Homework 5

Francesco BORDERI aka Luigi - Lorenzo fratello Soft Porno DUSO - Matteo NERI aka Mario

December 13, 2016

1 Statistical Inference and MSE

1.1

We consider a system that emits particles of decay length λ and an experimental setup that allows to detect the decay events in a range $1 < x < 20$. Thus, the probability distribution of the observed decays is an exponential restricted in the detection interval:

$$p_\lambda(x) = \begin{cases} \frac{1}{Z(\lambda)} e^{-x/\lambda} & \text{if } 1 < x < 20 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $Z(\lambda)$ assures the correct normalization:

$$Z(\lambda) = \int_1^{20} e^{-x/\lambda} = -\lambda[e^{-x/\lambda}]_1^{20} = \lambda \left(e^{-1/\lambda} - e^{-20/\lambda} \right) \tag{2}$$

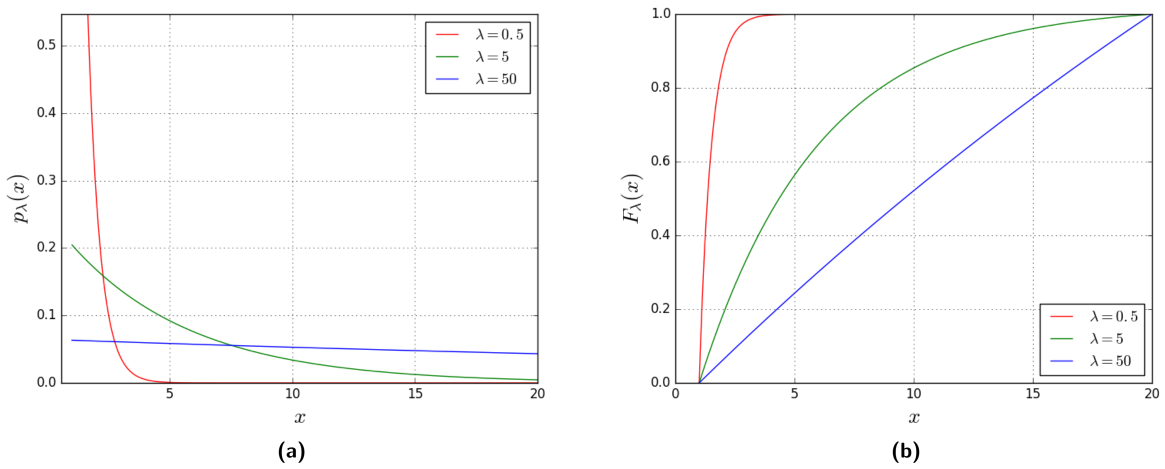


Figure 1: Some plots of the probability distribution 1 and of its cumulative distribution for different values of λ. Notice that $p_{0.5}(x)$ goes up to 2.0 for $x = 1$. As λ is increased, the distribution looks more and more like a uniform.

Since each decay is an independent event and is characterized by the same probability distribution, then all the decays are i.i.d. random variables and the probability $p_\lambda(\vec{x})$ of observing the set $\vec{x} = (x_1, x_2, ..., x_n)$ of n events factorizes:

$$p_\lambda(\vec{x}) = \prod_{i=1}^{n} p_\lambda(x_i) = \frac{1}{Z(\lambda)^n} \prod_{i=1}^{n} e^{-x_i/\lambda} = \frac{1}{Z(\lambda)^n} \exp\left(-\frac{1}{\lambda} \sum_{i=0}^{n} x_i \right) \tag{3}$$

If one wants to write a program that generates n samples of the distribution 1, in principle could naively draw random numbers from the full exponential distribution from 0 to $+\infty$ and then accept only the values in the region $1 < x < 20$. Anyway this procedure is inefficient because the number of rejections can be high. In this case it is possible to find an analytic formula to map the output of the uniform random number generator into the desired distribution. It is necessary to compute the cumulative distribution $F_\lambda(x)$ and invert the equality with the uniform distribution:

$$F_\lambda(x) = \int_{-\infty}^{x} p_\lambda(x) = \int_{1}^{x} \frac{1}{Z} e^{-x/\lambda} = \frac{\lambda}{Z}\left(e^{-1/\lambda} - e^{-x/\lambda}\right) = u$$

with $u \sim \text{Uniform}[0,1)$. Now explicit x:

$$x = -\lambda \log\left(e^{-1/\lambda} - \frac{Z}{\lambda} u\right) \tag{4}$$

1.2

To infer the value of λ from a set of n observations, a first possible approach is to use the Maximum Likelihood estimator, as in the "frequentist" point of view:

$$\hat{\lambda}_{ML} = \underset{\lambda}{\text{argmax}}\, p_\lambda(\vec{x}) \tag{5}$$

and to check the performance of this estimator, it is useful to consider the Squared Error:

$$SE(\hat{\lambda}(\vec{x}), \lambda) = (\hat{\lambda}_{ML} - \lambda)^2 \tag{6}$$

However, the use of the Maximum Likelihood estimator is unfeasible in practice: in fact the expression 3 gets exponentially small as n is increased, and the actual value of $p_\lambda(\vec{x})$ is lost due to the finite numerical precision. This problem is simply solved simply taking the logarithm of $p_\lambda(\vec{x})$, that is, working with the Log-Likelihood. Moreover, it is also convenient to divide the Log-Likelihood again by n. Both these choices make the values where the maximization has to perform more reasonable and do not shift the position of the maximum. Finally, the actual estimator that has been used is:

$$\hat{\lambda}_{ML} = \underset{\lambda}{\text{argmax}}\, \frac{1}{n} \log p_\lambda(\vec{x}) = \underset{\lambda}{\text{argmax}}\, \mathcal{L}(\lambda, \vec{x}) \tag{7}$$

where the rescaled Log-Likelihood can be rewritten as:

$$\mathcal{L}(\lambda, \vec{x}) = \frac{1}{n} \log\left(\frac{1}{Z(\lambda)^n} e^{-\frac{1}{\lambda}\sum x_i}\right) = -\frac{1}{\lambda}\frac{\sum x_i}{n} - \log(Z(\lambda)) \tag{8}$$

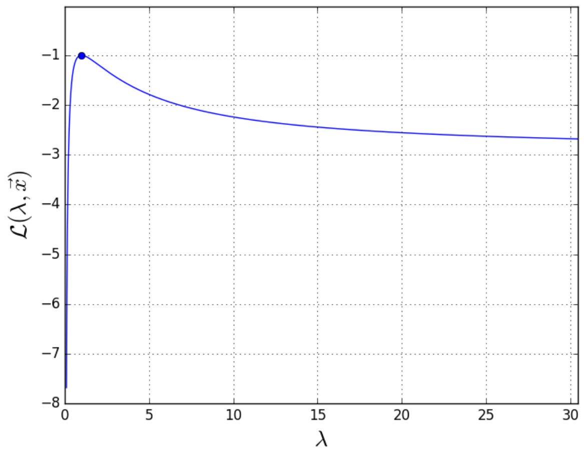
There are different ways to find numerically the value of $\hat{\lambda}$ that maximizes the expression 8. Here we present three intuitive ideas, explaining their positive and negative features:

- Gradient ascend algorithm: this method allows to get the precise value of $\hat{\lambda}$, within a tolerance, by computing numerically the slope of $\mathcal{L}(\lambda, \vec{x})$ and updating $\hat{\lambda}$ according to its direction. The drawback is that it requires some time to converge and there may be even some convergence issues if the updating parameters are not chosen properly.
- Define a vector of "proposed" λ's, evaluate the function $\mathcal{L}(\lambda, \vec{x})$ on each of them, and keep track of the maximum. This procedure is much faster and there are no convergence issues, but the obtained value of $\hat{\lambda}$ has a precision bounded by the discretization used to define the proposal vector. However, even if this method works well when sufficiently large data sets are used (large n), it gives WRONG results when applied to small data sets: especially when the data are generated with value of λ near or above 20, it happens quite often that the value of $\hat{\lambda}$ exceeds the upper range of the proposed vector over which the maximization is performed. This fact actually helps the estimation to work better, since $\hat{\lambda}$ is underestimated closer to the "true" value, and strange artifacts concerning the Cramer-Rao lower bound appear (see later).

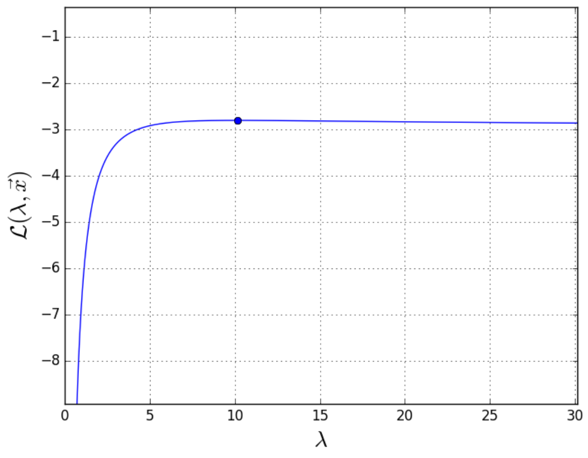
- Asymmetric ascend: the best idea is to exploit the fact that the Log-Likelihood is a concave function, which means that it has an unique absolute maximum, and do something that unifies the two precedent ideas. $\hat{\lambda}$ can be easily found starting from a very small value of λ and raising it with a certain small step until the first decrease in the Log-Likelihood is registered. In this way there is no risk to have initialized a proposal vector which is too small, the algorithm runs fast, but of course the value $\hat{\lambda}$ will be reached with a precision equal to the used step.

In the following results we applied the last strategy, usually using a spacing of $\Delta\lambda = 0.01$.

To directly see what looks like this maximization, we chose to plot the function $\mathcal{L}(\lambda, \vec{x})$, now applied on a set on $n = 10000$ samples, as shown in Figure 2. It is evident how the Log-Likelihood function changes shape when λ is increased: the maximum happens to be in a really flat region, and this is why for great values of λ but small n is becomes easy to make huge mistakes and obtain very large results of the estimator $\hat{\lambda}_{ML}$. This behavior is particularly evident in Figure 3, where the histograms for the obtained values of $\hat{\lambda}_{ML}$ are shown, for two different values of the data set size n and two different values of λ (now 10 and 20).

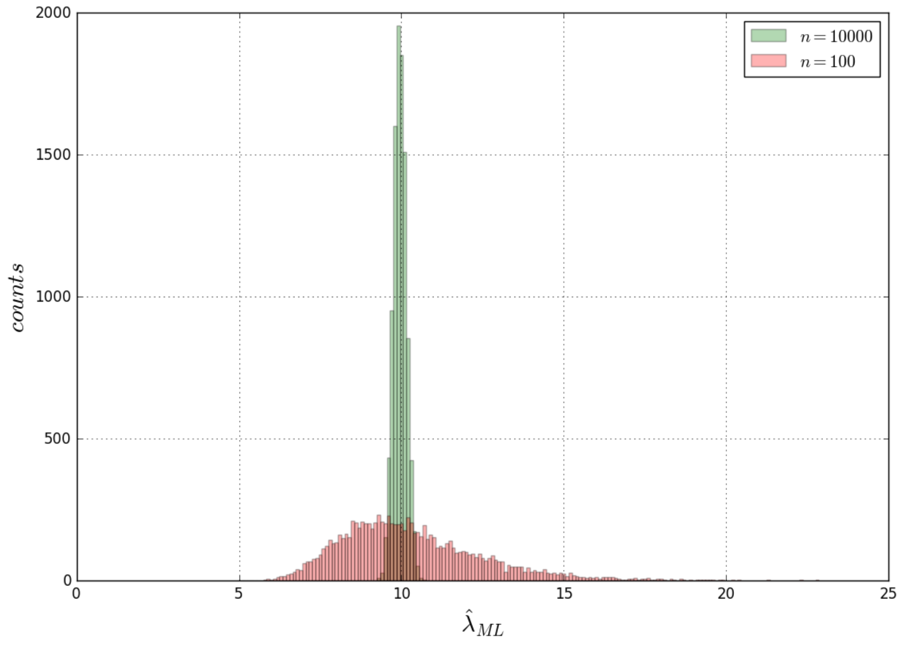


(a) $\lambda = 1$, $\hat{\lambda}_{ML} = 0.999$, $SE = 10^{-6}$

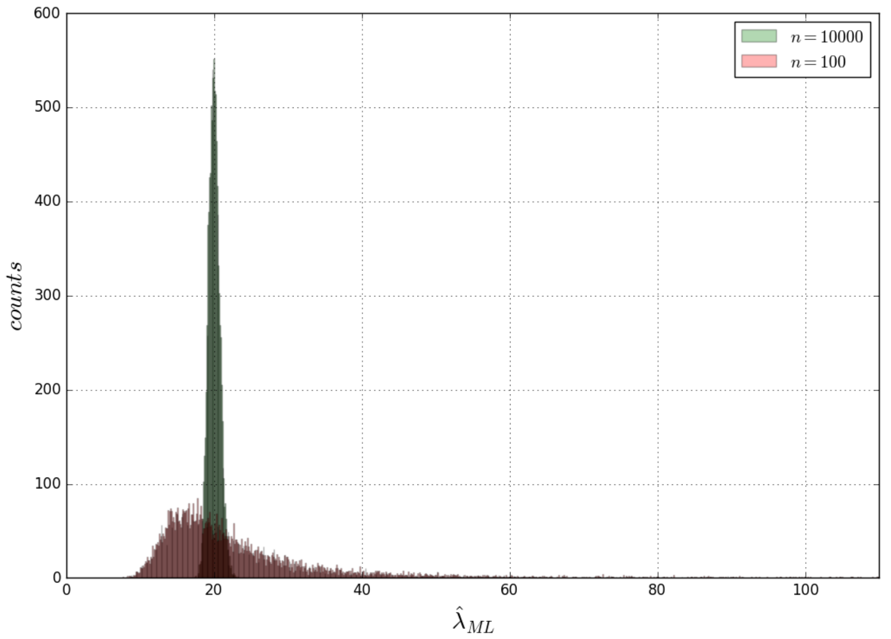


(b) $\lambda = 10$, $\hat{\lambda}_{ML} = 10.151$, $SE = 0.023$

Figure 2: Graphical examples of the result of $\hat{\lambda}_{ML}$ when applied to two set of $n = 10000$ samples generated with two different values of λ.



(a) $\lambda_{true} = 10$



(b) $\lambda_{true} = 20$

Figure 3: Histograms of the results of the Maximum Likelihood estimator using $n = 100$ or $n = 10000$ samples in the data sets. Of course the predictions get worse with lower values of n, but also when the data are generated with a larger λ_{true}.

1.3

If the process of evaluating $\hat{\lambda}$ and SE is repeated many times over many samples of size n, then we can obtain the Mean Squared Error:

$$MSE(\lambda, \hat{\lambda}, n) = \frac{1}{T}\sum_{m=1}^{T} SE(\hat{\lambda}(\vec{x}_m), \lambda) = \frac{1}{T}\sum_{m=1}^{T} (\hat{\lambda}(\vec{x}_m) - \lambda)^2 \tag{9}$$

where T is the number of considered data sets $\vec{x}_1, \vec{x}_2, ..., \vec{x}_T$, each one made of n samples.
We chose to compute the MSE curve for several values of λ, data sets of $n = 10000$ samples, and trying with different choices of T. Of course, the the bigger T is the more meaningful are the results, as discussed later. The maximization procedure has been done with step $\Delta\lambda = 0.01$. The MSE curves are plotted in Figure 4.

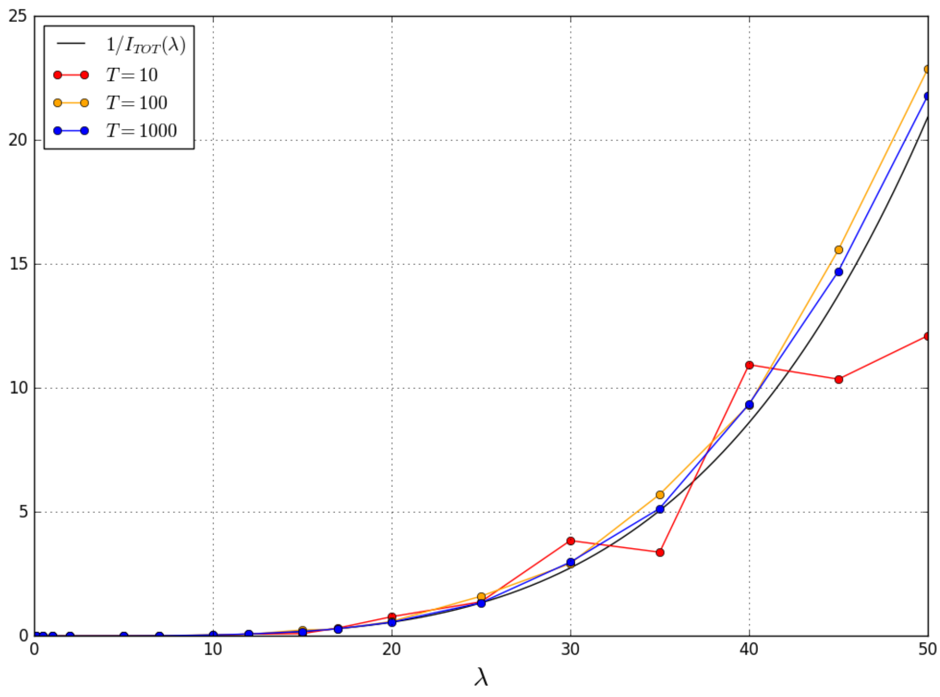


Figure 4: Comparison of the MSE curves obtained with the Maximum Likelihood estimator over data sets of size $n = 10000$ and the Cramer-Rao bound. $T = 10$, $T = 100$ and $T = 1000$ indicates the number of data sets averaged for each value of λ.

An important check that is required at this point is the behavior of the Mean Squared Error with respect to the Cramer-Rao bound. The expression of the Cramer-Rao inequality is different if considered for a biased or unbiased estimator. Recalling that an estimator is defined unbiased if:

$$\mathbb{E}\left[\hat{\lambda}_{unbiased}\right] = \lambda_{true} \tag{10}$$

then the Cramer-Rao bound for an unbiased estimator reads:

$$\mathbb{V}\left[\hat{\lambda}\right] \geq \frac{1}{I_{TOT}(\lambda)} \tag{11}$$

where $I_{TOT}(\lambda)$ is the Total Fisher information and $\mathbb{V}\left[\hat{\lambda}\right] = \int p_\lambda(x)(\hat{\lambda}-\lambda)^2$ is the variance of the unbiased estimator $\hat{\lambda}$. Indeed, looking back at equation 9, the Mean Squared Error plays the role of an unbiased estimator for the variance of the estimator $\hat{\lambda}_{ML}$ (unbiased because it is defined with the "true mean" λ). Instead, the general Cramer-Rao bound for an estimator having a bias $b(\lambda) = \mathbb{E}\left[\hat{\lambda} - \lambda\right]$ is:

$$\mathbb{V}\left[\hat{\lambda}\right] \geq \frac{1 + b'(\lambda)}{I_{TOT}(\lambda)} + b^2(\lambda) \tag{12}$$

Thanks to the property of Asymptotic Normality, we can treat the Maximum Likelihood estimator $\hat{\lambda}_{ML}$ as an unbiased estimator, up to order $n^{-1/2}$. In fact:

$$\lim_{x \to \infty} \sqrt{n}(\hat{\lambda}_{ML} - \lambda) = \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{I_1(\lambda)}\right) \tag{13}$$

Which means that:

$$\mathcal{P}(\hat{\lambda}_{ML}) \propto \exp\left(-\frac{(\hat{\lambda}(\vec{x}_m) - \lambda)^2}{2\frac{1}{n\, I_1(\lambda)}}\right) \tag{14}$$

So, this is why we can conclude that the Maximum Likelihood is unbiased estimator up to a factor $n^{-1/2}$, since:

$$\hat{\lambda}_{ML} = \lambda \pm \frac{1}{\sqrt{n\, I_1(\lambda)}} = \lambda \pm \frac{1}{\sqrt{I_{TOT}(\lambda)}} \tag{15}$$

Important Remark: from equation 13 to 15 the symbol n indicates the size of the data set $\vec{x}$. So equation 14 states that the variance of the estimator $\hat{\lambda}_{ML}$, for large n, approaches the inverse of the Total Fisher Information, so it is proportional to $1/n$:

$$\mathbb{V}\left[\hat{\lambda}_{ML}\right] \to \frac{1}{I_{TOT}(\lambda)} = \frac{1}{n\, I_1(\lambda)} \tag{16}$$

Instead, it is important to realize that what we did to compute the Mean Squared Error was to average the Squared Error of T different data sets of size n, for every value of λ considered. And we know that this procedure gives **an estimate of the variance of the estimator** $\hat{\lambda}_{ML}$ with a variance:

$$\mathbb{V}\left[MSE(\lambda, \hat{\lambda}, n)\right] = \mathbb{V}\left[\frac{1}{T}\sum_{m=1}^{T}(\hat{\lambda}(\vec{x}_m) - \lambda)^2\right] = \frac{1}{T}\mathbb{V}\left[(\hat{\lambda}(\vec{x}_m) - \lambda)^2\right] \propto \frac{1}{T} \tag{17}$$

So, we expect to obtain an $MSE(\lambda, \hat{\lambda}, n)$ curve that, for a given n sufficiently large, converges close to the Cramer-Rao lower bound with fluctuations of the order of $1/\sqrt{T}$, that are reduced as T is increased.

The computation of the Fisher Information for the distribution defined in equation 1 is reported in Appendix A, where there are also some dedicated plots. Thanks to this analytic result, it is possible to compare directly the behavior of our MSE curves with the Cramer-Rao bound. Figure 4 shows what deduced by the expressions 16 and 17: the empirical MSE curves follow the Cramer-Rao bound, and they oscillate around it in the two expected ways:

- as $1/\sqrt{n}$ and slightly form above, due to the asymptotic normality
- with fluctuations below and above proportional to $1/\sqrt{T}$, because of the variance of the MSE estimator itself

1.4

If one adopts a Bayesian point of view and uses the Jeffrey's prior $\mathcal{P}(\lambda) \propto \sqrt{I_1(\lambda)}$, the posterior probability for a single event reads:

$$\mathcal{P}(\lambda|x_i) = \frac{\mathcal{P}(\lambda \cap x_i)}{\mathcal{P}(x_i)} = \frac{\mathcal{P}(\lambda)\,\mathcal{P}(x_i|\lambda)}{\mathcal{P}(x_i)} \propto \mathcal{P}(\lambda)\; p_\lambda(x_i) \propto \sqrt{I_1(\lambda)}\;\frac{e^{-x_i/\lambda}}{Z(\lambda)} \tag{18}$$

If a data set on n samples is considered, the posterior probability is then:

$$\mathcal{P}(\lambda|\vec{x}) = \frac{\mathcal{P}(\lambda)\,\mathcal{P}(\vec{x}|\lambda)}{\mathcal{P}(\vec{x})} \propto \sqrt{I_1(\lambda)}\;\frac{e^{-\sum x_i/\lambda}}{(Z(\lambda))^n} \tag{19}$$

Now we are going to use the Maximum A Posteriori estimator, defined as:

$$\hat{\lambda}_{MAP} = \underset{\lambda}{\operatorname{argmax}}\,\mathcal{P}(\lambda|\vec{x}) = \underset{\lambda}{\operatorname{argmax}}\,\mathcal{P}(\lambda)\,p_\lambda(\vec{x}) \tag{20}$$

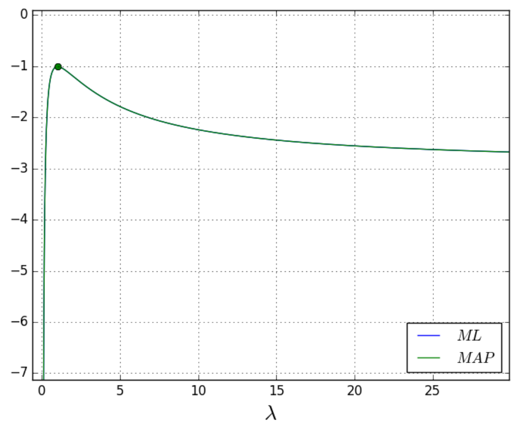
Again, in practice is convenient to take the logarithm and possibly also divide by n the argument of the maximization. Similarly to what done in equations 7 and 8 now we have:

$$\begin{aligned}\hat{\lambda}_{MAP} &= \underset{\lambda}{\operatorname{argmax}}\,\frac{1}{n}\log\mathcal{P}(\lambda|\vec{x}) = \underset{\lambda}{\operatorname{argmax}}\left(\frac{1}{n}\log p_\lambda(\vec{x}) + \frac{1}{n}\log\mathcal{P}(\lambda)\right) = \\ &= \underset{\lambda}{\operatorname{argmax}}\left(\mathcal{L}(\lambda,\vec{x}) + \frac{1}{n}\log\mathcal{P}(\lambda)\right) = \\ &= \underset{\lambda}{\operatorname{argmax}}\left(-\frac{1}{\lambda}\frac{\sum x_i}{n} - \log(Z(\lambda)) + \frac{1}{2n}\log I_{TOT}(\lambda)\right)\end{aligned} \tag{21}$$

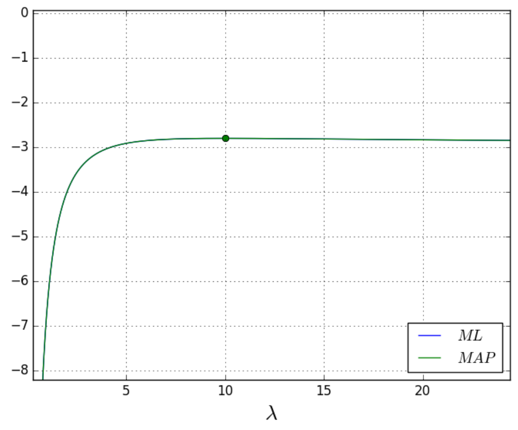
This expression shows clearly that the estimator $\hat{\lambda}_{MAP}$ contains in the argmax a term more than in the one of $\hat{\lambda}_{ML}$: since this term is in general different from zero and thanks to the fact that we already showed that $\hat{\lambda}_{ML}$ is unbiased using the Asymptotic Normality property, then we have a sufficient condition to say that $\hat{\lambda}_{MAP}$ is a biased estimator. So, the variance of the estimator $\hat{\lambda}_{MAP}$ has to be compared with the full Cramer-Rao lower bound for biased estimators as expressed in equation 12. Notice that, even if $b^2(\lambda)$ is for sure non-negative contribution, the factor $(1 + b'(\lambda))$ may be negative and this implies that a biased estimator could dominate an unbiased one. We anticipate that this case is exactly one of those, since we are going to find that the MAP estimator dominates the ML.

First, let's plot the function that enters in the argument of the MAP estimator as written in Equation 21. Figure 5 shows it together with the Log-Likelihood: qualitatively the two functions seem identical, and in fact it is required an high zoom to see their difference. Moreover, the estimated values are practically the same in this examples with $n = 10000$. Actually this strong similarity is reasonable: the term that is responsible of the difference between equation 21 and $\mathcal{L}(\lambda,\vec{x})$ has a factor $1/n$ in front: this explains why for large n the MAP and ML estimators behave equivalently; instead, for smaller n, this small difference is such that it slightly lowers the function for large values of λ and thus the MAP estimator dominates. Figure 6 is an "ad hoc" example chosen to show this phenomenon.

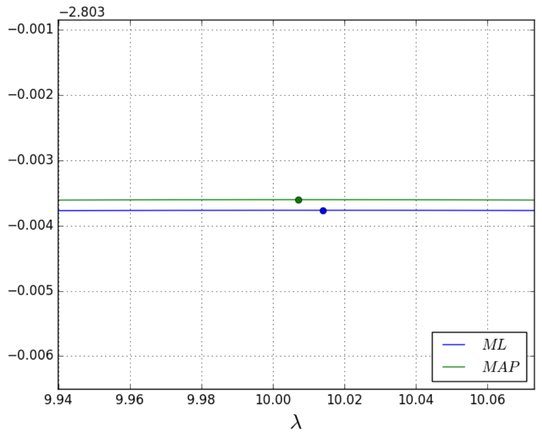
The right way to show the dominance of the MAP with respect to the ML estimator is looking at their MSE curves. We decided to plot them for $n = 10, 100, 1000$ together with the Cramer-Rao bound $1/(n \cdot I_1(\lambda)$. The results are reported in Figures 7,8 and 9.



(a) $\lambda = 1$, $\hat{\lambda}_{MAP} = 1.008$, $\hat{\lambda}_{ML} = 1.008$, $SE_{MAP} = SE_{ML} = 6.4 \cdot 10^{-5}$. The maximum difference between the two curves is only $6.9 \cdot 10^{-4}$



(b) $\lambda = 10$, $\hat{\lambda}_{MAP} = 10.007$, $\hat{\lambda}_{ML} = 1.014$, $SE_{MAP} = 4.9 \cdot 10^{-5}$, $SE_{ML} = 1.96 \cdot 10^{-4}$



(c) Zoom on the plot around $\lambda = 10$. Here it is possible to see the slight difference between $\hat{\lambda}_{MAP} = 10.007$ and $\hat{\lambda}_{ML} = 1.014$

Figure 5: Graphical examples of the result of $\hat{\lambda}_{MAP}$ and $\hat{\lambda}_{ML}$ when applied to two set of $n = 10000$ samples generated with two different values of λ.

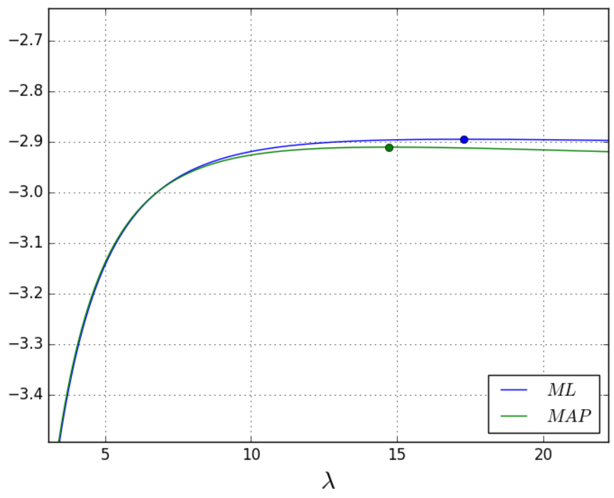


Figure 6: This exaggerated example with $\lambda_{true} = 15$ and $n = 10$ is meant to show, how with small n, the further term inside the argmax of the MAP produces a visible difference that shifts $\hat{\lambda}_{MAP}$ towards left with respect to $\hat{\lambda}_{ML}$.

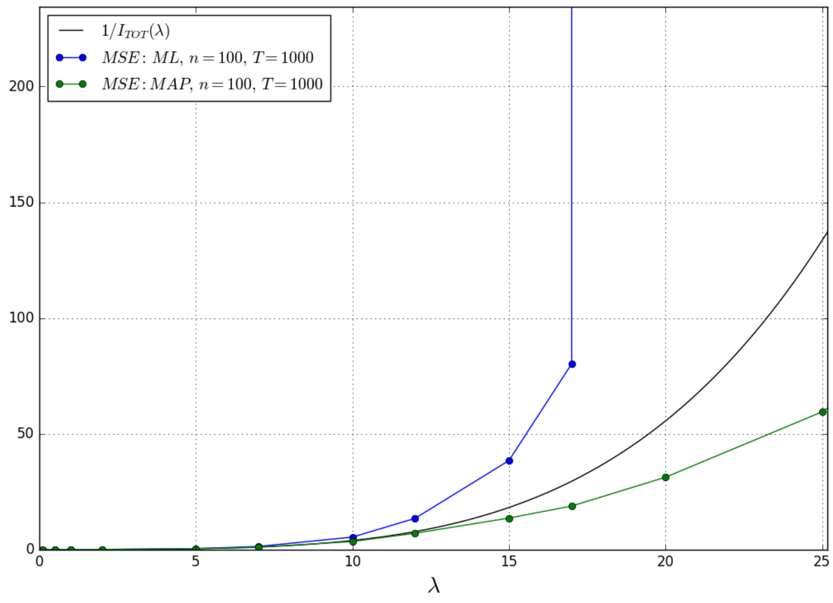


Figure 7: If the data sets are small ($n = 100$) it is clear that the MAP performs better than the ML estimator. Again, this is due to the correction term with the Fisher information that is more relevant. In particular, we can already notice that the bias of the MAP is such that it goes lower than the Cramr-Rao curve. Instead, as the data get generated with a greater value of λ, the ML estimator badly fails and predicts exaggerated estimates and this explains why the MSE curve explodes (consider again how the predictions are broad in the histrograms of Figure 3.)

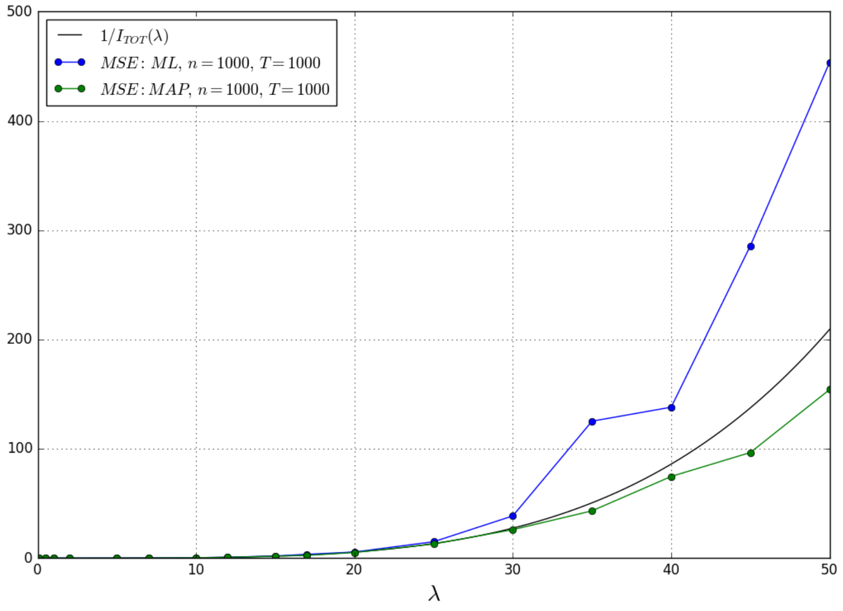


Figure 8: Increasing n to 1000, the ML estimator starts to work better but clearly it is still above the Cramr-Rao bound. The MAP estimator instead is still below.

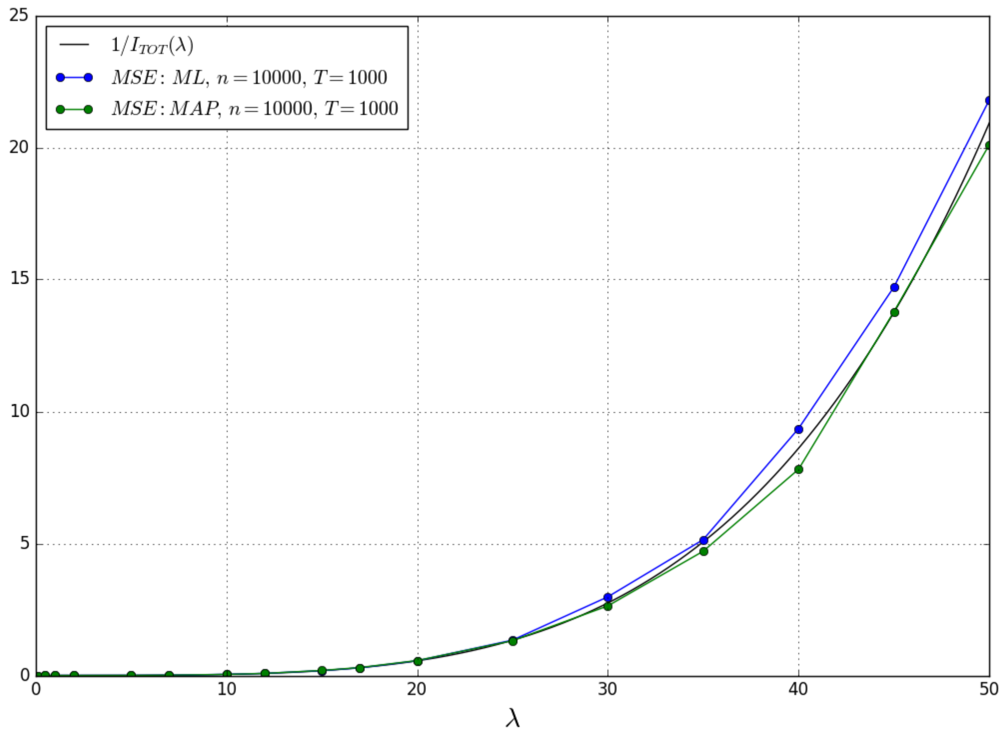


Figure 9: Reaching $n = 10000$ the two estimators work practically with the same performance, even if the ML will be always slightly dominating.

2 The MNIST dataset

For this section we worked on the training of a simple classifier on the MNIST dataset, hosted on Yann LeCun's website. In order to read in the MNIST images we used the tensorflow API and we reorganized the train, validation and test set in a set of 50 000 images, hereafter used as our training set, and a set of 10 000 images that will be our test set. Each image represents a digit and is a grayscale image. It is composed of $n = 784$ pixels, each of them is holding the intensity of the gray scale from zero to one. We consider a single image as a row vector in $\mathbb{R}^n$.
We are going to use a two layer neural networks in order to classify digits. The input layer has been built with a random projection matrix and the logistic function and it can be called the hidden layer since the values of its nodes are hidden from outside the net. It is composed of m nodes representing the m dimension in which the n-dimensional image vector is projected. The output layer is a linear combination of the m-features of the hidden layer.
Let $\mathbf{R}$ be the random projection matrix $\mathbf{R} \in \mathcal{M}_{n\times m}(\mathbb{R})$ from n to m dimensions, $f : \mathbb{R} \to [0, 1]$ be the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ and let f act memberwise where acting on a multidimensional object. Let $\mathbf{x}_\mu$ be the vector $\mathbf{x}_\mu \in \mathbb{R}^n$ storing the μ-image of the training set, θ the vector $\theta \in \mathbb{R}^m$ of the weights used to produce the output. Then we can write the output y of our network as:

$$y_\mu = \mathbf{z}_\mu \boldsymbol{\theta} = f(\mathbf{x}_\mu \mathbf{R}) \cdot \boldsymbol{\theta} \tag{22}$$

Let's now suppose that our classifier, once trained, will predict the data well and give us a value distributed around the true value as a Gaussian (we actually know that this is true only close to the true value, large deviations act differently). We have that:

$$\mathcal{P}(y_\mu^{\text{true}}|\boldsymbol{\theta}) \propto e^{\left(y_\mu^{\text{true}} - \sum_i \theta_i z_{\mu i}\right)^2 \big/ (2D)} \tag{23}$$

where D is the variance of the Gaussian and $\mathbf{y}^{\text{true}} \in \mathbb{R}^{50000}$ represents the vector of the labels of the images of the training set. Using Bayes:

$$\mathcal{P}(\boldsymbol{\theta}|\mathbf{y}^{\text{true}}) \propto e^{-\left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2 \big/ (2D)} \mathcal{P}(\boldsymbol{\theta}) \tag{24}$$

where $\mathbf{z}$ is the matrix having $\mathbf{z}_\mu$ in the μ-row and $\|\cdot\|_e$ is the euclidean norm. Using a uniform prior doing MAP is the same that doing ML. Then find the optimal θ means maximize the likelihood and the optimal θ will be:

$$\hat{\boldsymbol{\theta}} = \text{argmin}\left(\left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2\right) = (\mathbf{z}^{\mathrm{T}}\mathbf{z} + \Gamma\mathbf{I})^{-1}\mathbf{z}^{\mathrm{T}}\mathbf{y}^{\text{true}} \tag{25}$$

where $\Gamma \in \mathbb{R}^+$ is a regolarization term that guarantees that what is inside the brackets is invertible.

2.1 Optimal value of θ - proof of the previous result

Our aim is to maximinize the function $\boldsymbol{\theta} \to e^{-\left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2 \big/ (2D)}$, since $D > 0$ it's equivalent to minimize the function $\boldsymbol{\theta} \to \left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2$. Let's consider initially $\Gamma = 0$ and replace $\boldsymbol{\theta}$ with the value given in Equation 25. We obtain:

$$\left\|\mathbf{y}^{\text{true}} - \mathbf{z}\hat{\boldsymbol{\theta}}\right\|_e^2 = \left\|\mathbf{y}^{\text{true}} - \mathbf{z}(\mathbf{z}^{\mathrm{T}}\mathbf{z})^{-1}\mathbf{z}^{\mathrm{T}}\mathbf{y}^{\text{true}}\right\|_e^2 = \left\|\mathbf{y}^{\text{true}} - \mathbf{z}\mathbf{z}^{-1}\left(\mathbf{z}^{\mathrm{T}}\right)^{-1}\mathbf{z}^{\mathrm{T}}\mathbf{y}^{\text{true}}\right\|_e^2 = 0. \tag{26}$$

Since $\forall\boldsymbol{\theta}$ we have the inequality $\left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2 \geq 0$ we have proved that the chioice $\hat{\boldsymbol{\theta}}$ minimizes the function $\boldsymbol{\theta} \to \left\|\mathbf{y}^{\text{true}} - \mathbf{z}\boldsymbol{\theta}\right\|_e^2$. Now we can consider the regularization parameter $\Gamma > 0$, with a continuity argument the result still holds.

2.2 Creation of the classifiers

We have created ten classifier, one for each digit, and then a global classifier that select the best prediction adding an extra layer that take the maximum among the outputs of the previous layer as shown in Fig. 10. We report in the following figures the errors computed using the validation set and the training

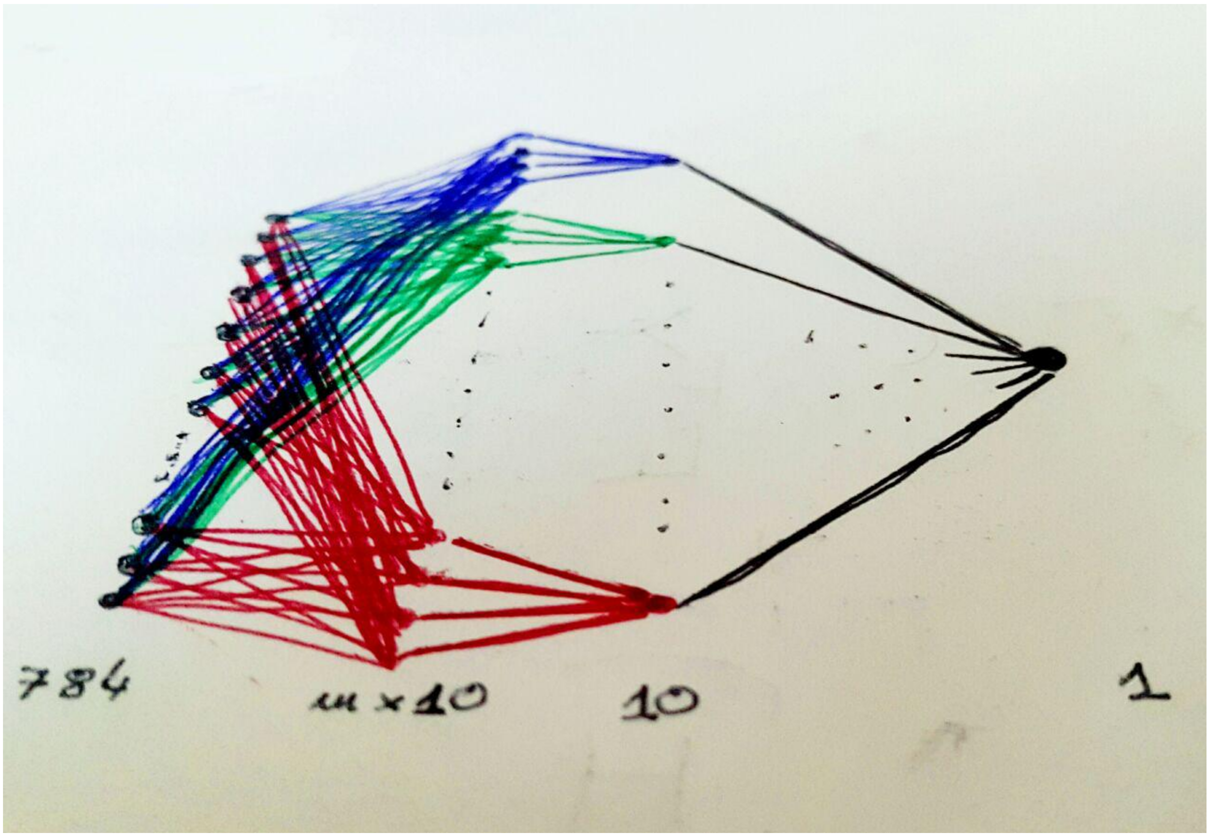
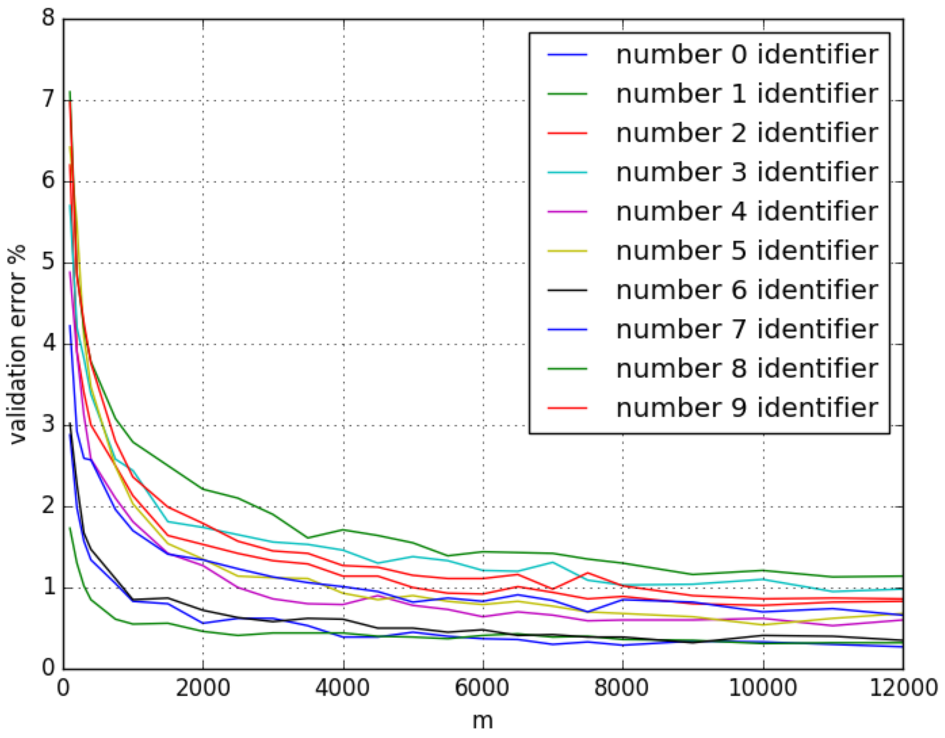
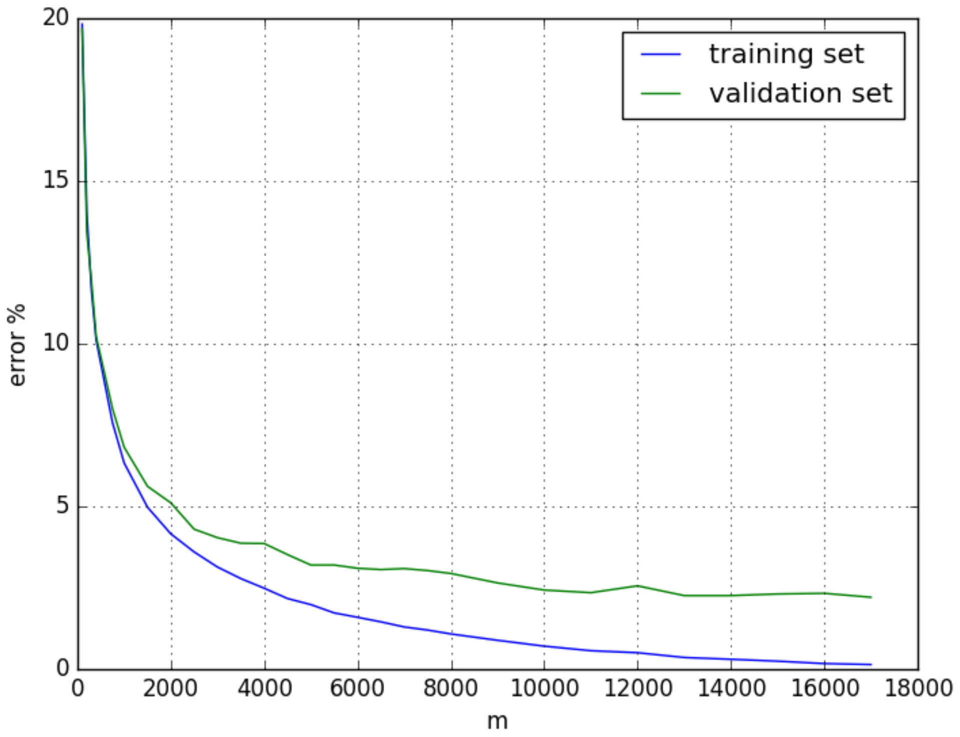


Figure 10: Neural Network structure. The input layer is composed of the 784 pixels of each images, the hidden layers are now two. One as before realize the projection from $n = 784$ to m dimension and applies the sigmoid function elementwise, the other applies the linear combination with each classifier theta vector. Eventually, the output vector use the max to choose the predicted digit.

set. Figure 11a show only the validation errors of the single digit classifiers. Instead Figure 11b show the overall error of the neural networks. In this Figure both training and validation set error are reported. The validation error always stay above the training error as expected. Increasing the dimension of space where we project our data, the validation error oscillates around 2.25%, while the error computed on the training set continues to decrease. Overfitting would be arise continuing to increase the dimension of m. In order to show it, even if it is not required, we trained the network inverting the dimension of the training and of the validation set. So now our training process has been done on a dataset of 10 000 elements, while the validation error has been computed on the other 60 000 pics. The computed errors are reported in Figure 12, it shows well the meaning of an overfitted neural network. Between $m = 4500$ and 5000 the training error goes to zero while the validation error starts to increase.



(a) Error (validation set) in percentage in each single classifier after training. The number of elements in the training set is 60 000 while in the validation set is 10 000. The gamma used during the whole process is 0.1.



(b) Errors in percentage of the networks after training. The number of elements in the training set is 60 000 while in the validation set is 10 000. The gamma used during the whole process is 0.1.

Figure 11

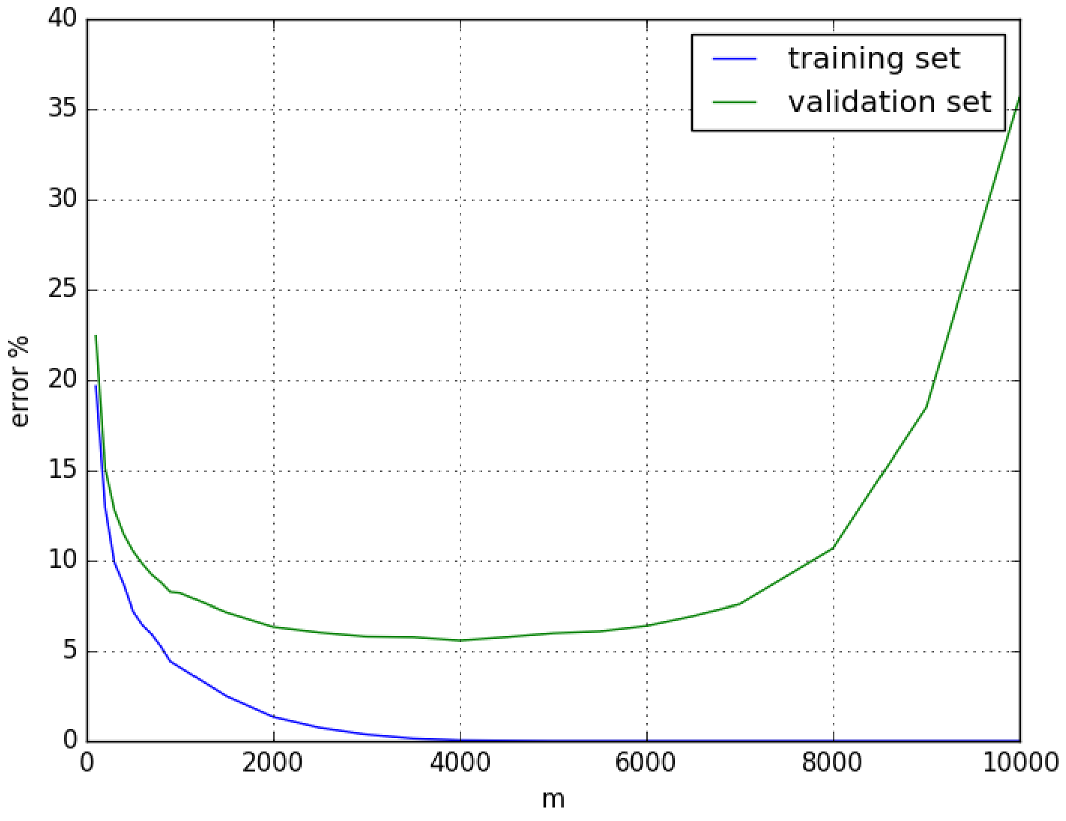


Figure 12: Example of overfitting obtained performing the training of the net with a small dataset (10 000 images) and computing the validation error on the other pics.

2.3 A brief story about memory and time

In order to look at the memory and at the computational time of our nets we firstly attached the code. We created two main classes. The first reproduce a single classifier (the node and the edges of one color in Figure 10) while the second `nist_classifier*␣reproduce␣all␣the␣network.␣The␣heavy␣coputational` cost of the network is consumed during the training phase.

Listing 1: HW5lib.py
This library contains the main classes of this second part of the exercise

```python
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
from scipy import linalg
from scipy import sparse
from functools import reduce

_mnst   = input_data.read_data_sets("../MNIST_data/", validation_size=10000)
imgs    = np.matrix(np.r_[_mnst.train.images,_mnst.test.images])
lbls    = np.matrix(np.r_[_mnst.train.labels,_mnst.test.labels]).T

imgs_val = np.matrix(np.r_[_mnst.validation.images])
lbls_val = np.matrix(_mnst.validation.labels).T


def static_vars(**kwargs):
    def decorate(func):
        for k in kwargs:
            setattr(func, k, kwargs[k])
        return func
    return decorate





@static_vars(_R = np.matrix([]))
class neural_net:
    def __init__(self, m, n, gamma=0.1, static_matrix=False, sparse=False):
        self._m = m
        self._n = n
        if static_matrix and not neural_net._R.any():
            np.random.seed(3141592653)
            neural_net._R = np.matrix(np.random.randn(n, m), dtype=np.float32)
        if not static_matrix:
            self._R = np.matrix(np.random.randn(n,m), dtype=np.float32)
        self._theta  = np.matrix(np.ones(m, dtype=np.float32))
        self._gamma  = np.float32(gamma)
        self._sparse = sparse

    @staticmethod
    def sigmoid(z):
        #return 1/(1+np.exp(-z))
        return np.reciprocal(np.add(1,np.exp(-z,z),z),z)

    def predict(self, x):
        return (self.sigmoid(x*self._R)*self._theta).A1

    @profile
    def train(self, x, y):
        z = neural_net.sigmoid(x*self._R)
        self._theta = np.matrix(linalg.solve(z.T*z+self._gamma*np.eye(self._m, dtype=np.
            float32), z.T*y))
        #self._theta = (z.T*z+self._gamma*np.eye(self._m, dtype=np.float32)).I * z.T*y
        return "done"





```

```python
class mnist_classifier:
    @staticmethod
    def y(lbls, i):
        return (lbls == i)*np.float32(2)-np.float32(1)

    @profile
    def __init__(self, m, imgs, lbls, gamma=0.1, static_matrix=False):
        self.nets = [neural_net(m, 28**2, gamma, static_matrix) for l in range(10)]
        print("training␣[{:<10}]␣{}%".format("", 0), end="\r")
        training  = [self.nets[l].train for l in range(10)]
        [f(imgs, self.y(lbls, i)) and \
                print("training␣[{:<10}]␣{}%".format("#"*(i+1), (i+1)*10),end="\r") \
                for i,f in enumerate(training)]
        print("␣"*70, end="\r")

    @profile
    def predict(self, img):
        predicting = [self.nets[l].predict for l in range(10)]
        m = map(lambda f: f(img), predicting)
        return map(lambda x: max(zip(x,range(10)))[1], zip(*m))

    @profile
    def compute_error(self, imgs, lbls):
        m = map(lambda x,y: x==y, self.predict(imgs), lbls.A1)
        return 100-sum(m)/len(lbls)*100
```

Let's now look at the memory used by our net. All the pics are extracted from the MNIST database are store by default as `numpy.float32` objects by tensorflow. We worked on them using `numpy.matrix` objects (compiled using BLAS libraries). Storing a single neural net means store in the memory a number of GiB of the order of the dimension of the random matrix. We will use in the following a projection space of dimension 1000 and we underline that we worked with `float32` because there is no need of more precision in order to work with the mnist database gray scale. The dimension of a random matrix with this characteristics is few MiB as we can see in the difference of height between the two light green delimiters in Figure 13. What is matter in the memory consumption is the handling and the operations performed on the huge) training set data. We focus on few consideration about the code.

1. First, the difference in performance and memory consumption between the regular numpy matrix inversion and the result obtained using the `scipy.linalg.solve` function. Inverting the matrix as in row 52 in Listing 1 is slower and more memory expensive than using scipy (row 51). This can be shown comparing the overall time of initialization of the network (57 versus 30 seconds).

2. Second, the peak that can be observed in the time profiler pictures is given by the sigmoid function (row42 and Figure 15) that has to inizialize a new matrix of dimensions 50000×1000 during the training and then apply the sigmoid function memberwise creating another matrix with the same huge dimension. We have avoided this let f acting in place (row43) using the `ufunc` of numpy, as it can be seen in Figure 16.

Figurer̃efmemoryfinal report the last implementation of our network focusing on the memory consumption due to the single digit classifier training inside our network.

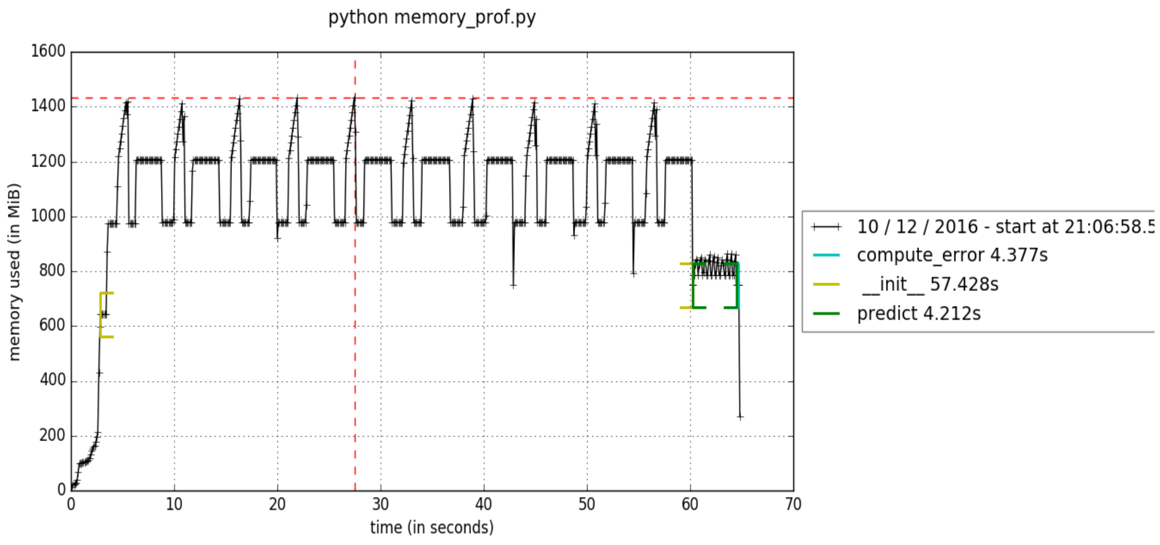


Figure 13: Memory and time profiler of the initialization of a net ($m = 1000$) trained inverting the matrix of row 52 in Listing 1.

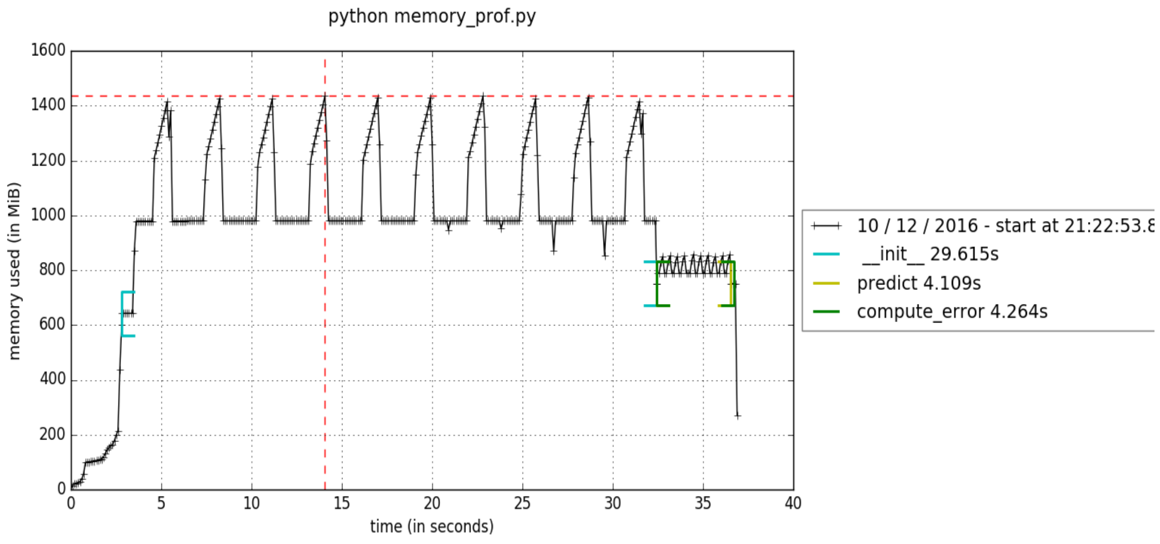


Figure 14: Memory and time profiler of the initialization of a net ($m = 1000$) trained with the solve function of scipy (row 51 in Listing 1). The computational time is less and there is no memory consumption.

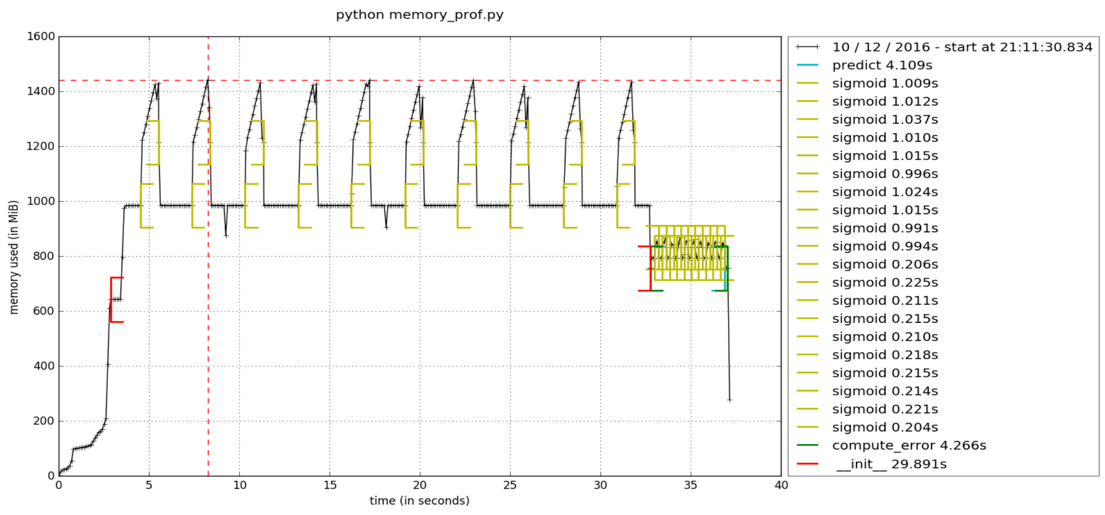


Figure 15: Focus on the memory consumption of the sigmoid function.

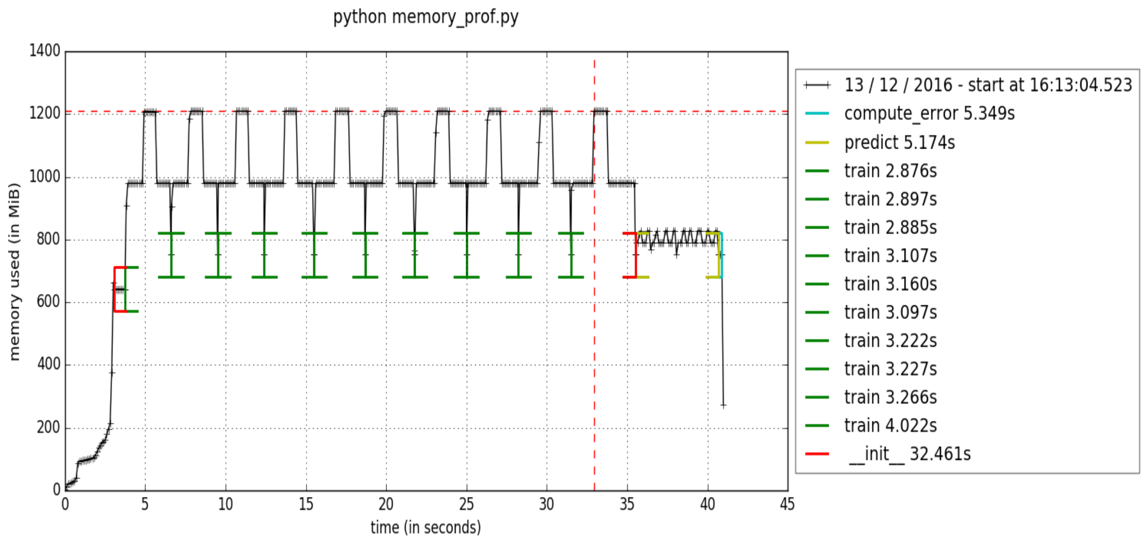


Figure 16: Memory over time profiler picture produce by the last version of the code in order to create one mnist classifier (m=1000) using the whole training set and computing the error on the validation set.

A Appendix: Computation of the Fisher Information

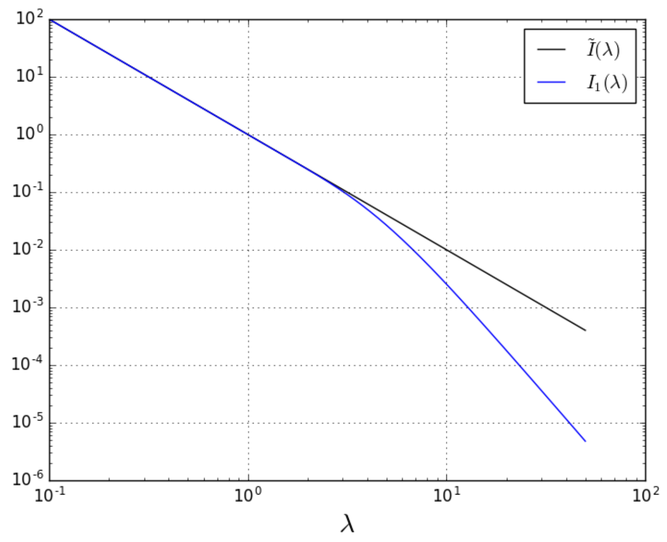
Now we are going to compute the Fisher Information $I_1(\lambda)$, where λ is the positive parameter in the probability density function $p_\lambda : [1, 20] \to \mathbb{R}$, $x \to \frac{e^{-x/\lambda}}{Z(\lambda)}$. The calculation is straightforward, we just use some fundamental concept of Information Theory and Statistical Mechanics as well as the basic proprieties of exponential functions under (in)definite integration:

$$
\begin{aligned}
I_1(\lambda) \triangleq \mathbb{E}\left[\left(\frac{\partial}{\partial\lambda}\log(p_\lambda(X))\right)^2\right] &= \int_1^{20} \frac{1}{p_\lambda(x)}\left(\frac{\partial}{\partial\lambda}p_\lambda(x)\right)^2 dx = \int_1^{20} p_\lambda(x)\left(\frac{x}{\lambda^2} - \frac{\mathbb{E}[X]}{\lambda^2}\right)^2 dx \\
&= \frac{1}{\lambda^4}\left(\mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2\right) = \frac{1}{\lambda^4}\left(\left[-\lambda p_\lambda(x)x^2\right]_1^{20} + 2\lambda\mathbb{E}[X] - \mathbb{E}[X]^2\right) \\
&= \frac{1}{\lambda^4}\left(\left[-\lambda p_\lambda(x)x^2\right]_1^{20} + 2\lambda\{[-\lambda p_\lambda(x)x]_1^{20} + \lambda\} - \{[-\lambda p_\lambda(x)x]_1^{20} + \lambda\}^2\right) \\
&= \frac{1}{\lambda^2} - \frac{1}{\lambda^4}\left(\lambda\left[p_\lambda(x)x^2\right]_1^{20} + \lambda^2\left([p_\lambda(x)x]_1^{20}\right)^2\right) \\
&= \frac{1}{\lambda^2} - \frac{1}{\lambda^4}\left(\frac{(20)^2e^{-20/\lambda} - e^{-1/\lambda}}{e^{-1/\lambda} - e^{-20/\lambda}} + \frac{\left[20e^{-20/\lambda} - e^{-1/\lambda}\right]^2}{\left[e^{-1/\lambda} - e^{-20/\lambda}\right]^2}\right) \\
&= \frac{1}{\lambda^2} - \frac{(20)^2 + 1 - 40}{\lambda^4}\frac{e^{-20/\lambda}e^{-1/\lambda}}{\left[e^{-1/\lambda} - e^{-20/\lambda}\right]^2} \\
&= \frac{1}{\lambda^2} - \frac{361}{\lambda^4}\frac{e^{-21/\lambda}}{\left[e^{-1/\lambda} - e^{-20/\lambda}\right]^2}.
\end{aligned}
$$

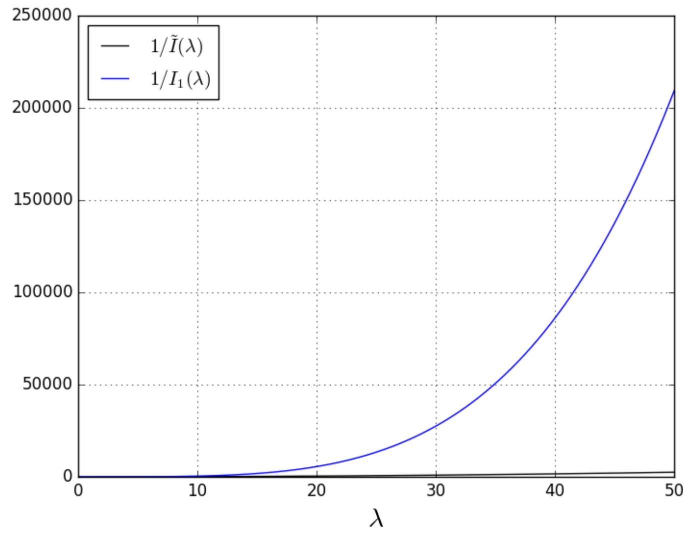
If now we compute the Fisher Information for an exponential distribution of meanλ with support $[0, +\infty)$:

$$
\tilde{I}(\lambda) = \mathbb{E}\left[\left(\frac{\partial}{\partial\lambda}\log\left(\frac{1}{\lambda}e^{-x/\lambda}\right)\right)^2\right] = \frac{1}{\lambda^2}
$$

So the Fisher information of the distribution $p_\lambda(x)$ is equal to $\tilde{I}(\lambda)$ minus a positive correction. This corrections turns out to be quite small, but it becomes really significant when considering $1/I_1(\lambda)$ for the correct behavior of the Cramer-Rao bound (see Figure 17).



(a) The difference between the two Fisher informations is really small. Notice the $\log - \log$ scale.



(b) From this plot one can realize how this small correction affects so importantly the Cramer-Rao lower bound.

Figure 17: Comparison of the Fisher informations $I_1(\lambda)$ and $\tilde{I}(\lambda)$ and their inverse, respectively for the distribution $p_\lambda(x)$ and the exponential with support $[0, +\infty)$.