Supervized Learning: Assignment 1

Thomas Nedelec
MSc Machine Learning
thomas.nedelec.15@ucl.ac.uk

Michal Daniluk
MSc Machine Learning
michal.daniluk.15@ucl.ac.uk

1 Exercise 1: Least Square Regression: effect of the training set size

We generated a noisy random data set, containing 600 samples, as $y_i = x_i' w + n_i$, where each x_i and n_i are drawn from the standard normal distribution. We splited the data into a training set of size 100 a test set of size 500. We train a linear regression model by Least Square Regression. To evaluate the model, we compute the mean squared error on both the training and test sets. Figure 12 shows examples of different training sets and estimated regression w.

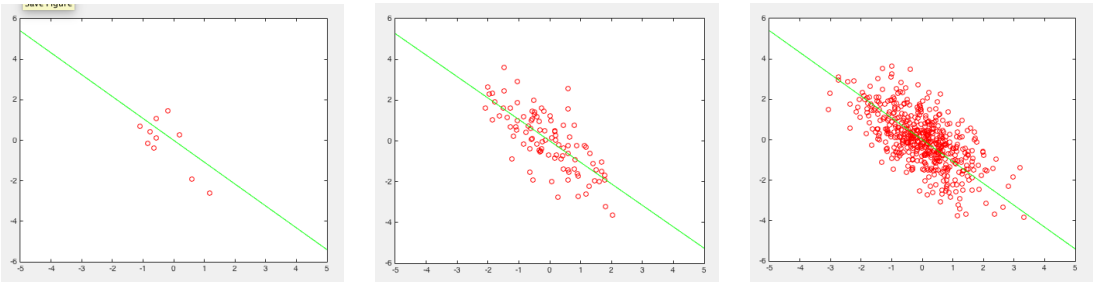


Figure 1: (a) training set of 10 points (b) training set of 100 points (c) the entire data set

	Train	Test
10	0.89	1.12
100	1.00	1.02

Figure 2: Average mean square error for both training and test sets, on both 10 and 100 element-size training set with running 200 times the algorithm.

Figure 2 illustrates the "2 x 2" table of averages mean square error for both training and test sets, on both 10 and 100 element-size training set. The average mean square run is computed with

running 200 times the algorithm.
We can observe that increasing the size of training set lets to decrease the average mean error on the test set. Indeed, a larger training set lets us to have more information on the distribution and lets to a better generalization of the model. With a small dataset, we do not have enough points to predict well datas in the test set. However, it increased the average mean error on train set because it is harder to have a line explaining all the data points. With increasing the size of training set, the average error on the test and train set becomes similar: the train set is big enough to predict well the test set. The average error on test set will remain a little bit higher than on the train set: normal because it is easy to predict the values of the training set: we have already seen them.

2 Exercise 2: Least Square Regression: effect of dimensionality

We repeated the task of exercise 1, but with 10-dimensional data sets. Figure 3 illustrates the "2 x 2" table of averages mean square error for both training and test sets, on both 10 and 100 element-size training set.

	Train	Test
10	0.0	2505.1
100	0.9	13.6

Figure 3: Average mean square error for 10-dimensional data for both training and test sets, on both 10 and 100 element-size training set with running 200 times the algorithm.

For 10-sample training set, we obtain the average mean square error 0 for train set and 2505.1 for test set. We have 10-dimensional data and 10 samples, so we can accurately predict y_i without any errors. However, the prediction is overfitted to the training data and we have a large error for test set. Increasing number of samples to 100 helps to decrease average error on test set because the biggest size of the training set lets to a better generalization ability. Average error for train set is not anymore 0 because we can't find a space of dimension 10 containing 100 points (except if they only form a 10-dimension space of course).

3 Exercise 3: Ridge regression

The regularization is a way to reduce the freedom of the classifier in order to improve the generalization and reach a better test set average error.
To implement ridge regression, we would like to minimize according to w the cost function,

$$\gamma w^T w + \frac{1}{l}\sum_{i=1}^{l}(x_i^T w - y_i)^2.$$

Using the notation $X = (x_1, x_2, ..., x_l)^T$, a matrix containing the training sample vectors as its rows, we can rewrite the cost function as:

$$\gamma w^T w + \frac{1}{l} Tr((Xw - Y)^T (Xw - Y))$$

Taking derivative according to w, we reach:

$$2\gamma w + 2\frac{1}{l}(X^T X w - 2Y^T X) = 0 \tag{1}$$

To reach the conditions for optimality, we set the previous equation to zero and we reach:

$$w = (\gamma l Id + X^T X)^{-1} Y^T X \tag{2}$$

because if $\gamma * l \neq 0$, $(\gamma l Id + X^T X)$ is non-singular.

$\gamma l Id + X^T X$ is symetric.

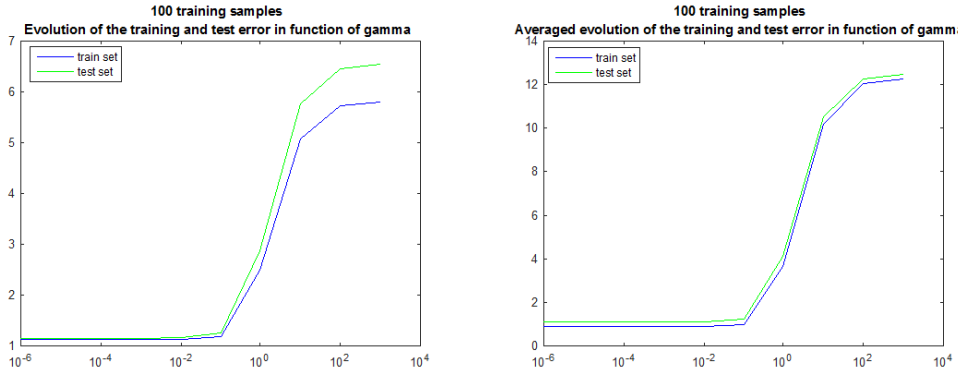
We consider $u \in \mathbb{R}^d$:

$$\begin{aligned} \langle (\gamma l Id + X^T X)u, u \rangle &= \langle \gamma l u, u \rangle + \langle X^T X, u \rangle \\ &= \gamma l \langle u, u \rangle + \langle Xu, Xu \rangle \\ &= \gamma l ||u||^2 + ||Xu||^2 > 0 \ for \ u \neq 0 \end{aligned}$$

Thus $\gamma l Id + X^T X$ is definite positve.

4 Effect of the regularisation parameter

In this exercise we perform ridge regression on the data sets with the regularisation parameter ranging from 10^{-6} up to 10^3. First, the graph 4 shows the training error and test error in function of gamma for 100 training samples. If gamma is too high, the training error and test error increase dramatically. It is intuitive because when gamma is high the algorithm is far more likely to minimize $||w||$ rather than the training error.



(a) One run.

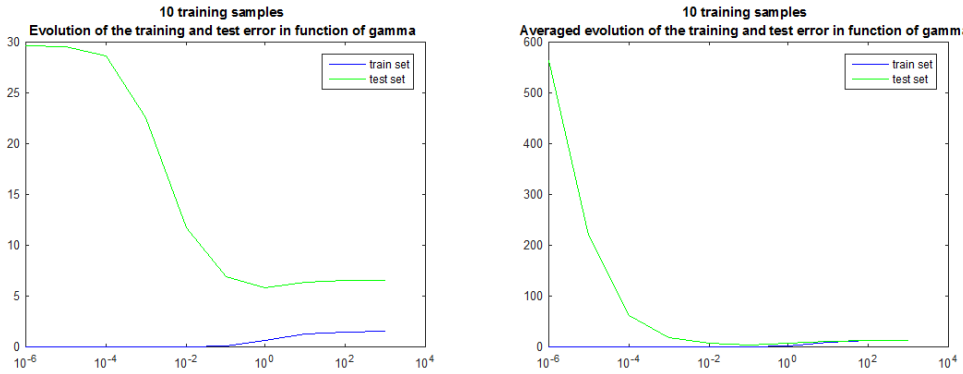
(b) Averaged on 200 runs.

Figure 4: Evaluation of the training error and test error in function of gamma for 100 training samples.

We have an optimal point corresponding to inflection point of the curve in order to select γ. Nevertheless, the training set error is not a sufficiently good guidance to select the regularization parameter because the charts are quite different.

However, when we average the training error on 200 runs, we can observe that now the test error and the training error are now almost identical. A way to select the optimal γ would be to plot the error on the training set averaged on a certain number of runs of the algorithm and select the optimal γ on this curve.

Figure 5 shows the training error and test error in function of gamma for 10 training samples. It



(a) One run.

(b) Averaged on 200 runs.

Figure 5: Evaluation of the training error and test error in function of gamma for 10 training samples.

looks different than for 100 training samples. Regularization prevented model from overfitting. The test error decreased with increasing regularization parameter until it reached the minimal value for $\gamma = 10^{-1}$. Basically trying to train a regression in dimension 10 with 10 points does not mean a lot of things. However, we can notice some interesting things.

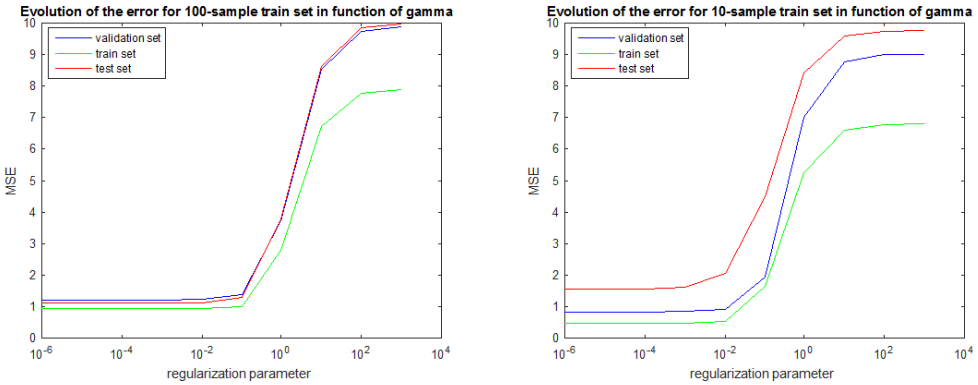
Regularization prevents algorithm to overfit the data set. In the case of 10 points, it is difficult to choose the optimal value of regularization parameter γ. Minimizing according to gamma would lead in this case to not take into account the data and choose w=0. T

The most important thing of this part is to see that with a reasonable size of data set there exists an optimal gamma that lets to lead to a good generalization ability.

5 Tuning the regularization parameter using a validation set

The goal of this exercise is to use a validation set to tune the regularization parameter for training set with 10 and 100 samples. Figure 6 shows the result error on training, validation and test sets for both data sets.

For 100 sample data set, the averaged validation error is almost identical as averaged test error, so it is a good approximation of the error on the test set. The drawback of this method is that we are losing information on the training data because we are only considering 4/5 of the data available to train our regression.



(a) 100 sample set (b) 10 sample set

Figure 6: Average MSE in function of gamma .

The averaged best γ on the validation set for 200 experiments are : $\gamma^{(100)} = 0.0417$ and $\gamma^{(10)} = 50.1787$. The averaged $\gamma^{(10)}$ is much bigger than $\gamma^{(100)}$. It is still because with only 10 points, ridge regression just leads to not take into account the training data. Notice that the training set has only 8 samples and the dimension is 10, so we have even less points than dimensions.

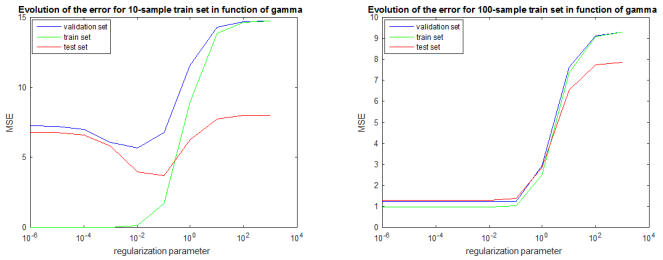
We performed RR on the full training set with 100 samples selecting the regularization parameter that gives rise to the smallest validation set error.Training error and validation error are equal to 0.2027 and 3.2162 respectively for 10 sample set and 0.8513 and 1.1106 for 100 sample set. It's obvious that for bigger training set we reached better results.

It is better to tune regularization parameter on validation set, which is independent from training set but we are losing information compare to having access to the full training set. That is why we need to introduce cross-validation.

With changing to one-dimensional datasets, regularization parameter increased for both 10 and 100 samples. That's because 1 dimension gives less of freedom and it is easier to overfit than for 10-dimensional dataset, so bigger γ leads to a good generalization ability.

6 Tuning the regularization parameter using cross validation

We use 5-fold cross validation to tune the regularization parameter. Figure 7 shows cross-validation score on top of the training and test set error for different values $\gamma = \{10^{-6}, 10^{-5}, \ldots, 10^{3}\}$ of the regularization parameter.



(a) 10-sample training set (b) 100-sample training set

Figure 7: Evaluation of the mean square error in function of gamma for different number of training examples.

Cross validation error is a good estimation of error for test set for both 100 samples and we do not lose anymore data in the training set.

7 Comparison of γ tuning methods

The goal of this exercise is to generate 200 such data as in above exercises and tune the regularization parameter γ using three methods. The results of average test error and standard deviation of the errors for 10 sample set-up are presented below:

1. By minimizing the training error
   - Test set error: 163.9892
   - Standard deviation of error: 508.3517
2. By minimizing the validation error (80% / 20 % split)
   - Test set error: 35.5974
   - Standard deviation of error: 192.3498
3. By minimizing the 5-fold cross validation error
   - Test set error: 6.2890
   - Standard deviation of error: 10.1677

The best results gives tuning regularization by minimizing the 5-fold cross validation error. Minimizing the 5-fold cross validation error gives us reasonable results despite the fact that training set size is very small. Minimizing the training error doesn't seem to be a good approach, because the test set error and standard deviation of error are very big. Indeed we do not have enough information to hope to predict well on the test set. Using only one validation set gives us better results but still not such as good as using cross validation method. Indeed in the validation set method, we are losing information compare to cross-validation methods.

The results of average test error and standard deviation of the errors for 100 sample set-up are presented below:

1. By minimizing the training error
   - Test set error: 1.1202

- Standard deviation of error: 0.0831

2. By minimizing the validation error (80% / 20 % split)
   - Test set error: 1.1529
   - Standard deviation of error: 0.1283
3. By minimizing the 5-fold cross validation error
   - Test set error: 1.1208
   - Standard deviation of error: 0.0848

First of all, the results are much better than for 10 sample set-up. We increased the size of the training set and now our model has a good generalization ability. Secondly, differences between results for each method are very small. It's because, we have a big enough train set and regularization isn't really necessary. We still can notice that cross validation is better than validation because thanks to cross-validation we do not need to loose information compare to cross-validation.

8 Load the data

We loaded the boston data set into Matlab.

9 Baseline versus full linear regression.

In this exercise we try a baseline method works for a problem instead of use all of our attributes for prediction. Firstly, we implemented Naive Regression, (approach whose name is well chosen..) which predicts with the mean y-value on the training set. The averaged training and test errors (after 20 runs) are equal to 84.5706 and 84.3933 respectively.

Then Linear Regression with single attributes was implemented. For each of the thirteen attributes, we performed a linear regression using only the single attribute but incorporating a bias term. Results are shown in figure 10.

In general, results are better than for Naive Regression, but they depend on which attribute we selected. It means that some attributes are more important (have more information that we can use to distinguish classes) that others. For example, Linear Regression with 13th attribute has error on test set: 39.1051 in contrast to with 4th attribute: 83.5478.

Linear Regression using all attributes outperforms obviously any of the individual regressors (MSE on test set: 27.8889.

10 Kernel Ridge Regression

In this exercise we will perform kernel ridge regression on the Boston data set with the Gaussian kernel, which is defined as:

$$K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

First, we compute the kernel matrix for all the data (when we implement cross-validation we select indexes which are meaningful for the training set and validation set).

Then, we used *kridgereg.m* to perform kernel ridge regression using equation $\alpha^* = (K + \gamma l I_l)^{-1} y$. We don't explicitly use a matrix inverse instead use the matrix left division operator. According to

the Matlab documentation, *mldivide* is a more efficient way to solve a linear equation $Ax = b$. If the solution does not exist or if it is not unique, the *mldivide* operator issues a warning.

Then, we created a function called *dualcost.m* to calculate the Mean Squared Error (MSE) using equation:

$$mse = \frac{1}{l}(K_{test}\alpha - y)'(K_{test}\alpha - y) \tag{3}$$

We perform kernel ridge regression on the training set using five-fold cross-validation with different values of γ and σ. Figure 8 shows the cross-validation error as a function of γ and σ. The best

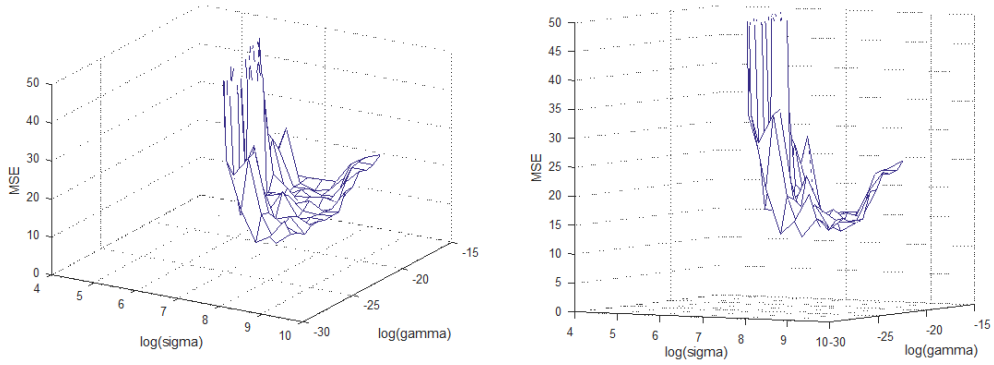


Figure 8: Cross-validation error as a function of γ and σ.

results are for values $\gamma = 2^{-31}$ and $\sigma = 2^{12}$. The MSE on the training data = 8.8392 and on the test set = 13.8215.

Then, we repeat those steps over 20 random splits of data. Our results for exercise 9 and 10 are summarized in the table:

Method	**MSE train**	**MSE test**
Naive Regression	84.5706 ± 3.7489	84.3933 ± 7.3810
Linear Regression (attribute 1)	71.9206 ± 3.7157	71.7623± 7.2524
Linear Regression (attribute 2)	73.4863 ± 4.2998	73.7122± 8.4777
Linear Regression (attribute 3)	64.8467 ± 4.6972	64.7529± 9.1959
Linear Regression (attribute 4)	81.3032 ± 3.2466	83.5478± 6.3118
Linear Regression (attribute 5)	69.1346 ± 3.6979	69.0250± 7.8448
Linear Regression (attribute 6)	43.0045 ± 4.6585	45.4416± 9.3804
Linear Regression (attribute 7)	72.6169 ± 4.3741	72.4109± 8.5975
Linear Regression (attribute 8)	79.1138 ± 4.3852	79.6780± 8.5893
Linear Regression (attribute 9)	72.4303 ± 4.2039	71.9495± 8.2573
Linear Regression (attribute 10)	66.3728 ± 4.5274	65.3491± 8.9139
Linear Regression (attribute 11)	63.3372 ± 3.9821	61.7821± 7.9450
Linear Regression (attribute 12)	75.1591 ± 3.8102	75.1225± 7.5406
Linear Regression (attribute 13)	38.2716 ± 2.1729	39.1051± 4.2303
Linear Regression (all attributes)	23.1232 ± 2.9711	27.8889± 6.2950
Kernel Ridge Regression	7.9308±1.3363	13.7928± 2.4398

11 Part 2

11.1 Simulating 1-Nearest Neighbor classifier with Linear Classifier

According to the previous part,

$$\alpha^* = (K + \gamma * l * I_l)^{-1} y$$

If β is big, $K_\beta(x,t) = exp(-\beta * ||x-t||^2) = 0$ if x is very different from t and not equal to zero if x is close from t. If β is really too big everything is equal to if x different from t. Thus, there is an optimal beta to try to consider the closest point with a linear classifier.

11.2 Using a perceptron to learn a linear classifier with a bias

$$f_{w,b}(x) = sign(w^T x + b)$$

To learn also the bias, we use the algorithm presented in different articles:

Algorithm 1 Perceptron

Input $D : trainingset, maxIter, learningRate$

$w_d \leftarrow 0$
initialize w=0, $b_0 = 0$, k=0
$R = max\{||x_i||, 1 \leq i \leq l\}$
While mistakes are made in the for loop repeat:
for $iter = 1 \dots l$ **do**
    if $y_i(w_k^T x_i + b_k) \leq 0$ **then**
        $w_{k+1} = w_k + y_i x_i$
        $b_{k+1} = b_k + y_i$
        k=k+1
    end if
end for
return w_d

According to us, this algorithm should increase of decrease the number of error depending on the dataset.
Indeed, if the dataset if linearly separable with b=0, it should be better to start with directly b=0 and does not tune a bias.
However, in the case where the data are translated from 0, introducing a bias can reduce the number of mistakes.
That is why we think that it should depend on the dataset.

11.3 Kernel modification

We consider the function: $K_c(x,z) = c + \sum_{i=1}^{n} x_i z_i$.

We assume $c \geq 0$.

We consider $\phi : \mathbb{R}^n \to \mathbb{R}^{n+1}$ such that:

$$\phi(x_1, ..., x_n) = (x_1, ..., x_n, \sqrt{(c)})$$

and we reach:

$$K(x,z) = \langle \phi(x), \phi(z) \rangle \text{ in } \mathbb{R}^{n+1} \tag{4}$$

Thus, for $c \in \mathbb{R}^+$ K_c is a positive semidefinite Kernel.

We suppose we do linear regression with K_c.

11.4 Sparse learning

We implemented the four different algorithms **perceptron, winnow, least squares and 1-nearest neighbours**.

To compute an estimator of the sample complexity we generate several test data sets (30 for winnow for instance) and we average the error on this 30 data sets. While this averaged error is higher than 0.1 we continue to increase the size of the training set. This is how we manage to draw the curve of the evolution of the sample complexity according to the dimension.

The first algorithm we implemented is perceptron.

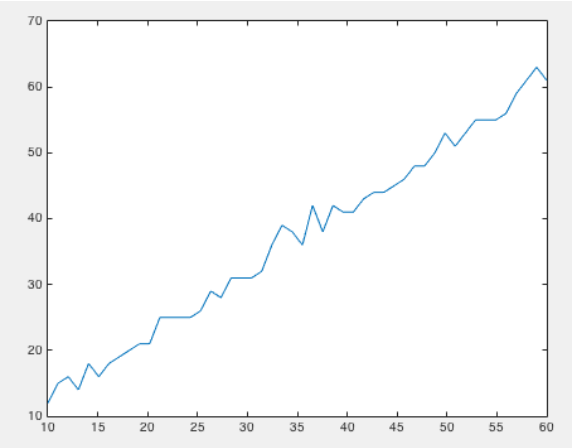


Figure 9: Evolution of sample complexity according to dimension for perceptron

M seems to grow linearly as a function of n with a slope close to one. It is quite good algorithm in terms of sample complexity.

Concerning least square algorithm, the relationship seems also to be linear with a lower slope. The slope is around 1/2. Thus linear regression seems to have a better sample complexity than perceptron.

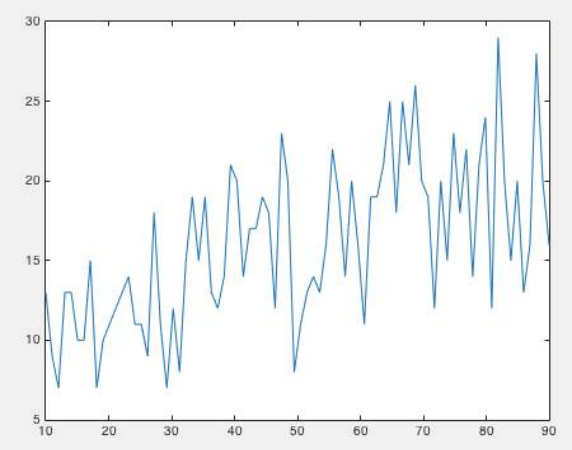


Figure 10: Evolution of sample complexity according to dimension for least square algorithm

The sample complexity of winnow seems to be very good and actually is the best of the 4 algorithms. The relationship seems to be $m = O(log(n))$.

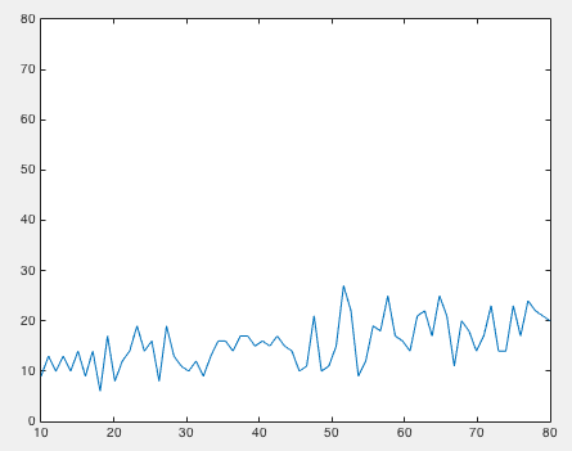


Figure 11: Evolution of sample complexity according to dimension for winnow algorithm

N-Nearest Neighbours is a very poor algorithm in this case. Indeed, N-Nearest Neighbours should use all the features with equal importance to classify whereas only the first one is necessary. We tried to draw the curve for a few points but the relationship seems to be exponential and for high points the algorithm never reaches at ecah iteration a generalisation error lower than 0.1.

Figure 12: Evolution of sample complexity according to dimension for perceptron

Thus, if we need to classify the algorithm according to their sample complexity (from the best to the worse) in this particular case where the label is the first column of the training data we reach:

$$winnowalgorithm \geq leastsquare \geq perceptron \geq NearestNeighbour$$