# CSI5180. Machine Learning for Bioinformatics Applications

**Regularized Linear** Models

by

Marcel Turcotte

Version **November 6, 2019**

# Preamble

# Preamble

**Regularized Linear Models**

In this lecture, we introduce the concept of regularization. We consider the specific context of linear models: Ridge Regression, Lasso Regression, and Elastic Net. Finally, we discuss a simple technique called early stopping.

**General objective :**

- **Explain** the concept of regularization in the context of linear regression and logistic

# Learning objectives

- **Explain** the concept of regularization in the context of linear regression and logistic

**Reading:**

- Simon Dirmeier, Christiane Fuchs, Nikola S Mueller, and Fabian J Theis, netReg: network-regularized linear models for biological association studies, **Bioinformatics 34** (2018), no. 5, 896898.

# Plan

1. Preamble
2. Introduction
3. Polynomial Regression
4. Regularization
5. Logistic Regression
6. Prologue

# Introduction

# Supervised learning

- The **data set** is a collection of **labelled** examples.
  - $\{(x_i, y_i)\}_{i=1}^N$
    - Each $x_i$ is a **feature vector** with $D$ dimensions.
    - $x_k^{(j)}$ is the value of the **feature** $j$ of the example $k$, for $j \in 1 \dots D$ and $k \in 1 \dots N$.
  - The **label** $y_i$ is either a class, taken from a finite list of classes, $\{1, 2, \dots, C\}$, or a **real number**, or a more complex object (vector, matrix, tree, graph, etc).
- **Problem**: given the data set as input, create a "**model**" that can be used to predict the value of $y$ for an unseen $x$.
  - **Classification**: $y_i \in \{\mathrm{Positive}, \mathrm{Negative}\}$, a binary classification problem.
  - **Regression**: $y_i$ is a real number.

# Linear Regression

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

# Linear Regression

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- Here, $\theta_j$ is the $j$the parameter of the (linear) **model**, with $\theta_0$ being the **bias** term/parameter, $\theta_1 \ldots \theta_D$ being the **feature weights**.

# Linear Regression

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- Here, $\theta_j$ is the $j$the parameter of the (linear) **model**, with $\theta_0$ being the **bias** term/parameter, $\theta_1 \ldots \theta_D$ being the **feature weights**.
- **Problem:** find values for all the model parameters so that the model **"best fit"** the training data.

# Linear Regression

- A **linear model** assumes that the value of the label, $\hat{y}_i$, can be expressed as a **linear combination** of the feature values, $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- Here, $\theta_j$ is the $j$the parameter of the (linear) **model**, with $\theta_0$ being the **bias** term/parameter, $\theta_1 \ldots \theta_D$ being the **feature weights**.
- **Problem:** find values for all the model parameters so that the model **"best fit"** the training data.
  - The **Root Mean Square Error** is a common performance measure for regression problems.

$$\sqrt{\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2}$$

# Polynomial Regression

# Polynomial Regression

- **What if** the data is more complex?

# Polynomial Regression

- **What if** the data is more complex?
- In our discussion on **underfitting** and **overfitting** the training data, we did look at polynomial models, but did not discuss how to learn them.

# Polynomial Regression

- **What if** the data is more complex?
- In our discussion on **underfitting** and **overfitting** the training data, we did look at polynomial models, but did not discuss how to learn them.
- Can we use our **linear model** to “fit” **non linear data**, and specifically data would have been generated by a polynomial “process”?

# Polynomial Regression

- **What if** the data is more complex?
- In our discussion on **underfitting** and **overfitting** the training data, we did look at polynomial models, but did not discuss how to learn them.
- Can we use our **linear model** to "fit" **non linear data**, and specifically data would have been generated by a polynomial "process"?
  - **How?**

# sklearn.preprocessing.PolynomialFeatures

- A surprisingly simple solution consists of generating **new features** that are **powers** of existing ones!

# sklearn.preprocessing.PolynomialFeatures

- A surprisingly simple solution consists of generating **new features** that are **powers** of existing ones!

# sklearn.preprocessing.PolynomialFeatures

- A surprisingly simple solution consists of generating **new features** that are **powers** of existing ones!

```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, include_bias=False)

X_poly = poly_features.fit_transform(X)
```

# sklearn.preprocessing.PolynomialFeatures

- A surprisingly simple solution consists of generating **new features** that are **powers** of existing ones!

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
print(lin_reg.intercept_, lin_reg.coef_)
```

# Example fitting a linear model

```
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()

lin_reg.fit(X, y)

lin_reg.intercept_, lin_reg.coef_

# [4.07916603] [[2.90173949]]
```

- $y = 4 + 3x + \text{noise}$
- $\hat{y} = 4.07916603 + 2.90173949x$

# Example fitting a polynomial model

```python
import numpy as np

X = 6 * np.random.rand(100, 1) - 3
y = 2 + 0.5 * X**2 + X + np.random.randn(100, 1)

from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)

lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_

# [1.701144] [[1.02118676 0.55725864]]
```

- $y = 2.0 + 0.5x^2 + 1.0x + \text{noise}$
- $\hat{y} = 1.701144 + 0.55725864x^2 + 1.02118676x$

# Remarks

- For higher degrees, **PolynomialFeatures** adds all the combination of features.

# Remarks

- For higher degrees, **PolynomialFeatures** adds all the combination of features.
  - Given two features $a$ and $b$, **PolynomialFeatures** generates, $a^2$, $a^3$, $b^2$, $b^3$, but also $ab$, $a^2b$, $ab^2$.

# Remarks

- For higher degrees, **PolynomialFeatures** adds all the combination of features.
  - Given two features $a$ and $b$, **PolynomialFeatures** generates, $a^2$, $a^3$, $b^2$, $b^3$, but also $ab$, $a^2b$, $ab^2$.
- Given $n$ features and degree $d$, **PolynomialFeatures** produces $\frac{(n+d)!}{d!n!}$ combinations!

# Regularization

# Bias/Variance trade-off

From [2] §4:

- "(. . . ) a models **generalization error** can be expressed as the sum of three very different errors:"

# Bias/Variance trade-off

From [2] §4:

- "(. . . ) a models **generalization error** can be expressed as the sum of three very different errors:"
  - **Bias:** "is due to **wrong assumptions**", "A **high-bias** model is most likely to **underfit** the training data"

# Bias/Variance trade-off

From [2] §4:

- "(. . . ) a models **generalization error** can be expressed as the sum of three very different errors:"
  - **Bias:** "is due to **wrong assumptions**", "A **high-bias** model is most likely to **underfit** the training data"
  - **Variance:** "the model's excessive sensitivity to small variations in the training data". A model with **many parameters** "is likely to have **high variance** and thus **overfit** the training data."

# Bias/Variance trade-off

From [2] §4:

- "(. . . ) a models **generalization error** can be expressed as the sum of three very different errors:"
    - **Bias:** "is due to **wrong assumptions**", "A **high-bias** model is most likely to **underfit** the training data"
    - **Variance:** "the model's excessive sensitivity to small variations in the training data". A model with **many parameters** "is likely to have **high variance** and thus **overfit** the training data."
    - **Irreducible error:** "noisiness of the data itself"

# Bias/Variance trade-off

From [2] §4:

- "(...) a models **generalization error** can be expressed as the sum of three very different errors:"
  - **Bias:** "is due to **wrong assumptions**", "A **high-bias** model is most likely to **underfit** the training data"
  - **Variance:** "the model's excessive sensitivity to small variations in the training data". A model with **many parameters** "is likely to have **high variance** and thus **overfit** the training data."
  - **Irreducible error:** "noisiness of the data itself"
- "**Increasing a models complexity** will typically **increase its variance** and **reduce its bias**. Conversely, **reducing a models complexity increases its bias** and **reduces its variance**."

# Overfitting and underfitting

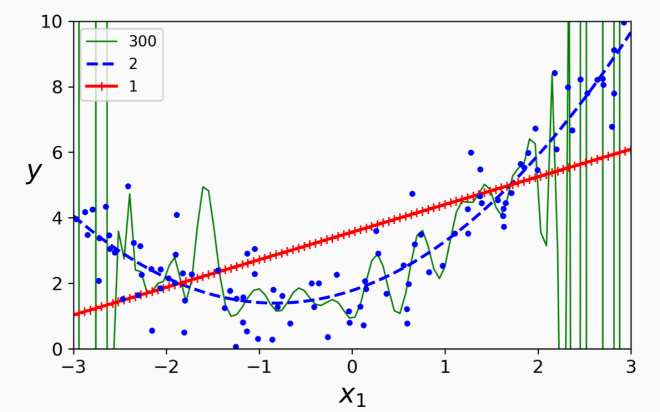



*Figure 4-14. High-degree Polynomial Regression*

**Source:** Géron 2019

# Linear model - underfitting

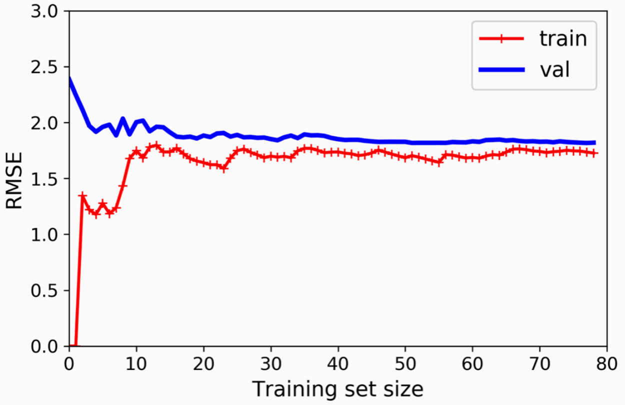



*Figure 4-15. Learning curves*

**Source:** Géron 2019

# Polynomial of degree 10 - overfitting

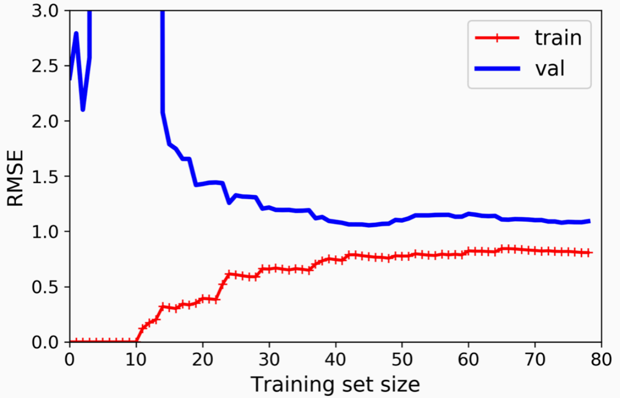



*Figure 4-16. Learning curves for the 10th-degree polynomial model*

**Source:** Géron 2019

# Regularization

- "Constraining a model to make it simpler and reduce the risk of overfitting is **called regularization**." [2]

# Regularization

- “Constraining a model to make it simpler and reduce the risk of overfitting is **called regularization**.” [2]
- One way to **regularized** a **polynomial model** is to restrict its degree.

# Regularization

- "Constraining a model to make it simpler and reduce the risk of overfitting is **called regularization**." [2]
- One way to **regularized** a **polynomial model** is to restrict its degree.
  - **How** would you do that?

# Regularization

- "Constraining a model to make it simpler and reduce the risk of overfitting is **called regularization**." [2]
- One way to **regularized** a **polynomial model** is to restrict its degree.
  - **How** would you do that?
    - Make the degree a **hyperpamater**, use a **holding set** or **cross-validation**.

# Regularization

- "Constraining a model to make it simpler and reduce the risk of overfitting is **called regularization**." [2]
- One way to **regularized** a **polynomial model** is to restrict its degree.
  - **How** would you do that?
    - Make the degree a **hyperpamater**, use a **holding set** or **cross-validation**.
- **Alternatively**, we can constraint the **weights** of the model.

# Norm

- A **norm** is a function that assigns a number (length, size) to a vector.
- $\ell_p$-norm

$$\ell_p\text{-norm} = ||\theta||_p = (\sum_{j=1}^{D} |\theta^{(j)}|^p)^{\frac{1}{p}}$$

- $\ell_1$-norm

$$\ell_l\text{-norm} = ||\theta||_1 = \sum_{j=1}^{D} |\theta^{(j)}|$$

- $\ell_2$-norm

$$\ell_2\text{-norm} = ||\theta||_2 = \sqrt{\sum_{j=1}^{D} |\theta^{(j)}|^2}$$

# Ridge Regression

- You will remember the objective function, **Mean Squared Error** (**MSE**), used by our gradient descent.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

# Ridge Regression

- You will remember the objective function, **Mean Squared Error** (**MSE**), used by our gradient descent.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

- In the case Ridge Regression, the objective function becomes:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \frac{1}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$

# Ridge Regression

- You will remember the objective function, **Mean Squared Error** (**MSE**), used by our gradient descent.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

- In the case Ridge Regression, the objective function becomes:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \frac{1}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$

- The regularization is applying at learning time only.

# Ridge Regression

- You will remember the objective function, **Mean Squared Error** (**MSE**), used by our gradient descent.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

- In the case Ridge Regression, the objective function becomes:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \frac{1}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$

- The regularization is applying at learning time only.
- $\alpha$ is a hyperparameter, with $\alpha = 0$, Ridge Regression is equivalent to a Linear Regression.

# Ridge Regression

- You will remember the objective function, **Mean Squared Error** (**MSE**), used by our gradient descent.

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2$$

- In the case Ridge Regression, the objective function becomes:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \frac{1}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$

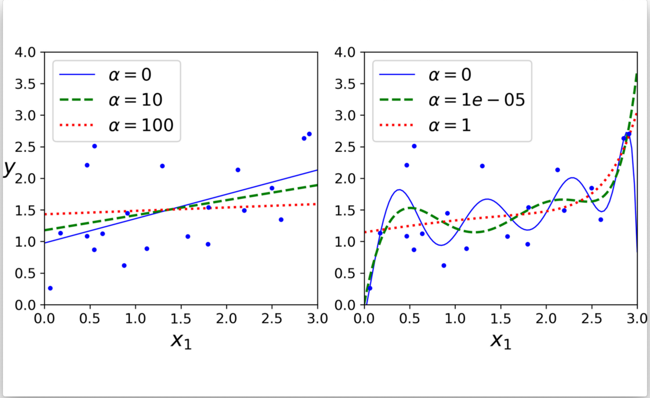
- The regularization is applying at learning time only.
- $\alpha$ is a hyperparameter, with $\alpha = 0$, Ridge Regression is equivalent to a Linear Regression.
- $\frac{1}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$ is the $\ell_2$-norm of the weight vector.

# sklearn.linear_model.Ridge

```python
from sklearn.linear_model import Ridge

ridge_reg = Ridge(alpha=1, solver="cholesky")
ridge_reg.fit(X, y)
```

# Ridge Regression




**Source:** [2] Figure 4.17

# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.

# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.
- Its objective function is:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \alpha\sum_{1}^{D}\theta^{(j)}$$

# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.
- Its objective function is:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \alpha \sum_{1}^{D} \theta^{(j)}$$

- The regularization is applying at learning time only.

# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.
- Its objective function is:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \alpha\sum_{1}^{D}\theta^{(j)}$$

- The regularization is applying at learning time only.
- $\alpha$ is a hyperparameter, with $\alpha = 0$, Lasso Regression is equivalent to a Linear Regression.

# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.
- Its objective function is:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i)-y_i]^2+\alpha\sum_{1}^{D}\theta^{(j)}$$

- The regularization is applying at learning time only.
- $\alpha$ is a hyperparameter, with $\alpha = 0$, Lasso Regression is equivalent to a Linear Regression.
- $\alpha\sum_{1}^{D}\theta^{(j)}$ is the $\ell_1$-norm of the weight vector.

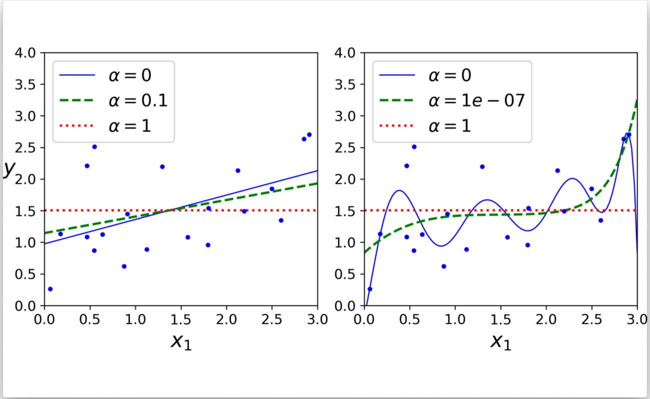
# Lasso Regression

- Another popular regularization is the **Least Absolute Shrinkage and Selection Operator Regression**, Lasso Regression.
- Its objective function is:

$$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + \alpha\sum_{1}^{D}\theta^{(j)}$$

- The regularization is applying at learning time only.
- $\alpha$ is a hyperparameter, with $\alpha = 0$, Lasso Regression is equivalent to a Linear Regression.
- $\alpha\sum_{1}^{D}\theta^{(j)}$ is the $\ell_1$-norm of the weight vector.
- Lasso regression favors sparse models (models with few terms with non-zero weights)

# Lasso Regression




**Source:** [2] Figure 4.18

# Ridge and Lasso regression

- “Your role as the data analyst is to find such a value of the hyperparameter [$\alpha$] that **doesn’t increase the bias too much** but **reduces the variance** to a level reasonable for the problem at hand.” [3]
- In practice, $\ell_1$-norm (Lasso) produces models that are **sparse**. Thus acting as a **feature selection** mechanism.
- However, $\ell_2$-norm (Ridge) usually gives better results in practice.
- These norms are frequently used with other models/objective functions.

# Elastic Net

- **Elastic Net** is a mixture of Ridge Regression and Lasso Regression.

# Elastic Net

- **Elastic Net** is a mixture of Ridge Regression and Lasso Regression.
- $$\frac{1}{N}\sum_{1}^{N}[h(x_i) - y_i]^2 + r\alpha\sum_{1}^{D}\theta^{(j)} + \frac{1-r}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$

# Elastic Net

- **Elastic Net** is a mixture of Ridge Regression and Lasso Regression.
- $$\frac{1}{N}\sum_{1}^{N}[h(x_i)-y_i]^2 + r\alpha\sum_{1}^{D}\theta^{(j)} + \frac{1-r}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$
- It adds a second hyperparameter $r$, to control ratio of $\ell_2$ and $\ell_1$ regularization.

# Elastic Net

- **Elastic Net** is a mixture of Ridge Regression and Lasso Regression.
- $$\frac{1}{N}\sum_{1}^{N}[h(x_i)-y_i]^2 + r\alpha\sum_{1}^{D}\theta^{(j)} + \frac{1-r}{2}\alpha\sum_{1}^{D}\theta^{(j)2}$$
- It adds a second hyperparameter $r$, to control ratio of $\ell_2$ and $\ell_1$ regularization.
- In all three cases, the summation starts at 1, i.e. the bias term (here, the intercept) is excluded from the regularization.

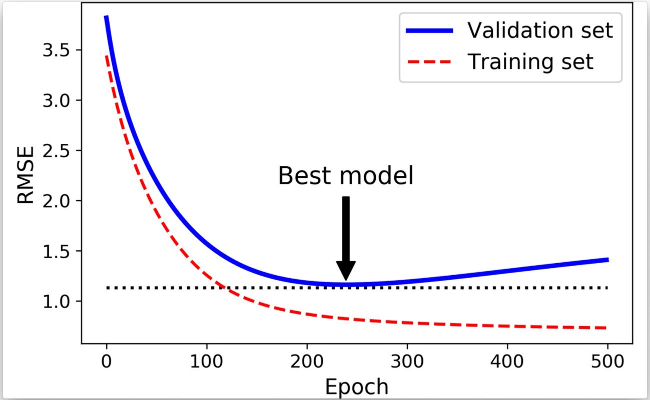
# sklearn.linear_model.ElasticNet

```python
from sklearn.linear_model import ElasticNet

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(X, y)
```

**Source:** [2] §4

# Early stopping




**Geoffrey Hinton** called this the "beautiful free lunch"

**Source:** [2] Figure 4.20

# Remarks

- The criteria used to drive the **optimization** (training) can be different than the criteria used for the **hyper parameter** selection procedure.
- Regularized models are known to be sensitive to the scale of features, thus the data should be "normalized".
- "(. . . ) the **fewer degrees of freedom** it has, the **harder it will be for it to overfit the data**."

# Logistic Regression

# Logistic (Logit) Regression

- Despite its name, **Logistic Regression** is a **classification** algorithm.

# Logistic (Logit) Regression

- Despite its name, **Logistic Regression** is a **classification** algorithm.
- The **labels** are binary values, $y_i \in \{0, 1\}$.

# Logistic (Logit) Regression

- Despite its name, **Logistic Regression** is a **classification** algorithm.
- The **labels** are binary values, $y_i \in \{0, 1\}$.
- It is formulated to answer the question, **"what is the probability that $x_i$ is a positive example, i.e. $y_i = 1$?"**

# Logistic (Logit) Regression

- Despite its name, **Logistic Regression** is a **classification** algorithm.
- The **labels** are binary values, $y_i \in \{0, 1\}$.
- It is formulated to answer the question, **"what is the probability that $x_i$ is a positive example, i.e. $y_i = 1$?"**
- Just like the **Linear Regression**, the **Logistic Regression** computes a weighted sum of the input features:

$$\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

# Logistic (Logit) Regression

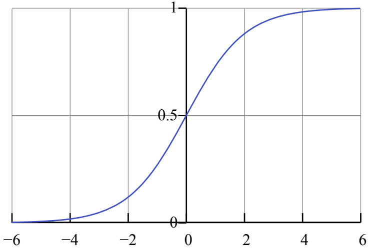
- Despite its name, **Logistic Regression** is a **classification** algorithm.
- The **labels** are binary values, $y_i \in \{0, 1\}$.
- It is formulated to answer the question, **“what is the probability that $x_i$ is a positive example, i.e. $y_i = 1$?”**
- Just like the **Linear Regression**, the **Logistic Regression** computes a weighted sum of the input features:

$$\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \ldots + \theta_D x_i^{(D)}$$

- The image of this function is $-\infty$ to $\infty$!

# Logistic Regression

- In mathematics, a **standard logistic function** maps a real value ($\mathbb{R}$) to the interval $(0, 1)$:




**Source:** Wikipedia

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

# Logistic Regression

- The **Logistic Regression** model, in its vectorized form is:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

# Logistic Regression

- The **Logistic Regression** model, in its vectorized form is:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

- **Predictions** are made as follows:

# Logistic Regression

- The **Logistic Regression** model, in its vectorized form is:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

- **Predictions** are made as follows:
  - $y_i = 0$, if $h_\theta(x_i) < 0.5$

# Logistic Regression

- The **Logistic Regression** model, in its vectorized form is:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

- **Predictions** are made as follows:
  - $y_i = 0$, if $h_\theta(x_i) < 0.5$
  - $y_i = 1$, if $h_\theta(x_i) \geq 0.5$

# Logistic Regression

- The **Logistic Regression** model, in its vectorized form is:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1 + e^{-\theta x_i}}$$

- **Predictions** are made as follows:
  - $y_i = 0$, if $h_\theta(x_i) < 0.5$
  - $y_i = 1$, if $h_\theta(x_i) \geq 0.5$
- The values of $\theta$ are learnt using **gradient descent**.

# 2020

- Include the derivation of the loss (objective) function.

# sklearn.linear_model.LogisticRegression

```python
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X, y)

# ...

y_proba = log_reg.predict_proba(X_new)
```

# Prologue

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.
  - **Ridge**: $\ell_2$-norm term is added to the objective function.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.
  - **Ridge**: $\ell_2$-norm term is added to the objective function.
  - **Lasso**: $\ell_1$-norm term is added to the objective function.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.
  - **Ridge**: $\ell_2$-norm term is added to the objective function.
  - **Lasso**: $\ell_1$-norm term is added to the objective function.
  - **Elastic Net**: both, $\ell_2$ and $\ell_1$-norm terms are added to the objective function.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.
  - **Ridge**: $\ell_2$-norm term is added to the objective function.
  - **Lasso**: $\ell_1$-norm term is added to the objective function.
  - **Elastic Net**: both, $\ell_2$ and $\ell_1$-norm terms are added to the objective function.
- **Early stopping** criteria is an effective and fairly general regularization, it can be applied iterative learning algorithms, such as batch gradient.

# Summary

- **Regularization** is the idea to constrain a model making it simpler, thus less prone to overfitting.
- Limiting the **complexity of the model** is one way to add regularization.
  - Limiting the degree of the polynomial in case of a polynomial model.
- Often, penalty terms are added to the objective (cost) function.
  - **Ridge**: $\ell_2$-norm term is added to the objective function.
  - **Lasso**: $\ell_1$-norm term is added to the objective function.
  - **Elastic Net**: both, $\ell_2$ and $\ell_1$-norm terms are added to the objective function.
- **Early stopping** criteria is an effective and fairly general regularization, it can be applied iterative learning algorithms, such as batch gradient.
- Contrary to **Principal Component Analysis**, the above techniques are of their impact on the performance of the learning algorithms (o the validation set).

# Next module

- Models related to **decision trees**

# References


Simon Dirmeier, Christiane Fuchs, Nikola S Mueller, and Fabian J Theis.
netReg: network-regularized linear models for biological association studies.
*Bioinformatics*, 34(5):896–898, 03 2018.

Aurélien Géron.
*Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*.
O'Reilly Media, 2nd edition, 2019.

Andriy Burkov.
*The Hundred-Page Machine Learning Book*.
Andriy Burkov, 2019.

Marcel **Turcotte**
Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
**University of Ottawa**