MSc Big Data for Business - MAP 534
Introduction to machine learning

Supervised classification (II)
*Logistic regression & feed forward neural networks*
Outline

Introduction

Logistic regression
  Inference

Feed Forward Neural Networks
  Model
  Implementation
Setting

→ Historical data about **individuals** \( i = 1, \ldots, n \).
→ **Features** vector \( X_i \in \mathbb{R}^d \) for each individual \( i \).
→ For each \( i \), the individual **belongs to a group** \( (Y_i = 0) \) or not \( (Y_i = 1) \).
→ \( Y_i \in \{0, 1\} \) is the **label** of \( i \).

Objective

→ Given a new \( X \) (with no corresponding label), **predict a label in** \( \{0, 1\} \).
→ Use data \( \mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\} \) to **construct a classifier**.
The best solution $f^*$ (which is independent of $D_n$) is

$$f^* = \arg\min_{f \in F} R(f) = \arg\min_{f \in F} \mathbb{E} \left[ 1_{Y \neq f(X)} \right] = \arg\min_{f \in F} \mathbb{P}(Y \neq f(X)).$$

Bayes Predictor (explicit solution)

→ Binary classification with 0–1 loss:

$$f^*(X) = \begin{cases} +1 & \text{if } \mathbb{P} \{ Y = 1 | X \} \geq \mathbb{P} \{ Y = 0 | X \} \\ \Leftrightarrow \mathbb{P} \{ Y = 1 | X \} \geq 1/2, & \text{otherwise}. \end{cases}$$

The explicit solution requires to know the conditional law of $Y$ given $X$...
How to estimate the conditional law of $Y$?

**Fully parametric modeling.**

Estimate the law of $(X, Y)$ and use the **Bayes formula** to deduce an estimate of the conditional law of $Y$: LDA/QDA, Naive Bayes...

**Parametric conditional modeling.**

Estimate the conditional law of $Y$ by a **parametric** law: linear regression, logistic regression, Feed Forward Neural Networks...

**Nonparametric conditional modeling.**

Estimate the conditional law of $Y$ by a **non parametric** estimate: kernel methods, nearest neighbors...
The **conditional densities are modeled as multivariate normal**. For all class $k \in \{0, 1\}$, conditionnally on \( \{Y = k\} \),

\[
X \sim \mathcal{N}(\mu_k, \Sigma_k).
\]

**Discriminant functions:**

\[
\psi_k : x \mapsto \ln(g_k(x)) + \ln(\mathbb{P}\{Y = k\}).
\]

In a two-classes problem, the optimal classifier is:

\[
f^* : x \mapsto \mathbb{1}\{\psi_1(x) > \psi_0(x)\}.
\]

QDA (differents $\Sigma_k$ in each class) and LDA ($\Sigma_k = \Sigma$ for all $k$)

→ May lead to poor results is the **model does not describe the data correctly**.
In the LDA case, the classification rule is of the form:

\[ f^*(x) = 1 \iff \langle w, x \rangle + b \geq 0, \]

where \( w \) and \( b \) depends on the model parameters.

→ How to relax the Gaussian assumption? (logistic model).

→ How to design nonlinear classification rules? (neural networks).
Outline

Introduction

Logistic regression
  Inference

Feed Forward Neural Networks
  Model
  Implementation
The objective is to **predict the label** $Y \in \{0, 1\}$ based on $X \in \mathbb{R}^d$.

Logistic regression **models the distribution of** $Y$ **given** $X$.

$$P(Y = 1|X) = \sigma(\langle w, X \rangle + b),$$

where $w \in \mathbb{R}^d$ is a vector of model **weights** and $b \in \mathbb{R}$ is the **intercept**, and where $\sigma$ is the **sigmoid** function.

$$\sigma: z \mapsto \frac{1}{1 + e^{-z}}.$$

The sigmoid function is a **model choice to map** $\mathbb{R}$ **into** $(0, 1)$.

Another widespread solution for $\sigma$ is $\sigma: z \mapsto P(Z \leq z)$ where $Z \sim \mathcal{N}(0, 1)$, which leads to a **probit** regression model.
Logistic regression

Log-odd ratio

\[
\log \left( \frac{P(Y = 1|X)}{P(Y = 0|X)} \right) = \langle w, X \rangle + b.
\]

Classification rule

Note that

\[ P(Y = 1|X) \geq P(Y = 0|X) \]

if and only if

\[ \langle w, x \rangle + b \geq 0. \]

→ This is a **linear classification** rule.

→ This classifier requires to **estimate** \( w \) and \( b \).
Outline

Introduction

Logistic regression
  Inference

Feed Forward Neural Networks
  Model
  Implementation
→ \{ (X_i, Y_i) \}_{1 \leq i \leq n} are i.i.d. with the same distribution as (X, Y).

**Likelihood**

\[
\prod_{i=1}^{n} P(Y_i | X_i) = \prod_{i=1}^{n} \sigma(\langle w, X_i \rangle + b)^{Y_i} (1 - \sigma(\langle w, X_i \rangle + b))^{1-Y_i},
\]

\[
= \prod_{i=1}^{n} \sigma(\langle w, x_i \rangle + b)^{Y_i} \sigma(-\langle w, X_i \rangle - b)^{1-Y_i}
\]

and the **normalized negative loglikelihood** is

\[
f(w, b) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-Y_i (\langle w, X_i \rangle + b)}) = \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, \langle w, X_i \rangle + b).
\]
Logistic regression

Compute $\hat{w}_n$ and $\hat{b}_n$ as follows:

$$(\hat{w}_n, \hat{b}_n) \in \arg\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-Y_i(\langle w, X_i \rangle + b)}) .$$

→ It is an **average of losses**, one for each sample point.

→ It is a **convex and smooth problem**.

Using the **logistic loss** function

$$\ell : (y, y') \mapsto \log(1 + e^{-yy'})$$

yields

$$(\hat{w}_n, \hat{b}_n) \in \arg\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, \langle w, X_i \rangle + b) .$$
Maximum likelihood estimate

Assume for now that the intercept is 0. Then, the likelihood is,

\[
L_n(w) = \prod_{i=1}^{n} \left( \frac{e^{X_i^T w}}{1 + e^{X_i^T w}} \right)^{Y_i} \left( \frac{1}{1 + e^{X_i^T w}} \right)^{1-Y_i} = \prod_{i=1}^{n} \left( \frac{e^{X_i^T w Y_i}}{1 + e^{X_i^T w}} \right).
\]

And the negative log-likelihood is

\[
\ell_n(w) = -\log(L_n(w)) = \sum_{i=1}^{n} \left( -Y_i X_i^T w + \log(1 + e^{X_i^T w}) \right).
\]

Derivatives

\[
\frac{\partial \log(L_n(w))}{\partial w_j} = \sum_{i=1}^{n} \left( Y_i X_{ij} - \frac{x_{ij} e^{X_i^T w}}{1 + e^{X_i^T w}} \right) = \sum_{i=1}^{n} X_{ij} (Y_i - \sigma(\langle w, X_i \rangle)).
\]

→ **No explicit solution** for the maximizer of the loglikelihood... Parameter estimate obtained using **gradient based optimization** (see next lesson).
Maximum likelihood estimate

The negative loglikelihood

\[ \ell_n(w) = -\log(L_n(w)) = \sum_{i=1}^{n} \left( -Y_i X_i^T w + \log(1 + e^{X_i^T w}) \right). \]

is minimized using a gradient descent algorithm.

Starting with an initial estimate \( w^{(0)} \), for all \( k \geq 1 \), set

\[ w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell_n(w^{(k-1)}). \]
Maximum likelihood estimate

Let \((w^*, b^*)\) be the parameter estimates after the gradient descent algorithm.

The usual logistic regression classifier is \(f^*(X) = 1 \iff \mathbb{P}(Y = 1|X) > 1/2\).

Sensitivity of the classifier to this threshold: for each value \(p^* \in (0, 1)\) the ROC curve classifies individuals using \(f^*(X) = 1 \iff \mathbb{P}(Y = 1|X) > p^*\) and plots the True positive rate as a function of the False positive rate.
The gradient of the negative loglikelihood is,

\[ \nabla \ell_n(w) = - \sum_{i=1}^{n} Y_i X_i + \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{1 + \exp(\langle X_i, w \rangle)} X_i. \]

On the other hand, for all \( 1 \leq i \leq n \) and all \( 1 \leq j \leq d \),

\[ \partial_j \left( \frac{\exp(\langle X_i, w \rangle)}{1 + \exp(\langle X_i, w \rangle)} X_i \right) = \frac{\exp(\langle X_i, w \rangle)}{(1 + \exp(\langle X_i, w \rangle))^2} X_{ij} X_i, \]

where \( X_{ij} \) is the \( j \)th component of \( X_i \).

Then, the Hessian matrix is

\[ (H_n(w))_{\ell j} = \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{(1 + \exp(\langle X_i, w \rangle))^2} X_{ij} X_{i\ell}, \]

that is,

\[ H_n(w) = \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{(1 + \exp(\langle X_i, w \rangle))^2} X_i X_i^T. \]

\( H_n(\beta) \) is a semi positive definite matrix, which implies that \( \ell_n(\beta) \) is convex.
Asymptotic properties

Assumptions

→ \( \hat{w}_n \to w^* \) almost surely.

→ There exists a continuous and nonsingular function \( H \) such that \( n^{-1}H_n(w) \) converges to \( H(w) \), uniformly in a ball around \( w^* \).

For all \( t \in \mathbb{R}^d \), using a Taylor expansion,

\[
\mathbb{E} \left[ \exp \left( -\frac{1}{\sqrt{n}} \langle t, \nabla \ell_n(w^*) \rangle \right) \right] \to_{n \to \infty} \exp \left( \frac{1}{2} t^T H(w^*) t \right).
\]

Therefore,

\[ -\nabla \ell_n(w^*)/\sqrt{n} \Rightarrow \mathcal{N}(0, H(w^*)). \]

On the other hand, by Slutsky lemma,

\[ \sqrt{n}(\hat{w}_n - w^*) \Rightarrow \mathcal{N}(0, H(w^*)^{-1}). \]
Confidence interval

$-\sqrt{n}(\hat{w}_j - w_j^*)$ converges in distribution to a centered Gaussian random variable with variance $(H(w^*)^{-1})_{jj}$.

Almost surely,

$$\hat{\sigma}_{n,j}^2 = (nH_n(\hat{w}_n)^{-1})_{jj} \rightarrow_{n \to \infty} (H(w^*)^{-1})_{jj}.$$ Then,

$$\sqrt{\frac{n}{\hat{\sigma}_{n,j}^2}}(\hat{w}_{n,j} - \beta_j^*) \rightarrow_{n \to \infty} \mathcal{N}(0, 1).$$

An asymptotic confidence interval $I_{n,\alpha}$ of level $1 - \alpha$ is then

$$I_{n,\alpha} = \left[ \hat{w}_{n,j} - z_{1-\alpha/2} \sqrt{\frac{\hat{\sigma}_{n,j}^2}{n}}, \hat{\beta}_{n,j} + z_{1-\alpha/2} \sqrt{\frac{\hat{\sigma}_{n,j}^2}{n}} \right],$$

where $z_{1-\alpha/2}$ is the quantile of order $1 - \alpha/2$ of $\mathcal{N}(0, 1)$.
Outline

Introduction

Logistic regression

  Inference

Feed Forward Neural Networks

  Model

  Implementation
Outline

Introduction

Logistic regression

Inference

Feed Forward Neural Networks

Model

Implementation
→ The objective is to **predict the label** $Y \in \{1, \ldots, M\}$ based on $X \in \mathbb{R}^d$.

→ Softmax regression **models the distribution of** $Y$ **given** $X$.

**The model**

For all $1 \leq m \leq M$,

$$z_m = \langle w_m, X \rangle + b_m,$$

$$P(Y = m|X) = \text{softmax}(z)_m,$$

where $z \in \mathbb{R}^M$, $w_m \in \mathbb{R}^d$ is a vector of model **weights** and $b_m \in \mathbb{R}$ is an **intercept**, and where **softmax** is the **softmax** function: for all $1 \leq m \leq M$,

$$\text{softmax}(z)_m = \frac{\exp(z_m)}{\sum_{j=1}^{M} \exp(z_j)}.$$
→ $X$ input in $\mathbb{R}^d$.

→ $z(X)$ pre-activation in $\mathbb{R}^M$, with weight $W \in \mathbb{R}^{d \times M}$ and bias $b \in \mathbb{R}^M$.

→ $g$ softmax function.

One neuron is a multi-class extension of the logistic regression model.
Layer of neurons and hidden states

\[ f(X) = g(z(X)) = g(WX + b) \]

- \( X \) input in \( \mathbb{R}^d \).

- \( z(X) \) pre-activation in \( \mathbb{R}^k \), with weight \( W \in \mathbb{R}^{d \times k} \) and bias \( b \in \mathbb{R}^k \).

- \( g \) any activation function (nonlinear & nondecreasing function).

- \( f(X) \) hidden state in \( \mathbb{R}^k \) which may be used as input of a new neuron...
→ $X$ input in $\mathbb{R}^d$.

→ $z^h(X)$ pre-activation in $\mathbb{R}^H$, with weight $W^h \in \mathbb{R}^{d \times H}$ and bias $b^h \in \mathbb{R}^H$. 
\( X \) **input in** \( \mathbb{R}^d \).

\( z^h(X) \) **pre-activation in** \( \mathbb{R}^H \), with **weight** \( W^h \in \mathbb{R}^{dxH} \) and **bias** \( b^h \in \mathbb{R}^H \).

\( g \) **any activation function** to produce \( h \in \mathbb{R}^H \).
→ \( X \) input in \( \mathbb{R}^d \).

→ \( z^h(X) \) pre-activation in \( \mathbb{R}^H \), with weight \( W^h \in \mathbb{R}^{d \times H} \) and bias \( b^h \in \mathbb{R}^H \).

→ \( g \) any activation function to produce \( h \in \mathbb{R}^H \).

→ \( z^o(X) \) pre-activation in \( \mathbb{R}^M \), with weight \( W^o \in \mathbb{R}^{H \times M} \) and bias \( b^o \in \mathbb{R}^M \).
→ X input in $\mathbb{R}^d$.

→ $z^h(X)$ pre-activation in $\mathbb{R}^H$, with weight $W^h \in \mathbb{R}^{d \times H}$ and bias $b^h \in \mathbb{R}^H$.

→ $g$ any activation function to produce $h \in \mathbb{R}^H$.

→ $z^o(X)$ pre-activation in $\mathbb{R}^M$, with weight $W^o \in \mathbb{R}^{H \times M}$ and bias $b^o \in \mathbb{R}^M$.

→ Apply the softmax function to produce the output, i.e. $\mathbb{P}(Y = m|X)$ for $1 \leq m \leq M$. 
As there is no modelling assumptions anymore, virtually any activation function may be used.

The rectified linear unit (RELU) activation function $\sigma(x) = \max(0, x)$ and its extensions are the default recommendation in modern implementations (Jarrett et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011a), (Maas et al., 2013), (He et al., 2015). One of the major motivations arise from the gradient based parameter optimization which is numerically more stable with this choice.
Outline

Introduction

Logistic regression
  Inference

Feed Forward Neural Networks
  Model
  Implementation
This dataset contains images representing handwritten digits. Each image is made of \(28 \times 28\) pixels, and each pixel is represented by an integer (gray level). These arrays can be flattened into vectors in \(\mathbb{R}^{784}\).

The labels in \(\{0, \ldots, 9\}\) are represented using one-hot-encoding and grayscale of each pixel in \(\{0, \ldots, 255\}\) are normalized to be in \((0, 1)\).

```python
from keras.datasets import mnist

# Number of classes
num_classes = 10

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)

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

print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

x_train shape: (60000, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)
y_train shape: (60000,)
y_test shape: (10000,)
60000 train samples
10000 test samples
```
This dataset contains images representing handwritten digits. Each image is made of \(28 \times 28\) pixels, and each pixel is represented by an integer (gray level). These arrays can be flattened into vectors in \(\mathbb{R}^{784}\).

The labels in \(\{0, \ldots, 9\}\) are represented using one-hot-encoding and grayscale of each pixel in \(\{0, \ldots, 255\}\) are normalized to be in \((0, 1)\).
The model with Keras

```python
model_ffnn = Sequential()
model_ffnn.add(Flatten(input_shape=input_shape))
model_ffnn.add(Dense(128, activation='relu'))
model_ffnn.add(Dense(num_classes, activation='softmax'))
model_ffnn.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adagrad(),
    metrics=['accuracy'])
model_ffnn.summary()
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 784)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 128)</td>
<td>100480</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 10)</td>
<td>1290</td>
</tr>
<tr>
<td>Total params: 101,770</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trainable params: 101,770</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-trainable params: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1:** Feed Forward Neural network. $h_1$ is obtained with the **RELU activation function** and is in $\mathbb{R}^{128}$. The last layer is $h_2 \in \mathbb{R}^{10}$ and is obtained with the **softmax activation function** so that each component $m$ models $P(Y = m | X)$. This neural network with one hidden layer relies on 101.770 parameters.
This model relies on more than **100,000 unknown parameters** which should be estimated.

As for the logistic regression and the discriminant analysis, a common choice is to **minimize the negative loglikelihood of the data**:

\[
\theta \leftarrow -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{10} \mathbb{1}_{Y_i = k} \log P_{\theta}(Y_i = k | X_i).
\]

The negative loglikelihood is computed using \( n = 60,000 \) training samples and **minimized using gradient descent algorithms** - see next lesson.

Then, the performance of the model is assessed using \( 10,000 \) new (test) samples: the **accuracy is the frequency of labels which are well predicted by the model with the estimated parameters**.
batch_size = 32
epochs = 8

# Run the train
history = model_ffnn.fit(x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test))
score = model_ffnn.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

plt.figure(figsize=(5, 4))
plt.plot(history.epoch, history.history['acc'], lw = 1, label='Training')
plt.plot(history.epoch, history.history['val_acc'], lw = 1, label='Testing')
plt.legend()
plt.title('Accuracy of softmax regression', fontsize=16)
plt.xlabel('Epoch', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.tick_params(labelright=True)
plt.grid(True)
plt.tight_layout()

Figure 2: Minimization of the negative liglikelihood using a gradient descent algorithm (here AdaGrad). The gradient is computed using batches of 32 observations and the whole data set is used 8 times.