Les données numériques: classification d'images - Regression Logistique

Diane Lingrand

UNIVERSITÉ CÔTE D'AZUR
Polytech SI3

2022 - 2023

Outline

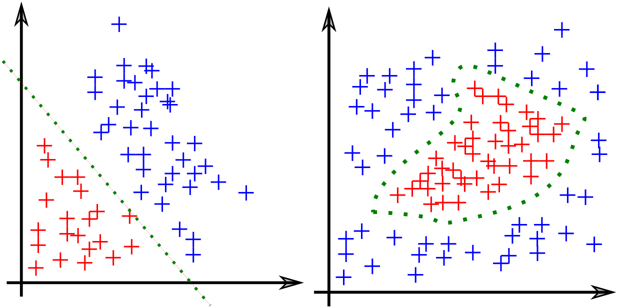
Classification supervisée des images

- classification des vecteurs représentant les images (*bow*)
- supervisée : on connait les labels (classes) des images et on utilise cette information lors de l'apprentissage
  - kNN
  - régression logistique linéaire
  - mais aussi : SVM, arbres de décisions, réseaux de neurones, ...
- utilisation des données
  - *train* : données utilisées pour l'apprentissage
  - *validation* : données utilisées pour le réglage des paramètres de l'apprentissage ou de la convergence de certains algorithmes
  - *test* : données utilisées pour évaluer les performances de la classification

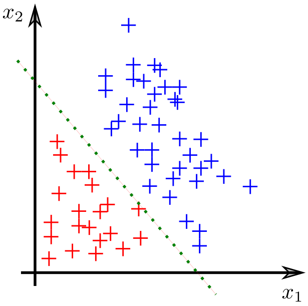
Outline

Régression logistique (*logistic regression*)

- Séparation binaire de données selon leurs labels (0 ou 1).
  - Séparation linéaire : droite ou hyperplan
  - (Séparation non-linéaire : polynomiale, gaussienne ...)
- Notations :
  - données : $\mathrm{x}^{\mathrm{j}} = [x_1^j \ x_2^j ...]$
  - labels : $y^j \in \{0, 1\}$
  - critère de décision h_θ de paramètre θ
    - $\theta = [\theta_0 \ \theta_1 \ ...]$



Séparation linéaire



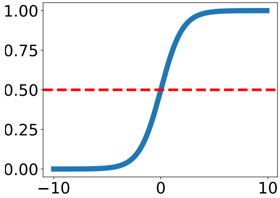
- Frontière de décision :
  - droite d'équation $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$
  - s'écrit aussi : $\theta^T \mathrm{x} = 0$
- Décision :
  - si $\theta^T \mathrm{x} \geq 0$ alors $y = 1$
  - si $\theta^T \mathrm{x} < 0$ alors $y = 0$

Function logistique

$$h_\theta(\mathrm{x}) = s(\theta^T \mathrm{x})$$

avec :

$$s(z) = \frac{1}{1 + e^{-z}}$$



- La décision devient :
  - si $h_\theta(\mathrm{x}) \geq 0.5$ alors $y = 1$
  - si $h_\theta(\mathrm{x}) < 0.5$ alors $y = 0$

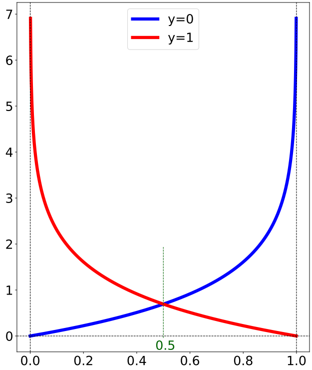
Régression logistique : apprentissage

- données d'apprentissage : $(\mathrm{x}^1, y^1), (\mathrm{x}^2, y^2), ..., (\mathrm{x}^m, y^m)$
- m données d'apprentissage
- but de l'apprentissage : trouver θ
- méthode :
  - minimisation d'une erreur
  - descente de gradient (ou autre méthode de minimisation)

Fonction de coût

$$J = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(h_\theta(\mathrm{x}^\mathrm{i})) + (1 - y^i)\log(1 - h_\theta(\mathrm{x}^\mathrm{i}))$$

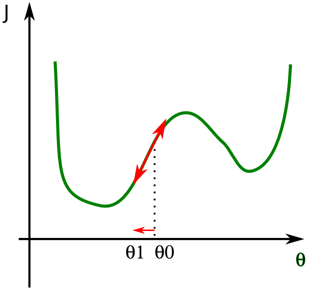
- valeurs de $h_\theta(\mathrm{x}^\mathrm{i})$:
  - entre 0 (négatif) et 1 (positif)
- donnée positive : $y^i = 1$
  - la prédiction est parfaite : $h_\theta(\mathrm{x}^\mathrm{i}) = 1 \Rightarrow$ coût nul
    - $\log(h_\theta(\mathrm{x}^\mathrm{i})) = 0$
  - la plus mauvaise prédiction : $h_\theta(\mathrm{x}^\mathrm{i}) = 0 \Rightarrow$ coût infini
- donnée négative : $y^i = 0$
  - la prédiction est parfaite : $h_\theta(\mathrm{x}^\mathrm{i}) = 0 \Rightarrow$ coût nul
    - $\log(1 - h_\theta(\mathrm{x}^\mathrm{i})) = 0$
  - la plus mauvaise prédiction : $h_\theta(\mathrm{x}^\mathrm{i}) = 1 \Rightarrow$ coût infini



Descente de gradient : le principe

- On cherche un minimum de la fonction de coût
- On va partir d'une valeur initiale pour θ, aléatoire.
- Si θ n'était pas un vecteur mais une variable scalaire :
  - Descente de gradient :

$$\theta = \theta - \alpha J'(\theta) = \theta - \alpha \frac{dJ}{d\theta}(\theta)$$



Descente de gradient : les calculs (1)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s(z) = \frac{1}{1+e^{-z}}$

$$s'(z) = \frac{-\frac{d(e^{-z})}{dz}}{(1+e^{-z})^2} \tag{1}$$

Descente de gradient : les calculs (1)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s(z) = \frac{1}{1+e^{-z}}$

$$
\begin{aligned}
s'(z) &= \frac{-\frac{d(e^{-z})}{dz}}{(1+e^{-z})^2} && (1) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} && (2)
\end{aligned}
$$

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s(z) = \frac{1}{1+e^{-z}}$

$$\begin{aligned} s'(z) &= \frac{-\frac{d(e^{-z})}{dz}}{(1+e^{-z})^2} && (1) \\ &= \frac{e^{-z}}{(1+e^{-z})^2} && (2) \\ &= s(z)\frac{e^{-z}}{1+e^{-z}} && (3) \end{aligned}$$

Descente de gradient : les calculs (1)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s(z) = \frac{1}{1+e^{-z}}$

$$
\begin{aligned}
s'(z) &= \frac{-\frac{d(e^{-z})}{dz}}{(1+e^{-z})^2} && (1)\\
&= \frac{e^{-z}}{(1+e^{-z})^2} && (2)\\
&= s(z)\frac{e^{-z}}{1+e^{-z}} && (3)\\
&= s(z)\frac{(1+e^{-z})-1}{1+e^{-z}} && (4)
\end{aligned}
$$

Descente de gradient : les calculs (1)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s(z) = \frac{1}{1+e^{-z}}$

$$
\begin{aligned}
s'(z) &= \frac{-\frac{d(e^{-z})}{dz}}{(1+e^{-z})^2} && (1)\\
&= \frac{e^{-z}}{(1+e^{-z})^2} && (2)\\
&= s(z)\frac{e^{-z}}{1+e^{-z}} && (3)\\
&= s(z)\frac{(1+e^{-z})-1}{1+e^{-z}} && (4)\\
&= s(z)(1-s(z)) && (5)
\end{aligned}
$$

Descente de gradient : les calculs (2)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) :

$$J_\theta(x) = -\log(h_\theta(\mathrm{x})) = -\log(s(\theta^T \mathrm{x}))$$

Descente de gradient : les calculs (2)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) :

$$J_\theta(x) = -\log(h_\theta(\mathrm{x})) = -\log(s(\theta^T \mathrm{x}))$$

$$\frac{dJ_\theta(x)}{d\theta} = -\frac{d(s(\theta^T \mathrm{x}))}{d\theta} \frac{1}{s(\theta^T \mathrm{x})} \tag{6}$$

Descente de gradient : les calculs (2)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) :

$$J_\theta(x) = -\log(h_\theta(\mathrm{x})) = -\log(s(\theta^T \mathrm{x}))$$

$$\begin{aligned} \frac{dJ_\theta(x)}{d\theta} &= -\frac{d(s(\theta^T \mathrm{x}))}{d\theta} \frac{1}{s(\theta^T \mathrm{x})} && (6) \\ &= -x \frac{s(\theta^T \mathrm{x})(1 - s(\theta^T \mathrm{x}))}{s(\theta^T \mathrm{x})} && (7) \end{aligned}$$

Descente de gradient : les calculs (2)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) :

$$J_\theta(x) = -\log(h_\theta(\mathrm{x})) = -\log(s(\theta^T\mathrm{x}))$$

$$\begin{aligned}
\frac{dJ_\theta(x)}{d\theta} &= -\frac{d(s(\theta^T\mathrm{x}))}{d\theta}\frac{1}{s(\theta^T\mathrm{x})} && (6)\\
&= -x\frac{s(\theta^T\mathrm{x})(1 - s(\theta^T\mathrm{x}))}{s(\theta^T\mathrm{x})} && (7)\\
&= (h_\theta(x) - 1)x && (8)
\end{aligned}$$

Descente de gradient : les calculs (3)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) : $\frac{dJ_\theta(x)}{d\theta} = (h_\theta(x) - 1)x$
  - pour une donnée négative ($y = 0$) :

$$J_\theta(x) = -\log(1 - h_\theta(\mathrm{x})) = -\log(1 - s(\theta^T \mathrm{x}))$$

Descente de gradient : les calculs (3)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) : $\frac{dJ_\theta(x)}{d\theta} = (h_\theta(x) - 1)x$
  - pour une donnée négative ($y = 0$) :

$$J_\theta(x) = -\log(1 - h_\theta(\mathrm{x})) = -\log(1 - s(\theta^T \mathrm{x}))$$

$$\frac{dJ_\theta(x)}{d\theta} = -\frac{d(1 - s(\theta^T \mathrm{x}))}{d\theta} \frac{1}{1 - s(\theta^T \mathrm{x})} \tag{9}$$

Descente de gradient : les calculs (3)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) : $\frac{dJ_\theta(x)}{d\theta} = (h_\theta(x) - 1)x$
  - pour une donnée négative ($y = 0$) :

$$J_\theta(x) = -\log(1 - h_\theta(\mathrm{x})) = -\log(1 - s(\theta^T \mathrm{x}))$$

$$\begin{aligned} \frac{dJ_\theta(x)}{d\theta} &= -\frac{d(1 - s(\theta^T \mathrm{x}))}{d\theta} \frac{1}{1 - s(\theta^T \mathrm{x})} && (9) \\ &= \frac{s'(\theta^T \mathrm{x}x)}{1 - s(\theta^T \mathrm{x})} = s(\theta^T \mathrm{x})x = (h_\theta(x) - 0)x && (10) \end{aligned}$$

Descente de gradient : les calculs (3)

- Pour réaliser la descente de gradient :
  - il faut dériver la fonction de coût
  - dérivée de la sigmoide : $s'(z) = s(z)(1 - s(z))$
  - pour une donnée positive ($y = 1$) : $\frac{dJ_\theta(x)}{d\theta} = (h_\theta(x) - 1)x$
  - pour une donnée négative ($y = 0$) :

$$J_\theta(x) = -\log(1 - h_\theta(\mathrm{x})) = -\log(1 - s(\theta^T \mathrm{x}))$$

$$\begin{aligned} \frac{dJ_\theta(x)}{d\theta} &= -\frac{d(1 - s(\theta^T \mathrm{x}))}{d\theta} \frac{1}{1 - s(\theta^T \mathrm{x})} && (9) \\ &= \frac{s'(\theta^T \mathrm{x}x)}{1 - s(\theta^T \mathrm{x})} = s(\theta^T \mathrm{x})x = (h_\theta(x) - 0)x && (10) \end{aligned}$$

  - pour toute donnée : $(h_\theta(x) - y)x$

- On cherche un minimum de la fonction de coût
- On va partir d'une valeur initiale pour θ, aléatoire.
- Descente de gradient :
  - Pour toutes les composantes θ_j de θ :

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

  avec

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathrm{x}^{\mathrm{i}}) - y^i) x_j^i$$

- convergence :
  - nombre d'itérations maximum
  - critére sur le coût

La regression logistique correspond à un neurone artificiel avec une fonction d'activation sigmoide.

Outline

- page de référence : https://scikit-learn.org/stable/
- installation : https://scikit-learn.org/stable/install.html
- pour ce cours :
  - régression logistique : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression
  - kNN : https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

régression logistique avec scikit-learn

```
from sklearn.linear_model import LogisticRegression
#linear regression object creation
lr = LinearRegression()
#learning
history = lr.fit(data, labels)
#prediction
pred = lr.predict(testdata)
#score (accuracy)
score = lr.score(testdata, testlabels)
print("test score =", score)
```

- Différents éléments sont cachées ici :
  - nombre d'itérations (`max_iter`)
  - régularisation
  - multiclasses

kNN avec scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
#kNN object creation with k=10
myKnn = KNeighborsClassifier(n_neighbors=10)
#learning
myKnn.fit(data, labels)
#prediction
pred = myKnn.predict(testdata)
#score (accuracy)
score = myKnn.score(testdata, testlabels)
print("test score =", score)
```

- Différents éléments sont cachées ici :
  - poids des données (par défaut 'uniform')
  - algorithme : kd-tree ou autre, pouvant être choisi automatiquement
  - ...

Les métriques d'évaluation

Décrites sur la page : https://scikit-learn.org/stable/modules/model_evaluation.html

```
from sklearn.metrics import f1_score, accuracy_score,
        ConfusionMatrixDisplay

yPredTest = clf.predict(xTest)

# cas binaire
print("F1 score (test): ", f1_score(yTest, yPredTest))
# cas multi-classes
print("F1 score (test): ", f1_score(yTest, yPredTest,
    average = 'micro')

print("accuracy (test): ", accuracy_score(yTest, yPredTest))
ConfusionMatrixDisplay.from_predictions(yTest, y_predTest)
```

Outline

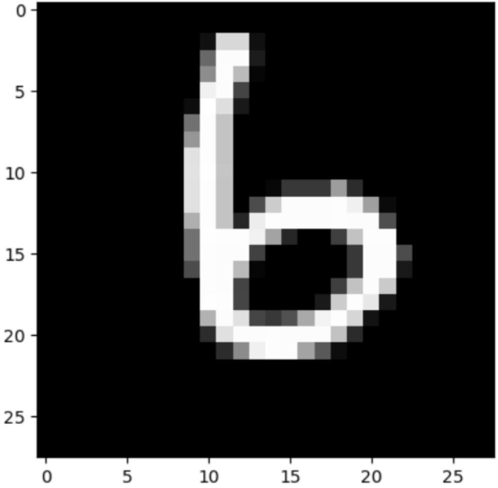
Dataset : Reduced version of MNIST

- 10 classes of gray images (28x28)
- only 2000 images in the train set (200 per class)
- only 1000 images in the test set (100 per class)
- How to get the files ?
  - download :
    - https://www.i3s.unice.fr/~lingrand/redMNIST-x-train.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-x-test.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-y-train.bin
    - https://www.i3s.unice.fr/~lingrand/redMNIST-y-test.bin
  - load in you python code :

```
import pickle
with open('redMNIST-x-train.bin', 'rb') as input:
    xTrain = pickle.load(input)
with open('redMNIST-x-test.bin', 'rb') as input:
    xTest = pickle.load(input)
with open('redMNIST-y-test.bin', 'rb') as input:
    yTest = pickle.load(input)
with open('redMNIST-y-train.bin', 'rb') as input:
    yTrain = pickle.load(input)
```

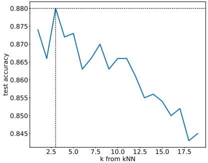
Looking at images

```
import random
n = random.randrange(0,len(yTrainr))
plt.imshow(xTrain[n], cmap=plt.cm.gray)
```



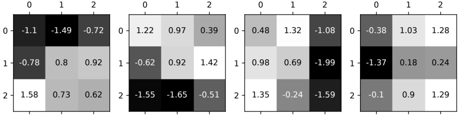
Baselines

- flat the train and test data
  - `x_train = xTrain.reshape(x_train.shape[0],784)`
- kNN
  - test the kNN with k=10
  - then search for the best k (according to the test accuracy)

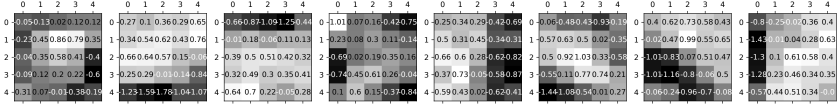


- logistic regression
  - again learn using the train set and test on the test set

- transform each image using few correlation filters :



or



  - do not consider borders
  - do not consider all raws and all columns : stride variable
  - 1 image will be represented by 1 vector : need to flat all results
- using the training dataset :
  - search for the best k in kNN
  - test the metrics on the train dataset AND the test dataset