes(['a','b','a','a','a','a','d'], 3, [12,32,11,3,44,32,0],
Etiquetas).
   Etiquetas = ['a','a','d'].
```

It will be easier for you to code the following exercise before completing this exercise.

Computation of the most relevant label. Code a predicate `calcular_etiqueta_mas_relevante/2` of parameters `calcular_etiqueta_mas_relevante(K_etiquetas, Etiqueta)` that having a label list `K_etiquetas` extracts the most common one in `Etiqueta`. That is, the label that appears the most number of times in the list `K_etiquetas`. Tip: It is possible that you may have to implement support functions as asking if an element is member of a list. Examples:

```
   calcular_etiqueta_mas_relevante(['a','c','c'], E).
   E = 'c'.
   calcular_etiqueta_mas_relevante(['a','b','a','c','d','b','a','c','a'], E).
```

```
E = 'a'.
```

Tip: It is possible that, in this exercise, you may want to convert the list ['a','b','a','c','d','b','a','c','a'] to a list of tuples like: [[1,'d'],[2,'c'],[2,'b'],[4,'a']]. You can do it via a support function `calcular_contadores/2`. Examples:

```
calcular_contadores(['a','b','a','c','c','c'], [[3, 'c'], [1, 'b'], [2, 'a']]).
L = [[3, 'c'], [1, 'b'], [2, 'a']].
```

9.4 Joining the predicates.

Code a predicate `k_vecinos_proximos/5` of parameters `k_vecinos_proximos/5(X_entrenamiento, Y_entrenamiento, K, X_test, Y_test)` whose implementation contains the predicates `calcular_distancias/3` and `predecir_etiquetas/4`. If you do this: you would have finished the implementation of the classifier! Examples:

```
k_vecinos_proximos([[2,3,1], [3,4,5], [2,3,2], [4,5,6], [6,5,8], [1,0,2], [3,4,3]],
['a','b','a','c','c','a','b'], 1, [[10,7,5], [2,3,1]], Etiquetas).
Etiquetas = ['c','a'].
k_vecinos_proximos([[2,3,1], [3,4,5], [2,3,2], [4,5,6], [6,5,8], [1,0,2], [3,4,3]],
['a','b','a','c','c','a','b'], 7, [[3,4,3], [10,10,10]], Etiquetas).
Etiquetas = ['a','a'].
```

9.5 Application to a real database.

In this section you are going to apply the classifier developed in the previous sections to a classification problem with real data. In particular, you are going to work with the Iris database available in Moodle.

Write the predicate `clasifica_patrones/4` with arguments `clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',K,tasa_aciertos)` that computes the averaged hit rate over the leave-one-out iterations. To load the data, it may be convenient to use the `csv_read_file/3` predicate. To select the test instance on each iteration, the `nth0/4` predicate may be useful. How does the value of K affect the classification?

10 Creation of fractals. (1.4 points)

In this section you are going to create fractals using prolog and the XPCE graphics library. Consider the code included below, obtained from `https://www.rosettacode.org/wiki/Fractal_tree#Prolog`:

```
fractal :-
    new(D, window('Fractal')),
    send(D, size, size(800, 600)),
    drawTree(D, 400, 500, -90, 9),
    send(D, open).

drawTree(_D, _X, _Y, _Angle, 0).

drawTree(D, X1, Y1, Angle, Depth) :-
    X2 is X1 + cos(Angle * pi / 180.0) * Depth * 10.0,
    Y2 is Y1 + sin(Angle * pi / 180.0) * Depth * 10.0,
    new(Line, line(X1, Y1, X2, Y2, none)),
    send(D, display, Line),
    A1 is Angle - 30,
    A2 is Angle + 30,
    From isDepth - 1,
    drawTree(D, X2, Y2, A1, From),
```

```
drawTree(D, X2, Y2, A2, From).
```

Guided generation of a fractal Write a predicate that allows you to obtain a fractal similar to the one in the figure.