Politecnico di Torino

STATISTICAL LEARNING AND NEURAL NETWORKS

A.A. 2021/2022

Professors ENRICO MAGLI, DIEGO VALSESIA

Students FRANCESCO DONATO (s304810)

MARCO COLOCRESE (s301227)

COMPUTER LAB1

Exercise 1

In the first exercise, we implemented a basic algorithm using the K-NN classifier.

For each K value (from 1 to 100) we computed the vector containing all distances between the point we were analyzing and all the training set points; then we ranked it. We did it for all test set points, hence we developed three nested 'for' loops. Having the sorted distance vector (regarding the current point), we counted the occurrences of each class among the nearest K points. The algorithm output is the class with the most occurrences, while in unbiased situations we let the algorithm choose a random value between the two classes.

The accuracy was evaluated by computing the right/total ratio.

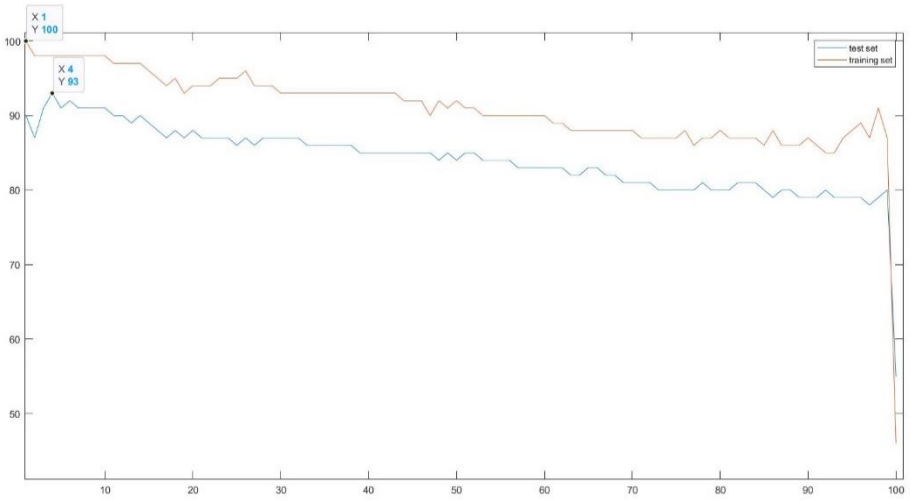

Fig. 1.1 - Accuracy of training set and test set as function of K (exercise 1).

As predictable, for the training set values, the accuracy is overall higher, and in particular, results to be 100% for K=1 (due to the fact that the nearest point is the point itself).

Regarding this particular value, we should observe the overfitting phenomenon in the Test set case, as the accuracy of the test set is different from the one on the training set.

In both cases, increasing the K value, the accuracy tends to decrease significantly, falling to around 50% when K equals to 100. This accuracy value fluctuated around 50% as we repeatedly ran the program, because of our choice of assigning a random class to the point.

This decrease shows the underfitting phenomenon, as with high values of K our algorithm is averaging too much.

Probably, a good choice for K is a small value (4 or 5), which makes the test accuracy reach its maximum value (93%).

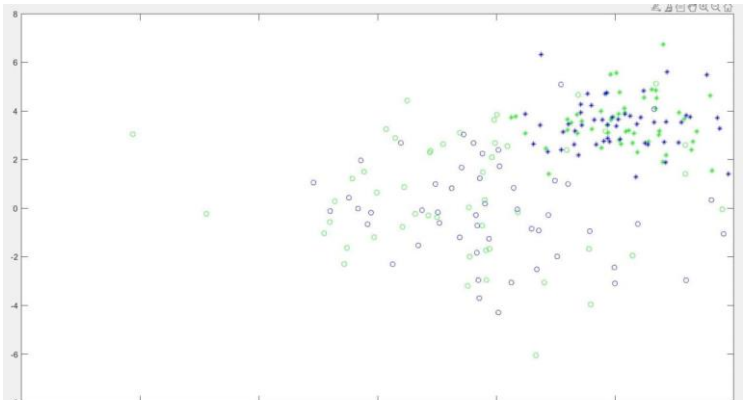

Fig. 1.2 – Data visualization.

Training set – Class1
Test set – Class1
Training set – Class2
Test set – Class2

We can see that the error between the training set and the test set, regarding the accuracy, in the cases of overfitting and underfitting, is not significant. In our opinion, this derives from the distribution homogeneity (FIG. 1.2) of the points belonging to the two different classes. In other words, the clusters of data look almost isolated from each other (not mixed).

Exercise 2

This exercise is very similar to the first one, in fact we implemented the same algorithm, but with few modifications related to the different number of features. In particular, distances were calculated using 5 “coordinates”. Having only the dataset, we divided it into training set and test set, specifically 85% for the training set and 15% for the test set.

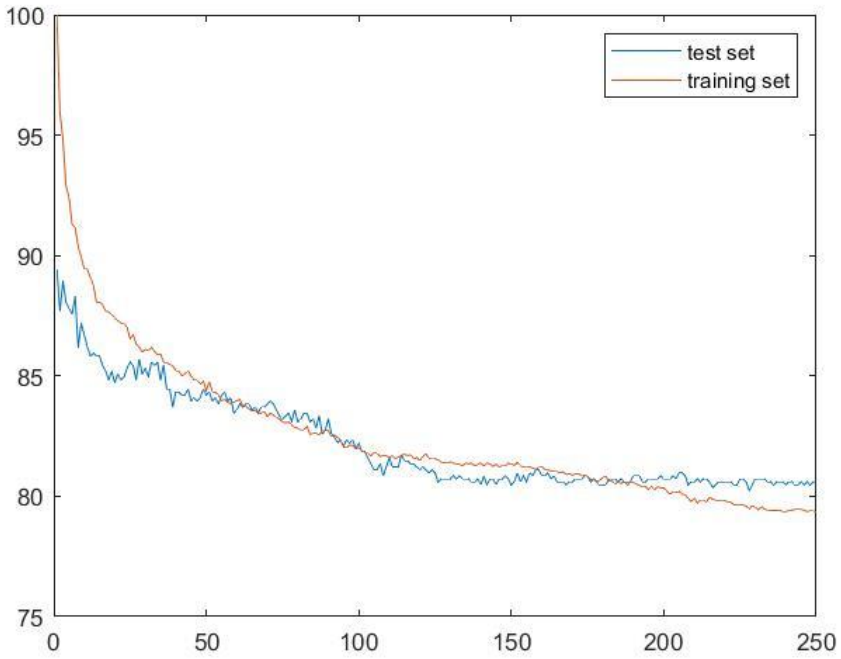

Fig. 1.3 – Accuracy of training set and test set as function of K (exercise 2).

For high values of K, we can observe the underfitting phenomenon, as the algorithm averages too much bringing to a decrease in terms of accuracy.

For small values of K, we notice the overfitting phenomenon, as the two accuracies differ from each other of a value of about 10%. A good model can show a difference of 2/3% between the two accuracies. So we need to find a trade-off between a small overfitting and a good accuracy.

Also in this case, a good choice for K would be a small value (between 4 and 7).

Exercise 3

As in previous exercises, we applied the K-NN classifier to solve this classification problem. For each test set's element we computed the "distances" (representing the absolute values of differences between the two RSSI values of the considered user and points belonging to the training set), we ordered the distance vectors and counted the occurrences of each class related to the K nearest points of each sensor.

In other words, we considered K*7 values for each user (K for each sensor). The classification output was obtained finding the output class with the most occurrences.

We implemented the algorithm with K going from 1 to 120 (because we had 120 points, hence the total number of measurements is 24*5).

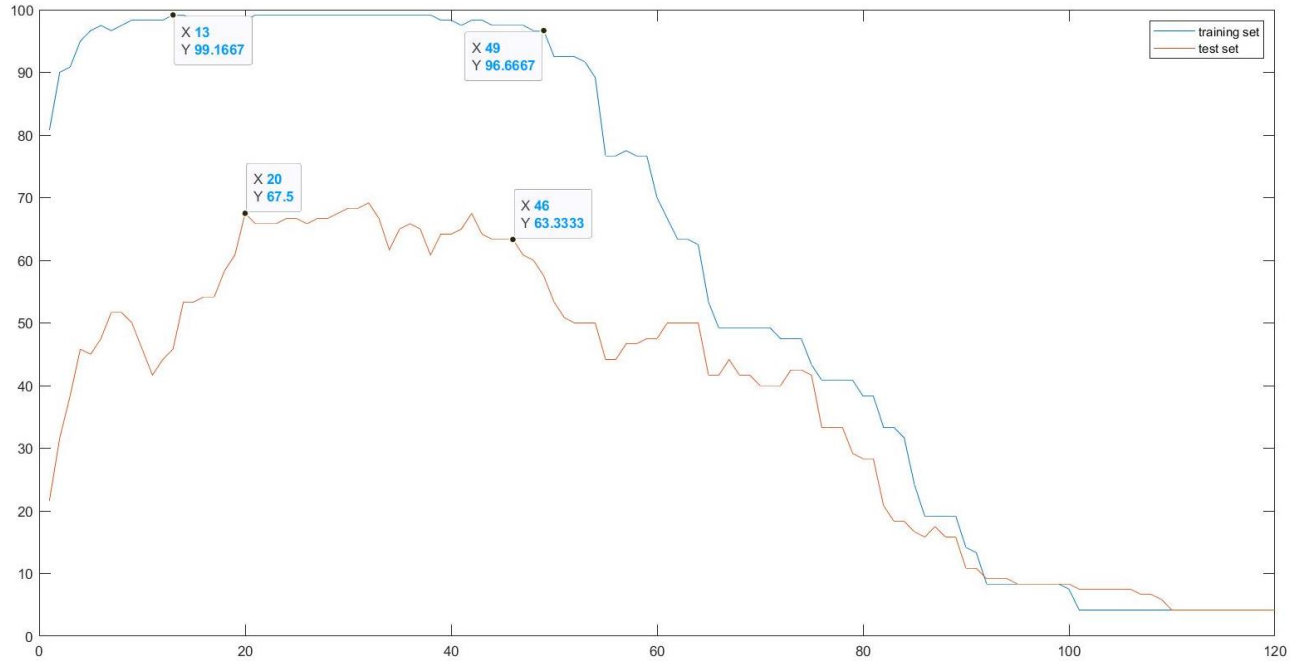

Fig. 1.4 – Accuracy of training set and test set as function of K (exercise 3).

We noticed that when K=1, the accuracy (regarding the training set plot) doesn't reach 100%. This happens because some measurements taken in different cells (classes) have the same value. Using the *max* function in Matlab, in case of equal values, the first one is taken. This implies that, among classes referring to distances between different points, the first one is taken as the output of our algorithm.

In the graph (FIG. 1.4), overfitting and underfitting phenomena are evident. When K is too small (left side of the graph), the complexity of the model implies overfitting), while on the right side the model is too much averaging because of the high K value, causing underfitting. In fact, when K=120, the accuracy is equal to 1/24, as the algorithm output is always class 1 (as specified about the *max* function implementation in Matlab).

An optimal K value would range from 20 to 40, where accuracy seems to maintain a good value, despite the overfitting.

COMPUTER LAB2

Exercise 1

In the first exercise, we used a dataset containing labeled data for two classes (men and women) containing the height and weight of each person. We fitted a class-conditional Gaussian multivariate distribution to these data, as the heights and weights can be represented as Gaussian distributions. In fact, visualizing the weight and height histograms, the distribution seemed to be Gaussian, except for some outliers.

Then we plotted the probability density functions.

Through the scattered plots, we guessed that a correlation between height and weight could exist (with the weight increasing, also the height tended to grow as well). This idea is confirmed by observing FIGs. 2.2 and 2.4, as the contours of equal probability are ellipses non-parallel to the axis.

Knowing that combining Gaussian distributions, we still obtain a Gaussian one, we imagined that a Gaussian model would be good enough for our data.

We calculated the Maximum Likelihood Estimate of the mean and of the covariance matrix, where each element of the mean vector is the average of the same feature (weight or height) of all males, and of all females.

$$\hat{\boldsymbol{\mu}}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x_i} = \bar{\boldsymbol{x}}$$

$$\hat{\boldsymbol{\Sigma}}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x_i} - \bar{\boldsymbol{x}})\,(\boldsymbol{x_i} - \bar{\boldsymbol{x}})^T$$

x_i is a two-element vector referring to the i-th person, containing the features. The mean vector x also has two elements (average weight and average height).

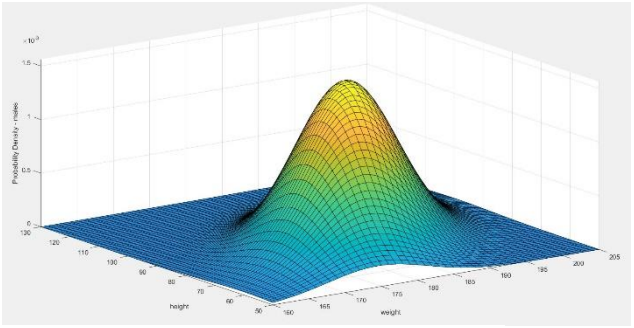

Fig. 2.1 – 3D male class conditional density.

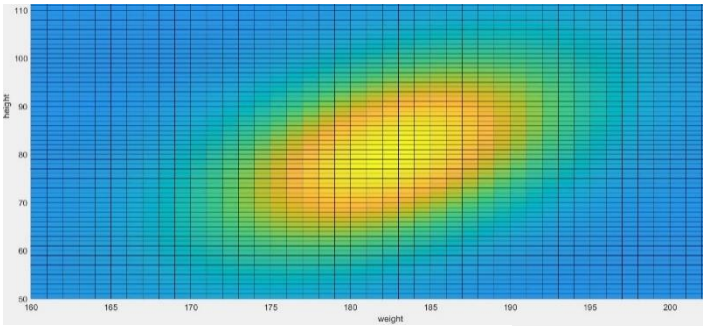

Fig. 2.2 – 2D male class conditional density.

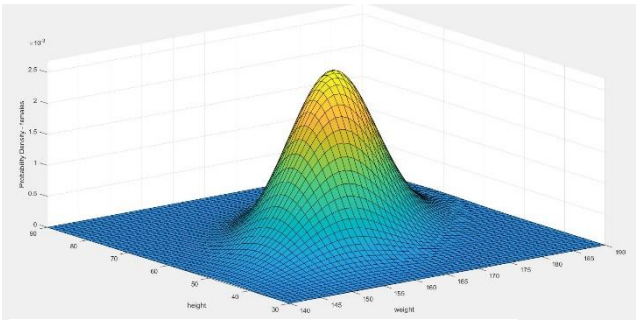

Fig. 2.3 – 3D female class conditional density.

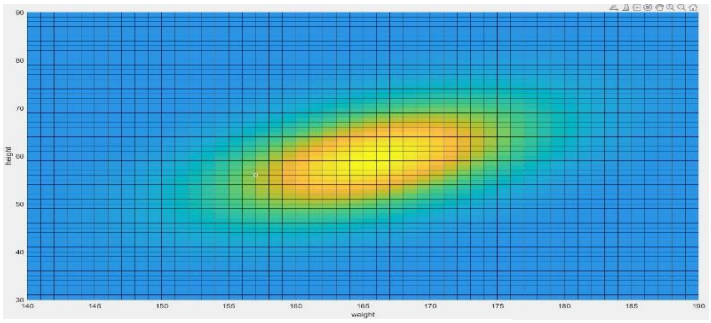

Fig. 2.4 – 2D female class conditional density.

In the FIGs. 2.1 and 2.3 we can appreciate the multivariate Gaussian distributions. In the FIGs. 2.2 and 2.4 right we can observe the rotated elliptical contours which demonstrate the correlation between the two features, which are statistically dependent.

Exercise 2

In this exercise we fitted the parameters that will be employed by a Naïve Bayes Classifier, using a Bernoulli model as each feature can assume two possible values representing word absence/presence (0 or 1). To do this we had to compute the prior probabilities of each class ($\pi_c = \frac{N_c}{N}$, where c={1,2}) and the probability that the feature j was equal to one (meaning the presence of the word) given the specific class ($\vartheta_{jc} = \frac{N_{jc}}{N_c}$, where j=[0,600]).

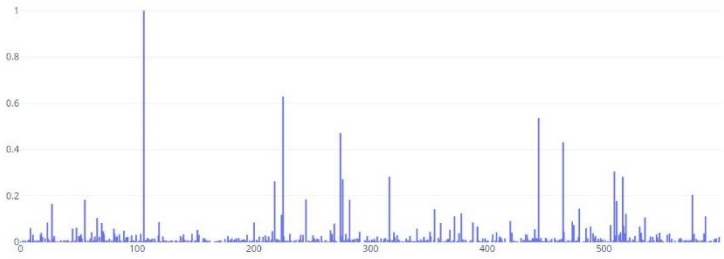

Fig. 2.5a – class conditional densities class 1.

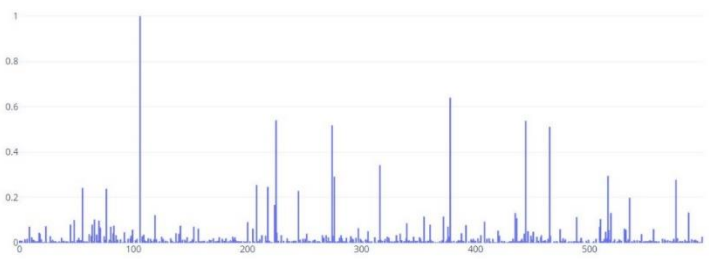

Fig. 2.5b – class conditional densities class 2.

It can be seen that the trend is very similar between the two classes: in fact, setting a margin of error of 0.2%, 79 features were considered uninformative ($\vartheta_{j1} \approx \vartheta_{j2}$).

Regarding the prior probabilities of each class, they were both equal to 0.5.

Exercise 3

In this exercise, we used the NBC model fitted in exercise 2 to compute the MAP estimate of the posterior probability used for document classification in the test set. We took advantage of the proportionality between the MAP and the class conditional probability (plus the prior probability computed in exercise 2). Assuming the features to be independent, we computed the class-conditional probability:

$$p(\boldsymbol{x}|y = c) = \prod_{j=1}^{D} p(x_j|y = c)$$

This formula allowed us to calculate the MAP estimate of the class the test vector belongs to:

$$\max \log\big(p(y = c|\boldsymbol{x})\big) \propto \log\big(p(\boldsymbol{x}|y = c)\big) + \log\big(p(c)\big)$$

It must be noted that, after the logarithm, the product becomes a summation in order to avoid numerical underflow. After the classification, we obtained accuracies equal to 92.78% for the training set and 74% for the test set.

Exercise 3 (optional)

Having observed that many features are uninformative, it is useful to compute the mutual information *I* between features and each class to choose the K most informative ones.

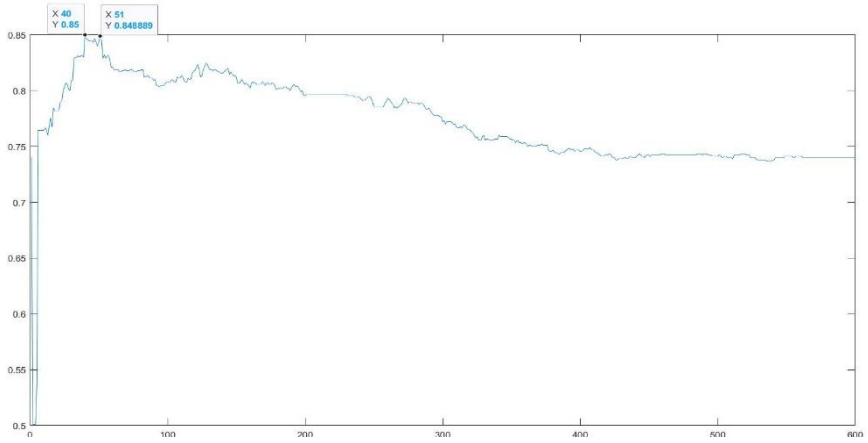

Fig. 2.6 – Accuracy of test set as function of K (number of features).

We computed the values of *I*, ranked the features by the decreasing values of *I* and run the classifier employing only the K most important features. The plot (FIG. 2.6) represents the accuracies as function of K.

The plot shows how, for small values of K, the accuracy grows almost linearly reaching the peak with K=40, showing a significantly better result than in the previous exercise. After K=50, the accuracy slightly decreases and fluctuates between 75% and 80%.

Exercise 4

In this exercise, we analyzed the performance of the classifier used in the previous exercises plotting the complete ROC curve, instead of simply measuring the accuracy. To make the prediction of the class, we defined a decision rule setting the value of a threshold *tau* to decide if an element belonged to class 1 or class 2 computing the ratio between p(y=1|x) and p(y=2|x) and comparing this ratio with *tau*. The threshold also determines the compromise between True Positive Rate (TPR) and False Positive Rate (FPR). Theoretically speaking, *tau* should assume values from 0 to infinite. Trying with different values and analyzing the results and the ROC obtained, we decided to make *tau* vary between 0 and 500 (with intervals=0.25).

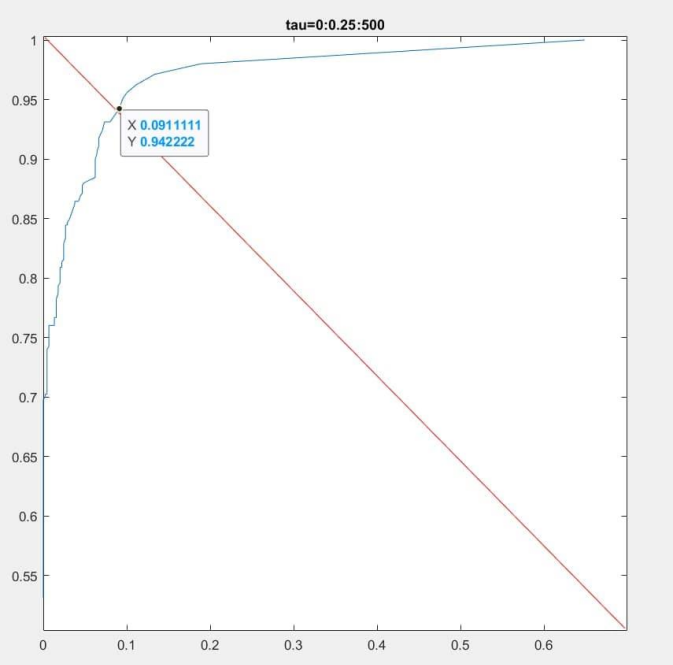

Fig. 2.7 – ROC curve 'bag of words'.

The curve (FIG. 2.7) shows that our classifier is pretty good as for most of the *tau* values the TPR is higher than the FPR.

Thinking about a good choice for *tau*, we can choose as a parameter the one nearer to the high-left corner: in our case TPR should be around 0.95.

The goodness of the classifier can also be proved by the fact that the ROC curve intersects the Equal Error Rate line in a point in which the FPR has a value of around 0.09.

Exercise 5

This exercise employs the height/weight data already employed in Exercise 1, and performs model fitting and classification using several versions of Gaussian discriminative analysis. For this exercise, the available data has been divided into two sets, *training* and *test* data. We decided to pick 35 females and 20 males to be used as test samples, removing them from the training set. We re-fitted the training data for each specific model and classified the test samples, calculating the accuracy of each classifier. We employed three classifiers:

1) Two-class quadratic discriminant analysis;
2) Two-class quadratic discriminant analysis with diagonal covariance matrices;
3) Two-class linear discriminant analysis.

Regarding the first one, both mean values and covariance matrices are class-specific, while for the second one, the off-diagonal entries of the class-specific covariance matrices were set to zero, employing a Naïve Bayes classifier (imposing statistical independence of the features). Finally, in the third classifier, a shared covariance matrix was calculated, putting together male and female training examples. The second and third methods can be possible solutions for preventing overfitting for Gaussian discriminant analysis.

We obtained the following accuracies:

1) 0.9091;
2) 0.9091;
3) 0.9091.

Depending on the chosen size for the test dataset, it is reasonable to obtain equal results for all the methods, as overfitting is unlikely to be present.

COMPUTER LAB3

Exercise 1

In this exercise, we used a hyperspectral image representing a scene of Indian Pines. It has a size of 145x145 pixels and 220 spectral bands, stored in vectors with length 220 (each value representing the value assumed by the considered pixel at the *i*-th wavelength).

Before applying the PCA, we centered the input data in order to make every feature of the pixels belonging to the chosen classes have zero mean. Without mean-centering, the first principal component found by PCA might correspond with the mean of the data instead of the direction of maximum variance[1].

We applied the PCA only for vectors belonging to two classes: class 1 and class 2. We used the following algorithm:

- we counted and stored all spectral vectors belonging to class1 and class2;
- we computed the means of all features of both vectors to subtract them obtaining zero-mean vectors;
- we computed the sample covariance matrix of the vector composed by spectral vectors belonging to class1 and class2;
- we computed eigenvectors and eigenvalues of this matrix sorting the resulted columns basing on eigenvalues values;
- we cycled (k=[1,220]) the following passages:
  - -we constructed the eigenvectors matrix W;
  - -we computed the PCA coefficients *z;*
  - -we obtained an approximation of the spectral vectors *x_hat;*
  - -we computed the MSE between the test set and *x_hat;*
- we plotted the $\sqrt{MSE(k)}$ obtaining the plot in FIG. 3.1.

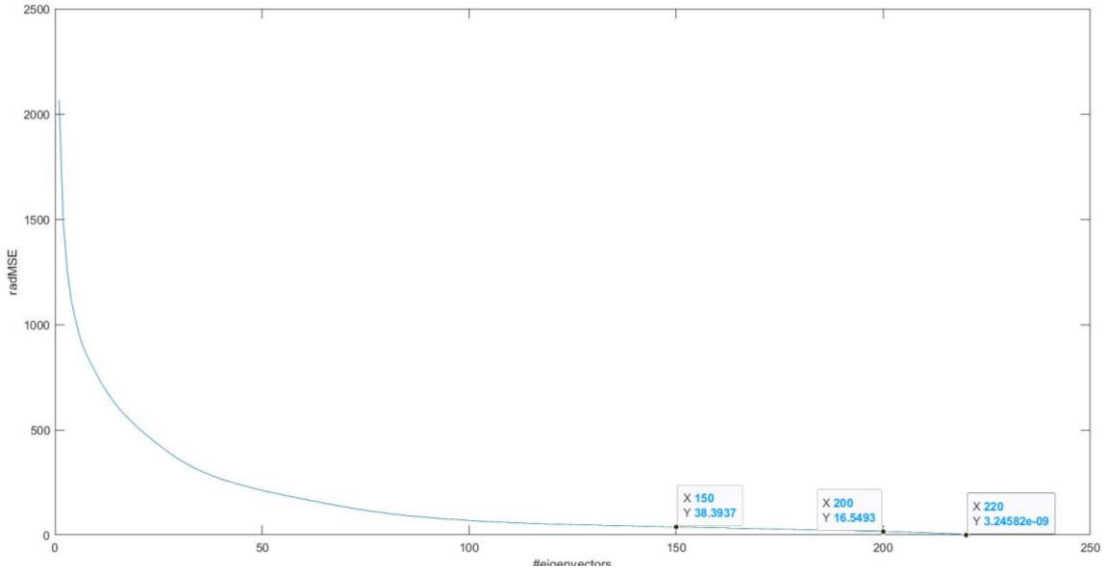

Fig. 3.1 – Accuracy of test set as function of K (number of eigenvectors building matrix W).

It can be observed how the error strongly decreases with an exponential trend. Therefore, after the first 100 k, it starts decreasing slowly. Knowing that the higher is the data correlation, the quicker the curve goes to 0, it can be said that there is a strong correlation between the features.

For k=150 the variance of the error in estimating $\hat{x}$ is lower than the variance of the spectral vectors (x).

Observing the values of the eigenvectors corresponding to the bigger eigenvalues, we noticed that they were the most similar to the data, except for a scale factor related to the need of unitary energy of the eigenvectors.

The smallest eigenvectors are the noisiest ones, hence less representing the data.

[1]= https://towardsdatascience.com/tidying-up-with-pca-an-introduction-to-principal-components-analysis-f876599af383

NEURAL NETWORKS-COMPUTER LAB1

Exercise 1

In this exercise, we built a CNN to predict values of given digit images using the MNIST Dataset.

This CNN was implemented as a Keras model using the Sequential Model, employing the TensorFlow library.

As loss function, the softmax cross-entropy was used. The cross-entropy is a cost function that can be used to avoid the learning slowdown incurred by the quadratic cost function, even if the initial choice of the parameters isn't really good. The aim is to minimize the cost function, as $C \approx 0$ when the neuron's output correctly estimates the input images. The learning rate is controlled by the error in the output.

The softmax layer is a particular type of activation of the output layer, composed of only ten neurons referring to the ten different classes (digits). Each neuron gives confidence related to the specific class determining the output answer.

The used Neural Network is composed of:

- one pooling layer (Average pooling) with **pools' size 2x2**;
- two convolutional layers: the first with **64 filters** (in order to extract 64 features) and **a 5x5 kernel** with **strides 4x4**; the second with **128 filters** and a **kernel 3x3** with **strides 2x2**. The common hyperparameters are the **RELU activation function** (in order to set to zero all negative values and not to modify the positive ones), the **glorot kernel initializer** (to initialize the kernel weights), and the bias set to zero;
  The reason why the number of filters is ascending is that at the input layer the network receives raw pixel data. Raw data are usually noisy, and this is especially true for image data.
  Because of this, we let CNN extract first some relevant information from noisy, "dirty" raw pixel data. Once the useful features have been extracted, then we make the CNN elaborate more complex abstractions on it using more filters.
- one flatten layer that makes the output of the previous layer (4D) be 2D in order to be compatible with the next layer;
- dense layer that gives the confidence of each class.

The **Adam** optimizer was used to reach the minimum of the cost function faster as it optimizes the direction of movement while reaching the minimum, starting with the value **0.001**. The **batch size** was set to **32**.

As it can be observed in FIG. 4.1, after the first 4/5 epochs the accuracy started growing very slowly. In fact, using 10 epochs, the accuracy on the test set is 98.61% but it is sufficient to use **4 epochs** to get an accuracy higher than 98%, before the network starts to learn training data peculiarities.

```
Epoch 1/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3004 - accuracy: 0.9147
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1145 - accuracy: 0.9653
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0804 - accuracy: 0.9754
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0631 - accuracy: 0.9805
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0514 - accuracy: 0.9834
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0419 - accuracy: 0.9863
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0360 - accuracy: 0.9885
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0308 - accuracy: 0.9902
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0261 - accuracy: 0.9917
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0223 - accuracy: 0.9929
```

Fig. 4.1 – Training of the CNN.

It can be seen how the loss function value decreases as the accuracy value increases, reaching, on the test set, a value of 0.0676 after 4 epochs and a value of 0.0452 after 10 epochs.

NEURAL NETWORKS-COMPUTER LAB2

Exercise 1

In this exercise, we employed transfer learning: we exploited an advanced CNN already trained on Imagenet to classify a different type of dataset, containing images of cats and dogs. The advantages of using transfer learning are the saving of resources and the improvement of efficiency when training new models. Transfer learning is useful as many times is not possible to train a neural network from scratch because of not having enough data.

In order to achieve it, we had to swap the last layer into a new layer with softmax activation function after having flattened the output to have a 2D model obtaining a layer composed of 2 neurons representing the 2 confidences for the classification of cats and dogs images. We used the softmax cross-entropy as loss function.

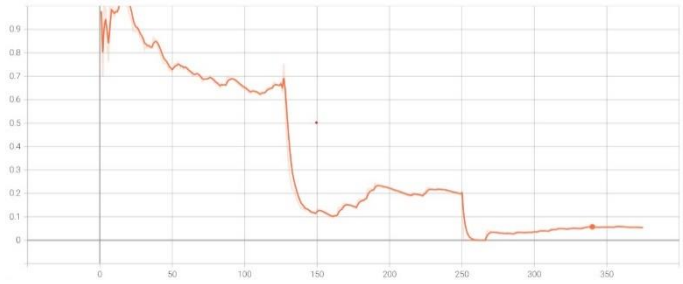

Fig. 4.2 – Loss function.

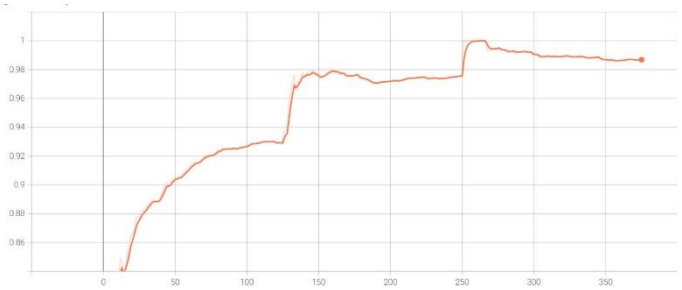

Fig. 4.3 – Accuracy.

Analyzing the obtained plots (FIG.s 4.2 and 4.3), it can be noticed that the accuracy increases and the loss function decreases as the size of batches grows. The number and the dimension of batches are really important when neural networks are used, as GPUs have tipically small memories and the use of too big batches makes the training to be very slow, as data have to be loaded many times. The dimension of the batches has an important influence on the accuracy as for each minibatch we get an estimate of the gradient which is noisy, hence the bigger is the size of the batches the less they are, producing less noise. In this case, the optimal value of the batch size is in the range [255,265].

Different optimizers were tried, reaching similar results. In particular, 2 of these attempts are reported:

```
Epoch 1/3
125/125 [==============================] - 9s 44ms/step - loss: 0.4247 - accuracy: 0.9320
Epoch 2/3
125/125 [==============================] - 6s 44ms/step - loss: 0.1354 - accuracy: 0.9770
Epoch 3/3
125/125 [==============================] - 6s 45ms/step - loss: 0.0501 - accuracy: 0.9915
```

1.Adam optimizer, learning rate=0.001, 3 epochs

```
Epoch 1/3
125/125 [==============================] - 9s 44ms/step - loss: 0.2152 - accuracy: 0.9165
Epoch 2/3
125/125 [==============================] - 6s 44ms/step - loss: 0.0704 - accuracy: 0.9770
Epoch 3/3
125/125 [==============================] - 6s 44ms/step - loss: 0.0333 - accuracy: 0.9920
```

2.SGD optimizer, learning rate=0.001, 3 epochs

It can be seen that the two methods produced almost the same results on the training set, apart from the first epoch.

On the test set, the results were:

- Adam optimizer: loss=0.36, accuracy=0.9600
- SGD: loss=0.12, accuracy=0.9570

COMPUTER LAB 4 – KALMAN FILTER

In this exercise, we studied an object moving on a 2D plane with constant (except for the noise) velocity. We simulated the motion of this object considering to have only the possibility to measure the positions at different time instants, not the velocities. We modelled the system noise as a Gaussian noise with diagonal covariance matrix (no correlation between noises referred to different variables). The observation noise vector is also modelled as a Gaussian noise with diagonal covariance matrix.

We built a Kalman filter based on a model with constant velocity (trying with velocity=0.5, 1.5 and 2.5) using our predictions on the states and the simulated measurements to give filtered estimates of the state of the analyzed system.

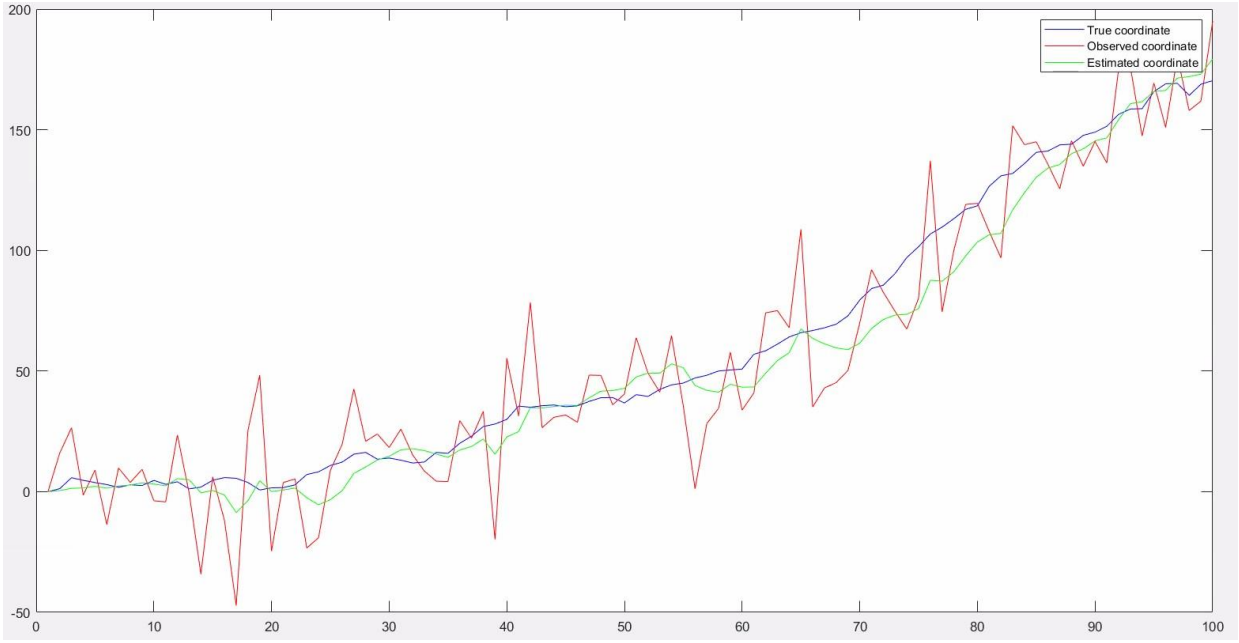

Fig. 5.1 – velocity=0.5.

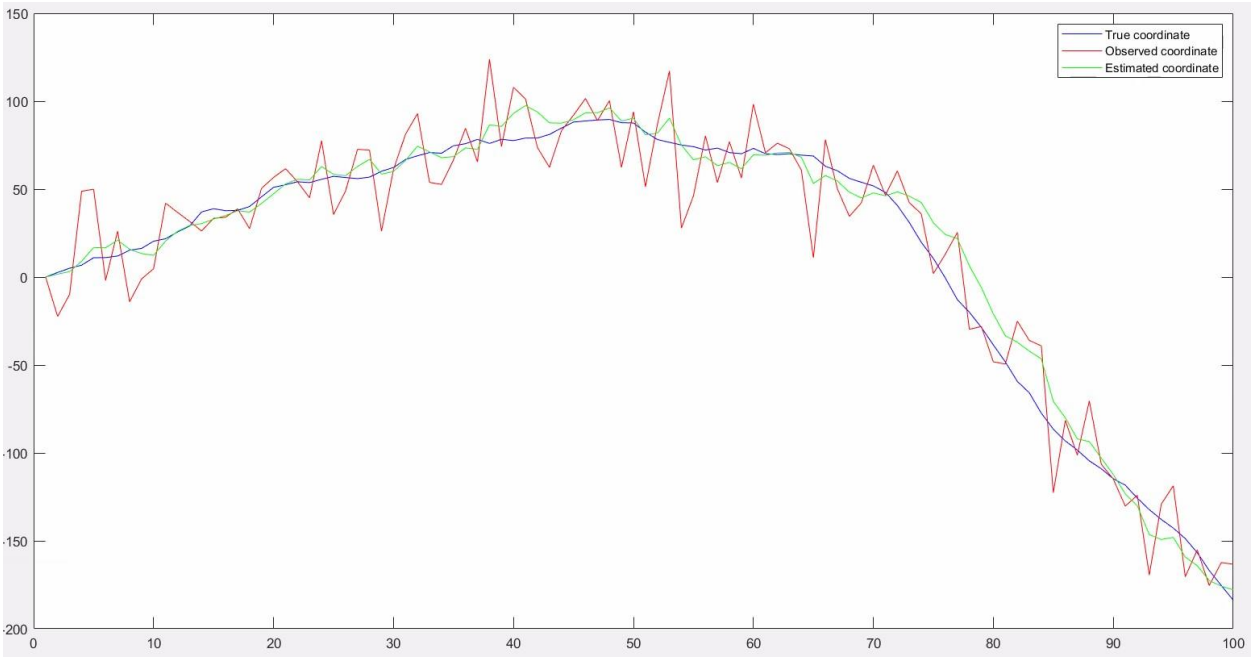

Fig. 5.2 – velocity=1.5.

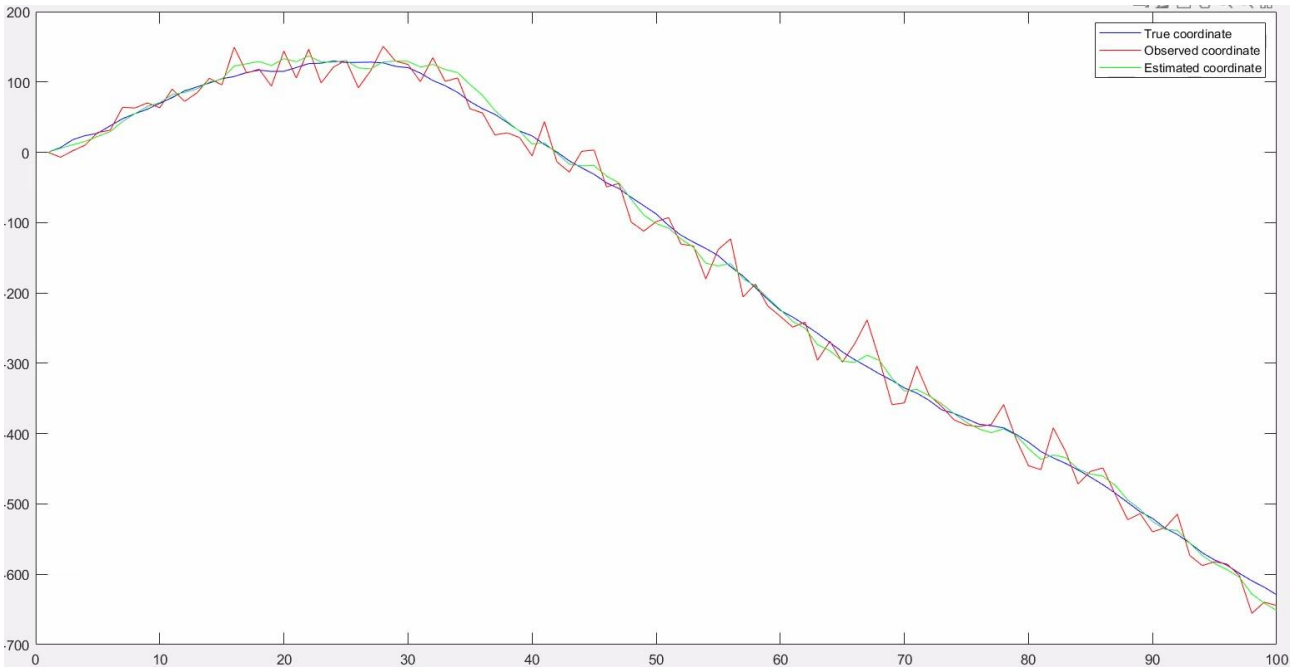

Fig. 5.3 – velocity = 2.5.

Analyzing the graphs representing the position of the object with respect to the time instants (trying with three different velocity values: 0.5, 1.5, 2.5), it can be seen (FIGs. 5.1, 5.2 and 5.3) that for higher values of the velocity there seem to be less influence of the noise on the system as the noise variance is the same.

We also tried to change some parameters: we modified the covariance matrix of the system by modifying the noise variances and the initial guesses of the mean.

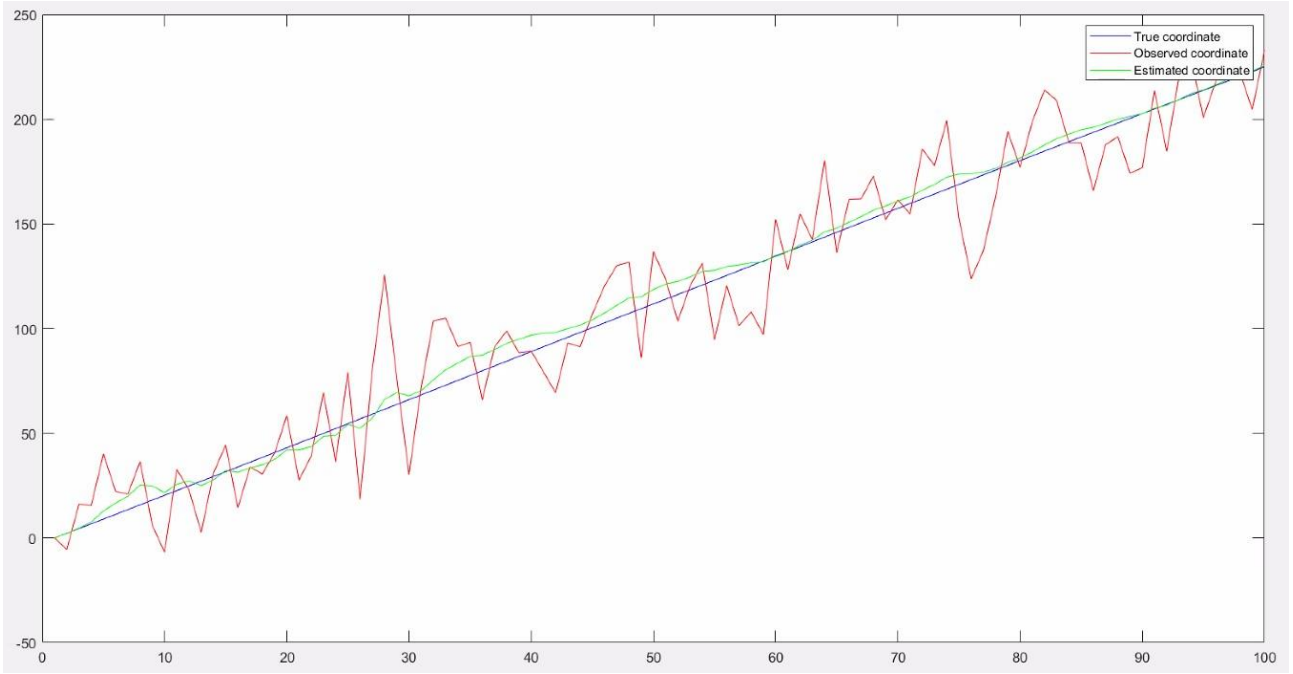

Fig 5.4 – small system std.

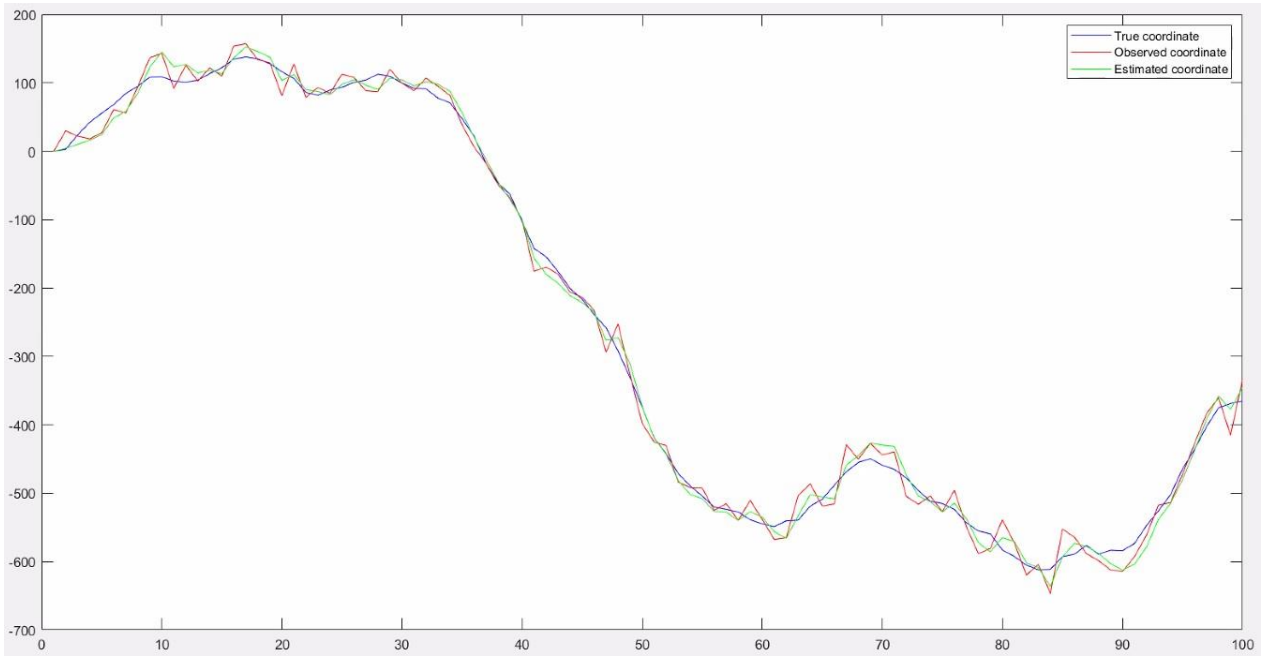

Fig 5.5 – big system std.

In FIGs. 5.4 and 5.5 we can observe how changes in the standard deviations of the system velocity and system position influence the filtered estimations. In particular, small standard deviations (0.005) make the final estimation to be closer to the prediction based on the chosen model. On the other hand, bigger standard deviations (5) make the estimates to be closer to the observed coordinate. This behavior can be explained through the analysis of the formula of the Kalman gain, which is proportional to the covariance matrix.

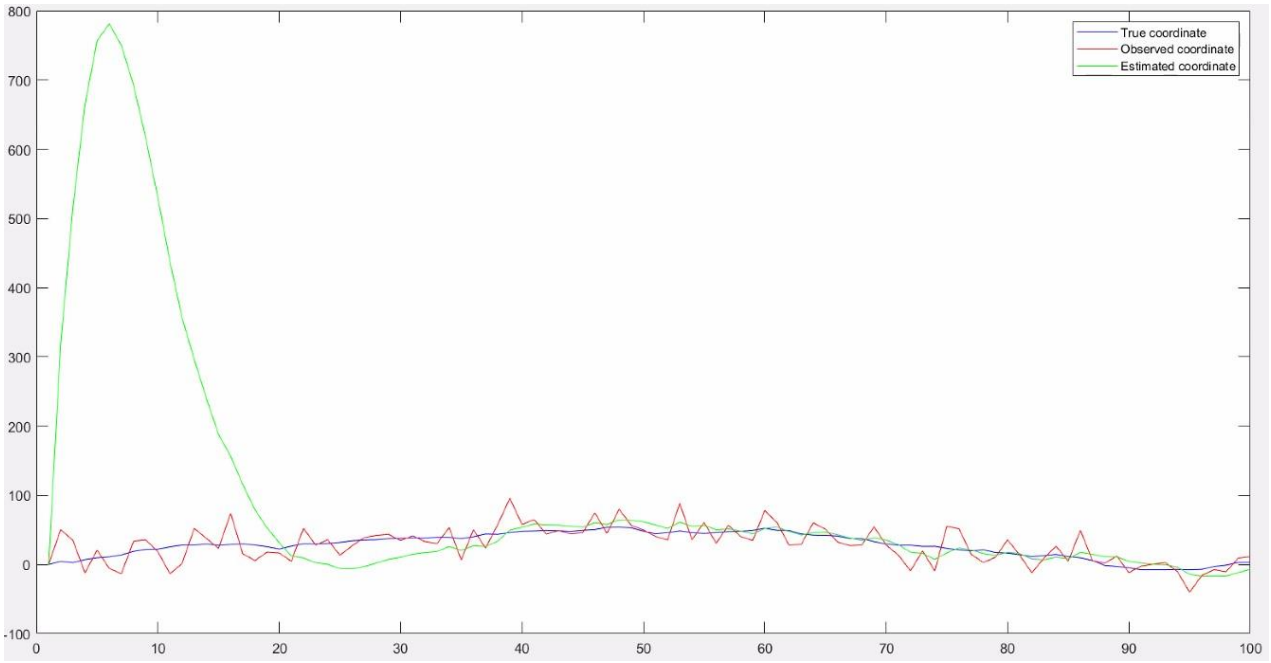

Fig 5.6 – wrong initial mean.

While in the previous cases the mean is chosen as the true one (0,0,1.5,1.5), FIG 5.6 refers to a wrong initialization of this vector (actually we could not know the real exact vector). We can observe how the Kalman filter is able to learn this parameter, also starting from an initial guess far from the correct one. In fact, this is not a critical parameter to be chosen using Kalman filer (this is not the case for the errors' standard deviations).