Numerical project 1

Machine Learning Course

by Morten Hjorth-Jensen at GANIL

Classification and Regression, from linear and logistic regression to neural networks

Paul GUIBOURG, Iftikhar SAFI, Tristan LE CORNU
Master 2 NAC Physique - UFR Science CAEN

February 21, 2020

Abstract

This document presents the code and the analysis on the subject of the project 1. It is focus on the Ising model, which we were given the experimental data. The Ising model is a model of statistical physics. It has been used to model different phenomena in which collective effects are produced by local interactions between two-state particles. The main example is ferromagnetism for which the Ising model is a model on a network of magnetic moments, in which the particles are always oriented along the same spatial axis and can only take two values, + M and -M. Using a variety of methods, we seeks to determine the best way to find the phase change. We applied linear regression, classification algorithms and neural networks using Python librairies such as Scikit-Learn or TensorFlow. In application of these methods, it was shown that the more interresting methods are the Ridge regression and Stochastic Gradient Descent method . In fact, the time of calculation are less than the neural networks and they have the same accuracy and mean squared error.

We want to thanks Sacha Daumas and Tom Genard for the discussion about this project and Christopher Jacquiot student of the master "Décision et optimisation . DOP".

Introduction

The aims of this work are to be able to evaluate the best method to analys data according to Ising model. It is a mathematical model for physicists to determine if a sample are ferromatic (according to the spins of the particules observed).
It's the reason why, in the first part, we consider some data in one dimension. We developpe few regression methods. After that, we can developpe the data through two dimensions and we needed to write some algorythm of logistic regression and neural network.
We analysed the result with a critical evaluation of the pros and the cons. This document will be finish wenn we conclude about the best method to analys the data according to the theoritical Ising model and then we compare with the the examples of Mehta et al.

The Ising model in one-dimensional problem

A - Exhibition part.

We start to work on the Ising model at only one dimension. We don't expect a phase transition. The problem should be reduce as nearest-neighbor interaction. We search the energy on the Ising method. It a sum over the nearest-neighbor interaction on a chain of N variable like this :

$$E[\hat{s}] = -J\sum_{j=1}^{N} s_j, s_{j+1},$$

where J is the coupling constant coeficient we search, and s the coefficient we work on. Like it explain in the subject, the problem at one dimension of the Ising model can be express again as :

$$E^i_{model} \equiv \mathbf{X}^i \cdot \mathbf{J}.$$

where $\mathbf{X}^i$ is a vector represent two-body interraction. In this part, we are looking for the best linear regression to estimate the coupling constant J. The data were generated with $J = 1$, and we need to find this result.

The discussion will focus on the perform of the regression : we used tree differents methods to determine the coupling constante J. We used linear regression, Ridge regression and Lasso regression. The performance of the regression are evaluated by the mean-square-error (MSE), the bias and the variance. Their expressions are different following to the hyperparameter λ in the last two regression.
The goal, is to find the minimal mean-square-error according to the hyperparameter λ. We can watch the R^2 score, as in the document of *Mehta et al.*

An another method, to specify our best value of MSE, is to use a resampling method. That means we pick randomly into the the training data and we re-calculate the quantities of interrest. The MSE will be the mean MSE over the number samples pick. However, we can apply a Cross-Validation method, to confirm our result.

B - The code

B.1 - Our own code

Two differents codes could be developed : the first one, is completely original and use numpy method only. We calculate by our own code each step of tree regressions.
We define variables we will use in the main function. In other function, we calculate the design matrix X we use in the third function which calculate the invert-matrix of X. She returns the β coefficients.

```
def regression_OLS(self):
   XT    = self.X.T
   A_inv = np.linalg.inv(XT @ self.X)
   self.BETA = ((A_inv @ XT) @ self.Y)
   self.Y_predict = self.X @ self.BETA
```

Python 3.7

All of these, could be put into a class (as it present above), it is simpliest to call it. We specify the variable at the class instantiation as the degree of the polynomial featureor the hyperparameter λ.

The quantities of interrest are calculate directly when we call from properties of the class. The simpliest code are given in the lecture, so we implement it in the class directly :

```
def properties(self):
   self.error = np.mean(np.mean((self.Y-
               self.Y_predict)**2,axis=1))
   self.bias = np.mean(np.mean(self.Y -
              np.mean(self.Y_predict,
              axis=1))**2)
   self.variance = np.mean(np.var(
                  self.Y_predict,axis=1))
```

Python 3.7

The resampling method is implemented into the main function, after then we have plit the total sample. The code is presented below :

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled =
        scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Python 3.7

The main problem of this code is that it doesn't take into account exceptions or incorrect data : if the X matrix cannot be invert, this cause some troubles. We should implement a SVD method to response.

B.2 - The scikit-learn methods

A second solution to solve the regression problem, is to use methods from scikit-learn. The code gets easier : we call the class for the specific regression we want like :

```
model = Pipeline([("poly",
        PolynomialFeatures(degree=maxdegree)),
        ("linear", Ridge(alpha=lamb,
        fit_intercept=False))])
fitted_model= model.fit(X_train_scaled, Y_train)
```

Python 3.7

This is the code we use and we compare with the *Mehta et al.*

C - Analysis

We plot the values of the mean-square-error(MSE) in function of the hyper-parameter λ for the train and test data for the linear regression, Ridge regression and Lasso regression :

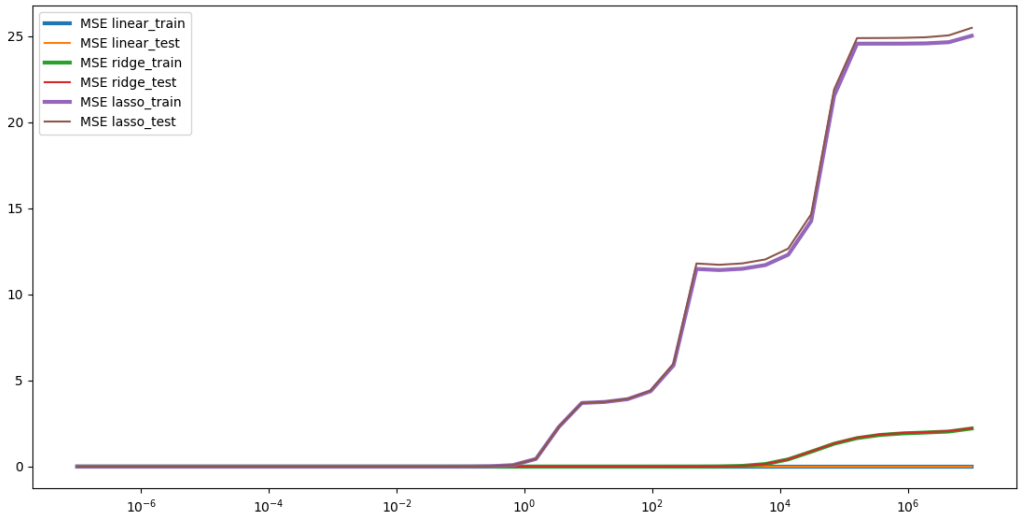

Figure 1: Graphics of the MSE in function of the hyper-parameter λ

We saw that the value of the mean-square-error is constant in the case of the ordinary-least-square : it's normal, it doesn't depend of λ. The second (Ridge) and the third (Lasso) depend of λ and it shows two differents things.

First, the optimal mean-square-error are not the same both cases.

The comparison with *Mehta et al* code (into its **jupyter notebook**) could be done with the R^2 score we plot below :

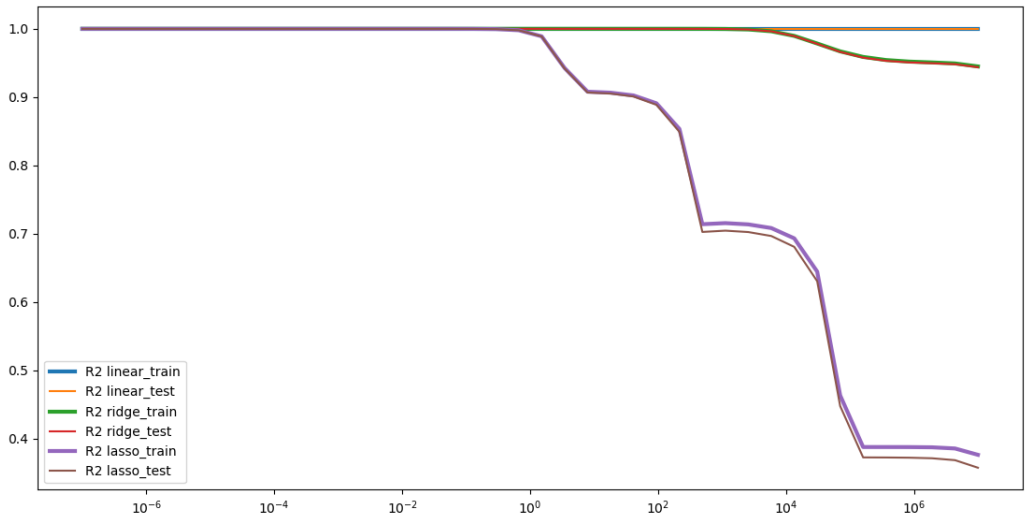

Figure 2: Graphics of the R^2 in function of the hyper-parameter λ

It show the best λ are equal to 10^{-7} for a Lasso regression and 10^{-7} for a Ridge regression. It's quite the same values then in the *Mehta et al* code. That is quite the same code. That's not enough to change strongly the result. We fixed the polynomial feature at one : J is constant, we fixed previously. We think just a linear regression is enough for this calculation, the MSE and R^2 are correct for this method but the Ridge regression give better results.
We use polynomial feature, it isn't suitable method for the given problem. The phase change seems to be a sigmoïd or tangent function. A logistic regression should be better to analys the data.

Determination of the phase of the two-dimensional Ising model

A - Exhibition part.

A.1. - Logistic regression

We need to implement a logistic regression because in fact, we have discrets variables. We know the theoritical critical phase transition at $T \simeq 2,269$ in units energy.
The phase transition follow a step function curve like :

$$f(x) = 1 \; for \; x < 0$$

$$f(x) = 0 \; for \; x \; \geq 0$$

We distinguish tree parts : ordered phase, spins in the lactice are homogeneous ; the second part are the critical part where the disorder comes to a peak in the third part. The figure are in the

Mehta et al **jupyter notebook**.

The aim of this method is to minimize a cost function as the same in linear regression. A form of the derivative cost function could be write like :

$$\frac{\partial C}{\partial \beta} = \mathbf{X}^{+}(\mathbf{y} - \hat{p}).$$

The 'X' matrix is the lactice data and the 'y' matrix energies expected.

A.2. - Other method to classify datas

An other method exists to classify datas. It is the SGD classifying method (Stochastic Gradient Descent). We always search to minimize the cost function, it set some random point and calculate the cost function. The gradient descends in the most favourable direction.
We evaluate the relevance of the logistic regression with the accuracy score. It's note the how the model correspond to the prediction according to binaries values (correct values return 1, something else 0) given by the datas. That can be express like :

$$accuracy = \frac{Number\ of\ correct\ labels}{Total\ labels}$$

The best regressor return 1 : all label are correctly predict.

B - The code

Now, we recover the data from the code given to us in the Github. Those code give the 'X' matrix which compound of lactices at each temperature but also the 'Y' matrix, energies linked with at each lactice. It's the result of previous formula in two dimensions.

Here, we give the code we need, to obtain the accuracy of the matrix manualy. We had describe the method to get this in the previous topic.

```
def meanConfig(x):
  bit_1 = 0
  bit_0 = 0
  #x = np.reshape(x,(1600,1))
  for j in range(x.shape[0]):
      #print("Taille de x : {}".
             format(x.shape))
      if x[j] == 1 :
          bit_1 += 1
      elif x[j] == 0 :
          bit_0 += 1
      else :
          print("ERROR value of the data
            (bits [{},{}])".format(k,h))
      bit_total = x.shape[0]
```

```
  return bit_0/bit_total*100,
         bit_1/bit_total*100, bit_total

meanBits = []
for i in range (16):
  bit0 = np.zeros((1,N))
  bit1 = np.zeros((1,N))
  for j in range(N):
    x = X[j+i*N]
    bit0[0][j],bit1[0][j],bitTot =
              meanConfig(x)
    if abs(bit0[0][j]-bit1[0][j])/100
            >= threshold:
       acurateTab[i][j] = 1
  meanBits.append(np.mean
           (abs((bit0-bit1))))

print("Les valeurs moyennes : \n {}"
      .format(meanBits))
```

Python 3.7

The difference between both matrix are made into those loops on the X matrix dimension. The results should be set in range zero and one as we expected.
Now, we can employ the logistic regression from Scikit-learn.

```
from sklearn import linear_model

clf = linear_model.LogisticRegression()
clf.fit(X,Y)
```

Python 3.7

To have the logistic regression, we can use the method "`LogisticRegressionCV`()" from scikit-learn. This method add a method of cross-validation wich we expect more precize accuracy. It split in k sample the data and the program run on each sample. The mean-square-error is taken to estimate the prediction error of the logistic regression. By default, the methods `score`() include into scikit-learn are setup on "accuracy" mode. The penalty are setup default parameter "l2" i.e. the penalty from Ridge regression.

C - Analysis

Two codes are made : the first use the simplest method of logistic regression from scikit-learn, we devellop after this. The second method, include directly a physical sens : we set the X matrix as vector matrix compound of the homogeneous score. It's mean we calculate the proportion on each lactices of spin value we divide by total number of label.
We obtain this result of the homogeneous of lactice at each temperature :

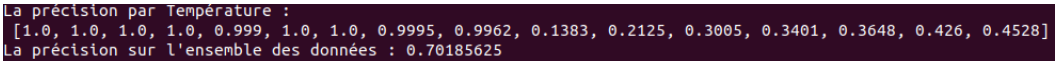

Figure 3: Accuracy calcul by our method

We see that we could separate the sample of data in tree part (ordered, critic and disordered) as in the **jupyter notebook** of *Mehta et al*. An another result we can show up are describe below. We calculate the Ising energies value at each lactice end we put it into the logistic regression. We do the linear regression on the homogeneity of spin system and it's interresting because we obtain better results :

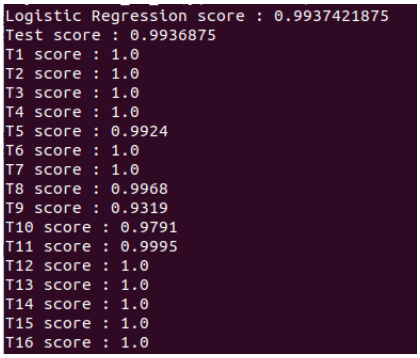

Figure 4: Accuracy calcul by logistic regression of Scikit-Learn

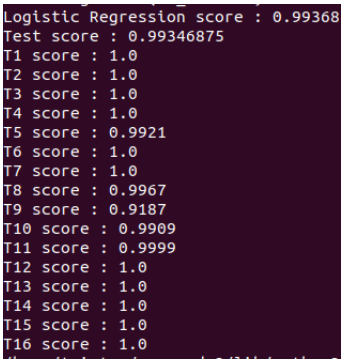

Figure 5: Accuracy calcul by cross-validation logistic regression of Scikit-Learn

Compare to *Mehta et al*, we got better logistic regression because we do the regression on the homogeneity of spin system and not on the spin. We use this method because the material has ferromagnetic effects if the system has a homogeneous spin system and is independent of the sign of the spins. It seems more natural to check whether our material has a homogeneous spin surface or not. Our results seem rather good and we can identify the ordered zone, the critical zone and the disordered zone thanks to the following diagram and logistic regression.

```
SDG score : 0.9938359375
Test score : 0.9936875
T1 score : 1.0
T2 score : 1.0
T3 score : 1.0
T4 score : 1.0
T5 score : 0.9923
T6 score : 1.0
T7 score : 1.0
T8 score : 0.9967
T9 score : 0.9259
T10 score : 0.9862
T11 score : 0.9998
T12 score : 1.0
T13 score : 1.0
T14 score : 1.0
T15 score : 1.0
T16 score : 1.0
```

Figure 6: Accuracy calcul by Stochastic Gradient Descent methode of Scikit-Learn

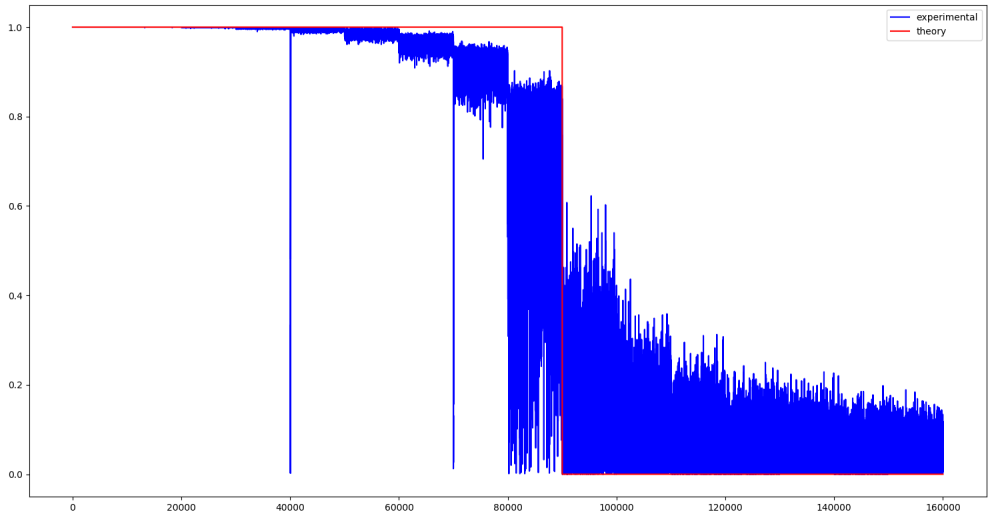

Figure 7: homogeneity of spin system in function of the temperature

The SDG method and the logistic regression have the same accuracy for the test data, however logit cross-validation regression gives less good results. I therefore recommend using either the SDG method or logistic regression.

About neural network

A - One dimension problem

At only one dimension, the problem are simple. The previous formula of Ising energy doesn't change, we take data as in the first part.
Some methods exist and the first one is to use a perceptron. It's the simpliest network compound of only one neuron whose take all the input data and return outputs. It take an activation function as sigmoïd or tangente. It's quite the same thing than a logistic regression.
To use this, we call from scikit learn the adequat method below :

```
from sklearn.neural_network
             import MLPClassifier

dnn = MLPClassifier(
      hidden_layer_sizes=50,
      activation='logistic',alpha=lmbd,
      learning_rate_init=eta)
dnn.fit(Xtrain,Ytrain)
```

Python 3.7

That give back the accuracy of the model we predict like the previous method of logistics regression.

A second method is to creat more complexe neural network. The output correspond to the first layer of neurons. After, we set some hidden layers until the last one which return the correct value expect. Between each layers, neurons are interlinked according to a weight. With scikit-learn, the code is called as :

```
from sklearn.linear_model \
             import Perceptron

clf = Perceptron()
clf.fit(X, y)
```

Python 3.7

Several problems occurs : the network need a hard training to predict something, a too large input may cause a long time to calculate gradient. In our case, we need absolutly to train over a bootstrap method or cross validation method.
The code behind a neural network are compound of some methods then we present here :

- Back-propagation algorythm : it's a method whose calculate the gradient of the cost function in regards to the weights.
- Feed-Forward algorythm : it's the first neural network devellop, and in those, it means that the information cannot turn back.

In our code we used TensorFlow to do the neural-network and this is our code :

```
# Neuronal Network
def DNN():
    N_train = 10000
    L = 40
    maxdegree = 10
    states_train = np.random.choice([-1, 1], size=(N_train,L))
    from keras.utils import to_categorical
    tmp = ising_energies(states_train, L)[:,np.newaxis]+40
```

```
Y_train = to_categorical(tmp)

N_test = 5000
states_test = np.random.choice([-1, 1], size=(N_test,L))
Y_test = to_categorical(ising_energies(states_test, L)+40)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.regularizers import l2
from keras.optimizers import Adam
lmbd=1e-5
eta =1e-5
dnn = Sequential([
    Dense(1024, input_shape=(states_train.shape[1],), activation='relu', kernel_regularizer
    Dense(1024, activation='relu', kernel_regularizer=l2(lmbd)),
    Dense(1024, activation='relu', kernel_regularizer=l2(lmbd)),
    Dense(Y_train.shape[1], activation='softmax'),
])

adam = Adam(learning_rate=eta)
dnn.compile(loss='binary_crossentropy', optimizer=adam, metrics=['acc'])

history = dnn.fit(states_train, Y_train, epochs=150, batch_size=32, validation_split=0.25)

# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

score = dnn.evaluate(states_test, Y_test, batch_size=32)
print(score)
```

Python 3.7

B - The Ising model phase

Now, we search to determine the Ising phase using neural network. The first solution is to use the previous code from scikit-learn and Tensorflow. We change now our cost function to the log cross-entropy classification cost function for the case discussed in part c). Train our network again and compare the results with those from your logistic regression code in c):

```
# Neural Network
def DNN(X_train, X_test, Y_train, Y_test, lmbd, eta, epochs=1000):
dnn = Sequential([
Dense(1024, input_shape=(X_train.shape[1],),
activation='relu', kernel_regularizer=l2(lmbd)),
Dense(Y_train.shape[1], activation='softmax'),
])
adam = Adam(learning_rate=eta)
dnn.compile(loss='categorical_crossentropy',
optimizer=adam, metrics=['acc'])

history = dnn.fit(X_train, Y_train, epochs=epochs, batch_size=32,  validation_split=0.25, callb
EarlyStopping(monitor='val_loss', min_delta=1e-5, patience=10,
verbose=0, mode='auto', baseline=None, restore_best_weights=True)
])

# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

score = dnn.evaluate(X_test, Y_test, batch_size=32)
print(score)
```

Python 3.7

C - Analysis

This is our results for the regression analysis of the one-dimensional Ising model using neural networks :

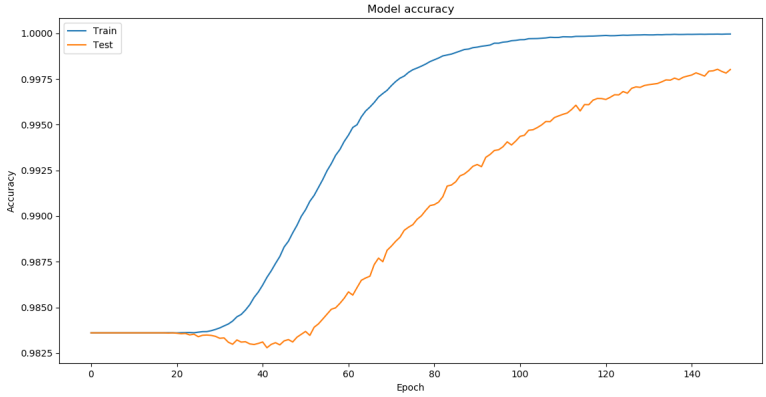

Figure 8: Accuracy of train data and test data in function of the epoch with the model train by neural network

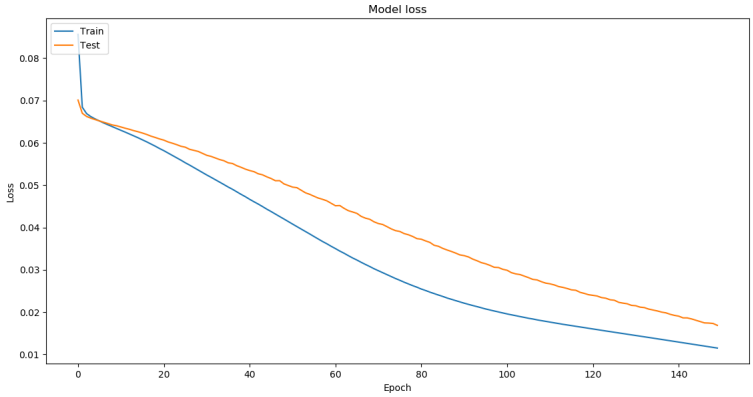

Figure 9: MSE of train data and test data in function of the epoch with the model train by neural network

Compared to linear regression, the neural network takes more computing time and is less precise with 150 epoch of calculations. But otherwise it tends towards the same result which is a good sign.

This is our results for the classifying the Ising model phase using neural networks.

The neural network gives results equivalent to SDG and logistic regression.

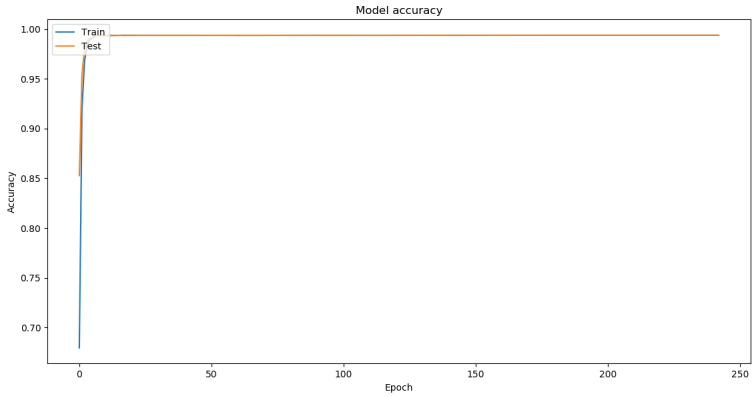

Figure 10: Accuracy of train data and test data in function of the epoch with the model train by neural network

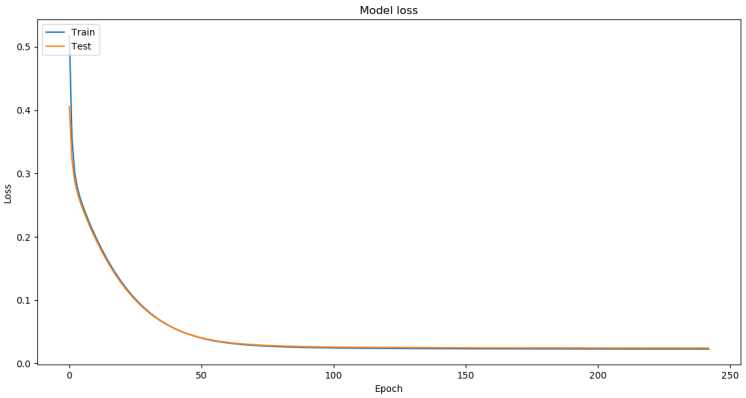

Figure 11: MSE of train data and test data in function of the epoch with the model train by neural network

Conclusion

We have been able to see in this work that the fastest methods and with good results are the Ridge regression and the SDG method. Neural networks give the same results but take more computing time.
The difference with the work of *textitMethaetal* is the way we treated logistic regression. We think that studying whether the surface is homogeneous spin or not is a better method than studying s