1 Implementation of k-NN

Since k-NN needs just to store the training data for then performing predictions, there is not any particular way of “training” the model, we just need to store somewhere our training samples (instance-based ML model). However, regarding the implementation of the predicting phase we had to made some decisions in the minkowski_dist function. We explored two possible approaches:

- Exploiting broadcasting to obtain a giant tridimensional matrix where $M[i][j][k] = x_{i,[k]} - x_{j,[k]}$ where i indicates the testing sample, j the training sample, and k indicates the index of the single feature of a sample.
- Using a for loop and exploiting broadcasting to obtain in one line all the distances of a given test sample from the training samples

Running some raw benchmarks with our chosen model configuration and with different test set sizes (synthethic datasets) on our machine we found out that the second approach was faster.

Test samples	102008	70966	39975	25584
No loop	13.4s	5.8s	2.7s	1.7s
One loop	5.1s	3.5s	2s	1.3s

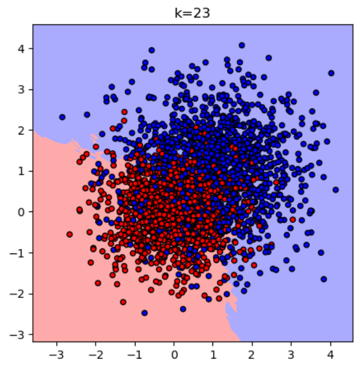
Furthermore, we may also encounter some memory issues as the number of test samples grows, since we always need to generate a matrix of float with dimensions MxN where M=number of training samples, N=number of testing samples. This problem becomes more severe using the first approach, since before generating our distance_matrix we first generate an MxNxD tridimensional matrix, where D=dimensionality of each sample.

2 Evaluation for Hyperparameter Tuning

To decide which values of k and p best fit our model, we need to perform hyperparameter tuning using a validation set that is different from the training set. In this case, we have both a training and a validation set, so we will train our model using the training set, then make predictions on the validation set, varying k and p. Finally, we will choose the pair (k,p) that gives us the best accuracy on our validation set. To do this, we choose a range of k up to 3.6% of our training samples (from 1 to 100), which seems to be a reasonable range. We also try different values of p, which is used to compute the Minkowski distance.

```
(k, p, accuracy) [23, 5, 0.827]
```

As we can see, the best accuracy (82.7%) is obtained with k=23 and p=5, thus we will choose these as hyperparameters for our model.
By selecting those parameters, we derive the decision boundaries illustrated in the image on the right.
Another interesting configuration was $k = 27$ and $p = 2$ (82.5%) which was faster due to the lower p parameter that corresponds to the exponent of the Minkowski distance.

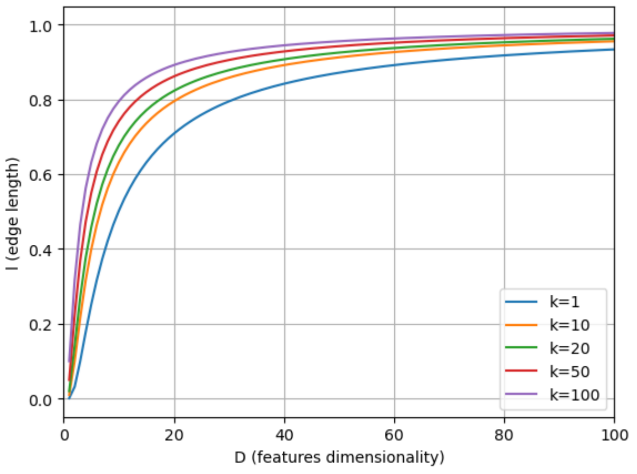


3 Explaining curse of dimensionality in KNN

K-NN is strongly affected by the *curse of dimensionality*, since as dimensionality of data grows, data becomes increasingly sparse, leading to the loss of the concept of local neighborhoods. We now explain this phenomenon using an example.
Assuming we have a D-dimensional hypercube (our sample space) with all sides of length 1, and we sample the training data uniformly from this hypercube, we want to analyze the edge length l of the smallest hypercube that contains all k-nearest neighbors of a test point, with respect to D(feature dimensionality), k, N(number of training samples).

- Since the data is uniformly distributed in our hypercube (volume $V_{tot} = 1$), we can approximate the volume of the smallest hypercube that contains a neighborhood made of k neighbors as $V_{fraction} = \frac{k}{N} V_{tot} = \frac{k}{N}$
- We can now recover l from the volume as $\left(\frac{k}{N}\right)^{\frac{1}{D}}$

We now plot l as a function of D, varying k and fixing $N = 1000$, in order to see how the dimensionality growth affects the edge length of the hypercube containing the neighborhood.



For example, we can see that with $k = 10$ (so we use a local neighborhood that covers 1% of our data), in order to obtain locality (with a rather small dimensionality of the features, $D = 20$), we have to cover the 80% of the range of each input variable. We can see that our data becomes sparse, so our neighborhoods are no longer “local”, and the performance of k-nn decreases dramatically.