Artificial Intelligence. Practice 3.

Pablo Cuesta Sierra, Álvaro Zamanillo Sáez
Group 2351. Pair 5.

April 16, 2022

Contents

1 Reading exercise.

The predicate `slice/4` extracts a slice from a list, between two given indices of that list. The code is the following:

Listing 1: *slice/4*

```
slice([X|_],1,1,[X]).

slice([X|Xs],1,K,[X|Ys]) :-
    K > 1,
    K1 is K - 1,
    slice(Xs,1,K1,Ys).

slice([_|Xs],I,K,Ys) :-
    I > 1,
    I1 is I - 1,
    K1 is K - 1,
    slice(Xs,I1,K1,Ys).
```

1.1 Declarative reading.

The previous definition of `slice/4` has 3 clauses. We will now read the definition declaratively:

1. The slice of a list *L* between indices 1 and 1 is the list containing only the first element of *L*.
2. If `K>1` and `Ys` is the slice of `Xs` between 1 and `K-1`, then `[X|Ys]` is the slice of `[X|Xs]` between 1 and `K`.
3. If `I>1` and `Ys` is the slice of `Xs` between `I-1` and `K-1`, then `Ys` is the slice of `[_|Xs]` between `I` and `K`.

1.2 Procedural reading.

Query: `slice([1, 2, 3, 4], 2, 3, L2).`

1. Does the first clause apply? No: 2 does not unify with 1.
2. Does the second clause apply? No: 2 does not unify with 1.
3. Does the third clause apply? Yes: `Xs:=[2,3,4]`, `I:=2`, `K:=3`, `Ys:=L2`.
4. The three first goals are: `I>1` (true), `I1` unifies with 1 and `K1` unifies with 2.
5. Now, the goal: `slice(Xs, I1, K1, Ys)`, in this case: `slice([2,3,4], 1, 2, L2)`.
   (a) Does the first clause apply? No: 2 does not unify with 1.
   (b) Does the second clause apply? Yes: `X':=2`, `Xs':=[3,4]`, `K':=2`, `[2|Ys']:=L2`.
   (c) The two first goals are: `K'>1` (true), `K1'` unifies with 1.
   (d) Now, the goal: `slice(Xs', 1, K1', Ys')`, in this case: `slice([3,4], 1, Ys')`.
      i. Does the first clause apply? Yes: `X'':=3`, `[X'']:=[3]`.
   (e) The previous goal exits with `Ys'=[3]`
6. The previous goal exits with `L2=[2|Ys']=[2,3]`

Finally the query exits with `L2=[2,3]`.

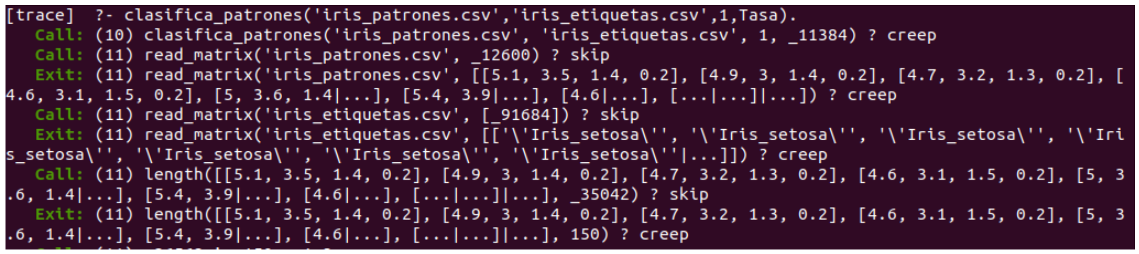
9 K nearest neighbours.

9.5 Application to a real database.

The nearest neighbours algorithm predicts the class to which a specfic instance belongs given a collection of training data. The K refers to the number of elements around the instance (the metric used in this case is the euclidean distance) that are going to be considered to determine its class.

If $K = 1$, the class assigned to our instance is the one from the closest neighbour. As we increase K, the class assigned is the most frequent from the K closest neighbours. Having said this, it is clear that the performance of the algorithm does not only depend on the training data given but also in the value that K takes. The optimal value of K depends on the size of the training data and its accuracy.

In the last line of the next image, it is observed that the size of our training data is 150. In general, larger sets of training data make the optimal value of K smaller. This is due to the fact that it is likely to happen that for any point we want to classify, it exists another in the training set which is actually really close to it, and therefore, they can be considered of the same class with a low probability of error.

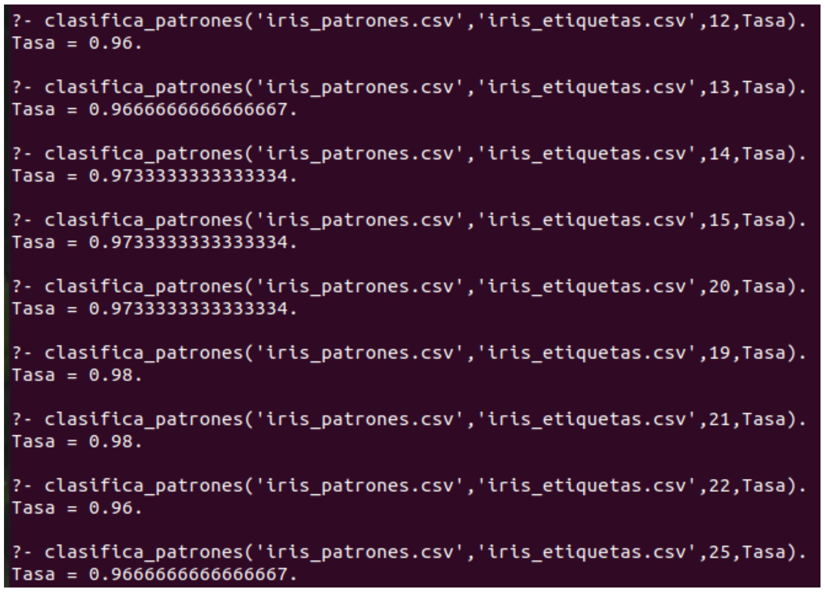


```
[trace]  ?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',1,Tasa).
   Call: (10) clasifica_patrones('iris_patrones.csv', 'iris_etiquetas.csv', 1, _11384) ? creep
   Call: (11) read_matrix('iris_patrones.csv', _12600) ? skip
   Exit: (11) read_matrix('iris_patrones.csv', [[5.1, 3.5, 1.4, 0.2], [4.9, 3, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [
4.6, 3.1, 1.5, 0.2], [5, 3.6, 1.4|...], [5.4, 3.9|...], [4.6|...], [...|...]|...]) ? creep
   Call: (11) read_matrix('iris_etiquetas.csv', [_91684]) ? skip
   Exit: (11) read_matrix('iris_etiquetas.csv', [['\'Iris_setosa\'', '\'Iris_setosa\'', '\'Iris_setosa\'', '\'Iri
s_setosa\'', '\'Iris_setosa\'', '\'Iris_setosa\'', '\'Iris_setosa\''|...]]) ? creep
   Call: (11) length([[5.1, 3.5, 1.4, 0.2], [4.9, 3, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5, 3
.6, 1.4|...], [5.4, 3.9|...], [4.6|...], [...|...]|...], _35042) ? skip
   Exit: (11) length([[5.1, 3.5, 1.4, 0.2], [4.9, 3, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5, 3
.6, 1.4|...], [5.4, 3.9|...], [4.6|...], [...|...]|...], 150) ? creep
```

Figure 1: Partial trace of the K nearest neighbours algorithm. The length of our data is 150.

Firstly, we will comment the effect of K being too large. In our case, selections of K near to 150 would obviously be useless as we would be considering almost every point of the data set as a "near" neighbour. Thus, the distinction to assign a class would be the frequency of the training set rather than the distance of our point to the rest.

On the other hand, extremely small values of K may be susceptible to noise and are too dependant to the training data. As commented before, if the training set is large enough, this may not be a real problem.



```
?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',12,Tasa).
Tasa = 0.96.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',13,Tasa).
Tasa = 0.9666666666666667.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',14,Tasa).
Tasa = 0.9733333333333334.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',15,Tasa).
Tasa = 0.9733333333333334.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',20,Tasa).
Tasa = 0.9733333333333334.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',19,Tasa).
Tasa = 0.98.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',21,Tasa).
Tasa = 0.98.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',22,Tasa).
Tasa = 0.96.

?- clasifica_patrones('iris_patrones.csv','iris_etiquetas.csv',25,Tasa).
Tasa = 0.9666666666666667.
```

Figure 2: Hit rate varying K.

As seen in figure 2, the maximum hit rate (0.98) in our case is obtained with values of $K \in \{19, 21\}$. It is worth mentioning that the performance of the algorithm for small values of K ($K \in \{1, 2, ..., 20\}$) does not differ considerably from the maximum hit rate. Indeed the difference is about just a 2%. This small difference can be explained with the size of the training set; once again, larger training sets make the optimal value of K smaller.

Finally, if we analyze the rest of values (figure 3), we observe that after the values that maximize the hit rate, the performance of the algorithm decreases and once K reaches $2/3$ of the size of the training data, the hit rate is below the 52%. This makes sense, as there are exactly 50 samples of each of the 3 classes. Therefore, when taking $K \geq 100$, we will be considering, at most, only 49 samples of the correct class, which is less than a halve of K. This explains the drastic drop of the hit rate.

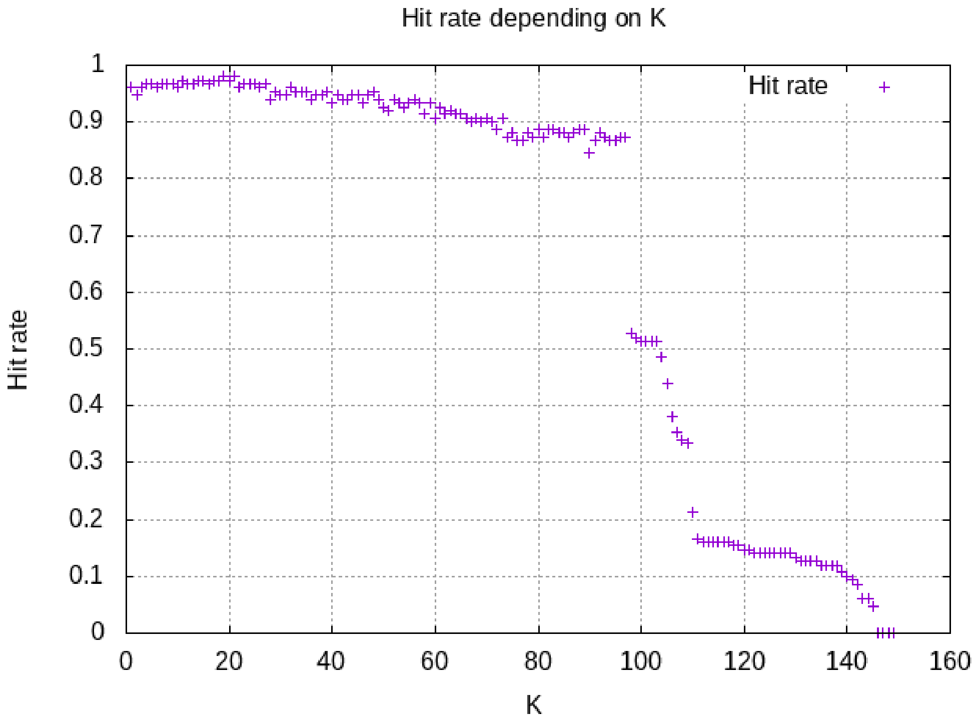


Figure 3: Graph of the hit rate with respect to K.

One would expect that, if there is a higher proportion of a class C_1 within the training data, using K close to the number of samples (150 in this case) would make the hit rate close to the proportion of said class C_1. However, the classes are equally represented in our data (3 classes with 50 samples); therefore, when we take one out in order to make a prediction, for K close to 150, the prediction fails and the hit rate drops to 0.

10 Creation of fractals.

The idea of the fractal proposed in the assignment is the following:

1. Depth 0 paints a line between two given points.
2. Depth $d > 0$ takes two points (P and Q) and paints four fractals of depth $d - 1$ between points P and R, R and S, S and T, T and Q; where points R, S, T are points defined relative to P and Q:

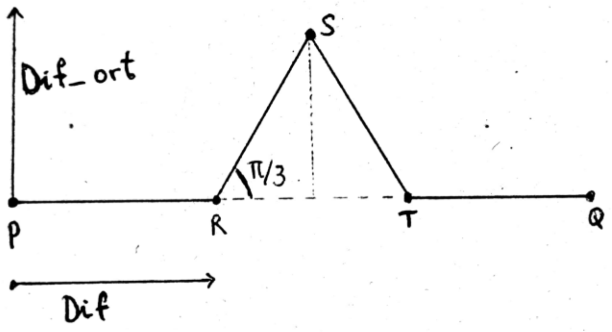


Figure 4: Visual representation of the relative positions of the points.

If points are seen as vectors, with respect to any origin, the definitions are the following (note that P and Q are given):

$$\text{Dif} = \frac{Q - P}{3}$$
$$\text{Dif\_ort} = \mathbf{R}_{\pi/2}(\text{Dif})$$
$$R = P + \text{Dif}$$
$$T = R + \text{Dif}$$
$$S = R + \cos\left(\frac{\pi}{3}\right)\text{Dif} + \sin\left(\frac{\pi}{3}\right)\text{Dif\_ort} = R + \frac{\text{Dif} + \sqrt{3}\text{Dif\_ort}}{2}.$$

Note: $\mathbf{R}_{\pi/2}(v)$ denotes the rotation of a vector $v \in \mathbb{R}^2$ of $\pi/2$ radians with respect to the origin.

Therefore, the base case (depth 0) displays a single line; whereas the general case: depth d calculates the R, S, T points with the given P and Q, and calls the recursive function with depth $d - 1$ on the four lines that were before explained.

Pictures of our fractals:

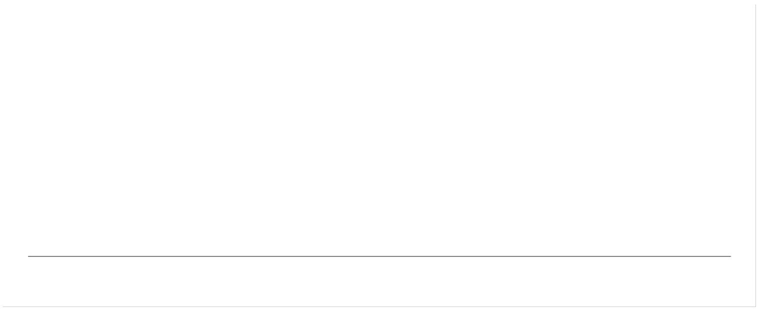


Figure 5: Depth 0.

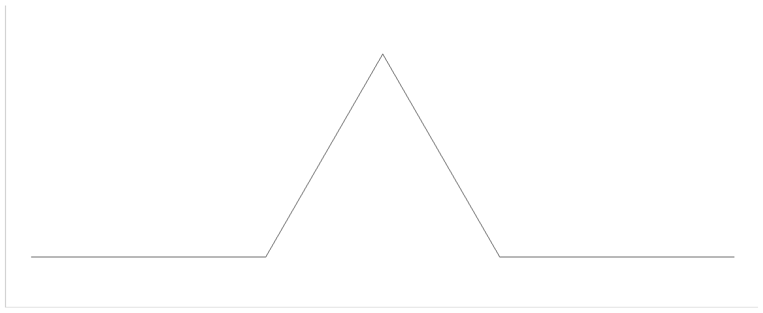


Figure 6: Depth 1.

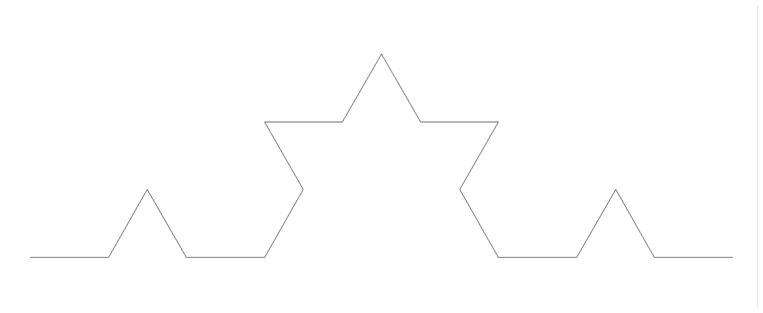


Figure 7: Depth 2.

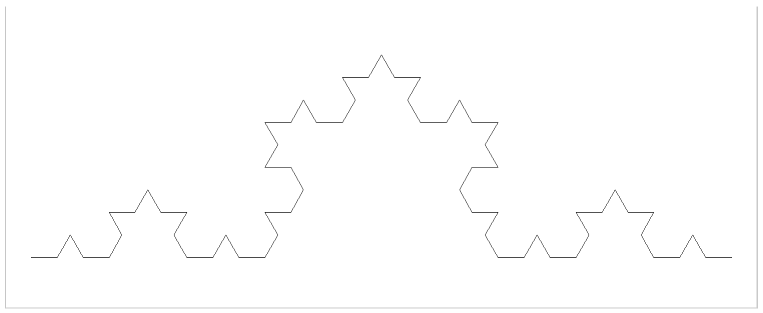


Figure 8: Depth 3.

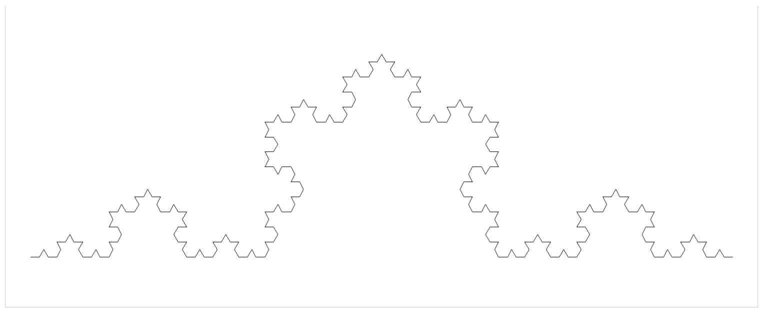


Figure 9: Depth 4.

Appendix A Alternative codification.

Computation of the most relevant labels (9.3.1), another implementation.

In this exercise, we had to implement the predicate `calcular_K_etiquetas_mas_relevantes(Y_entrenamiento, K, Vec_distancias, K_etiquetas)`.

The implementation that we included in the code file makes use of the data structure 'Pairs', from prolog; with built-in predicates. The idea is to sort the tags from `Y_entrenamiento` by the distances in `Vec_distancias` and select the `K` first ones.

Another option, to implement the same idea, would be to manually code the sort predicate:

Listing 2: Alternate implementation of `calcular_K_etiquetas_mas_relevantes/4`.

```
/***************
* EJERCICIO 9e. calcular_K_etiquetas_mas_relevantes/4
*
*       ENTRADA:
*               Y_entrenamiento: Vector de valores alfanumericos de una distribucion
    categorica. Cada etiqueta corresponde a una instancia de X_entrenamiento.
*               K: Numero de valor entero.
*               Vec_distancias: Vector de valores reales correspondiente a una fila de
    Matriz_resultados.
*       SALIDA:
*   K_etiquetas: Vector de valores alfanumericos de una distribucion categorica.
*
****************/
calcular_K_etiquetas_mas_relevantes(Y_Entrenamiento, K, Distancias, Etiquetas):-
    create_tuples(Y_Entrenamiento,Distancias,Tuples),
    sort_tuples(Tuples,SortedTuples),
    first_K_tags(K,SortedTuples,Etiquetas).


/**
* Extract the first K tags of a list of tuples [Tag,Value]
* Input: List of tuples
* Output: List of K tags
*/
first_K_tags(0,_,[]):-!.
first_K_tags(K,[[Tag,_]|SortedTuples],[Tag|Etq]):-
    Ks is K-1,
    first_K_tags(Ks,SortedTuples,Etq).

/**
* create_tuples(Y_Entrenamiento,Distancias,Tuples)
* Creates list of tuples [tag,distance]
* Input: List of tags (['a','b','a']) list of distances ([1,5,4])
* Output: list of tuples ([['a',1],['b',5],['a',4]])
*/
create_tuples([],[],[]):-!.
create_tuples([E|Es],[D|Ds],Tuples):-
    create_tuples(Es,Ds,Rs),
    Tuples=[[E,D]|Rs].

/**
* sort_tuples(UnsortedList, SortedList).
* Sorts a list of elements of type [TAG,DISTANCE]
* Input: unsorted list
* Output: sorted list
*/
sort_tuples([],[]):-!.
sort_tuples([X|Xs],Sorted):-
    sort_tuples(Xs,Xs_Sorted),
    insert_in_order(X,Xs_Sorted,Sorted).

/**
* insert_in_order([TAG,DISTANCE],List,NewSortedList)
* Inserts an element of type [tag,distance ]in a sorted list by distance
* Input: element to insert and sorted list
* Output: New sorted list that contains the element given as input
*/
insert_in_order([Tag,K],[],[[Tag,K]]):-!.
insert_in_order([Tag,K],[[T,N]|Tail],[[Tag,K]|[[T,N]|Tail]]):- K<N,!.
insert_in_order([Tag,K],[[T,N]|Tail],[[T,N]|Xs]):-
    insert_in_order([Tag,K],Tail,Xs).
```