Deep Learning

A course about ***theory*** *&* ***practice***

Predictions

Marco Piastra

Feed-Forward Neural Network

Target function: $y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$

Dataset

$$D := \{(\boldsymbol{x}^{(i)},\, y^{(i)})\}_{i=1}^{N}$$

Representation

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{W} \in \mathbb{R}^{h \times d},\ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}$$

Evaluation *(Mean Squared Error)*

$$L(D) := \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

Optimization *(Gradient descent and its variants)*

$$\Delta \boldsymbol{W} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{W}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta \boldsymbol{b} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{b}} L(\tilde{y}^{(i)}, y^{(i)})$$

$$\Delta \boldsymbol{w} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{w}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta b = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial b} L(\tilde{y}^{(i)}, y^{(i)})$$

Predictions?

Optimization:
The aim is finding the parameters that make the representation best approximating the target function over the dataset

Fundamental question:
How good is the approximator with data items that are *not* in the dataset?

Independent, Identically Distributed (iid)

Dataset

$$D = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}) \right\}_{i=1}^{N}$$

this what we use for optimization (a.k.a. learning)

Identically distributed

$$p^*(\boldsymbol{x}^{(i)}) = p^*(\boldsymbol{x}^{(j)}), \quad \forall i, j$$

where p^* is the (unknown) true probability that generated the sample

Independent

$$p^* \left(\{\boldsymbol{x}^{(i)}\}_{i=1}^{N} \right) = \prod_{i=1}^{N} p^*(\boldsymbol{x}^{(i)})$$

What we might look for

Evaluation *(Mean Squared Error)*

$$L(D) = \frac{1}{N} \sum_{i=1}^{N} \left(\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2$$

this what we use for optimization (a.k.a. learning)

Expected error *(over a population of interest)*

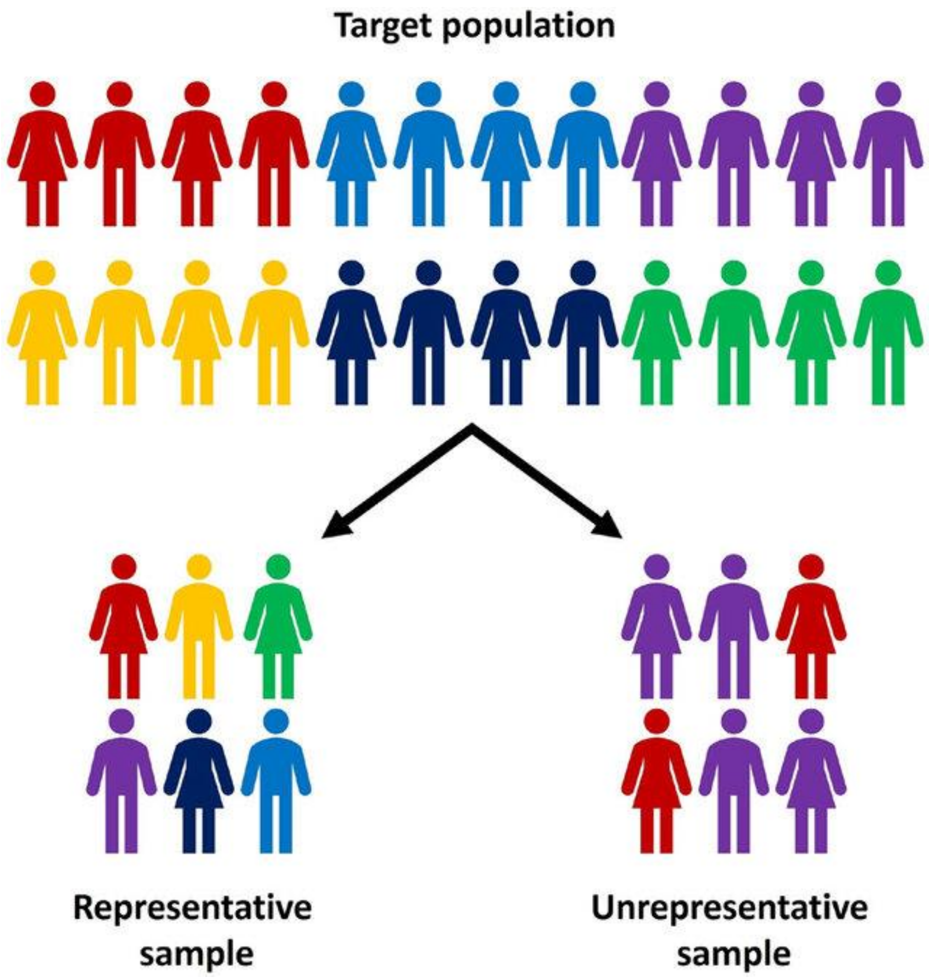
$$L_{p^*} := \mathbb{E}_{p^*} \left[\left(\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 \right]$$

this what we might want to minimize

where p^* is the (unknown) true probability of the population

Representativeness

Is the dataset representative of <u>input features</u> $p^*(\boldsymbol{x}^{(i)})$ *?*



[Image from https://cms.galenos.com.tr/Uploads/Article_53618/Diagn%20Interv%20Radiol-28-450-En.pdf]

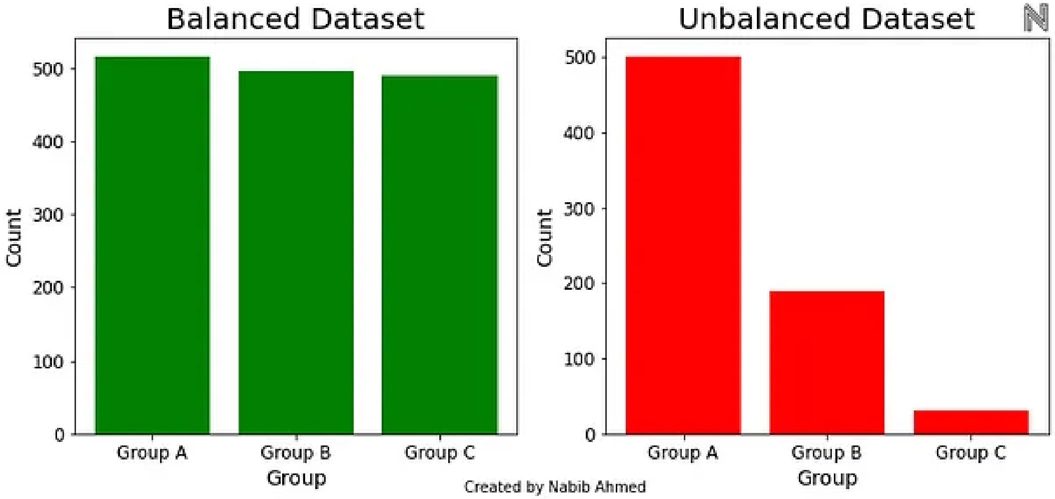
Balanced vs. unbalanced

Sometimes, even the minimization of

$$L_{p^*} := \mathbb{E}_{p^*}\left[\left(\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2\right]$$

might not be enough

When feature distributions are <u>unbalanced</u> a predictor minimizing the <u>expected error</u> will be <u>biased</u> towards over-represented classes



[Image from https://medium.com/@nahmed3536/data-bias-what-all-data-practitioners-should-be-aware-of-115eaeae48c]

Noisy observations

So far, we assumed that in a dataset D

$$y^{(i)} = f^*(\boldsymbol{x}^{(i)}), \quad \forall i$$

namely, that all annotations are noise-free

What if we have instead

$$y^{(i)} = f^*(\boldsymbol{x}^{(i)}) + \epsilon$$

where ϵ is some random noise?

If $\epsilon \sim \mathcal{N}(0, \sigma^2)$, namely if noise is gaussian with zero mean (and any variance), we can still use the <u>expected error</u> as a target

If this is not the case, our predictions will be <u>biased</u>

Training Set
Validation Set
Test Set

Overfitting

When the training process becomes too specific to the training set

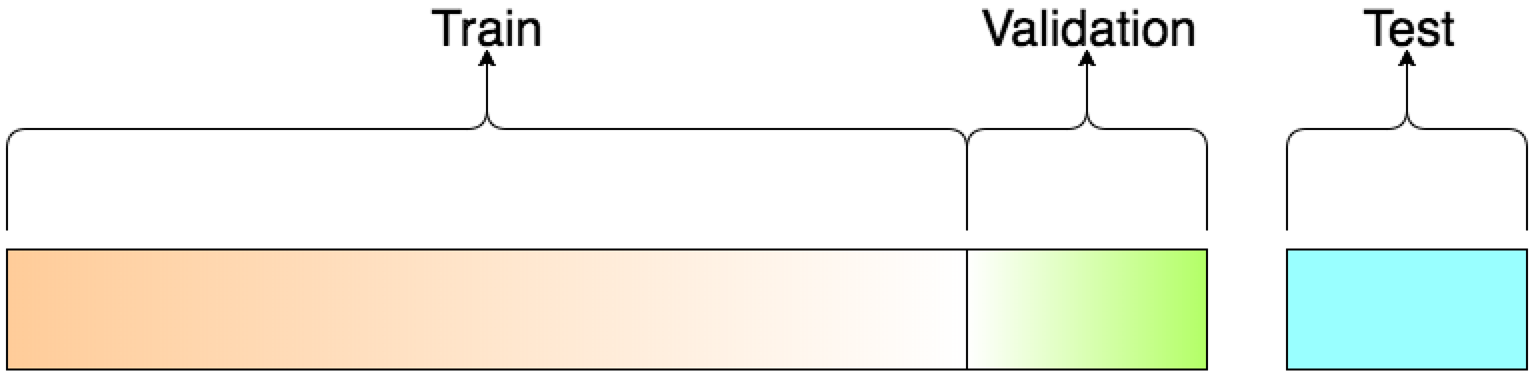
- ***Training set, validation set, test set***

  Splitting the dataset

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \;\cup\; \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \;\cup\; \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$



Overfitting

When the training process becomes too specific to the training set

- ***Training set, validation set***

Splitting the dataset

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$

Training is made on D_{train} only

At each *epoch*

when the whole D_{train} has been processed

the loss function is evaluated on D_{val}

After some epochs, the performance on D_{val} might get ***worse***

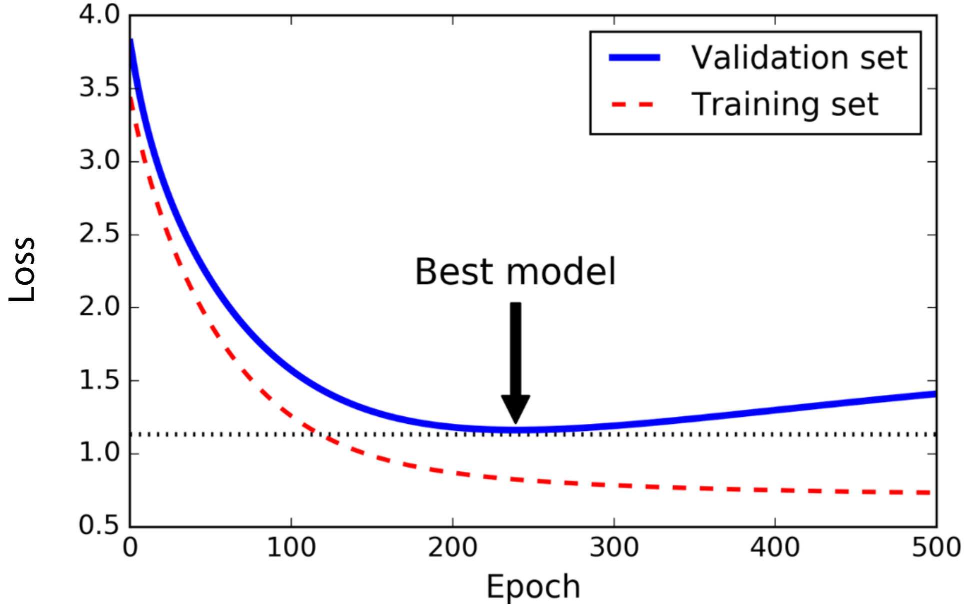


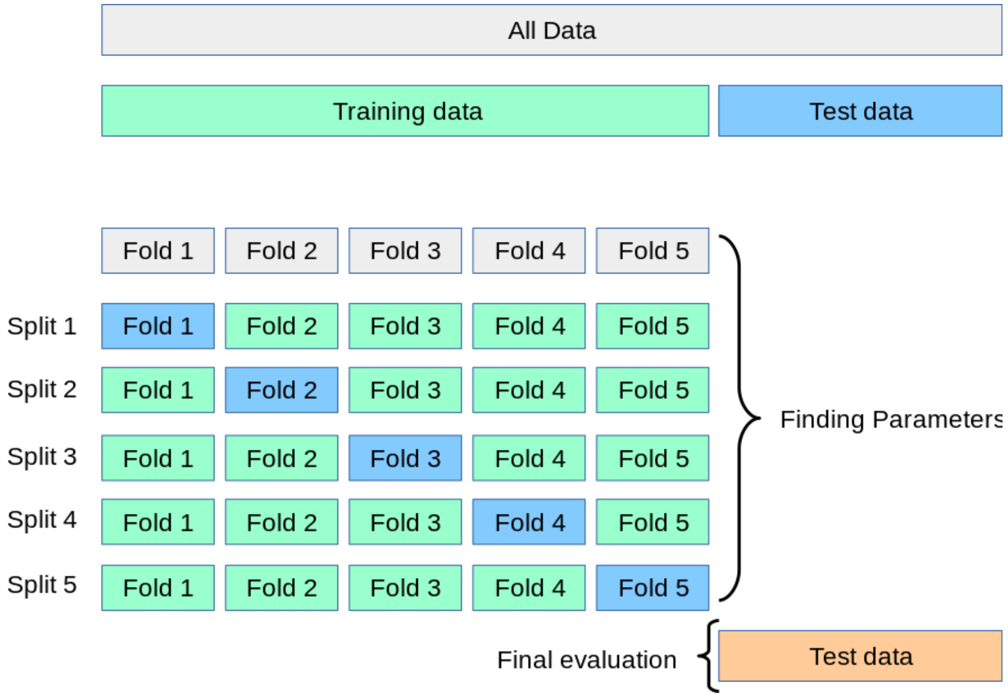
Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

k-Fold Cross-Validation

- ***One dataset, multiple splits***

  1) Divide the dataset into k splits (i.e. *folds*)
  2) Use k - 1 folds for training and 1 fold for testing
  3) Unless all combinations have been considered, change combination and go back to 2)

Consider the *average test loss* across all possible combinations



All Data

Training data | Test data

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 1	🟦 Fold 1	🟩 Fold 2	🟩 Fold 3	🟩 Fold 4	🟩 Fold 5
Split 2	🟩 Fold 1	🟦 Fold 2	🟩 Fold 3	🟩 Fold 4	🟩 Fold 5
Split 3	🟩 Fold 1	🟩 Fold 2	🟦 Fold 3	🟩 Fold 4	🟩 Fold 5
Split 4	🟩 Fold 1	🟩 Fold 2	🟩 Fold 3	🟦 Fold 4	🟩 Fold 5
Split 5	🟩 Fold 1	🟩 Fold 2	🟩 Fold 3	🟩 Fold 4	🟦 Fold 5

Finding Parameters

Final evaluation: Test data

Image from https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html

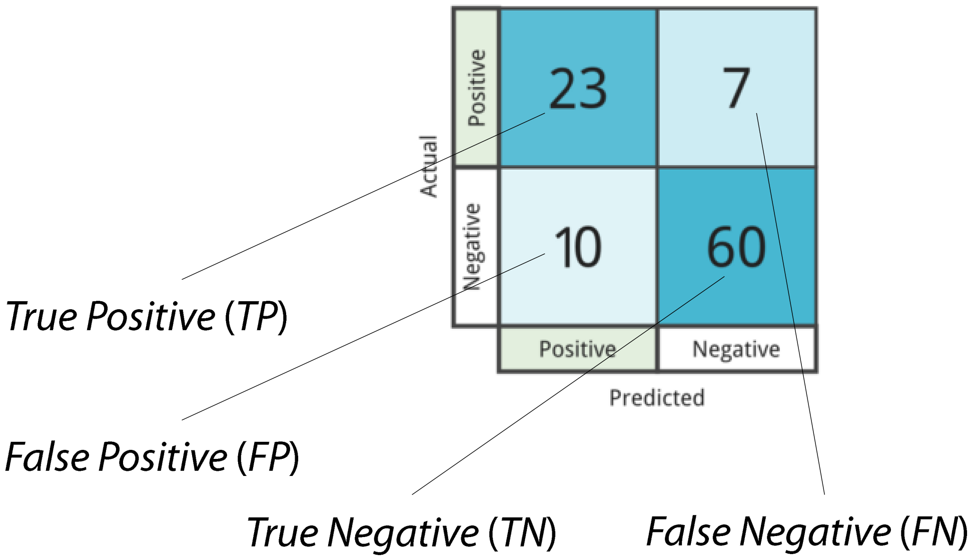
Evaluating a Classifier

Classifier: Confusion Matrix

- Actual VS. Predicted Classes

Predicted:
the class with the highest probability

Binary



Actual

	Positive	Negative
Positive	23	7
Negative	10	60

Predicted

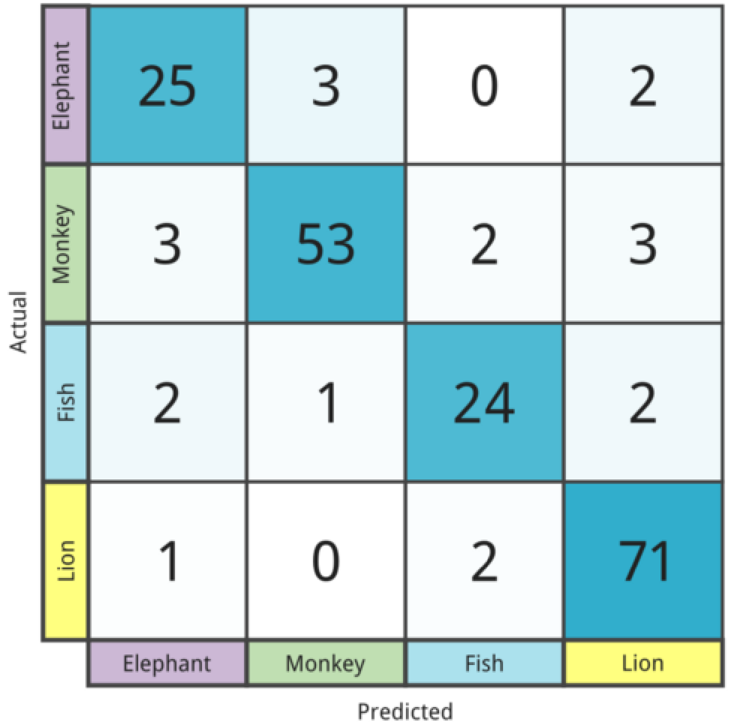
True Positive (TP)

False Positive (FP)

True Negative (TN)

False Negative (FN)

Multi-Class



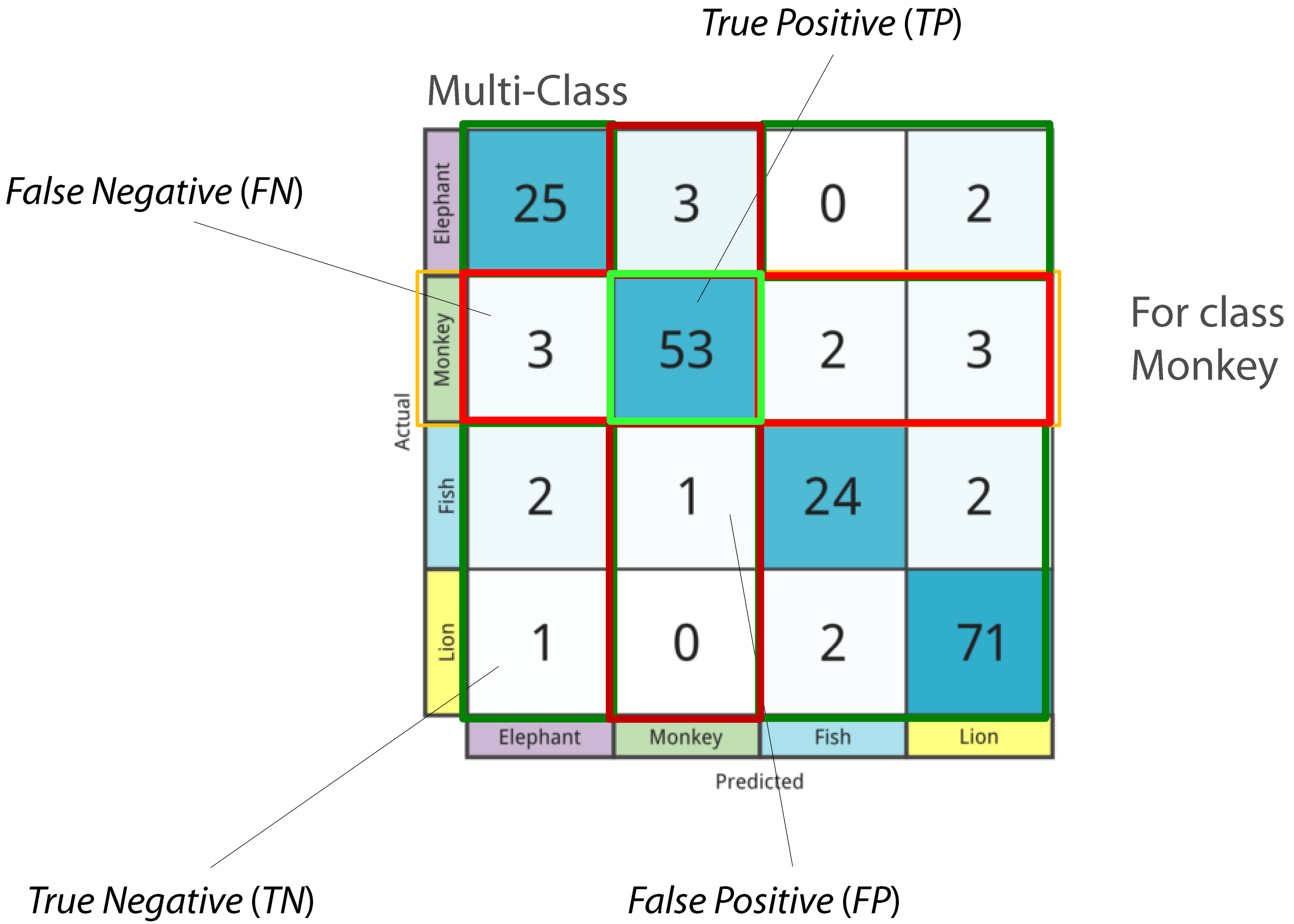
Actual

	Elephant	Monkey	Fish	Lion
Elephant	25	3	0	2
Monkey	3	53	2	3
Fish	2	1	24	2
Lion	1	0	2	71

Predicted

Classifier: Confusion Matrix

- Actual VS. Predicted Classes



Multi-Class

Actual \ Predicted	Elephant	Monkey	Fish	Lion
Elephant	25	3	0	2
Monkey	3	53	2	3
Fish	2	1	24	2
Lion	1	0	2	71

True Positive (*TP*)

False Negative (*FN*)

True Negative (*TN*)

False Positive (*FP*)

For class Monkey

Classifier: Metrics

- **Accuracy**

$$\text{accuracy} := \frac{TP + TN}{TP + TN + FP + FN}$$

- **Recall**

$$\text{recall} := \frac{TP}{TP + FN}$$

also called 'sensitivity'

- **Precision**

$$\text{precision} := \frac{TP}{TP + FP}$$

- **F$_1$**

$$F_1 := 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

typically preferred when
when positive and negative cases are highly <u>unbalanced</u>

Receiver operating characteristic (ROC)

Typically, a classifier produces a probability distribution
Where do we put the threshold γ *?*

Note that now we are considering

$$p^*(y^{(i)} \mid \tilde{y}(\boldsymbol{x}^{(i)}))$$

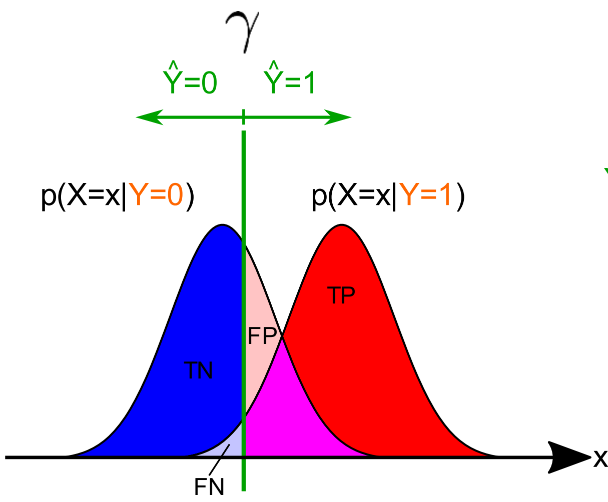
which could be estimated as

same as 'recall'

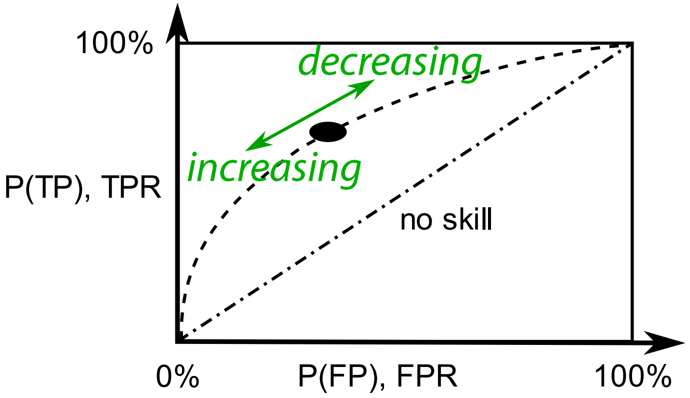
$$\text{true positive rate (TPR)} := \frac{TP}{TP+FN}$$

$$\text{false positive rate (FPR)} := \frac{FP}{FP+TN}$$

these values depend on the threshold γ *we choose*



$\hat{Y}$ Y	1	0
1	TP	FP
0	FN	TN



[Images from https://en.wikipedia.org/wiki/Receiver_operating_characteristic]

Area under curve (AUC)

A random classifier is right / wrong an equal number of times, regardless of γ

$$\text{TPR} = \text{FPR}, \quad \forall \gamma$$

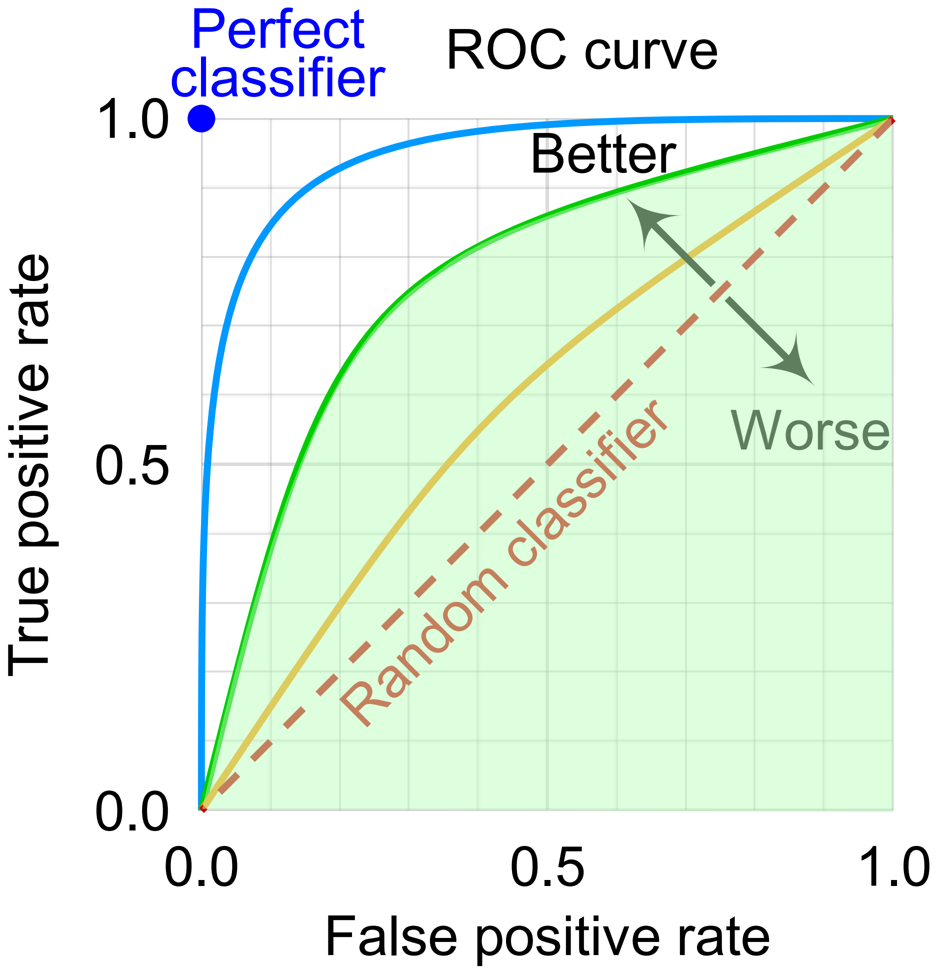
whereas a good classifier should have

$$\text{TPR} \geq \text{FPR}, \quad \forall \gamma$$

The Area Under Curve of the ROC measures the overall efficiency of a classifier

For a random classifier: $\text{AUC} = 0.5$

For a perfect classifier: $\text{AUC} = 1.0$



[Images from https://en.wikipedia.org/wiki/Receiver_operating_characteristic]