04 – Linear Models

Data Science and Management

Corso di Laurea Magistrale in Ingegneria Gestionale

Marco Mamei, Natalia Hadjidimitriou, Fabio D’Andreagiovanni, Matteo Martinelli

{marco.mamei, selini, fabio.dandreagiovanni, matteo.martinelli}@unimore.it

- Linear Classifier
- Perceptron
- Logistic Regression
- Example

Linear Classifiers

Discriminant function as a **linear combination** of features

$$f(x) = \beta^T x + \beta_0$$

- Quite **simple**: one of the first algorithms to try
- With **very large datasets** and **complex tasks** it could be the only available options (being simple, it has a low cost)

Perceptron

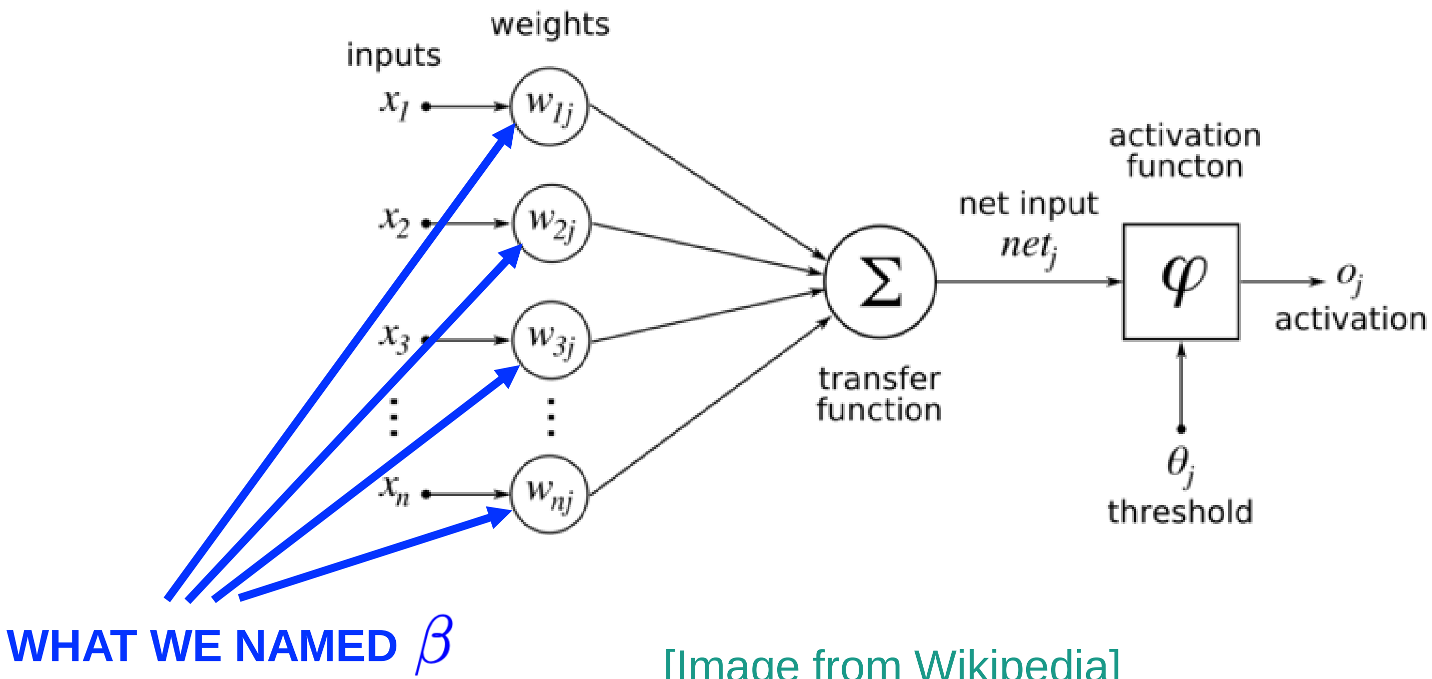
Architecture with a **single neuron**

$$f(x) = \mathrm{sign}(\beta^T x + \beta_0)$$

- Biological inspiration: **synapsis activation** when the neuron potential exceeds a certain **threshold**
- Capable to classify correctly **linearly separable** examples

Perceptron

Architecture with a **single neuron** (Rosenblatt, 1958)



[Image from Wikipedia]

Perceptron

How to learn parameters

- Define a **loss** function to be **minimized**
- Typically the **classification error** on the training set

$$E(\beta, \mathcal{D}) = \sum_{(x_i, y_i) \in \mathcal{D}} \ell(y_i, f(x_i))$$

- How can we **minimize** this error?

Perceptron

For the classic perceptron the **loss function** is defined as:

$$\ell(y_i, f(x_i)) = -y_i f(x_i)$$

Idea: **different signs** between target and prediction imply **wrong classifications**, so we should correct them

Perceptron

Gradient descent algorithm

- Compute the **gradient** of error function wrt parameters
- **Iterate** until gradient is approximately zero

$$\beta^i = \beta^{i-1} - \eta \nabla E(\beta, \mathcal{D})$$

i IS THE ITERATION

- η is called the **learning rate**

Perceptron

The learning rate is a **crucial** parameter

- Too **small** implies slow **convergence**
- Too **high** causes **instability**
- There exist algorithms to **adapt it during training**

Guarantee to reach a **local minimum** of the error function

Perceptron

Batch vs. **stochastic** gradient descent

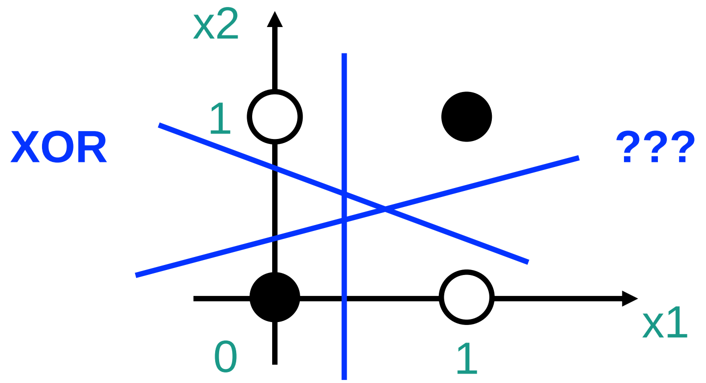
- In general, the gradient is obtained as the **sum** of gradients computed for all the **training examples**
- In principle, we could also update the vector of parameters **after each example** rather than after all the training set
- Update after each example is called **stochastic** update
- **Faster** training, which can help **avoiding** local minima
  - Helps avoiding positive and negative example cancelling each other

- **Mini-batch**

Perceptron

The perceptron finds a separating **hyperplane**

This is often **not enough**, because in many cases there exist **non-linear relationships** between input (features) and output (target)



Perceptron

Perceptron can also be used for **regression** problems

- Do **not** use threshold over the neuron output
- Exploit a **different loss function** (e.g., RMSE)

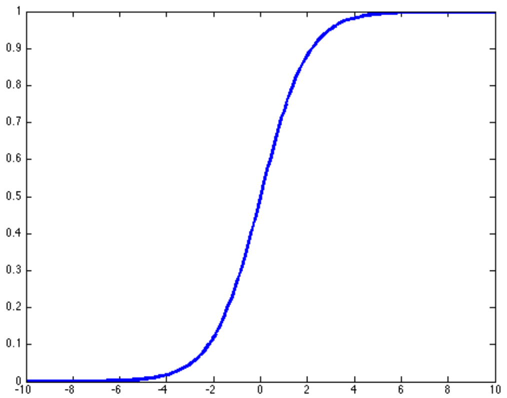
$$\ell(y_i, f(x_i)) = (y_i - f(x_i))^2$$

Logistic Regression

Modeling the probability of target being 1, given evidence

$$h(x) = p(y=1|x) = \frac{1}{1+e^{-(\beta^T x+\beta_0)}} = \frac{e^{\beta^T x+\beta_0}}{1+e^{\beta^T x+\beta_0}}$$

The output lies between 0 and 1, as in the **sigmoid** function



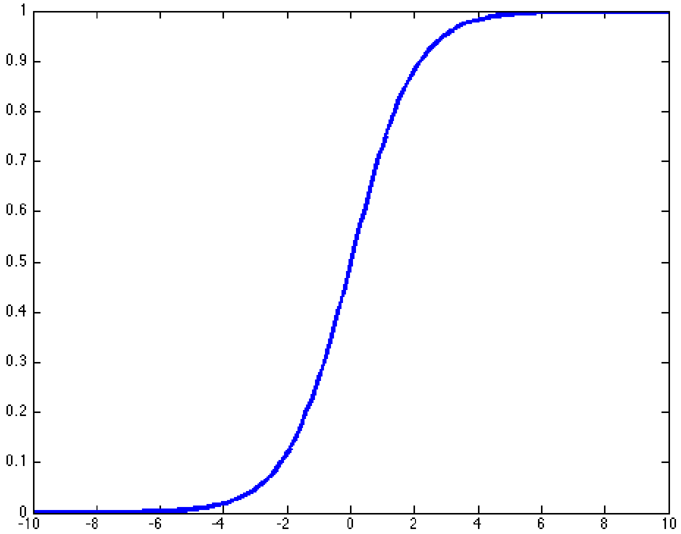
$$\sigma(t) = \frac{1}{1+e^{-t}}$$
$$t = \beta^T x + \beta_0$$

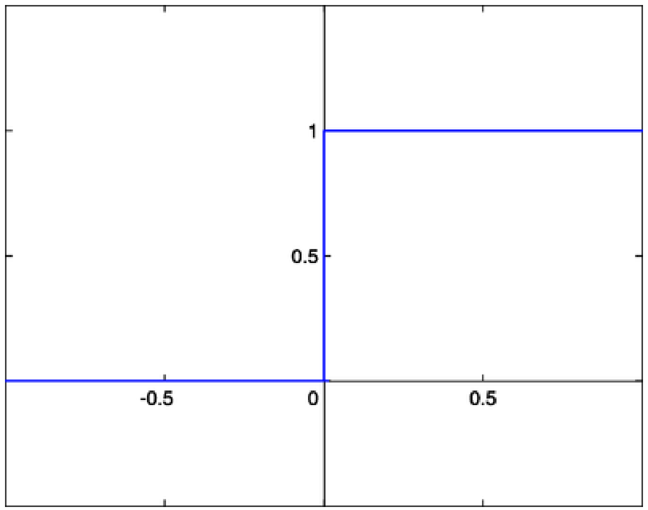
$$d/dt\ \sigma(t) = \sigma(t)(1-\sigma(t))$$

2024 – Marco Mamei, Full Professor (marco.mamei@unimore.it)

Logistic Regression

It is a sort of **smooth version** of the perceptron which exploits the **sigmoid** function in place of the **hard** threshold





Logistic Regression

In order to estimate the β coefficients, a typical approach is to **maximize** a so-called **likelihood function β** (note that the negative of the likelihood can be interpreted as as loss; mazimize likelihood = minimize loss)

$$\ell(\beta, \mathcal{D}) = \sum_{(x_i, y_i) \in \mathcal{D}} y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i))$$

Basic idea: **fit** the β parameters so that the **output** of the logistic function is close to 1 for positive examples, and close to 0 for negative examples

Logistic Regression

Interpretability of the results

A **high-valued** parameter β means that the corresponding feature, if positive, is a strong indicator for **positive** class

A parameter β **close to zero** means that the corresponding feature is **irrelevant** for the classification

A **low-valued** parameter β means that the corresponding feature, if positive, is a strong indicator for **negative** class

Example with Scikit-Learn

Bank customer churn dataset
https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset

This dataset is for ABC Multistate bank with the aim to predict customer churn as a function of client features

Example with Scikit-Learn

Bank customer churn dataset

1. customer_id, unused variable.
2. credit_score, used as input.
3. country, used as input.
4. gender, used as input.
5. age, used as input.
6. tenure, used as input.
7. balance, used as input.
8. products_number, used as input.
9. credit_card, used as input.
10. active_member, used as input.
11. estimated_salary, used as input.
12. churn, used as the target. 1 if the client has left the bank during some period or 0 if he/she has not

Example with Scikit-Learn

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

df = pd.read_csv("bank_customer_churn.csv", index_col="customer_id")

one_hot_country = pd.get_dummies(df.country, prefix='country')
one_hot_gender = pd.get_dummies(df.gender, prefix='gender')
df = df.drop(["country","gender"],axis=1)
df = pd.concat([df, one_hot_country, one_hot_gender], axis=1)

y = df["churn"]
X = df.drop("churn", axis=1)
```

TRANSFORM WITH ONE-HOT ENCODING

SEPARATE FEATURES AND TARGETS

Example with Scikit-Learn

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

clf = LogisticRegression(class_weight='balanced')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

print(clf.coef_)
print(X.columns)

plt.hist(df[df["churn"] == 0]["age"], density=True, histtype='step', bins=20)
plt.hist(df[df["churn"] == 1]["age"], density=True, histtype='step', bins=20)
plt.show()
```

NORMALIZE FEATURES IN [0,1] INTERVAL

BALANCE CLASSES

INVESTIGATE THE IMPACT OF AGE

Example with Scikit-Learn

```
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier

clf = Perceptron(class_weight='balanced')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

GENERALIZATION OF STOCHASTIC GRADIENT DESCENT

TEST THE PERCEPTRON