Project 1 Statistical Machine Learning

- Edgar E[redacted] -

- Tim [redacted]

- Thibaud [redacted]

- Henrik SÖDERHIELM: [redacted]

November 2018

Contents

1 Stochastic Sub-gradients for Training Support Vector Machines

Let's describe the general setup. We want to predict y which is one of K labels for a given image x. The image is $8 \times 8 = d$ pixels. Thus we decode the input space as $X = [0,1]^d$ and the output space $Y = \{i\}_{0 \le i < K}$. Our classifier is described by $\Theta \in \mathbb{R}^{d \times K}$. For given input vector $x \in X$ and weight matrix Θ the predicted label $\hat{y}$ is determined by

$$\hat{y} = \underset{j=0,\dots,K-1}{\arg\max} \sum_{i=0}^{d-1} x_i \Theta_{ij} \tag{1}$$

1.1 Question a

Let λ be in (0,1) and (α, β) in $\mathbb{R}^{d\times K} \times \mathbb{R}^{d \times K}$. We have:

$$R^{\text{emp}}\left(\lambda\alpha + (1-\lambda)\beta\right) = \frac{1}{n}\sum_{k=1}^{n} \max_{j \neq y_k} \left(1 + \sum_{i=1}^{d} x_{k,i}\left(\lambda\alpha_{ij} + (1-\lambda)\beta_{ij} - \lambda\alpha_{ik} - (1-\lambda)\beta_{ik}\right)\right)$$

We can separate the sum in two parts and use the fact that $1 = \lambda + (1-\lambda)$:

$$R^{\text{emp}}\left(\lambda\alpha + (1-\lambda)\beta\right) = \\ \frac{1}{n}\sum_{k=1}^{n} \max_{j \neq y_k} \left(\lambda + \sum_{i=1}^{d} x_{k,i}(\lambda\alpha_{ij} - \lambda\alpha_{ik}) + (1-\lambda) + \sum_{i=1}^{d} x_{k,i}\left((1-\lambda)\beta_{ij} - (1-\lambda)\beta_{ik}\right)\right).$$

And then, since the maximum of a sum is less or equal to the sum of the maximums, we have the inequality :

$$R^{\text{emp}}\left(\lambda\alpha + (1-\lambda)\beta\right) \le \\ \frac{1}{n}\left[\sum_{k=1}^{n} \max_{j \neq y_k}\left(\lambda + \sum_{i=1}^{d} x_{k,i}(\lambda\alpha_{ij} - \lambda\alpha_{ik})\right) + \sum_{k=1}^{n} \max_{j \neq y_k}\left((1-\lambda) + \sum_{i=1}^{d} x_{k,i}\left((1-\lambda)\beta_{ij} - (1-\lambda)\beta_{ik}\right)\right)\right]$$

And thus we have, after reorganizing the terms :

$$R^{\text{emp}}\left(\lambda\alpha + (1-\lambda)\beta\right) \le R^{\text{emp}}(\lambda\alpha) + R^{\text{emp}}\left((1-\lambda)\beta\right)$$

so R^{emp} is a convex function.

1.2 Question b

Now we want to compute a subgradient G of $R^{\text{emp}}(\Theta)$ for given pair (x, y). First assume we have only one data point i.e. $n = 1$ and thus $L(\Theta, (x, y)) = R^{\text{emp}}(\Theta)$. Let's consider j^* given by

$$j^* := \underset{j \neq y}{\arg\max} \left[1 + \sum_{i=1}^{d} x_i(\Theta_{ij} - \Theta_{iy})\right]_+ \tag{2}$$

Thus we have

$$L(\Theta, (x,y)) = \left[1 + \sum_{i=1}^{n} x_i(\Theta_{ij^*} - \Theta_{iy})\right]_+ \tag{3}$$

Now we have two cases: either $L(\Theta, (x, y)) = 0$ or $L(\Theta, (x, y)) > 0$.

Case 1 $L(\Theta, (x, y)) = 0$. This means that

$$1 + \sum_{i=1}^{d} x_i(\Theta_{ij^*} - \Theta_{iy}) \leq 0 \qquad \forall j \tag{4}$$

If the inequality is strict then it is obvious that it is differentiable and the gradient is just the Zero Matrix (same size as Θ). If we have equality we still have the same Zero Matrix as a subgradient. Thus we can have

$$G = \mathbf{0} \in \mathbb{R}^{d \times K} \tag{5}$$

Case 2 $L(\Theta, (x, y)) > 0$. This means that

$$L(\Theta, (x, y)) = \left[1 + \sum_{i=1}^{n} x_i(\Theta_{ij^*} - \Theta_{iy})\right]_+ \tag{6}$$

$$= 1 + \sum_{i=1}^{n} x_i(\Theta_{ij^*} - \Theta_{iy}) \tag{7}$$

If we now take the derivative of L w.r.t. the j-th column, we get

$$\frac{\partial L}{\partial [\Theta_{ij}]_{1 \leq i \leq d}} = \begin{cases} x & \text{if } j = j^* \\ -x & \text{if } j = y \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

If we put everything together, in the second case we get a subgradient of

$$G = \begin{array}{cccccccccc} \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \\ 0 & \cdots & 0 & x & 0 & \cdots & 0 & -x & 0 & \cdots \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \\ & & & \uparrow & & & & \uparrow & & \\ & & & j^* & & & & y & & \end{array} \tag{9}$$

1.3 Question c

In this section, we implement four important functions to compute the subgradient. **The Python Code can be found in the appendix.** `loss_function` just computes the value of $L(\Theta, (x, y))$ and j^*. `stochastic_subgradient` computes one stochastic subgradient for given (x, y). `svmsubgradient` computes the subgradient as the mean of all stochastic subgradients. `sgd` is the (convex-)stochastic gradient descend method.

1.4 Question d

We tested the performance on a test set of size 100 and on training sets of size 20, 50, 100, 500, 1000, and 1500. In Figure 1 the score (#correct classified / #total), f1 and precision is showed. With enlarged training set all these scores become better. The bars show the standard deviation over 30 trials, which is quite big.

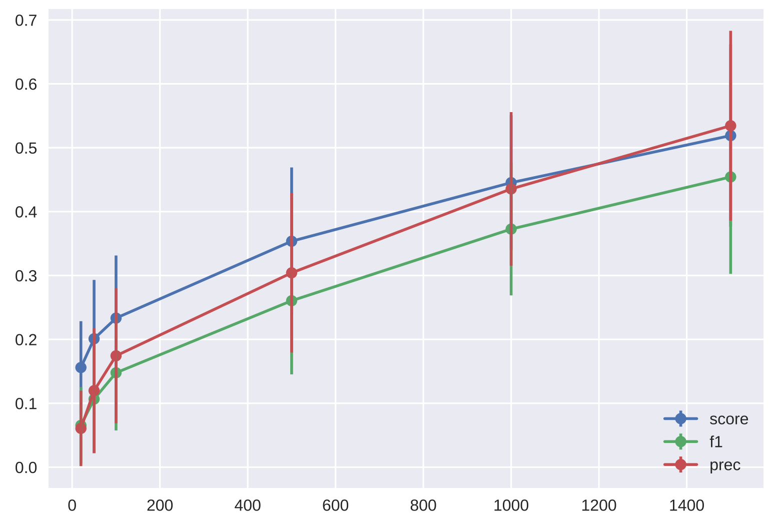


Figure 1: Shows the importance of the size of the training set and how the test error decreases with increasing size of training set. The error bars show the std over 30 different rounds.

The method was also implemented in `R`, through the function `sgd`. The function can be found in the appendix. We included an argument, `metodo`, which indicates the method to be employed, as follows. If it takes the value `nonsto` (non-stochastic), it implements the subgradient algorithm for minimizing the empirical risk

$$R^{\text{emp}}(\Theta) = \frac{1}{n}\sum_{k=1}^{n} z_k \quad \text{with } z_k = L\left(\Theta,(x_k,y_k)\right) = \max_{j\neq y_k}\left[1+\sum_{i=1}^{d} x_i(\Theta_{ij}-\Theta_{iy})\right]_+ . \tag{10}$$

If it takes the value `sto` (stochastic), it implements a stochastic subgradient algorithm by randomly selecting one z_k as defined in (10) and using it as an estimate for R^{emp}. If it takes the value `strata`, it implements a variation of the stochastic subgradient algorithm by stratifying the z-values with respect to the class of y and selecting a sample of size one in each stratum. The empirical risk is then unbiasedly estimated by

$$\hat{R}^{\text{emp}}(\Theta) = \frac{1}{n}\sum_{h=1}^{H} n_h z_h$$

where n_h and z_h are, respectively, the size and the value of the selected element in the hth stratum.
The function also includes an argument, `radius0`, that indicates the radius of the ball around the starting value Θ_0 where we want to constraint our search for Θ. If `radius0=NULL`, the "non-projected" algorithm is implemented.
This function was used as follows. We take $\Theta_0 = 0$ as starting value and run the subgradient algorithm (non-projected–non-stochastic). The last value is used as our new starting value. (This can be seen as a burn-in period.) Then we run the six variations of the subgradient algorithm for training sets of sizes $n_{\text{train}} = 20, 50, 100, 500, 1000$ and 1500 and a disjoint testing set of size $n_{\text{test}} = 100$. The results are shown in Figure 2. It can be seen that there is a big improvement with respect to the results shown in Figure 1. We also observe the following:

- Even for moderate size of the training set (50) the proportion of correctly classified elements is larger than 80% with every method. For training sizes of 500 or larger, it grows above 90%.
- The variations using the complete empirical function perform better than the stochastic ones. However, these differences are rather small, in particular for the stratified version. This is even

more relevant when we take into account that it takes several more times to run the algorithm using all the observations.

- The stochastic algorithm that uses stratified sampling performs virtually as well as the one using the complete empirical function, while, at the same time, running almost as fast as the stochastic version that uses only one element.

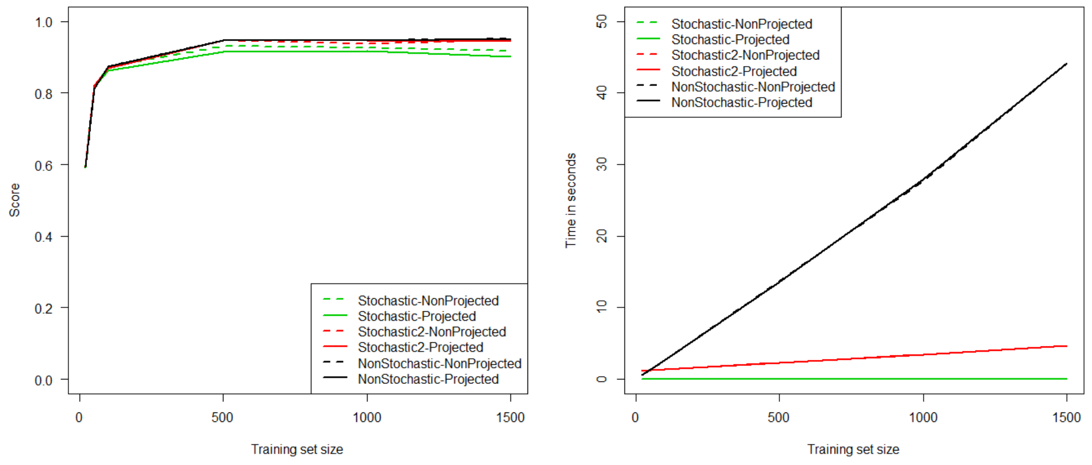


Figure 2: Performance of six subgradient methdos after ten iterations. Left panel: Training set size vs. Average score. Right panel: Training set size vs. Average running time.

1.5 Question e

Now we evaluate what the impact of the initial step-size is. In Figure 3 we can see a small peak around $10^{1.8}$.

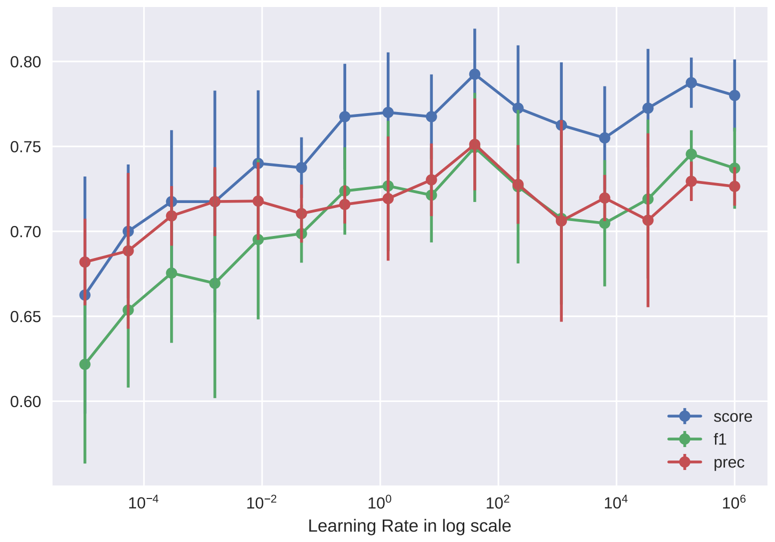


Figure 3: Different initial learning rates. For each learning rate we tested the performance five times. The error bars show the standard deviation over these trials.

2 Gaussian Process Classification

In this section we take a look at image classification using Gaussian Processes. We will look at Gaussian Process Classification and Regression.

2.1 Question a

We want to make an investigation about the classification performance of a Gaussian Process Classifier with various choices of covariance function and training dataset size. The algorithms were tested on the same test sets in order to obtain consistent prediction testing. L represents the length-scale which determines the smoothness of the covariance function, i.e. the degree to which data points influence one another. The shorter the length scale, the more one data point depends on another. The covariance functions used were the Radial basis function (isotropic kernel), Dot Product kernel and the Materné function.

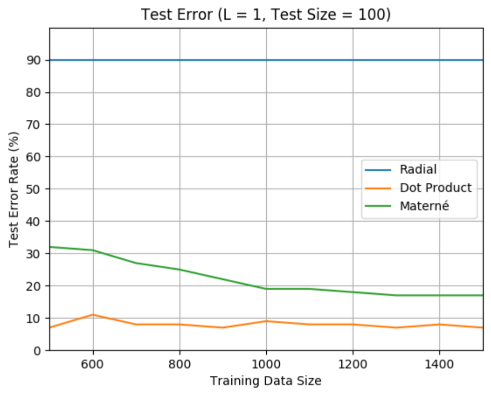


Figure 4: GPC Test Error Rates with $L = 1$ and test size 100.

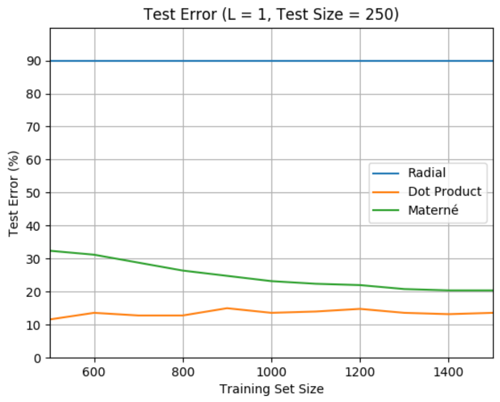


Figure 5: GPC Test Error Rates with $L = 1$ and test size 250.

Figure 4 and 5 show the test error relative to training data size. The success rate was tested for test sets of 100 and 250 images. It can be seen that there is a large variation in test error rates for the chosen kernel functions. We can see that the use of the radial basis function as the choice of covariance function performs the worst and it does not seem to improve with larger Training Sets. The linear kernel, or dot-product kernel, shows promising results under 10% even at smaller Training Sets. Additionally, the Materné function provides a steady decrease in the test error rate with increasing training data. The script to this problem can be found in the appendix.

As far as increasing length scales goes, the test error rates improves slightly for with a minimum test error rate of 4.8% for $L = 2$ instead of 6% for $L = 1$ in the radial basis function. Decreasing the length scale in the radial basis function to $L = 0.5$ significantly increased the test error rate to almost 50%. The classifier using the dot product (linear) did not show any improvement in the test error rates. The Materné covariance functions showed dramatic increase in accuracy for length scales $L \geq 2$ and for $L < 1$ the accuracy becomes as bad as with the radial basis function.

2.2 Question b

In this section, we use Gaussian Process Regression as a replacement for Gaussian Process Classification. The classification can be treated as a multivariate regression problem in which each class is a variable. The classification is done by predicting the class with the largest predicted value. One can do this by using the `np.argmax()` command in Python for each output vector and the position of that value as the class according to the specified one-hot vector. **The Python code for this section can be found in the Appendix**.

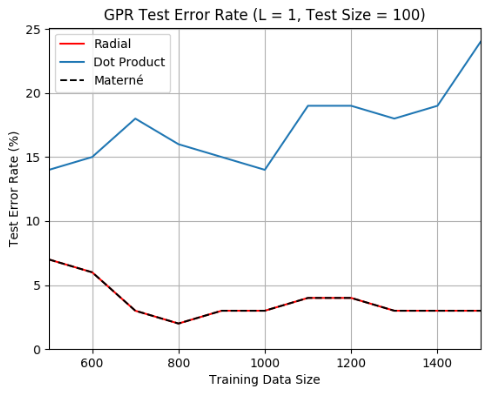


Figure 6: GP Regression Test error rates with $L = 1$ and test size 100.

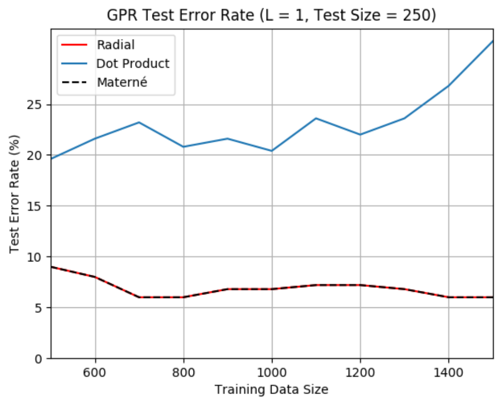


Figure 7: GP Regression Test error rates with $L = 1$ and test size 250.

Figures 6 and 7 show the test error rates of classification using Gaussian Process regression and using various covariance functions. The Radial Basis Function and Materné functions produced the same, gradually improving accuracy for both test sets of 100 and 250 images and for the most part returned error rates under 10%. Surprisingly, the dot product (linear) kernel function produced worse error rates as the training set size increased.

Looking at different length scales, the Materné function and Radial Basis function provided greater classification accuracy for length scales $L \geq 1$ as in the figures shown before, produce the same accuracy. The same occurred when switching to the dot-product (linear) kernel except the positive relationship between test error rates and training set size persisted.

3 Convolutional Neural Network Classification

In this part, we will classify our data using CNN (Convolutional neural network). This model is often used in real application for sight's recognition, because it is considered one of the best for those tasks.

3.1 Question a

First, we built the simplest CNN as possible. Then, we trained and tested it with our dataset.
We started by the basic routine : loading the dataset and creating the training set / testing set. Next, we created our model using Keras.

We initiazed our model and named it "CNN". Then, we added two layers and an operation :

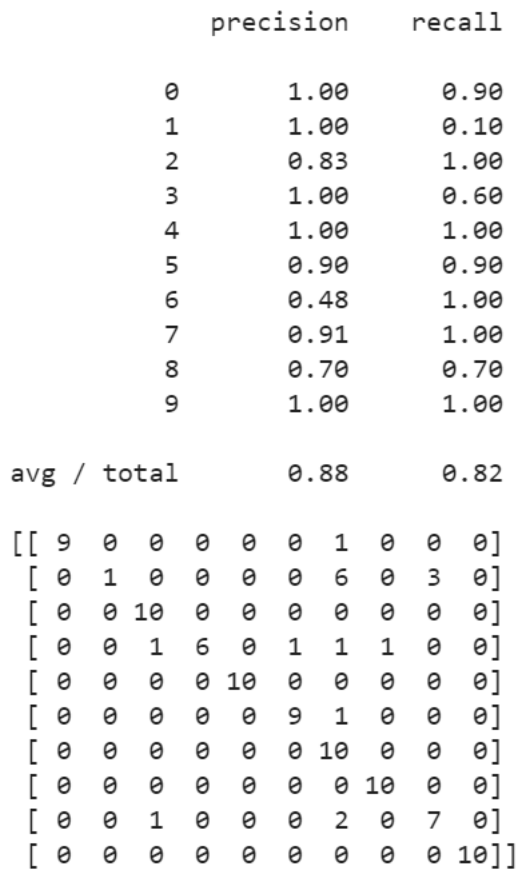
- The first one is "Conv2D", a 2D Convolutional layer. This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

- The second one is "Flatten", an operation which flattens the input. This operation is necessary for shape's purposes.

- The last one is the output layer. It is a "Dense" one with size 10 (number of labels) and "softmax" as the activation function (the common one for multi-class problems).

Once we have created our Network, we have to compile it and train it on our training set. This model only take a few seconds to train on the whole set. We evaluated it on the testing set and got an 94% accuracy. The **Python code can be found in the appendix.**

We can push our evaluation further, for instance by plotting a classification report which shows the precision, recall and f1_score for each class. We can also plot the Confusion matrix. Using those new tools, we can see that most of our mistakes are due to the digit "1". We will try to fix this in the next section.



```
             precision    recall  f1-score   support

          0       1.00      0.90      0.95        10
          1       1.00      0.10      0.18        10
          2       0.83      1.00      0.91        10
          3       1.00      0.60      0.75        10
          4       1.00      1.00      1.00        10
          5       0.90      0.90      0.90        10
          6       0.48      1.00      0.65        10
          7       0.91      1.00      0.95        10
          8       0.70      0.70      0.70        10
          9       1.00      1.00      1.00        10

avg / total       0.88      0.82      0.80       100

[[ 9  0  0  0  0  0  1  0  0  0]
 [ 0  1  0  0  0  0  6  0  3  0]
 [ 0  0 10  0  0  0  0  0  0  0]
 [ 0  0  1  6  0  1  1  1  0  0]
 [ 0  0  0  0 10  0  0  0  0  0]
 [ 0  0  0  0  0  9  1  0  0  0]
 [ 0  0  0  0  0  0 10  0  0  0]
 [ 0  0  0  0  0  0  0 10  0  0]
 [ 0  0  1  0  0  0  2  0  7  0]
 [ 0  0  0  0  0  0  0  0  0 10]]
```

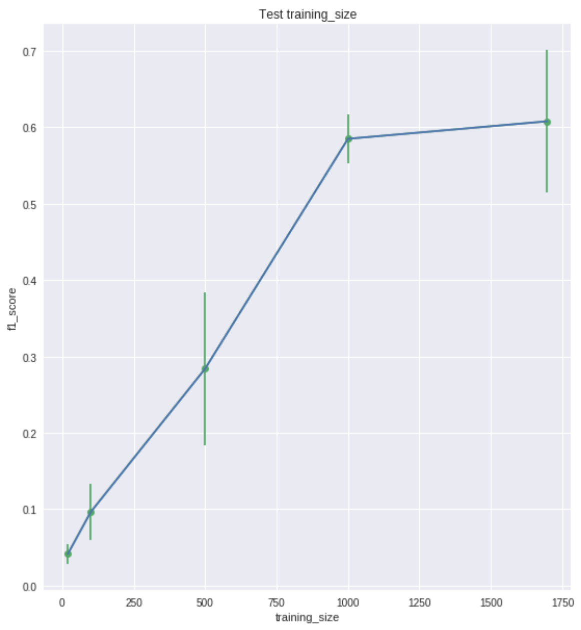
Figure 8: Classification report and Confusion matrix (1)

3.2 Question b

Now, we will try to improve our model. Indeed, even if it is already the best one, we can still probably improve its accuracy.

3.2.1 Influence of the size of the training data

Here we will discuss the influence training data. We will especially investigate on the size of the dataset and on the number of epochs. In order to have more flexibility for creating and training our models, we defined a function which is able to train a CNN for given arguments :
Once this function has been written, we used it to plot box-plots for some values of training size and epochs. The means and standard deviation are based on 5 trials. This is not that much but more trials would make the execution too long. **The Python code for plotting the figures can be found in the appendix**. We can obtain figures such as figure 9:



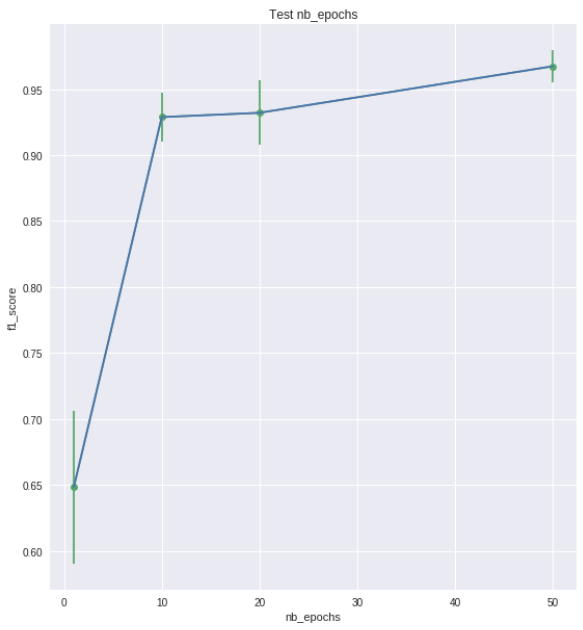


Figure 9: Influence of the training size and the number of epochs

Those results show that we should always seek for more data, and be sure to give our model enough time to train its parameters. The last script was quite long to run, because CNNs are slower models to train than the previous ones. However, when we provide enough data and/or epochs, the results can be outstanding (up to 98% accuracy, by modifying only one parameter !).

3.2.2 Influence of the characteristics oh the layer

Here we will try more advanced design for our CNN. We will first list the parameters that can be modified, and then study the influence of a few of them.

Indeed, they are many possibilities for modifying our network. The main ones are :

- Training on "batches" of data. This is a common method in Machine Learning, because it can make the training a lot faster. The "batch_size" is a parameter we decided to investigate on.
- Dividing the training set in two parts: a training one and a validation one. This is also a common method in Machine Learning, because it can prevent the over-fitting by doing cross-validation. The validation split ratio ("validation_split") is a parameter we decided to investigate on.
- Modifying the parameters of the convolutional layer. There are a few parameters we can change: the dimensionality of the output space (basic one: 32), the kernel size (basic one: (2,2)), the activation function (basic one: Relu), etc. However, we decided not to investigate on them.
- Adding a pooling layer. This a way to reduce dimensionality which is often used in CNN. However, a few tests showed that it was not improving the model.
- Adding a Dropout layer. This is a way to prevent the over-fitting. However, a few tests showed that it was not improving the model.
- Adding other layers, such as "Dense" ones. This is an easy way to add more parameters to our model. However, the classification problem is here quite "simple", and we tought that there was no need for a too complex neural network.

Thus, we decided to investigate on two parameters: the ”batch_size” and the ”validation_split” ratio. The Python script used is very similar to the previous one and can be found in the appendix.

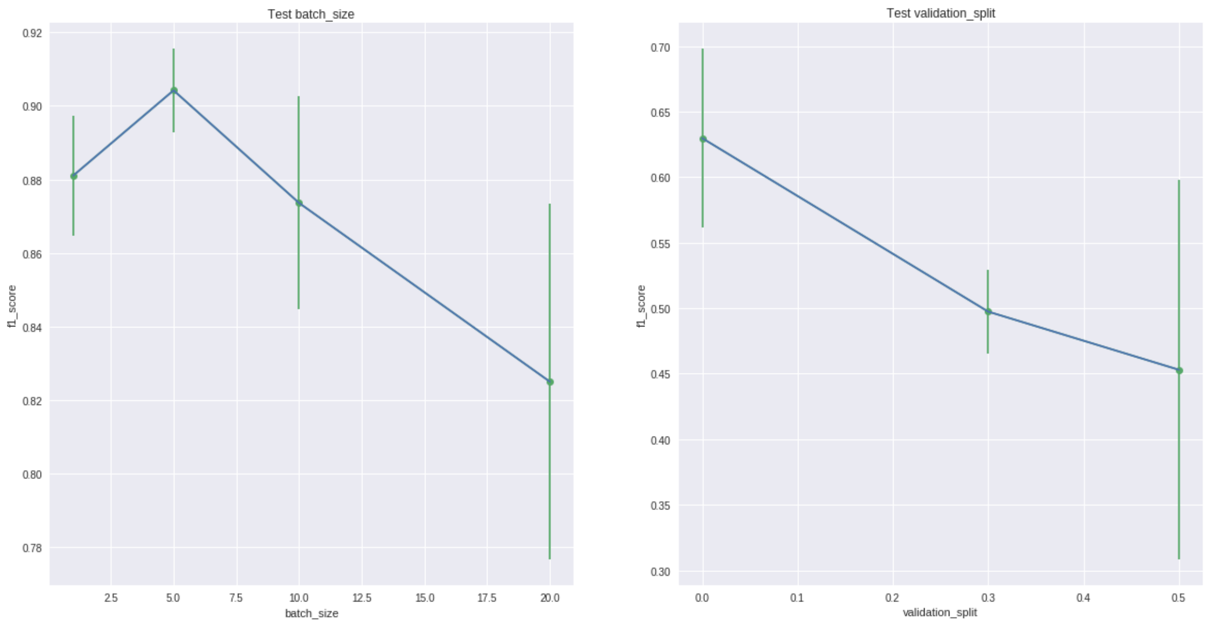
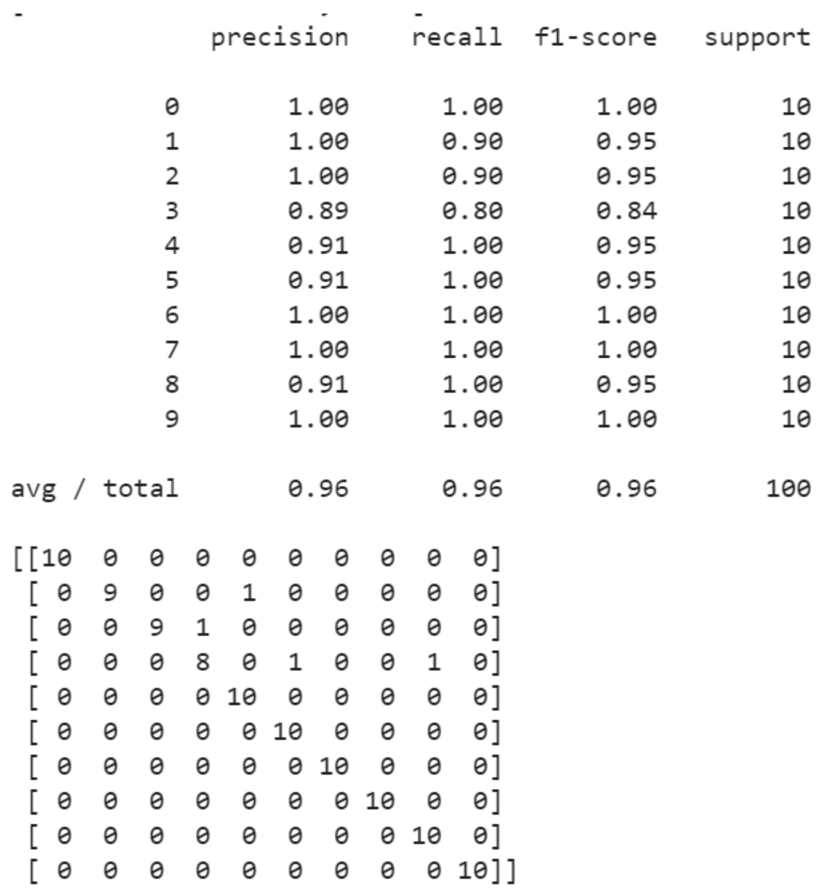


Figure 10: Influence of the batch size and the validation split ratio

We can draw several conclusion from those figures. For the batch size, we can see that it may significantly reduce the accuracy of the model if it is too high. Yet, if carefully chosen (here, the best one is 5), it may increase the network's precision, besides making the training faster. Indeed, the time required for the training is roughly $\frac{nb\_epochs \times trainingsize}{batch\_size} \leq training\_time(ms) \leq 10\frac{nb\_epochs \times trainingsize}{batch\_size}$.

The second plot show that it is not interesting to use cross validation in our case. This method is always a trade-off: on the one hand, it may significantly reduce over-fitting, but on the other hand, it reduces the size of our training set. And, in this example, it does not seem worth it.

Thanks to those investigation, we can then try to build the best CNN possible. According to the previous results, we should use the whole training set, as much epochs as possible (50 in our case), a batch size of 5 and no cross validation. With those parameters, we can reach the following results :



```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        10
          1       1.00      0.90      0.95        10
          2       1.00      0.90      0.95        10
          3       0.89      0.80      0.84        10
          4       0.91      1.00      0.95        10
          5       0.91      1.00      0.95        10
          6       1.00      1.00      1.00        10
          7       1.00      1.00      1.00        10
          8       0.91      1.00      0.95        10
          9       1.00      1.00      1.00        10

avg / total       0.96      0.96      0.96       100

[[10  0  0  0  0  0  0  0  0  0]
 [ 0  9  0  0  1  0  0  0  0  0]
 [ 0  0  9  1  0  0  0  0  0  0]
 [ 0  0  0  8  0  1  0  0  1  0]
 [ 0  0  0  0 10  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 10  0  0  0]
 [ 0  0  0  0  0  0  0 10  0  0]
 [ 0  0  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0 10]]
```

Figure 11: Classification report and Confusion matrix (2)

The results can still be improved. For instance, the CNN created with "`CNN_best = CNN(x_train, y_train, nb_epochs = 50, batch_size = 10, pooling = True, hidden_layer = [500])`" usually have 99% accuracy on the test set. However, this model is more complex, and probably have too many parameters. Even if it is not the case here, such models are slower to train and can easily be affected by issues such as over-fitting.

4 Comparison and discussion

Several methods were implemented throughout this report for classifying a set of images according to the digit they represent. Figure 12 shows examples of those images. In the first section, we implemented subgradient algorithms. In the second one, we used gaussian processes. In the third section, we used Convolutional Neural Networks. In brief, it can be said that all the methods (when properly specified) were able to correctly classify a large proportion of the elements in the test set. This proportion was already larger than 50% for even moderate sizes of training sets, and it was around 95% for training sets of 500 or larger.

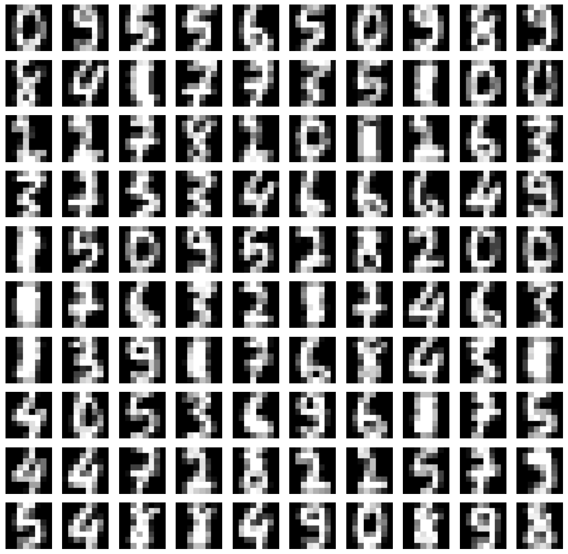


Figure 12: The test set with 100 8×8 pixel handwritten digits.

Regarding the subgradient methods, the stochastic subgradient method using stratified sampling, yielded the best results. It was efficient both in terms of processing time and precision of classification.
The Gaussian Process Classification method showed low test error rates and hence high accuracy. The best choice of covariance function showed to be the dot-product (linear) kernel. The Materné kernel provided a gradually decreasing trend in the test error rate, and perhaps it would decrease with even larger training sets. The minimum test error rates found were 7% and 11.6% for 100 and 250 test images respectively (length scale $L = 1$). The computation time required for both of these kernel functions is also quite low.
In Gaussian Process Regression, the Radial Basis Functions and Materné functions produced the exact same results with the minimal test error rate being 2% and 6% for 100 and 250 test images respectively (length scale $L = 1$). The dot-product (linear) kernel provided increasingly worse results with larger training set sizes. The computation times were also very low for this method. In comparison, the Gaussian Process Regression outperforms the Gaussian Process Classifier in all aspects.
Convolutional Neural Networks are famous to be able to tackle difficult classification problems, especially in image classification. Thus we were expecting them to be able to deal with our problem quite easily.
If we look at our results, we can see that even with a few parameters, we managed to get great accuracy (¿ 90%). However, they also required a significant training time (more than a minute), and we had to add several parameters in order to reach the best results as possible (99% with CNN).
We think that CNNs are at there best in those kind of problems, but this one is probably somewhat ”too easy” for them. Indeed, we think that, for problems with such a low resolutions and so few classes, it is better to use SVM or Gaussian Processes since they give similar results with shorter training time and/or less data.

References

[1] NumPy python library *http://www.numpy.org/*

[2] Scikit-Learn python library *https://www.scikit-learn.org/*

[3] TensorFlow python library *https://www.TensorFlow.org/*

[4] Keras python library *https://keras.io/*

A Appendix

A.1 Stochastic Subgradients for Training SVMs

A.1.1 Part (c). Python code.

Functions for computing subgradient

```
def loss_function(Theta,x,y):
    '''theta Matrix R^(d x K), x vector R^d, y integer returns the loss_value and the index jm
        = argmax'''
    x= np.reshape(x,[-1,1])
    y = int(y)
    d,K=Theta.shape
    jm=-1
    loss_value=0
    loss_vector = np.ones([K,1])-x.transpose().dot(Theta -
        np.reshape(Theta[:,y],[-1,1])).transpose()
    loss_vector = np.maximum(loss_vector,np.zeros([K,1]))
    loss_vector[y]=-1 #s.t. the argmax doesn't take the y-th entry.
    jm = np.argmax(loss_vector)
    loss_value = loss_vector[jm][0]
    return loss_value, jm

def stochastic_subgradient(Theta,x,y):
'''computes a stochastic subgradient of the loss function for given input Theta, x, y'''
    x= np.reshape(x,[-1,1])
    y = int(y)
    d,K=Theta.shape
    g=np.zeros([d,K])
    loss_value, jm = loss_function(Theta,x,y)
    if loss_value == 0:
        return g
    else:
        g[:,jm]=x.transpose()
        g[:,y]=-x.transpose()
    return g

def svmsubgradient(Theta, x, y):
'''computes the a subgradient of the loss function for data Theta, x, y where x and y are a
    collection of n data points'''
    #  Returns a subgradient of the objective empirical hinge loss
    G = np.zeros(Theta.shape)
    for x_i,y_i in zip(x,y):
        G=G+stochastic_subgradient(Theta,x_i,y_i)
    G=G/x.shape[0]
    return(G)

def sgd(Xtrain, ytrain, maxiter = 10, init_stepsize = 1.0, l2_radius = 10000):
'''train a SVM as in eq (1) described'''
    K = 10
    NN, dd = Xtrain.shape
    Theta = np.zeros(dd*K)
    Theta.shape = dd,K
    mean_Theta = np.zeros(dd*K)
    mean_Theta.shape = dd,K
    for i in range(maxiter):
      random_index = np.random.randint(0,NN)
      alpha = init_stepsize/((i+1)**0.5)
      Theta  = Theta - alpha *
          stochastic_subgradient(Theta,Xtrain[random_index,:],ytrain[random_index])
      radius = np.linalg.norm(Theta.flatten())
      if radius > l2_radius:
        Theta = Theta/radius*l2_radius
      mean_Theta = mean_Theta+Theta
    mean_Theta/i
    return Theta, mean_Theta
```

A.1.2 Part (c). R code.

Functions for computing subgradient

```
#This function computes L(y,a) (Pg 46 of the lecture notes)
L<- function(y,a) {a<- 1+a-a[y]; return(max(c(a[-y],0)))}

#This function computes L(Theta,X,Y,k)=L(X2,Y,k) where X2=X%*%Theta and k is the k-th element
    in X (see Pg 1 of the Project I)
L2<- function(i,Y,X2) {return(L(Y[i],X2[i,]))}

#This function computes Remp(Theta,X,Y) (Pg 1 of the Project I)
Remp<- function(Theta,X,Y) {
   X2<- X%*%Theta
   N<- nrow(X)
   salida<- apply(as.matrix(1:N),1,L2,Y=Y,X2=X2)
   return(mean(salida))
}

#This function computes a subgradient of Theta given x and y
svmsubgradient0<- function(Theta,y,x) {
   Theta[,y]<- Theta[,y]-1
   g<- matrix(0,nrow(Theta),ncol(Theta))
   X2<- t(as.matrix(x))%*%Theta
   indica<- (X2==max(X2))*(1:l)
   indica<- indica[indica>0]
   if (y%in%indica==FALSE) {g[,indica]<- x; g[,y]<- -x}
   return(g)
}

#This function computes the subgradient of Theta for the ith element of X and Y
svmsubgradient1<- function(i,Theta,Y,Z) {svmsubgradient0(Theta,Y[i],Z[i,])}


#This function computes a subgradient of Theta given x and y
svmsubgradient<- function(Theta,Y,Z) {
   N<- length(Y)
   g<- sapply(1:N,svmsubgradient1,simplify="array",Theta=Theta,Y=Y,Z=Z)
   g<- rowMeans(g,dims=2)
   return(g)
}


#This function implements a projected subgradient method for finding Theta that minimizes the
    Loss
sgd<- function(Theta,X,Y,radius0=NULL,maxiter,stepsize,metodo="nonsto") {
   N<- length(Y)
   Theta0<- matrix(0,nrow(Theta),ncol(Theta))
```

```
  Thetaold<- Theta
  for (i in 1:maxiter) {
    if (metodo=="sto") {
      k<- sample(N,1)
      yest<- svmsubgradient1(k,Theta,Y,X)
    }
    if (metodo=="strata") {
      id<- 1:N
      muestra<- numeric(0)
      clases<- as.vector(by(Y,Y,mean))
      yest<- array(0,dim=c(nrow(Theta),ncol(Theta),length(clases)))
      Nh<- as.vector(by(Y,Y,length))
      for (j in 1:length(table(Y))) {
        k<- sample(id[Y==clases[j]],1)
        yest[,,j]<- Nh[j]*svmsubgradient1(k,Theta,Y,X)
      }
      yest<- rowSums(yest,dims=2)/N
    }
    if (metodo=="nonsto") {yest<- svmsubgradient(Theta,Y,X)}
    Theta<- Theta-stepsize*yest/sqrt(i)
    if (is.null(radius0)==FALSE) {
      radius<- sqrt(sum((Theta-Thetaold)^2))
      if (radius>radius0) {
        Theta<- radius0*Theta/radius
        Theta2<- Theta-Thetaold
        Theta<- (radius0*Theta2/radius)+Thetaold
      }
    }
    Theta0<- Theta0+Theta
  }
  salida<- list(Theta=Theta,mean_Theta=Theta0/maxiter)
  return(salida)
}
```

A.2 Gaussian Processes

A.2.1 Part (a)

Script for building a Gaussian Process Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct
from sklearn.gaussian_process.kernels import Matern

# import some data to play with
digits = datasets.load_digits()

X = digits.data
y = np.array(digits.target, dtype = int)

N,d = X.shape

N = np.int(1797)
```

```
Ntrain = np.int(100)
Ntest = np.int(100)


Xtrain = X[0:Ntrain-1,:]
ytrain = y[0:Ntrain-1]
Xtest = X[N-100:N,:]
ytest = y[N-100:N]


#kernel = 1.0 * RBF([3.0]) #isotropic kernel
kernel = DotProduct(1.0) #dotproduct kernel
#kernel = Matern(1.0) #matern kernel

gpc_rbf = GaussianProcessClassifier(kernel=kernel).fit(Xtrain, ytrain)
yp_train = gpc_rbf.predict(Xtrain)
train_error_rate = np.mean(np.not_equal(yp_train,ytrain))
yp_test = gpc_rbf.predict(Xtest)
test_error_rate = np.mean(np.not_equal(yp_test,ytest))
print('Training error rate')
print(train_error_rate)
print('Test error rate')
print(test_error_rate)
```

A.2.2 Part (b)

Script for building a Gaussian Process Regression Model

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct
from sklearn.gaussian_process.kernels import Matern

# import some data to play with
digits = datasets.load_digits()

X = digits.data
y = np.array(digits.target, dtype = int)

N,d = X.shape

N = np.int(1797)
Ntrain = np.int(100)
Ntest = np.int(100)

Xtrain = X[0:Ntrain-1,:]
ytrain = y[0:Ntrain-1]
Xtest = X[N-100:N,:]
ytest = y[N-100:N]

#create one-hot vector for the targets. i.e. the labels we want to correctly classify
yhotvec = [ ]
for n in range(0, len(y)):
    zeros = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    yvalue = y[n]
    zeros[yvalue] += 1
    yhotvec.append(zeros)

yhotvectrain = yhotvec[0:Ntrain-1]
yhotvectest = yhotvec[N-100:N]

kernel = 1.0 * RBF(1.0) #isotropic
#kernel = DotProduct(1.0)
#kernel = Matern(1.0)
gpc_rbr = GaussianProcessRegressor(kernel = kernel, normalize_y = False).fit(Xtrain,
    yhotvectrain)
yp_train = gpc_rbr.predict(Xtrain)
train_error_rate = np.mean(np.not_equal(yp_train, yhotvectrain))
yp_test = gpc_rbr.predict(Xtest)
yp_test = np.argmax(yp_test, axis = 1)
yhotvectest = np.argmax(yhotvectest, axis = 1)
test_error_rate = np.mean(np.not_equal(yp_test, yhotvectest))
print('Training error rate')
print(train_error_rate)
print('Test error rate')
print(test_error_rate)
```

A.3 Convolutional Neural Network Classification

A.3.1 Part (a)

Script for a simple CNN

```
import numpy as np
np.random.seed(100)
import keras as kr
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from sklearn import datasets
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics as met
import matplotlib.pyplot as plt

# Load the dataset and separate it into train set and test set

digits = datasets.load_digits()
X = digits.data
y = np.array(digits.target, dtype = int)
N,d = X.shape
Ntest = np.int(100)
Ntrain = np.int(1697)
Xtrain = X[0:Ntrain,:]
ytrain = y[0:Ntrain]
Xtest = X[Ntrain:N,:]
ytest = y[Ntrain:N]

input_shape=(8,8,1)

y_train = kr.utils.to_categorical(ytrain,10)
y_test = kr.utils.to_categorical(ytest,10)
```

```
x_train = Xtrain.reshape(Xtrain.shape[0],*input_shape)
x_test = Xtest.reshape(Xtest.shape[0],*input_shape)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

print(y_train.shape)
print(y_test.shape)

#Creation of the CNN

CNN = Sequential()
CNN.name = "CNN"
CNN.add(Conv2D(32,kernel_size=(2,2), activation="relu", input_shape = input_shape))
CNN.add(Flatten())
CNN.add(Dense(10,activation="softmax"))
print(CNN.output_shape)

CNN.compile(loss="categorical_crossentropy",
    optimizer=kr.optimizers.Adadelta(),metrics=["accuracy"])

CNN.fit(x_train,y_train)

score = CNN.evaluate(x_test,y_test,verbose=1)
print(score)

test_predictions = CNN.predict(x_test)

Y_test = np.argmax(y_test, axis =1)
test_predictions = np.argmax(test_predictions, axis =1)

print(classification_report(Y_test, test_predictions, target_names=[str(i) for i in range(10)]))
print(confusion_matrix(Y_test, test_predictions, labels=range(10)))
```

A.3.2 Part (b)

Function for the creation of a CNN

```
def CNN(x_train, y_train,nb_epochs = 1, batch_size = None, validation_split = 0, pooling =
    True, dropout = False, hidden_layer = None) :
  CNN = Sequential()
  CNN.name = "CNN"
  CNN.add(Conv2D(32,kernel_size=(2,2), activation="relu", input_shape = input_shape))
  if pooling :
    CNN.add(MaxPooling2D(pool_size=(2, 2)))
  if dropout :
    CNN.add(Dropout(0.3))
  CNN.add(Flatten())
  if hidden_layer != None :
    for i in hidden_layer :
      CNN.add(Dense(i,activation="relu"))
  CNN.add(Dense(10,activation="softmax"))
  print(CNN.output_shape)

  CNN.compile(loss="categorical_crossentropy",
      optimizer=kr.optimizers.Adadelta(),metrics=["accuracy"])
  CNN.fit(x_train,y_train, batch_size=batch_size, epochs=nb_epochs, verbose=1,
      validation_split=validation_split)

  score = CNN.evaluate(x_test,y_test,verbose=1)
  print(score)
  return CNN
```

Script for plotting the influence of the training size and the number of epochs

```
#Parameters
Ntrain = 1697
training_size=np.array([20,100,500,1000,Ntrain])
nb_epochs = np.array([1,10,20,50])

for i,size in enumerate(training_size):
  for rou in range(rounds):
    idx = np.random.randint(Ntrain, size=size)
    train_data_x = x_train[idx,:]
    train_data_y = y_train[idx]
    CNN_temp = CNN(train_data_x,train_data_y)

    test_predictions = CNN_temp.predict(x_test)
    Y_test = np.argmax(y_test, axis =1)
    pred = np.argmax(test_predictions, axis =1)
    f1_1[i,rou]=met.f1_score(Y_test,pred,average='weighted')

for i,epc in enumerate(nb_epochs):
  for rou in range(int(rounds)):
    idx = np.random.randint(Ntrain, size=Ntrain)
    train_data_x = x_train[idx,:]
    train_data_y = y_train[idx]
    CNN_temp = CNN(train_data_x,train_data_y,nb_epochs = epc)
    test_predictions = CNN_temp.predict(x_test)
    Y_test = np.argmax(y_test, axis =1)
    pred = np.argmax(test_predictions, axis =1)
    f1_2[i,rou]=met.f1_score(Y_test,pred,average='weighted')

mean_1 = np.mean(f1_1,axis=1)
variance_1 = np.std(f1_1,axis=1)
mean_2 = np.mean(f1_2,axis=1)
variance_2 = np.std(f1_2,axis=1)
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,10))
ax[0].plot(training_size,mean_1)
ax[0].errorbar(training_size,mean_1,variance_1,fmt="-o",barsabove=3)
ax[1].plot(nb_epochs,mean_2)
ax[1].errorbar(nb_epochs,mean_2,variance_2,fmt="-o",barsabove=3)
ax[0].set_title("Test training_size")
ax[1].set_title("Test nb_epochs")
ax[0].set_xlabel('training_size')
ax[1].set_xlabel('nb_epochs')
ax[0].set_ylabel('f1_score')
ax[1].set_ylabel('f1_score')
```

Script for plotting the influence of the batch size and of the validation split ratio

```
#3.2.2
batch_size = np.array([1,5,10,20])
```

```
validation_split = np.array([0,0.3,0.5])

rounds=5
test_size=100
f1_3 = np.zeros([batch_size.shape[0],rounds])
f1_4 = np.zeros([validation_split.shape[0],rounds])



for i,btc in enumerate(batch_size):
  for rou in range(rounds):
    idx = np.random.randint(Ntrain, size=Ntrain)
    train_data_x = x_train[idx,:]
    train_data_y = y_train[idx]
    CNN_temp = CNN(train_data_x,train_data_y,batch_size= btc)

    test_predictions = CNN_temp.predict(x_test)
    Y_test = np.argmax(y_test, axis =1)
    pred = np.argmax(test_predictions, axis =1)
    f1_3[i,rou]=met.f1_score(Y_test,pred,average='weighted')

for i,val in enumerate(validation_split):
  for rou in range(int(rounds)):
    idx = np.random.randint(Ntrain, size=Ntrain)
    train_data_x = x_train[idx,:]
    train_data_y = y_train[idx]
    CNN_temp = CNN(train_data_x,train_data_y,validation_split = val)
    test_predictions = CNN_temp.predict(x_test)
    Y_test = np.argmax(y_test, axis =1)
    pred = np.argmax(test_predictions, axis =1)
    f1_4[i,rou]=met.f1_score(Y_test,pred,average='weighted')

mean_3 = np.mean(f1_3,axis=1)
variance_3 = np.std(f1_3,axis=1)
mean_4 = np.mean(f1_4,axis=1)
variance_4 = np.std(f1_4,axis=1)
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,10))
ax[0].plot(batch_size,mean_3)
ax[0].errorbar(batch_size,mean_3,variance_3,fmt="-o",barsabove=3)
ax[1].plot(validation_split,mean_4)
ax[1].errorbar(validation_split,mean_4,variance_4,fmt="-o",barsabove=3)
ax[0].set_title("Test batch_size")
ax[1].set_title("Test validation_split")
ax[0].set_xlabel('batch_size')
ax[1].set_xlabel('validation_split')
ax[0].set_ylabel('f1_score')
ax[1].set_ylabel('f1_score')
```