ASI Assessed Exercise 2016/2017

Simone Rossi
Advanced Statistical Inference
EURECOM 2017

7. Juni 2017

Introduction

This notebook will be focused on two approaches of Bayesian inference: **Naive Bayesian Classifier** and the **Bayesian Linear Regressor**. Both of them will be implemented from scratch and tested on two well known dataset: **MNIST** and **CIFAR10**.

```
Using TensorFlow backend.
```

The MNIST and CIFAR10 datasets

Let's start by downloading and importing the two datasets. Since these are two famous datasets, some Python modules (in this case `Keras`) already provide support for handling training and testing.

```
MNIST dataset correctly downloaded and imported.

CIFAR10 dataset correctly downloaded and imported.
```

Dataset analysis

Let's look at some numbers: first of all let's comment on the dimensionality of the input.

```
Number of training samples for MNIST: 60000
Number of testing samples for MNIST : 10000
The shape of a MNIST sample is (28, 28)

Number of training samples for CIFAR10: 50000
Number of testing samples for CIFAR10 : 10000
Shape of a CIFAR10 sample is (32, 32, 3)
```

Each sample in MNIST is a 28×28 B&W image and therefore it has a dimensionality of 784 features; on the other hand, CIFAR is a collection of bigger colored images and, in particular, each sample has 3072 features: this is a huge difference with respect to MNIST, and since the number of training samples is even less, we will expect a great difference in the performances between MNIST and CIFAR. Let's now have a look on how many labels there are and how they are distributed.

```
Possible labels for MNIST: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Possible labels for CIFAR: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Both of them has 10 possible classes but since MNIST is database of handwritten digits, there's immediately a correspondence between the numerical label and the description. For CIFAR, though, the possible descriptive labels are the following.

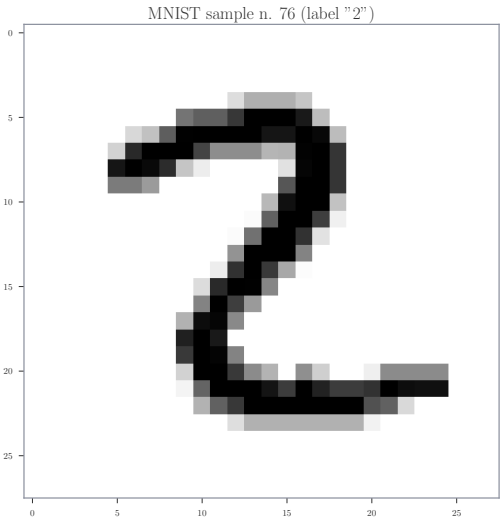
```
Possible descriptive labels for CIFAR:
          ['airplane', 'automobile', 'bird', 'cat', 'deer']
          ['dog', 'frog', 'horse', 'ship', 'truck']
```

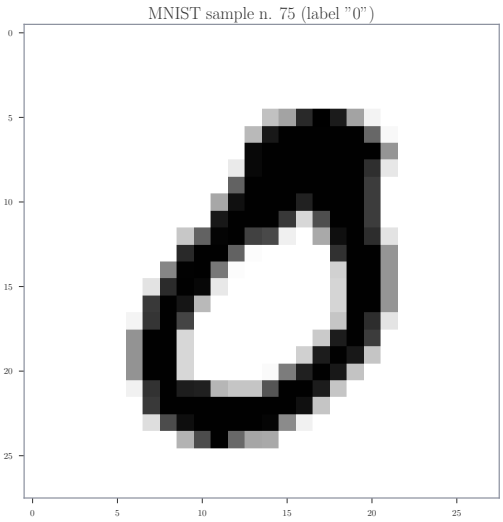
Let's look at the distribution of labels in the two training datasets

	MNIST	CIFAR
Label 0:	5923	5000
Label 1:	6742	5000
Label 2:	5958	5000
Label 3:	6131	5000
Label 4:	5842	5000
Label 5:	5421	5000
Label 6:	5918	5000
Label 7:	6265	5000
Label 8:	5851	5000
Label 9:	5949	5000

It's interesting to notice that MNIST labels are not uniformly distributed; nevertheless, since the difference is not so big, for the sake of simplicity for both MNIST and CIFAR all labels will be treated as equally probable.

Let's show some pictures to better understand the problem we're working with.





Naive Bayes Classifier

Without describing in details how the **Naive Bayes Classifier** works, here we just present some useful identities and equations for the implementation. Let's start by looking at the Bayes rule.

$$P(t_{new} = k|\mathbf{X}, \mathbf{t}, \mathbf{x_{new}}) = \frac{P(\mathbf{x_{new}}|t_{new} = k, \mathbf{X}, \mathbf{t})P(t_{new} = k)}{\sum_{j=0}^{K-1} P(\mathbf{x_{new}}|t_{new} = j, \mathbf{X}, \mathbf{t})P(t_{new} = j)} \tag{0.1}$$

In our case, we set a **Uniform Prior** and a **Gaussian Likelihood** whose parameters are obtained as follows.

$$P(\mathbf{x}|t = k, \mathbf{X}, \mathbf{t}) = \prod_{d=0}^{D-1} \mathcal{N}(\mu_{kd}, \sigma_{kd}) \tag{0.2}$$

where $\mu_{kd} = \frac{1}{N_k} \sum_{n:t_n=k} x_{nd}$ and $\sigma_{kd} = \frac{1}{N_k} \sum_{n:t_n=k} (x_{nd} - \mu_{nd})^2$

For better numerical stability, also the approximated **Gaussian LogLikelihood** has been defined and implemented as follows:

$$\log P(\mathbf{x}|t = k, \mathbf{X}, \mathbf{t}) = \sum_{d=0}^{D-1} \log \mathcal{N}(\mu_{kd}, \sigma_{kd}) \tag{0.3}$$

Taking advantage of the Jensen's Inequality, the label prediction of a samples reduces as follows

$$t_{new} = argmax_k \left\{ \log P(\mathbf{x}|t = k, \mathbf{X}, \mathbf{t}) + \log P(t = k) - \sum_{j=0}^{K-1} \log P(\mathbf{x}|t = j, \mathbf{X}, \mathbf{t}) \right\} \tag{0.4}$$

Implementation of the Naïve Bayes classifier

To avoid filling the report with pages of code, the implementation can be viewed on the HTML version of the notebook. The important remark is that all the possible computations are done in matrix form taking advantage of the `C`-like performance of Numpy.

Let's run the training and the predictions with few images, just to prove the correctness of this implementation.

```
Training completed in 1.10 seconds.
```

```
k = 0 : 345729.851221
k = 1 : 362996.976449
k = 2 : 343982.455369
k = 3 : 439455.35421
k = 4 : 439927.686773
k = 5 : 440354.172781
k = 6 : 262095.888977
k = 7 : 441194.107172
k = 8 : 439030.597706
k = 9 : 441133.748314
```

```
k = 0 : 642525.431998
k = 1 : 654336.234654
k = 2 : 666777.161808
```

```
k = 3 : 589085.642204
k = 4 : 641678.075029
k = 5 : 662105.351307
k = 6 : 661814.741789
k = 7 : 386607.822711
k = 8 : 635668.013868
k = 9 : 450131.715342
```

The predictions are, respectively, 7 and 2. Let's see which are the true labels.

```
True labels: [7 2]
```

This is encouraging. Afterward we will run training and prediction on the whole dataset for both MNIST and CIFAR.

Strengths and weaknesses of the Bayesian Classifier

The naive Bayes assumption is helpful when the dimensionality D of the input space is high, making density estimation in the full D-dimensional space more challenging. Another strength of this classifier is its non-parametric nature: training and prediction are done without writing down any equation that relates pixels to labels.

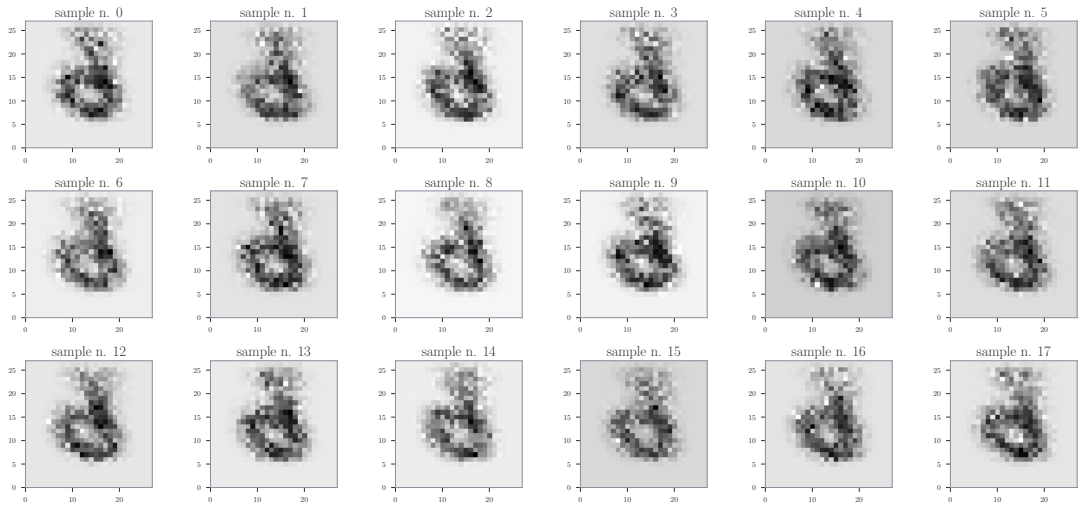
On the other hand, though, to make it scalable we have done strong assumptions on the distribution of pixel values; for instance, to write the likelihood we made the approximation that each component of $\mathbf{x}$ is independent from the others: this is questionable. Intuitively, it's reasonable to assume that neighbors of a given pixel will share similar gray level and will not have significant differences. Nevertheless, this cross-correlation it's not taken into account.

Data preprocessing

In this particular case, the first data preprocessing to be do is to flatten all the images: samples come in matrix form but they have to be converted into feature vectors.

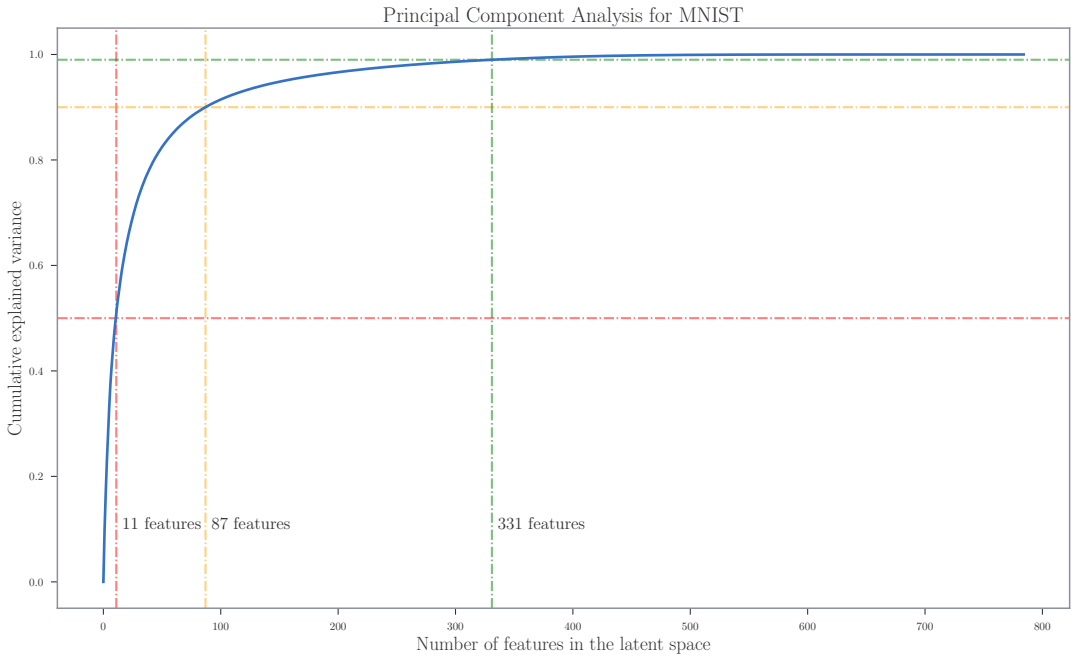
Secondly, since pixel ranges from 0 to 255, to guarantee better numerical stability (in particular to avoid overflows while computing intermediate results), it's better to normalize them all between 0 and 1.

Finally, the dimensionality on the input space is much more than the required to distinguish ten classes. For instance, look at the next picture.



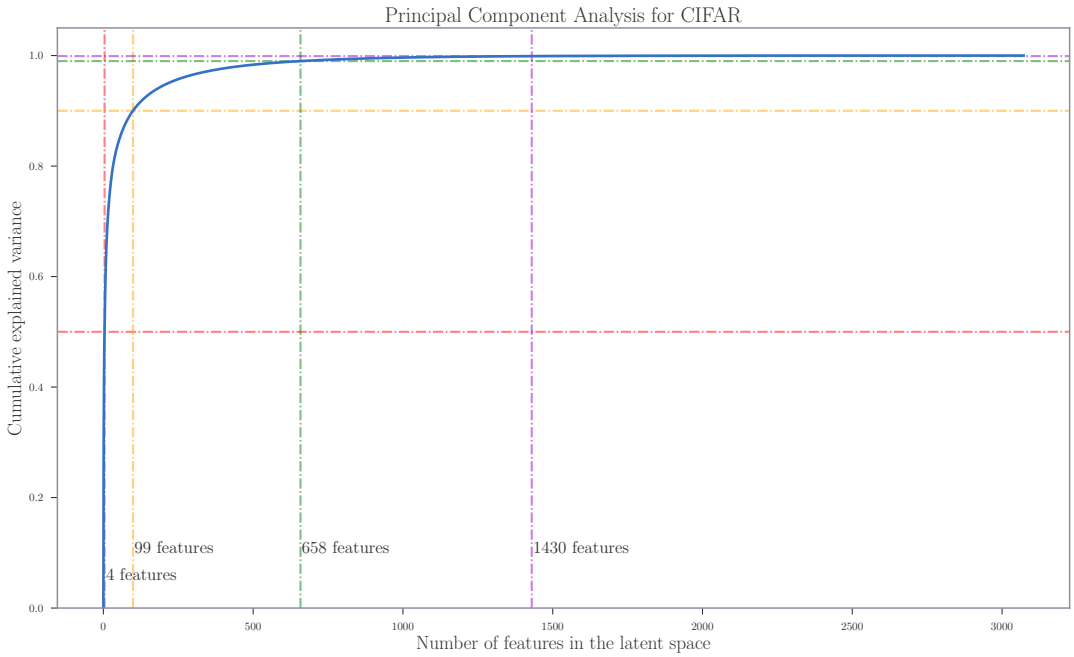
These are 18 images coming from sampling a normal distribution with mean and variance of the digit `9`. As we can see, none of them is close to the original number; this suggest us that either the number of training samples is too low or the number of dimensions is too high. Since we cannot change the dataset, the only way to mitigate this effect is to reduce the dimensionality of the input. Therefore, a `PCA` or `KernelPCA` is strongly suggested to increase the performances.

Let's see how much the dimensionality can be reduced without losing too much information.



This is really interesting. The analysis of the cumulative explained variance suggests that in order to maintain 50% of the information we need only 11 dimensions. The number increase to 87 for the 90% of information and to 331 for 99% of information. This means that if we accept to loose 1% of data we can reduce the number of feature to more than one half.

Let's do the same on the CIFAR and discuss the results.



Here the result are more impressive. For the 99% of information, only 658 features are needed. Even if the accuracy has to be 99.9%, the number of features can be more than halved. This is an huge saving.

Performance Analysis

Let's train the classifier on both MNIST and CIFAR, without dimensionality reduction.

```
Training completed in 1.17 seconds.
Training completed in 4.30 seconds.
```

Let's run now the predictions.

```
Starting prediction...


100%|#########|Time: 0:00:09


Starting prediction...


100%|#########|Time: 0:00:18
```

Let's now look to some simple metric for performance evaluation. The first one is the accuracy score, which simply counts one many predictions were correct on the whole test set.

```
Accuracy for MNIST: 0.6713
Accuracy for CIFAR: 0.2976
```

As expected accuracy is not so high and in particular our hypothesis on the high dimensionality of CIFAR was correct.

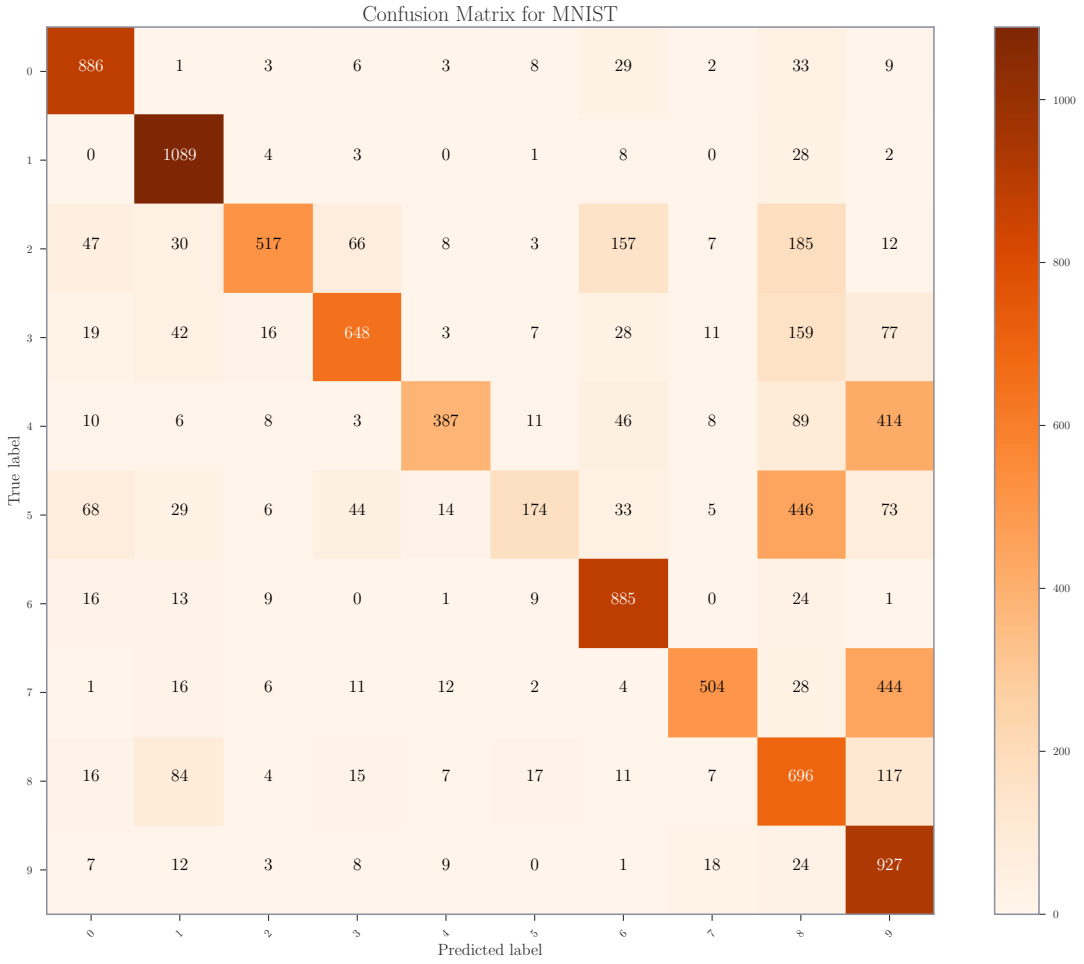
Let's see now the **loglikelihood** on the test data.

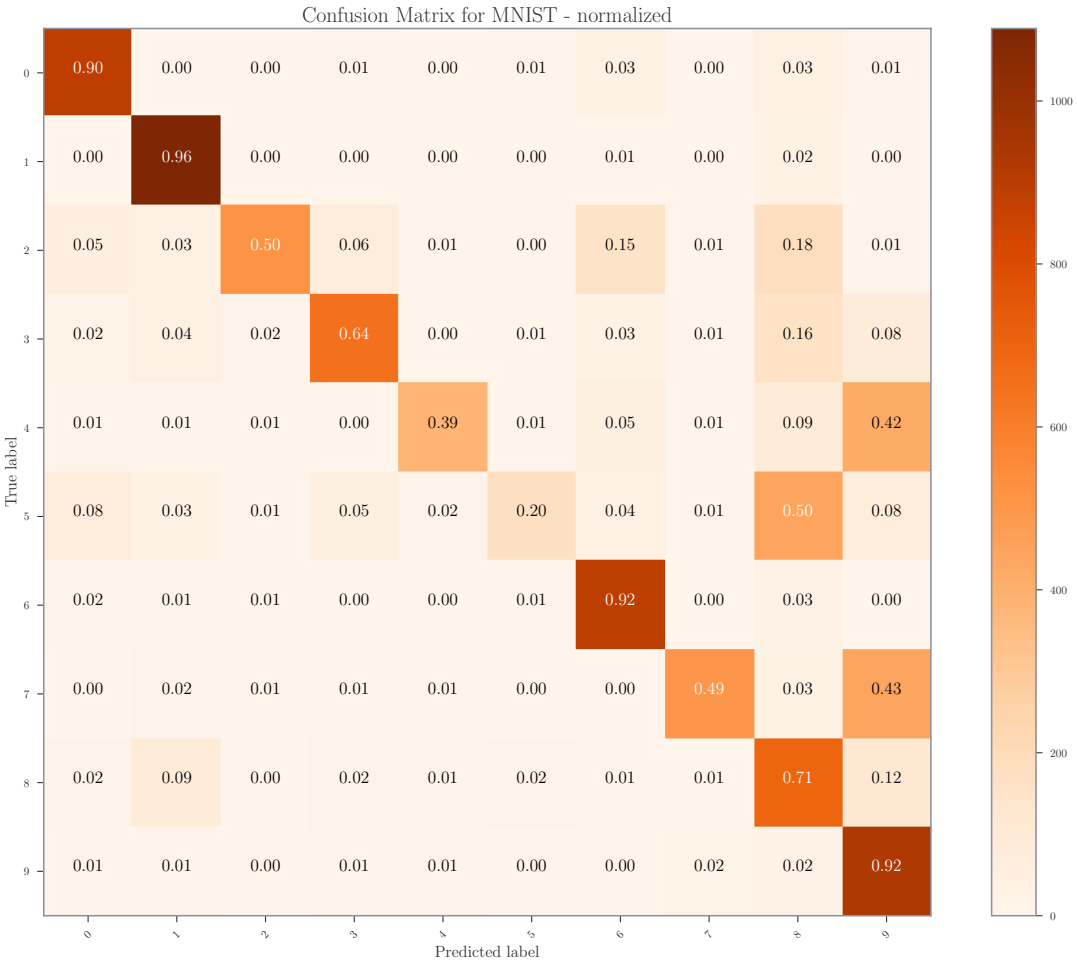
```
Log-Likelihood on test for MNIST: -10.26
Log-Likelihood on test for CIFAR: -23.55
```

This is another figure of merit for analysing the performances of the classifier. In the contex of maximizing the likelihood, the Bayesian classifier for MNIST performs way better than for CIFAR.

Confusion Matrix

Let's now see the **confusion matrix** for both the datasets.





From the analysis of this confusion matrix, we can notice some interesting trends. For instance, 0, 1, 6, 8 and 9 are the best well-classified digits; on the other hand, though, due to similarity in the shapes, 4 and 5 are nearly never correctly labeled: 4 is most of the times confused with 9 and 5 with 8. Finally, 2, 3 and 7 are classified with the correct label about 50% of the times.

Here, the situation is much more tragic: almost all the classes are never correctly labeled and, even for the best one, more than half of the samples are missclassified.

Final discussion on performance

As said before, this simple classifier assumes independence between features (see Eq. 2). This underestimates the spacial information that each pixel carries on: as a consequence in order to increase the performance, the first thing to do is to break this assumption using, for instance, a **Multivariate Gaussian Likelihood**.

Even though the performances of the Naive Bayesian Classifier do not overcome the state-of-art, intuitively, we think that it can easily outperform a random classifier (which ideally should have $1/K$ accuracy). Let's see if it's the case or not.

```
Accuracy for random classifier with MNIST: 0.1007
Accuracy for random classifier with CIFAR: 0.1001
```

Bayesian Linear Regression

Implementation of Bayesian Linear Regression

Same as before, to avoid filling the report with pages of code, the implementation can be viewed on the HTML version of the notebook. The important remark is that all the possible com-

putations are done in matrix form taking advantage of the C-like performance of Numpy.

Regression for a classification problem?

Let's now train and predict on MNIST, for the moment with order 2.

```
Training completed in 23.11 seconds.


Predictions done in 13.19 seconds.
```

And now on CIFAR.

```
Training completed in 394.72 seconds.


Predictions done in 128.63 seconds.
```

Performance evaluations

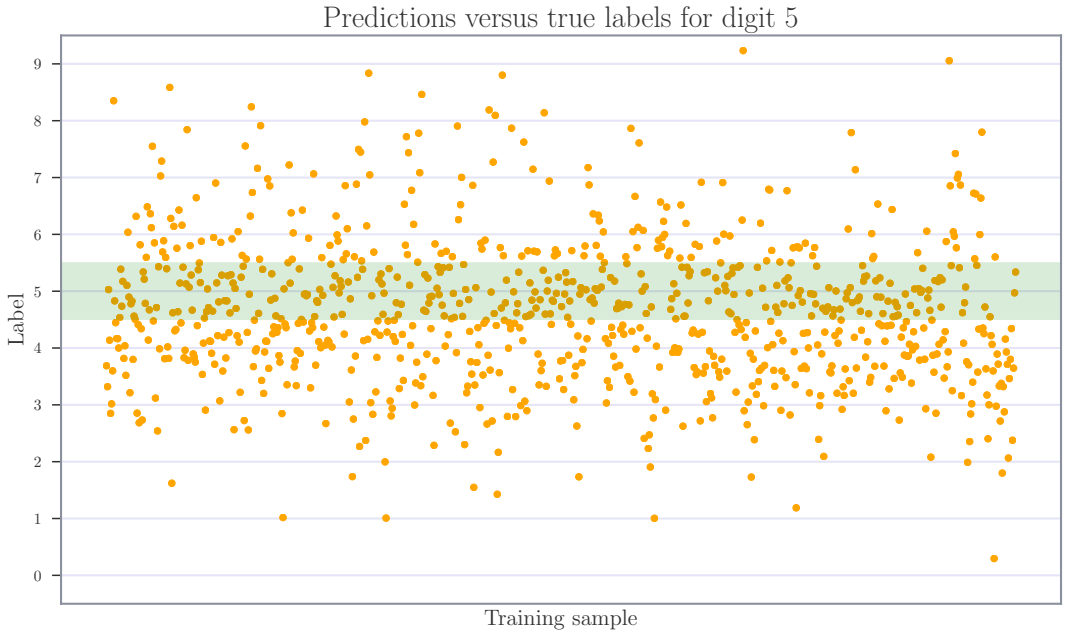
Let's print some results, to see what we're dealing with.

```
[MNIST] True labels: [7 2 1 0 4]
[MNIST] Prediction means: ['6.44', '0.70', '1.43', '0.29', '4.44']
[MNIST] Prediction variances: ['4.1e12', '4.2e12', '4.0e12', '4.6e12', '4.1e12']

[CIFAR] True labels: [3 8 8 0 6]
[CIFAR] Prediction means: ['6.84', '5.16', '5.22', '3.79', '3.62']
[CIFAR] Prediction variances: ['7.5e13', '9.8e13', '8.7e13', '9.5e13', '7.5e13']
```

We can already appreciate some problems here: even though the prediction means for MNIST are not far away from the true labels, the variance on the predictions is huge, meaning that the model is completely insecure on the actual outcomes. For CIFAR, on the other hand, also predictions seem to be quite wrong.

Let's now see some plots and let's see how the digit `5` is labeled.

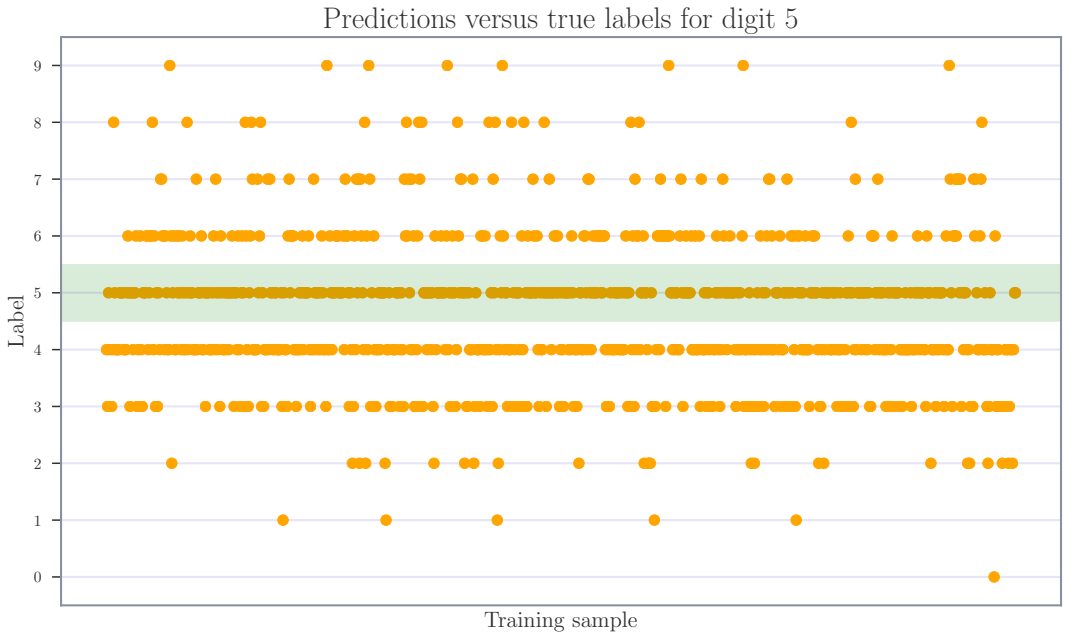


This plot displays all the samples associated to label 5 and prediction of those samples (the green band represents the range $\pm$ 0.5). Since this is a regression problem, we could use also the Mean Squared Error as figure of merit for analyzing the performances.

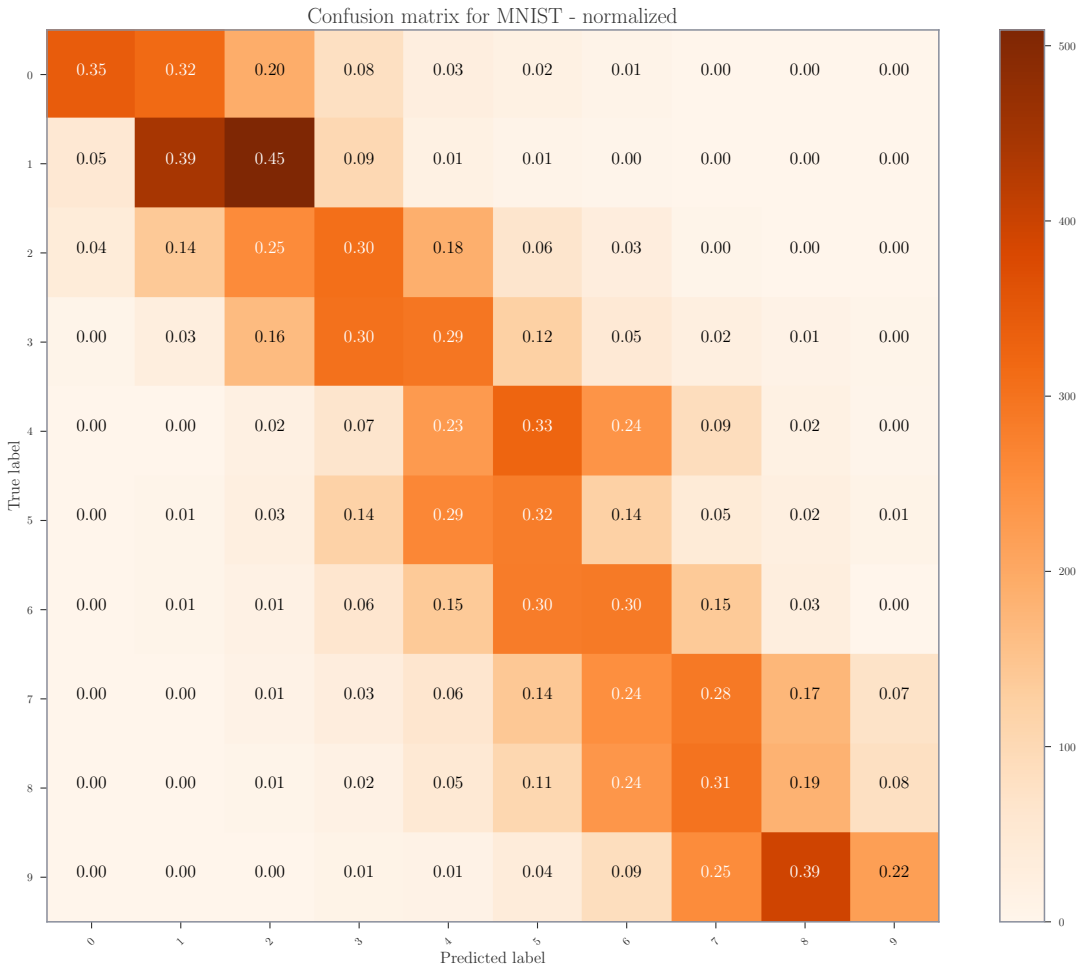
```
MeanSquaredError for MNIST: 2.4805
MeanSquaredError for CIFAR: 8.1734
```

Label discretization

The most immediate way to perform label discretization is to look into ranges of values and associate for each of them a label. The next plot is the same as before but with label discretization.

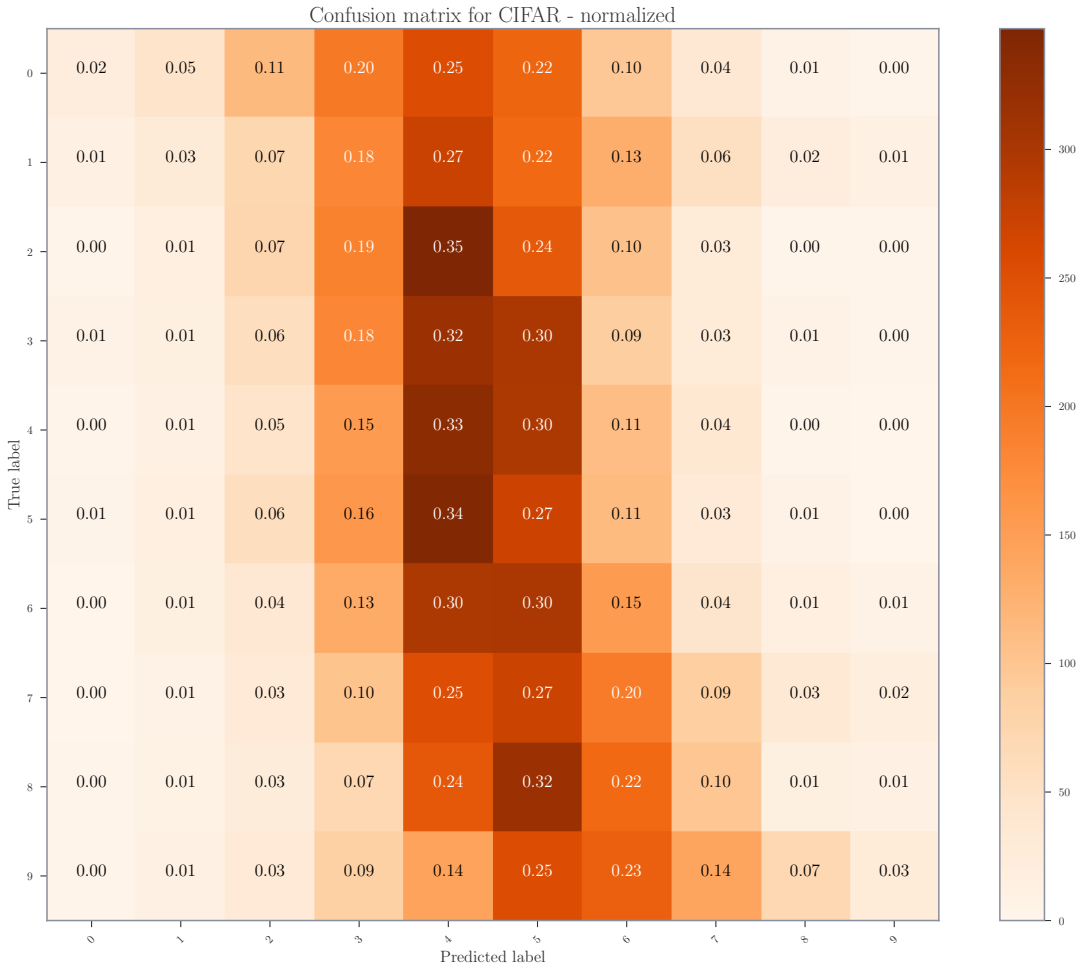


Using this discretization strategy, let's see the the confusion matrix for both the datasets.



Some interesting patterns can be individuated here. Since we're treating this problem like a regression problem, where we implicitly assume labels to be continuous, is correct that predictions are (*equally*) spread around the ground truth: ideally one should have differences smaller than 0.5 to be correctly labeled. This is not the case, but it's likely to be the case increasing the model complexity.

Let's do the same also for CIFAR.



No much to say here, this model for CIFAR is completely useless. Maybe increasing the complexity, the situation would change. Unfortunately, this was not possible to do due to the huge computation time needed in this use case.

As figure of merit, we can also use the accuracy ratio.

```
Accuracy for MNIST: 0.2838
Accuracy for CIFAR: 0.1188
```

Not much surprise, performances are quite poor.

Classification or regression?

Theoretically, classification and regression are two different point of view of the same problem: model a relationship between inputs and outputs. The reasons why regression performs so badly on these dataset must be found on the high dimensionality of the input and on the low order of polynomials used to create the design matrix.

Weaknesses of linear regression for classification

Linear regressions are sensitive to **outliers**; in fact, if most of the data lives in a small neighborhood, but there are one or two samples completely different, this could significantly swing the regression results.

Secondly, it is easy to **overfit** the model such that the regression begins to model the random error (noise) in the data, rather than just the relationship between the variables. This most commonly arises when you have too many parameters compared to the number of samples (it's unlikely in this task, though).

Finally, linear regressions are meant to describe **linear relationships** between variables. So, if there is a nonlinear relationship, then the resulting model will be wrong. In our case, we cannot guarantee to be in this privileged condition.

Bonus: Multilabel Linear Regression with Soft Max Layer

We can move from the multiclass problem to a binary problem: this is justified by the fact that in general multilabel classification is much more challenging than binary. Therefore, for instance, we can use the **One-vs-All Strategy**: we can build N different binary classifiers and for the i^{th} classifier, let the positive examples be all the points in class i, and let the negative examples be all the points not in class i . Let f_i be the i^{th} classifier. According to this rule, we now classify with

$$\tilde{y} = f(\mathbf{x}) = \text{argmax}_i(f_i) \tag{0.5}$$

In addition to this, in the spirit of **Deep Learning** approach, we add an additional level of *Soft Max* before drawing conclusions on the outcomes, so that the results can be interpreted as probabilities.

$$y = \text{argmax}_i(\tilde{y}_i) \quad \text{where} \quad \tilde{y}_i = \frac{\exp\left(f(\mathbf{x})_i\right)}{\sum_{k=0}^{K-1} \exp\left(f(\mathbf{x})_k\right)} \tag{0.6}$$

Before doing so, though, we need to change the encoding of the label to the **Hot Encoding** format.

```
4 -> [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
1 -> [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
9 -> [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
```

Let's now run the training and the predictions on the MNIST dataset and, if the results are encouraging, also on CIFAR. Let's start on MNIST with regression order three.

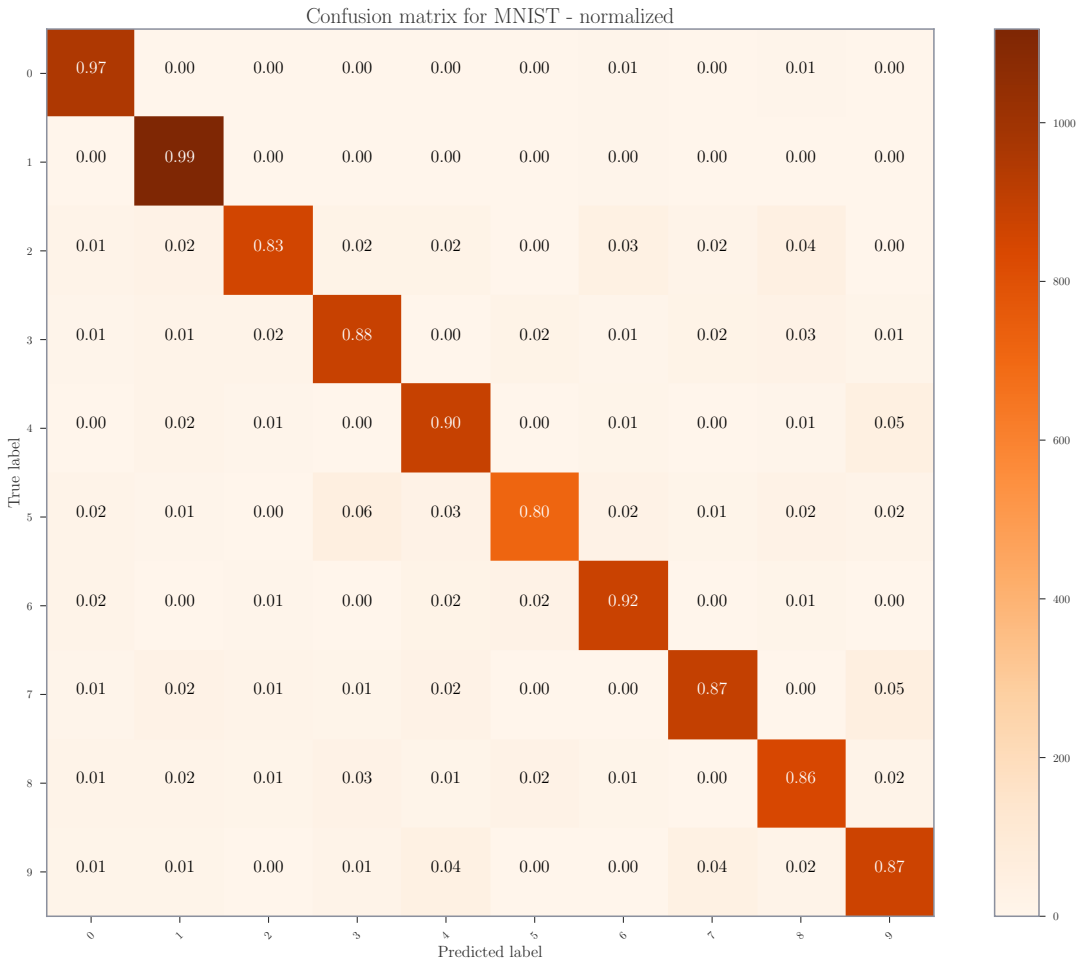
```
Training completed in 57.35 seconds.
Training completed in 51.51 seconds.
Training completed in 53.65 seconds.
Training completed in 45.97 seconds.
Training completed in 45.08 seconds.
Training completed in 52.02 seconds.
Training completed in 49.00 seconds.
Training completed in 44.90 seconds.
Training completed in 50.06 seconds.
Training completed in 48.53 seconds.


Predictions done in 18.84 seconds.
Predictions done in 18.73 seconds.
Predictions done in 18.77 seconds.
Predictions done in 19.24 seconds.
```

```
Predictions done in 18.65 seconds.
Predictions done in 18.75 seconds.
Predictions done in 22.23 seconds.
Predictions done in 19.20 seconds.
Predictions done in 21.36 seconds.
Predictions done in 21.82 seconds.
```

Let's see how it performs using the `Accuracy score`, the `Loglikelihood` and the `Confusion Matrix`.

```
Accuracy score for MNIST...:  0.8912
Log-Likelihood on test for MNIST: -1.7589
```



Now, this is much more interesting than before! From the analysis of the accuracy, this model is more than 35% better than the Naive Bayesian classifier and overcome twice the performance of the plain Bayesian Linear Regression. This is one of the top scores that can be achieved with the MNIST dataset without using Deep Learning models.

This is really encouraging. We can move to CIFAR.

```
4 -> [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
8 -> [ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
```

```
6 -> [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
```

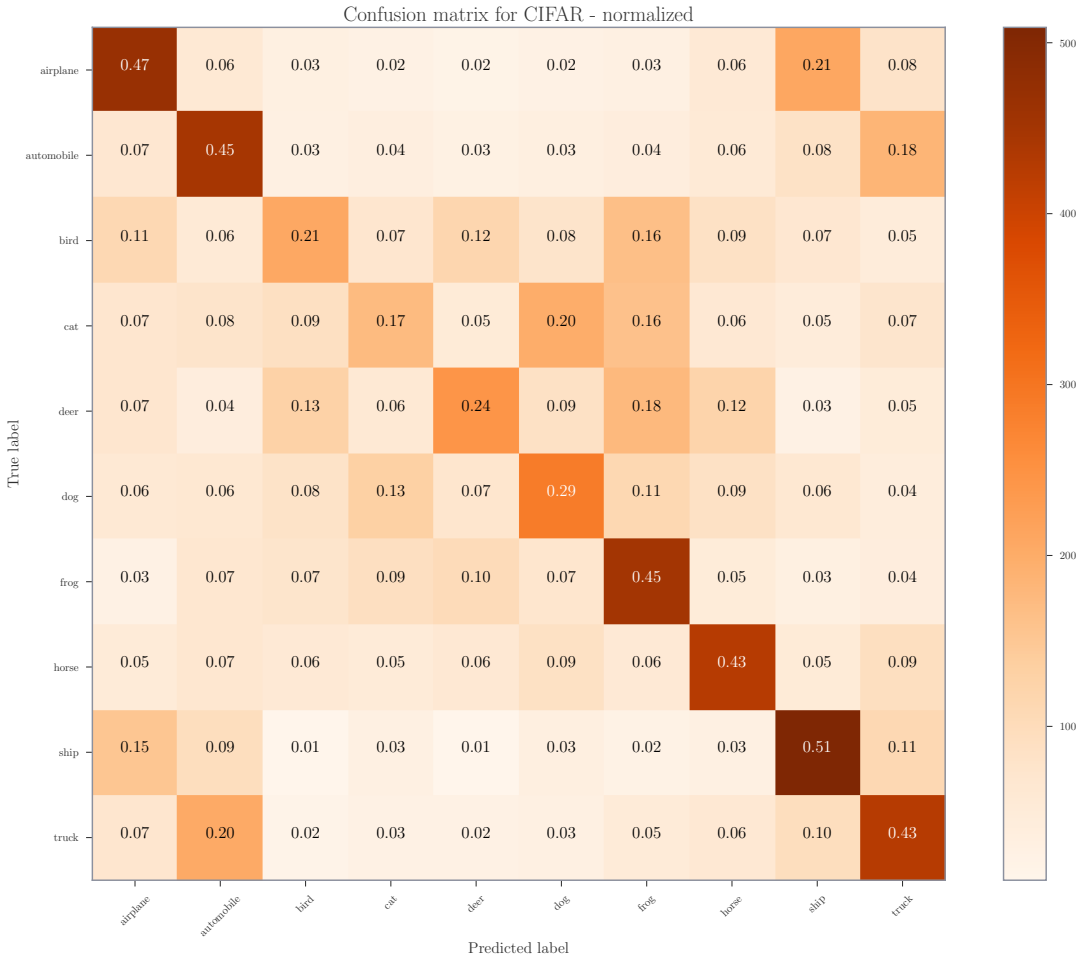
Let's run.

```
Training completed in 140.56 seconds.
Training completed in 130.41 seconds.
Training completed in 126.49 seconds.
Training completed in 149.08 seconds.
Training completed in 141.59 seconds.
Training completed in 140.85 seconds.
Training completed in 124.96 seconds.
Training completed in 141.81 seconds.
Training completed in 153.30 seconds.
Training completed in 139.50 seconds.


Predictions done in 55.36 seconds.
Predictions done in 62.05 seconds.
Predictions done in 68.58 seconds.
Predictions done in 74.07 seconds.
Predictions done in 64.78 seconds.
Predictions done in 86.18 seconds.
Predictions done in 81.42 seconds.
Predictions done in 67.25 seconds.
Predictions done in 55.41 seconds.
Predictions done in 62.74 seconds.
```

And let's measure the performances.

```
Accuracy score for CIFAR...:  0.3636
Log-Likelihood on test for CIFAR: -2.1891
```



The same trends can be drawn also here; as expected CIFAR is much more challenging dataset but this method outperforms both the Naive Bayesian classifier and the standard Bayesian Linear regression.

Conclusions

In this notebook, we implemented and tested two of the most well known models for statistical inference: the **Naive Bayesian Classifier** and the **Bayesian Linear Regression**. For both of them, the most informative metrics have been used for evaluating their performances on two classic dataset **MNIST** and **CIFAR**.