Machine Learning for the Social Sciences:
Preliminaries and Algorithms

Gian Maria Campedelli, PhD
gianmaria.campedelli@unitn.it
Department of Sociology and Social Research - University of Trento

November 17, 2020

Contents

Quick Recap

In the first lecture we covered:

- *Historical cross-disciplinary origins of AI and ML*
- *Difference between Supervised and Unsupervised Learning*
- *Difference between Classificationn and Regression tasks in Supervised Learning*
- *Main areas of application of Unsupervised Learning*
- *Differences between Statistics and ML*

Resources

For those of you who are particularly interested in the mathematical backbone and functioning of ML, DL and learning algorithms in general, I warmly suggest:

- *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Hastie, Tibshirani, and Friedman 2013)
- *Pattern Recognition and Machine Learning* (Bishop 2006)
- *Deep Learning* (Goodfellow, Bengio, and Courville 2017)
- *An Introduction to Statistical Learning* (James et al. 2013)
- *Mathematics for Machine Learning* (Deisenroth 2020)

First Technical Concepts

What are we going to cover?

- Train and Test
- Generalizability
- Performance Evaluation

Learning from Data

OLS/logistic regression from a statistics standpoint $\rightarrow$ we use the entire dataset to estimate our θ

ML does not work this way $\Rightarrow$ assuming a dataset with $i.i.d.$ observations, we split it in 2 sets:

1. **Train Set**: set of examples (e.g., pairs $(\mathbf{x_i}, y_i)$) used to the train the algorithm $\Rightarrow$ these inputs are used to fit the parameters through optimization methods (e.g., SGD)
2. **Test Set**: the independent set of unseen examples that is used to evaluate the performance of the algorithm on the task, based on the learned weights in the training phase

+ sometimes we have an intermediate set, called **validation set**, in which hyperparameters are tuned to further optimize the performance of the algorithm

Example on a Regression task

Let's pretend we have a dataset containing 150,000 obs measuring individual school performance (y), in the range 0-100. To train a ML regression model we need to:

1. Split the dataset in train (80%?) and test (20%?) sets.
2. Fit the model on train set, evaluate error, through e.g., $\text{MSE}_{train} = \frac{1}{m} \sum_i (\hat{\mathbf{y}}^{\text{train}} - \mathbf{y}^{\text{train}})_i^2$
3. Fit the model on the test set, evaluate its error and performance ($\text{MSE}_{test} = \frac{1}{m} \sum_i (\hat{\mathbf{y}}^{\text{test}} - \mathbf{y}^{\text{test}})_i^2$)

What is the performance of our model in predicting individual school performance scores? We answer to this question by looking at the **test** set

Small Datasets: K-Fold Cross Validation

Small datasets (e.g., 100 observations): k**-fold cross validation** is good strategy to evaluate and validate model performance.
Procedure:

1. Shuffle the observations randomly (we can $\rightarrow$ *i.i.d.*)
2. Split dataset into k independent sets
3. Per each group:
   - Use it as test set
   - Use the remaining $k-1$ groups as training sets
   - Fit a model
4. Assess overall model through evaluation of each sub-model

Iteration	Train on	Test on	RMSE (Test)
1	*S1, S2, S3, S4*	S5	0.188
2	*S1, S2,S3,S5*	S4	0.211
3	*S1, S2, S4, S5*	S3	0.195
4	*S1, S3, S4, S5*	S2	0.299
5	*S2, S3, S4, S5*	S1	0.201

Overall RMSE= 0.204

Generalizability

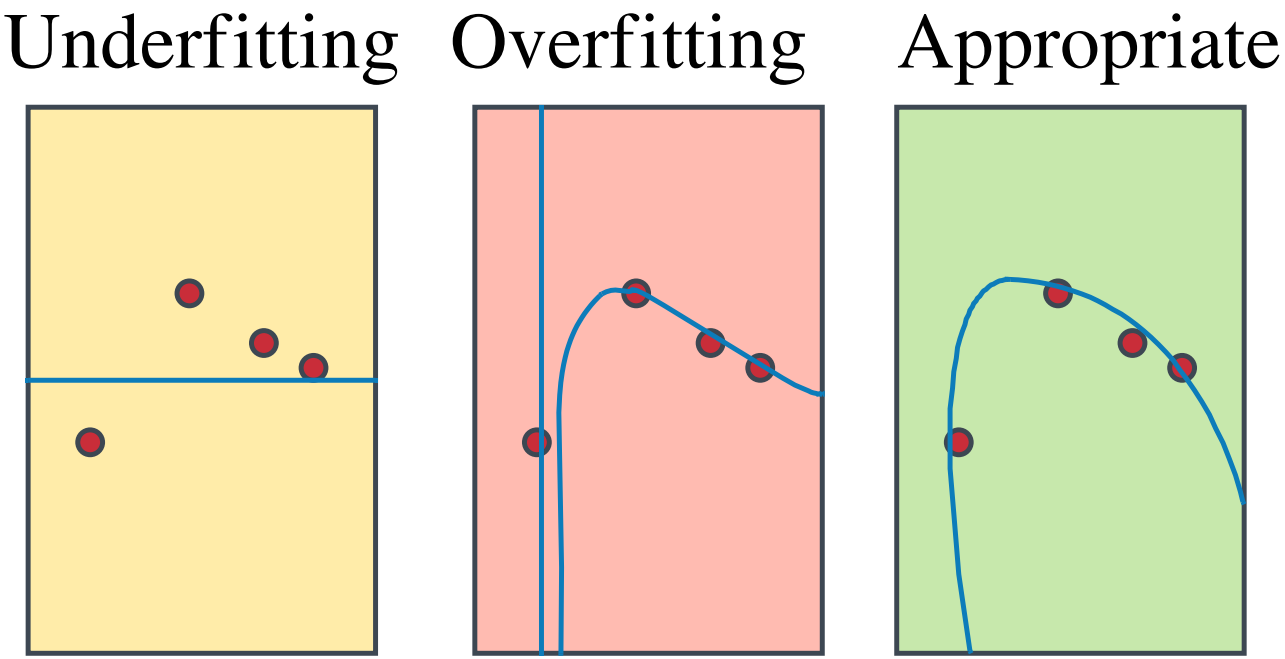
The most important aim of a ML model is **generalizability**.

The chosen algorithm must able to learn patterns that can be useful to *capture the dynamics*, *relationships* in the test set.
To achieve this, one's goals should be to:

1. **Minimize the training error** as much as possible
2. **Reduce the gap between training and test error** as much as possible

$\Rightarrow$ this brings us to two core challenges of ML: **underfitting and overfitting**

Visualizing under- and overfitting



Defining under- and overfitting

- **Underfitting:** generally occurs when the model is not able to reach a sufficiently low error in the training set
- **Overfitting:** the gap between the train and test error is particularly large

Defining under- and overfitting

- **Underfitting:** generally occurs when the model is not able to reach a sufficiently low error in the training set
- **Overfitting:** the gap between the train and test error is particularly large

A complementary explanation is the following, given an hypothesis class $\mathcal{H}$:

Defining under- and overfitting

- **Underfitting:** generally occurs when the model is not able to reach a sufficiently low error in the training set
- **Overfitting:** the gap between the train and test error is particularly large

A complementary explanation is the following, given an hypothesis class $\mathcal{H}$:

- **Underfitting:** $\mathcal{H}$ is less complex than the underlying function(s) governing the data (e.g., fitting a line to data generated from a cubic polynomial)
- **Overfitting:** $\mathcal{H}$ is more complex than the underlying function(s) governing the data (e.g., fitting a cubic polynomial to data generated from a line)

How to prevent these two phenomena from occurring? $\Rightarrow$ assess **model capacity**

ML/DL: Lost in Space?

Traditional statistical approaches are often used *off-the-shelf* in the social sciences ⇒

```
rreg income nationality education crimhistory
```

Look for the **significance stars**, rule out some predictors, try others based on previous multicollinearity tests.

ML is often extremely different, especially for complex models ⇒ *many decisions to make, hyperparameters to tune*

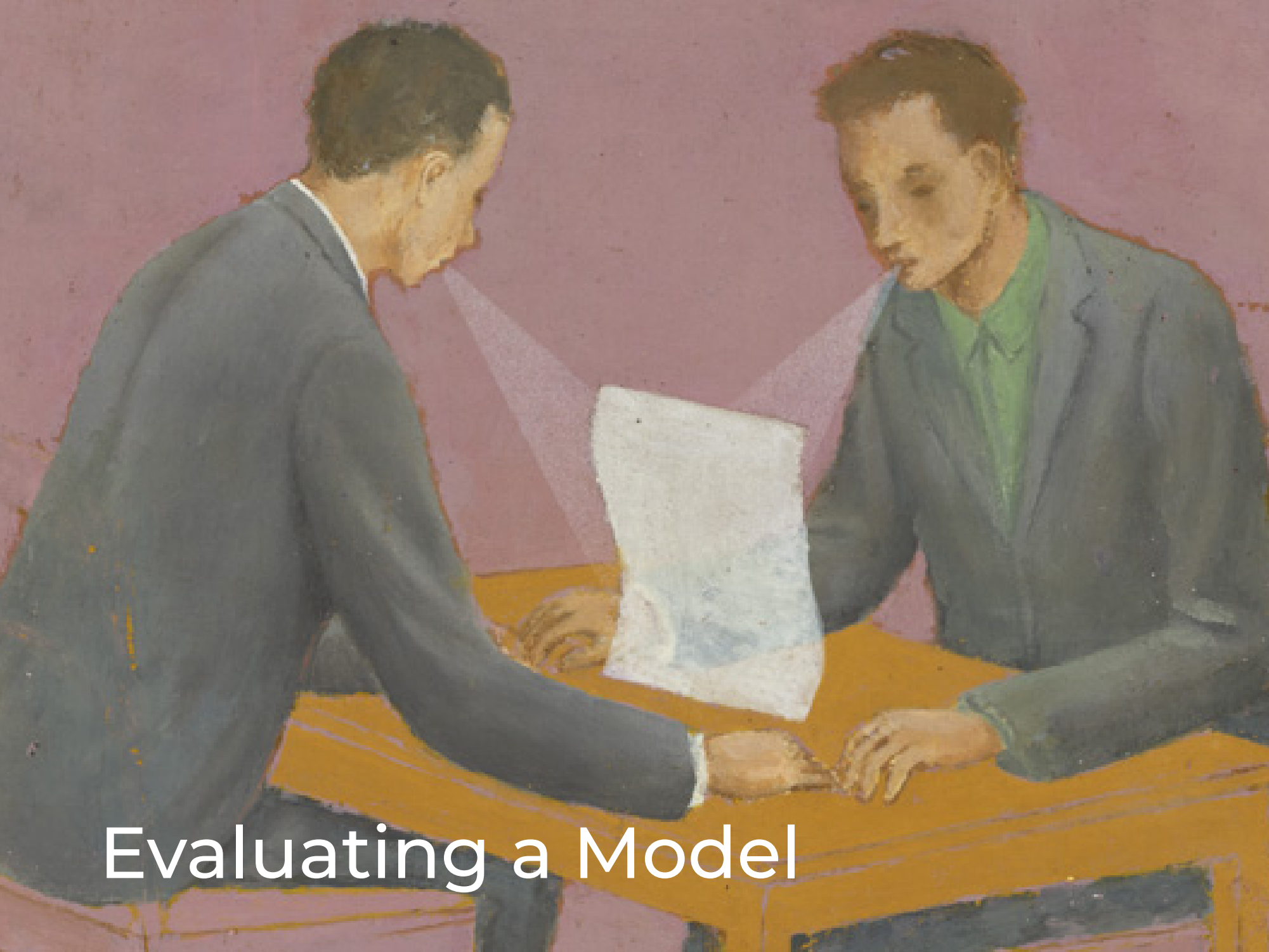
Don’t Fall in the Hype Trap

R and Python enable you to run a ML model in two lines of code. However, employing ML meaningfully is **no easy task**.

Don’t Fall in the Hype Trap

R and Python enable you to run a ML model in two lines of code. However, employing ML meaningfully is **no easy task**.

- *Know your data*
- *Ground your approach in theory/motivated hypotheses*
- *Dig into the technical aspects of the selected approach*
- *Engineer your feature space*
- *Test, evaluate, experiment*



Evaluating a Model

Evaluating: Another Dichotomy

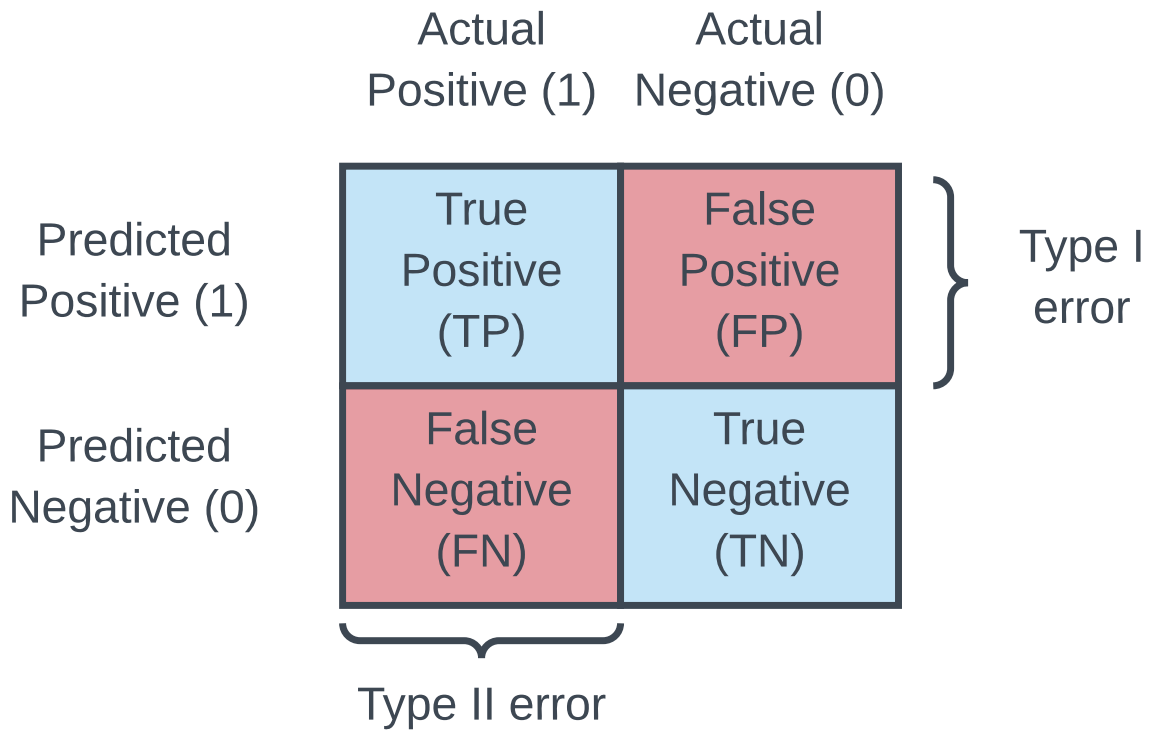
Evaluating a model is a decisive and *complex* task (it requires domain expertise, besides technical knowledge).

We need to distinguish between evaluation metrics for

1. *Classification*
2. *Regression*

Classification: Confusion Matrix

Assume (for the sake of simplicity) to have a binary classification problem.



Classification: Confusion Matrix

The Taxonomy of a Confusion Matrix:

- **True Positive (TP):** the subject i has a disease and the algorithm correctly predicts this condition
- **True Negative (TN):** the subject i does not have a disease and the algorithm correctly predicts this condition
- **False Positive (FP):** the subject i does not have a disease but the algorithm predict that he/she has a disease
- **False Negative (FN):** the subject i has a disease but the algorithm predict that he/she has not a disease

The relative weight of FP and FN is tightly connected with the domain of application $\Rightarrow$ in this scenario a high FN is an extremely bad result

Classification: Accuracy

Accuracy is the **most common** evaluation metric.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

It can easily lead to wrong interpretations $\Rightarrow$ especially in the case of an imbalanced dataset (*if 95% of the instance are negative, can we really say that 96% accuracy is a good result?*)

Classification: Precision/PPV

The **Precision** score (also called **Positive Predictive Value)** indicate show many times the predicted positive prediction was actually correct

In other words, it gives as a measure of the ability of a classifier not to label as positive a sample that is negative

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Classification: Recall/Sensitivity

Recall (also called **Sensitivity**) quantifies the ability of the algorithm to find all the positive samples

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

This metric is extremely important, for instance, in cases where a disease has to be detected by a machine

Classification: F1-score

Sometimes, we may be interested in combining together **Recall** and **Precision** to evaluate our model performance.

F1-score is generally the solution. F1-score is the harmonic mean of the precision and recall.

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

This measure however calls for an important caveat: *it does not consider TN in its computation* $\Rightarrow$ misleading in unbalanced datasets.

Furthermore, it has been criticized in the literature because it gives equal weight to both *precision and recall*

Classification: Cross-entropy Loss

When working with algorithms that assign probabilities to each class for all samples (e.g., *logistic regression*), one suggested solution is to use **Cross-entropy Loss** (also called **Log-loss**).

Cross-entropy increases when the predicted probability diverges from the actual data. Algorithmically:

```
def CrossEntropy(yHat, y):
    if y == 1 :
        return -log(yHat)
    else:
        return -log(1 - yHat)
```

Classification: Cross-entropy Loss

When working with algorithms that assign probabilities to each class for all samples (e.g., *logistic regression*), one suggested solution is to use **Cross-entropy Loss** (also called **Log-loss**).

Cross-entropy increases when the predicted probability diverges from the actual data. Algorithmically:

```
def CrossEntropy(yHat, y):
    if y == 1 :
        return -log(yHat)
    else:
        return -log(1 - yHat)
```

The equation for the full sample of N instances is:

$$\text{Cross Entropy} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log p_i + (1-y_i) log(1-p_i)] \qquad (5)$$

Regression: Beyond R^2

From a classical statistical standpoint, regression models are usually evaluated using R^2 or similar measures (e.g., Pseudo R^2) $\Rightarrow$ *How much of the variance in the data is explained by the model?*

In ML, other metrics are instead generally used to assess the extent to which our algorithm gets closer to y. Among these are:

1. **Mean Absolute Error**
2. **Mean Squared Error**

Regression: Mean Absolute Error

MAE is simply the average of the absolute differences between predicted and actual values $\Rightarrow$ it quantifies the error without capturing the direction of the error

$$\mathrm{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{6}$$

MAE is thus robust to outliers

Regression: MSE and RMSE

In MSE we take the square of the absolute errors:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{7}$$

By taking the square of the error, we **penalize larger errors more**

A similar measure is **Root Mean Squared Error** (RMSE) $\Rightarrow$ *how spread out are the errors?*

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{8}$$

Algorithms

Initial Caveat

In this section of the course we are going to cover some of the most popular ML algorithms.

Fundamental: be aware that these only represents a **minority** of the entire universe of models that are being developed in the community.

From a theoretical/foundational point of view, ML continuously produces new approaches/methods

Initial Caveat

In this section of the course we are going to cover some of the most popular ML algorithms.

Fundamental: be aware that these only represents a **minority** of the entire universe of models that are being developed in the community.

From a theoretical/foundational point of view, ML continuously produces new approaches/methods

Nonetheless, you are **social scientists** $\Rightarrow$ our fields are *lagging behind*: many of the traditional models that have been around for decades in ML are still considered innovative in *Sociology* and *Economics* journals.

Back to Basics: Logistic Regression/1

Logistic regression is considered one of the standard methods for binary classification problems in ML.

Back to Basics: Logistic Regression/1

Logistic regression is considered one of the standard methods for binary classification problems in ML.

However it is not properly a classification method, but rather a **regression** method that is used for classification when coupled with a decision rule.

Back to Basics: Logistic Regression/1

Logistic regression is considered one of the standard methods for binary classification problems in ML.

However it is not properly a classification method, but rather a **regression** method that is used for classification when coupled with a decision rule.

...What?

Back to Basics: Logistic Regression/2

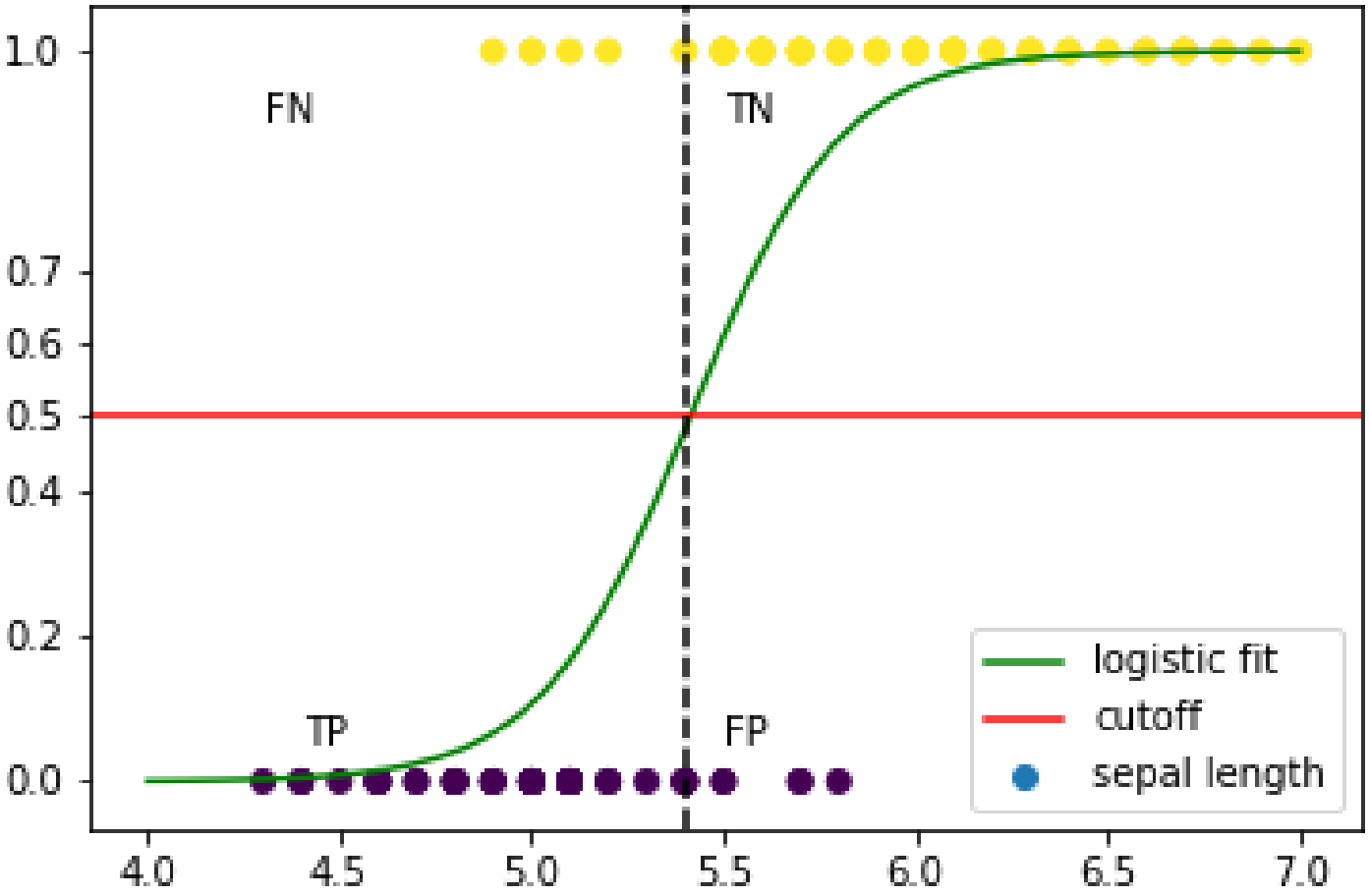
Logistic regression can be seen as a **probabilistic classification model**.

Per each instance, the algorithms assign probabilities to each of the classes $\Rightarrow$ **Decision Boundary:**

$$\begin{aligned} p \geq 0.5,\ class = 1 \\ p < 0.5,\ class = 0 \end{aligned} \tag{9}$$

The task becomes a classification one because we represent our target response in a binary state, while it was rather continuous

Back to Basics: Logistic Regression/3



Back to Basics: Logistic Regression/4

While in Linear Regression: $h(x) = \theta^T x$, in Logistic regression we apply the sigmoid function to the output of the linear regression, then: $h(x) = \sigma(\theta^T x)$ with:

$$\sigma(t) = \frac{1}{1 + e^{-z}} \tag{10}$$

At this point, the hypothesis $h(x)$ actually becomes a probability p due to the equation:

$$p(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{11}$$

The probability then can be reduced to:

$$p(x) = \begin{cases} \geq 0.5 \rightarrow class = 1 & \text{if } \theta^T x \geq 0 \\ < 0.5 \rightarrow class = 0 & \text{if } \theta^T x < 0 \end{cases} \tag{12}$$

Visualizing a Learning Algorithm

Let's visualize how a logistic regression (or a neural network) learns: TensorFlow Playground

Regularization Methods

OLS and Logistic regression may suffer from several problems, including **multicollinearity** (good old *Stats*) and **overfitting/generalizability**.

Regularization methods are useful to deal with such issues: they reduce *model dimensionality*, increase *generalizability* and also *interpretability*.

Regularization Methods

OLS and Logistic regression may suffer from several problems, including **multicollinearity** (good old *Stats*) and **overfitting/generalizability**.

Regularization methods are useful to deal with such issues: they reduce *model dimensionality*, increase *generalizability* and also *interpretability*.

⇒ among regularization methods the most popular are: **Ridge** and **LASSO** regression

Back to Basics 2: Linear Regression

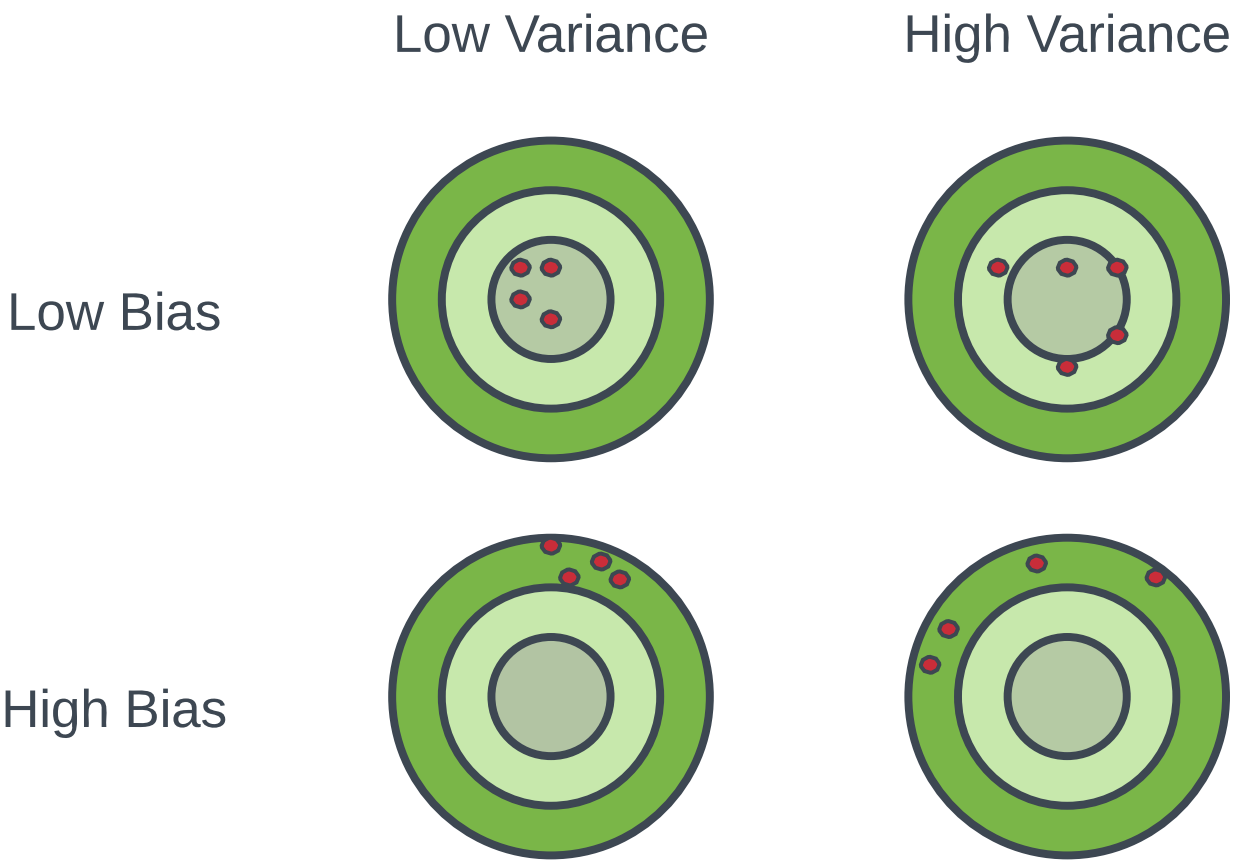
Assume we have a simple linear regression model $h(x) = f(x)$:

$$h(x) = \theta^T x = \theta_0 + \sum_{j=1}^{p} \theta_j x_j \tag{13}$$

$\Rightarrow$ p+2 parameters: p-vector of θ-coefficients, one intercept, one variance parameter (*Gaussian error*). The parameters are generally esimated using OLS:

$$\mathrm{RSS}(\theta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 = \sum_{i=1}^{N} (y_i - \theta_0 - \sum_{j=1}^{p} \theta_j x_{ij})^2 \tag{14}$$

The Bias/Variance Tradeoff



Ridge (L2) Regression

A method to achieve regularization is **Ridge Regression** (also called **L2 Regression**).

$$\mathcal{L}_{Ridge}(\theta) = \sum_{i=1}^{N}(y_i - \theta_0 - \sum_{j=1}^{p}\theta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\theta_j^2 \quad (15)$$

It penalizes coefficients shrinking them towards 0.

We introduce a penalty term λ that allows decreasing model complexity + decreasing variance $\Rightarrow$ we need to find the optimal value to maximize the **bias/variance** trade-off

ISSUES: the model may have low estimate precision + difficult interpretability due to *the high number of predictors* (it does not exclude them!)

Lasso (L1) Regression

Lasso Regression seeks to solve the limitations of **Ridge Regression**.

Unlike Ridge Regression, Lasso:

1. Minimizes the sum of absolute values of θ
2. For high values of λ, certain coefficients are zeroed and thus excluded from the model $\Rightarrow$ better handling of *multicollinearity and interpretability*

The equation of the Loss function in **Lasso regression** is

$$\mathcal{L}_{Lasso}(\theta) = \sum_{i=1}^{N}(y_i - \theta_0 - \sum_{j=1}^{p}\theta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\theta_j| \tag{16}$$

Ridge vs Lasso

There is not a clear and universal answer to the question *"what model should I choose?"*

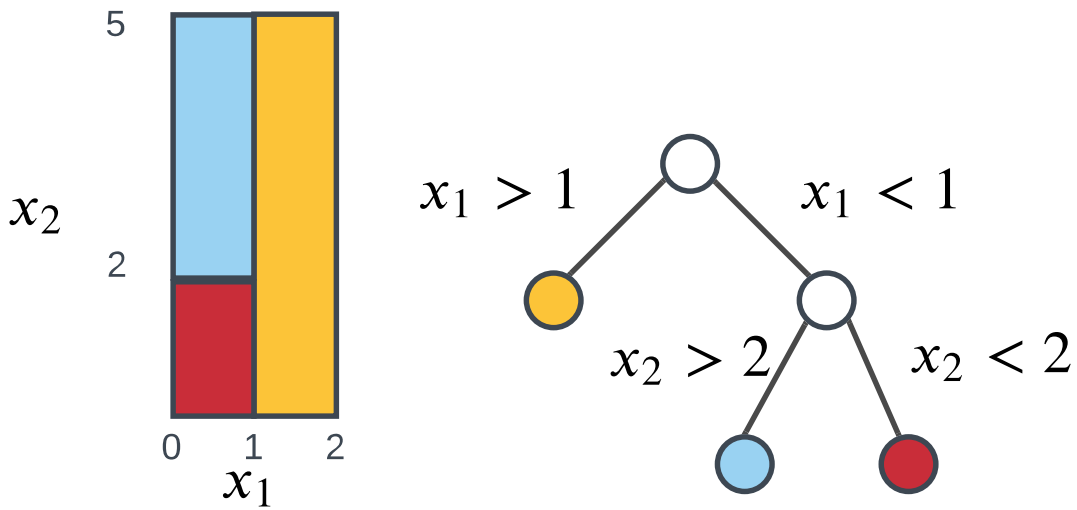
If you can, try to compare them. In general:

- **Ridge regression** likely performs better when many predictors p really impact the target/response/dependent variable
- **Lasso regression**, conversely, generally outperforms Ridge Regression when we have a high number of p but only a small subset really impact Y

Decision Trees

Decision Trees are a non-parametric supervised learning method that can be used for both *regression* and *classification*.

They involve **stratification/segmentation** of the feature space into a number of finite regions



Decision Tree: How to Split?

For classification and regression trees we rely on different criteria to split each node.

- **Classification Tree:** maximize information gain candidate on each possible split is computed via *Entropy*

$$H(S) - \left[\frac{|S_1|}{|S|} H(S_1) + \frac{|S_2|}{|S|} H(S_2) \right]$$

- **Regression Tree**: minimize objective function considering quality of partitions $S = S_1 \cup S_2$ via:

$$\sum_{i \in S_1} (y_i - \frac{1}{|S_1|} \sum_{i \in S_1} y_i)^2 + \sum_{i \in S_2} (y_i - \frac{1}{|S_2|} \sum_{i \in S_2} y_i)^2$$

Pros and Cons of Decision Trees

PROS:

- They are generally good when interpretability is necessary (*White box*)
- Little data preparation
- Computational cost: acceptable

CONS:

- Risk of overfitting/instability
- Low performance compared to other methods

Next Week

- Alternatives to Decision Trees (e.g., *Random Forests*)
- Unsupervised Learning Approaches (*Clustering or Dimensionality Reduction?*)
- Societal impact of AI (+ Discussion)


Thanks for your attention!
email: gianmaria.campedelli@unitn.it
Twitter: @CampedelliGian

Bibliography I

Christopher M. Bishop (Aug. 1, 2006). *Pattern Recognition And Machine Learning*. New Ed edizione. New York: Springer Nature. 738 pp. ISBN: 978-0-387-31073-2.

Marc Peter Deisenroth (Apr. 23, 2020). *Mathematics for Machine Learning*. Cambridge ; New York, NY: Cambridge University Press. 390 pp. ISBN: 978-1-108-45514-5.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville (Jan. 1, 2017). *Deep Learning*. Cambridge, Massachusetts: Mit Pr. 775 pp. ISBN: 978-0-262-03561-3.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman (June 20, 2013). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2 edizione. New York, NY: Springer Nature. 745 pp. ISBN: 978-0-387-84857-0.

Bibliography II

Gareth James et al. (2013). *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York. ISBN: 978-1-4614-7137-0 978-1-4614-7138-7. DOI: 10.1007/978-1-4614-7138-7. URL: http://link.springer.com/10.1007/978-1-4614-7138-7 (visited on 11/16/2020).

Artwork Reference

Title Slide: *Peinture (Etoile Bleue), Joan Miro (1927)*

First Technical Concepts: *Stati d'animo serie II. Gli addii, Umberto Boccioni (1911)*

Model Evaluation *La Leçon de Musique, Francis Alÿs (2000)*

Algorithms *Structural Constellation "To Ferdinand Hodler", Josef Albers (1954)*

Closing Slide: *The Persistence of Memory, Salvador Dalì (1931)*