K-Nearest Neighbours

Giuseppe L'Erario

Introduction

The *K-Nearest Neighbours* is one of the simplest machine learning algorithms, particularly useful in pattern recognition. It is called *lazy* algorithm, because it does not learn a function from train dataset, but instead learns it.

In this report is analysed a classification problem on a the famous *Iris Dataset*.

Implementation

The KNN algorithm can be summarized in few steps:

1. Choose the number k of the neighbours;
2. Choose a weight function;
3. Find the k-nearest neighbours of the object to classify;
4. Assign the label based on majority vote.

The KKN algorithm finds the most similar k samples in training dataset to the object to classify. This similarity is determined by the weight function.

The weight function used in this homework are:

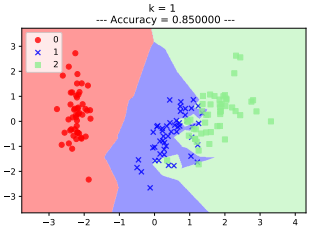
Uniform : all points in each neighbourhood are equally weighted;

Distance : points are weighted with the inverse of the distance. Points closer to the query point have greater influence than further points;

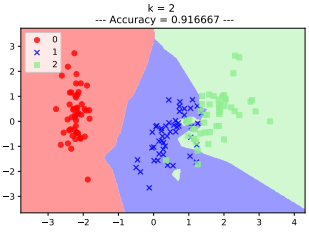
Custom : a gaussian function in the form $w = e^{-\alpha d^2}$, non-linearly dependent from the inverse of the distance.

As usual the dataset is preprocessed and then reduced in dimensionality with PCA.

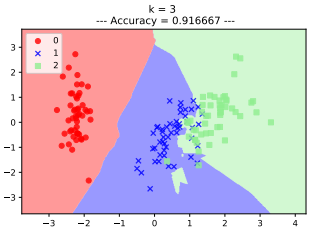
The variation of the decision boundaries can be seen in fig.1. The core of the problem is to find the right number of the neighbors. A small k leads to a classification linked too much with the single sample (in other words, to much sensible to outliers). With the increasing of k the decision boundaries become smoother. They are, indeed, more dependant from a group of samples. A too high k could lead to a too generalized classification: the algorithm will consider samples that are too distant, and probably are uncorrelated, from the query point.



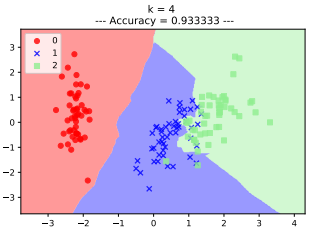
(a) k = 1



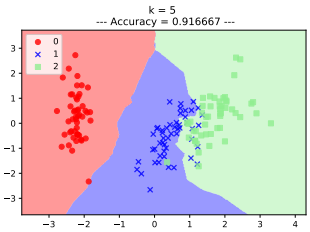
(b) k = 2



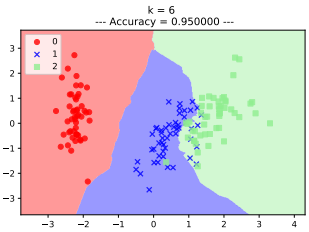
(c) k = 3



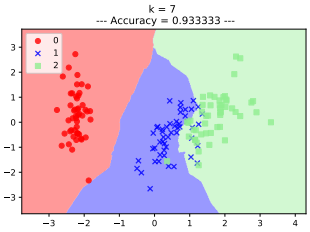
(d) k = 4



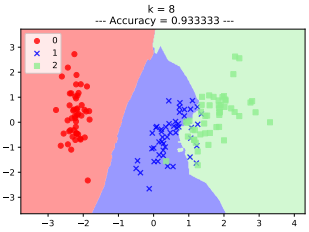
(e) k = 5



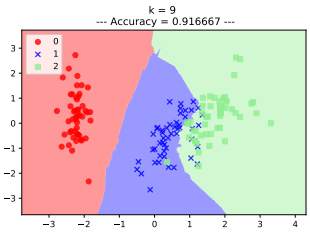
(f) k = 6



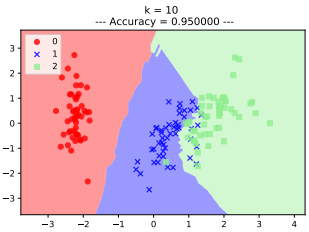
(g) k = 7



(h) k = 8



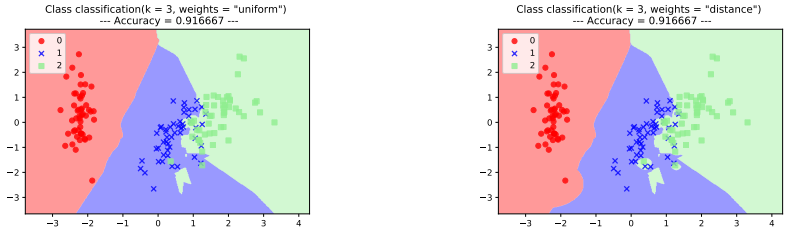
(i) k = 9



(j) k = 10

Figure 1: Variation of decision boundaries with the number of neighbors

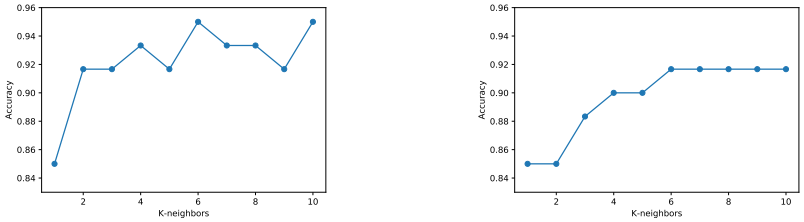
For $k = 3$ neighbors are used the *'uniform* and *'distance'* weight function (fig.2). The accuracy is the same.



(a) Uniform (b) Distance

Figure 2: Weight functions

In fig.3 can be seen that the accuracy wrt. the number k.



(a) Uniform weight function (b) Distance weight function

Figure 3: Accuracy with the number of neighbors

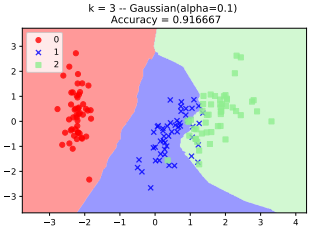
Gaussian weight function

With a custom weight function we can tune how much the distance parameter is important in the classification problem, making this weight function more versatile. With the increasing of α, the distance parameter is more and more important.

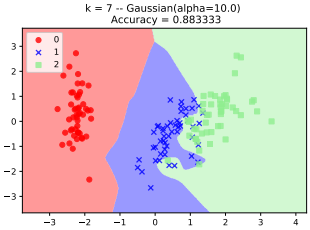
In fig.4 are plotted the decision boundaries, for $k = 7$, on varying of α. In correspondence of $\alpha = 1000$ the fig.4(d) shows a visible difference in the decision boundaries. With huge values of α the weight function comes to zero quickly: the "influence zone" of the samples *1* and *2* becomes smaller[1], only the closer samples are classified as belonging to the same class.

In fig.5 is plotted the accuracy on varying of α: the best performance are obtained with low value of α.

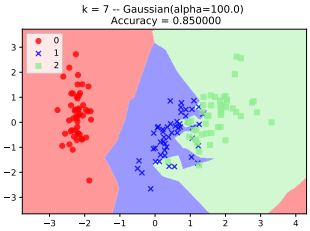
[1]Probably the red zone, relative to *0* group, is the last zone drawed



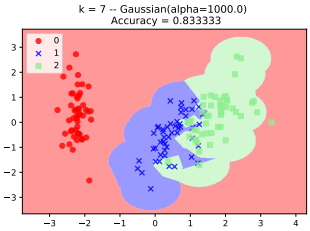
(a) $\alpha = 0.1$



(b) $\alpha = 10$



(c) $\alpha = 100$



(d) $\alpha = 1000$

Figure 4: Gaussian weight functions

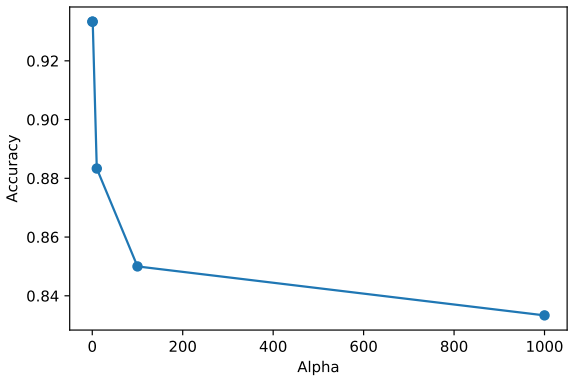


Figure 5: Accuracy on varying of α.

Best model

We can find the best model with a grid search, including number k of neighbors, weight functions and α (using gaussian weight function).

The model with the best performance is the one shown in fig.6.

However, there is not a univocal best model with a precise weight function. In fact, it can be different from one simulation to another (due to the necessary random split of the data), or, in the same simulation, there also can be models with the same accuracy score. The shape of the boundaries are often very similar between the models.

For example, after a different simulation comes out that the best model is one

with a gaussian weight function (fig.7), but clearly leads to a misclassification of class *0*.

The only way to find a good model is to measure its accuracy, trying not to fall in overfitting issues.

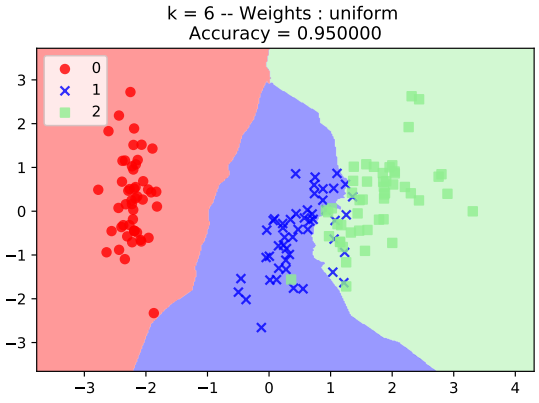


Figure 6: Best model.

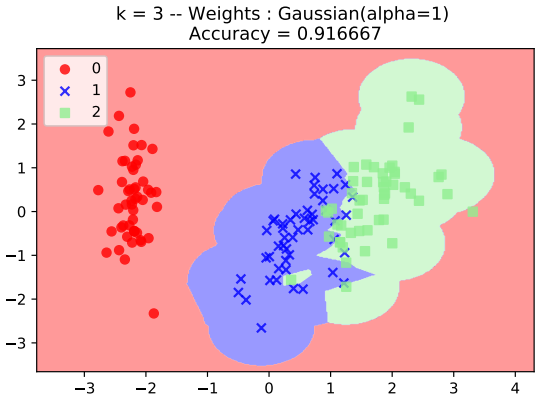


Figure 7: Best model with gaussian function.