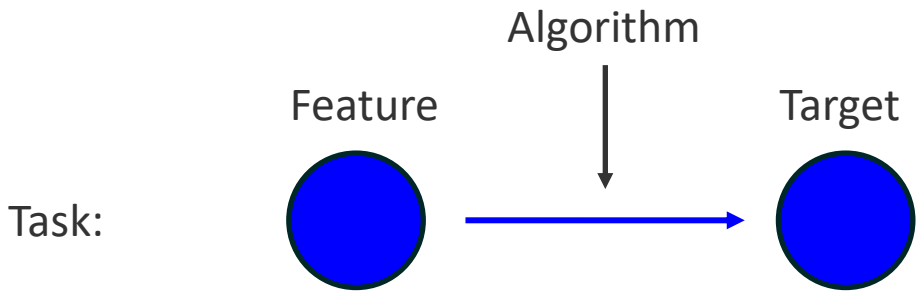
Day 2: Simple Perceptron, Classification, ROC/AUC

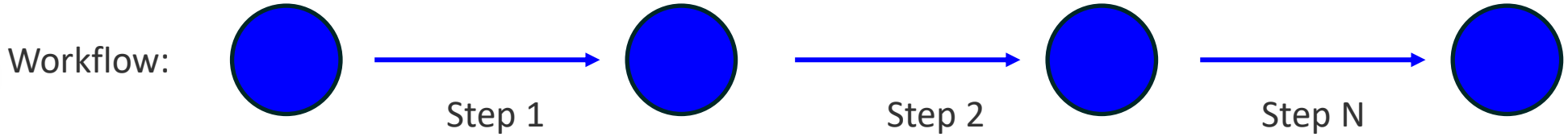
Sergei V. Kalinin

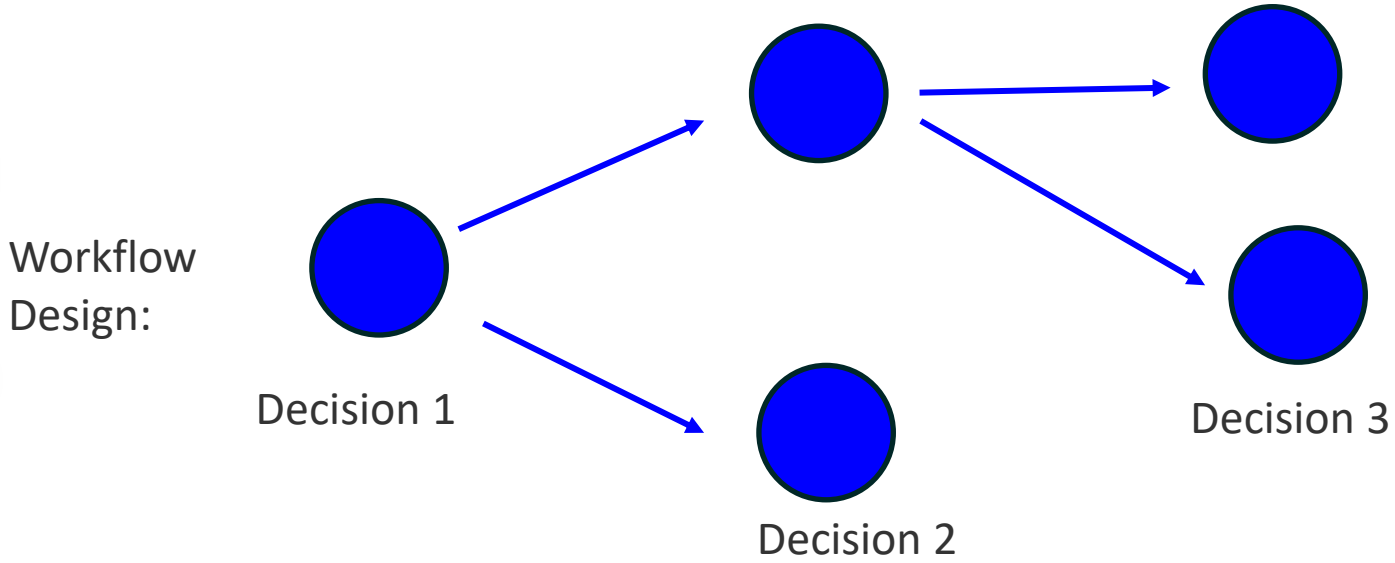
Tasks, workflows, and workflow planning

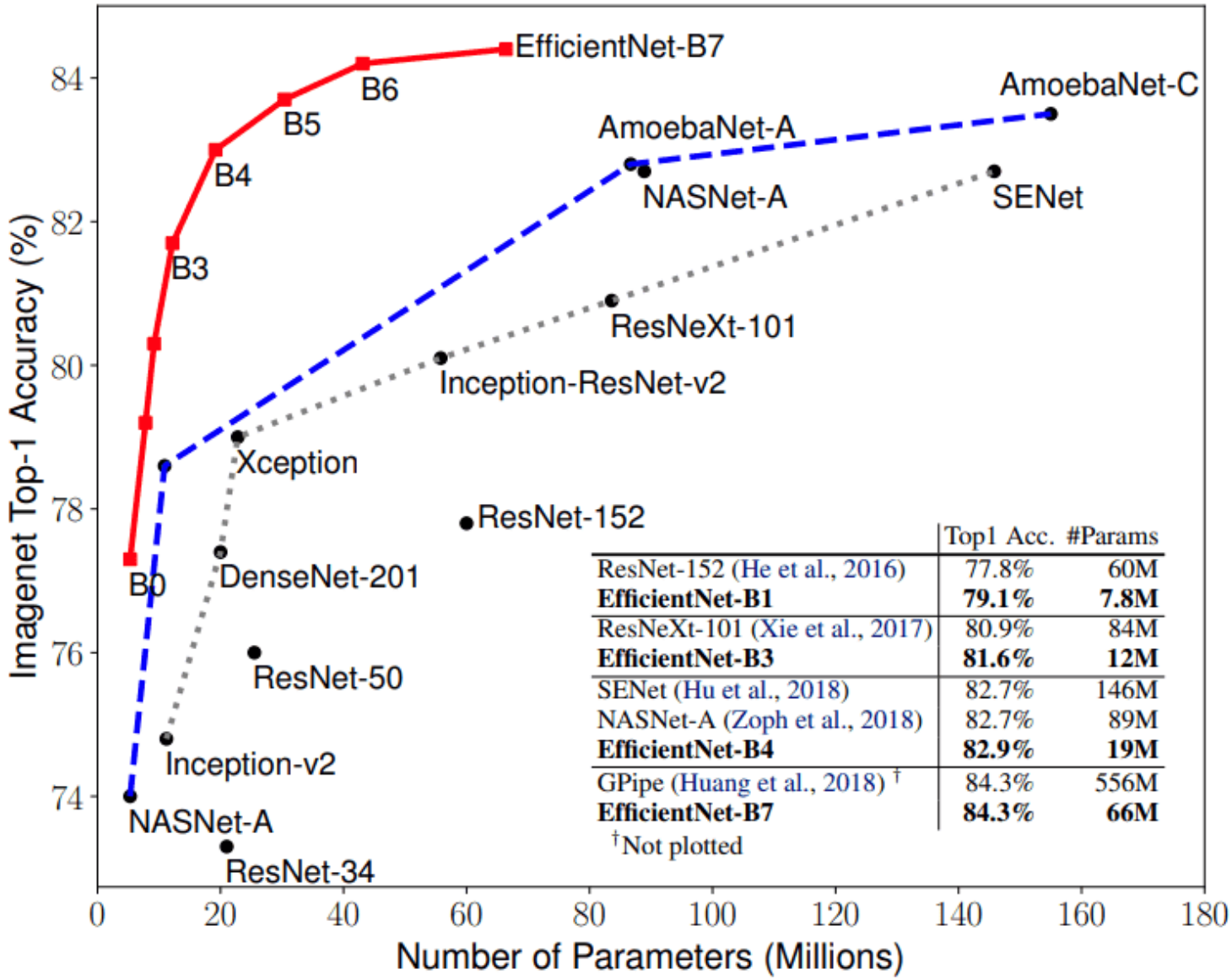


Task can be classification, regression, clustering, optimization.
Algorithm can be kNN, perceptron, DCNN, etc.

We can use complex algorithms for “simple” tasks and simple algorithms for complex tasks.



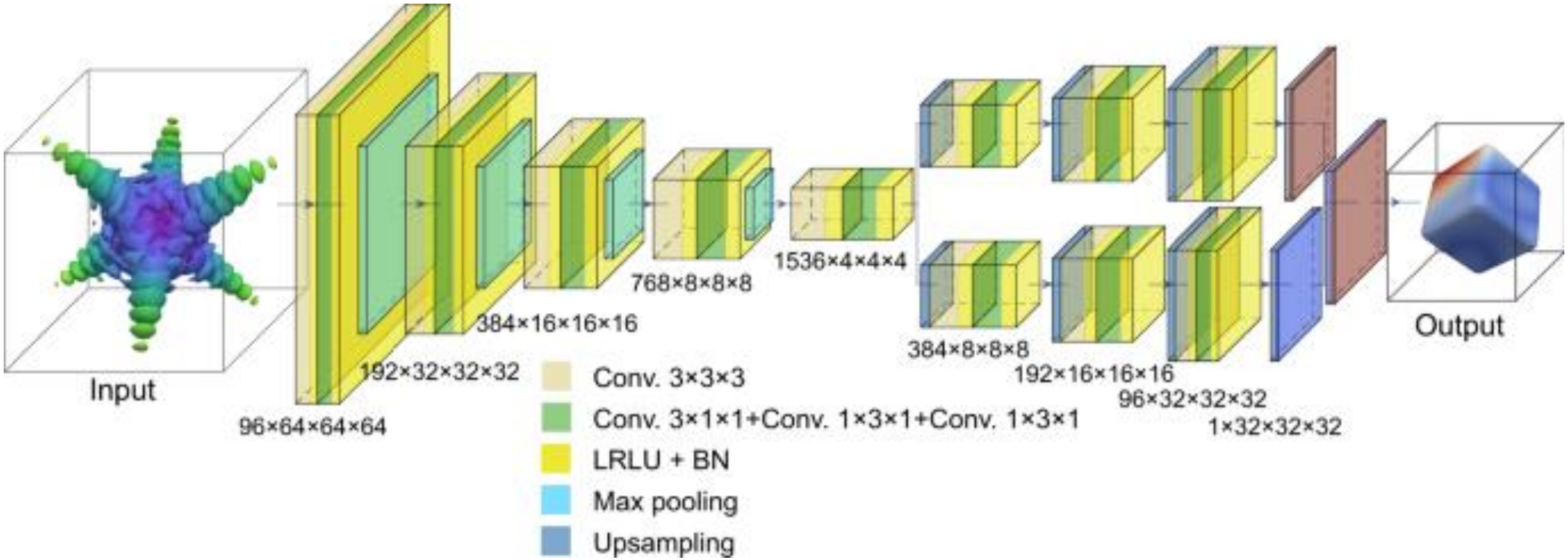




	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

https://theaisummer.com/cnn-architectures/

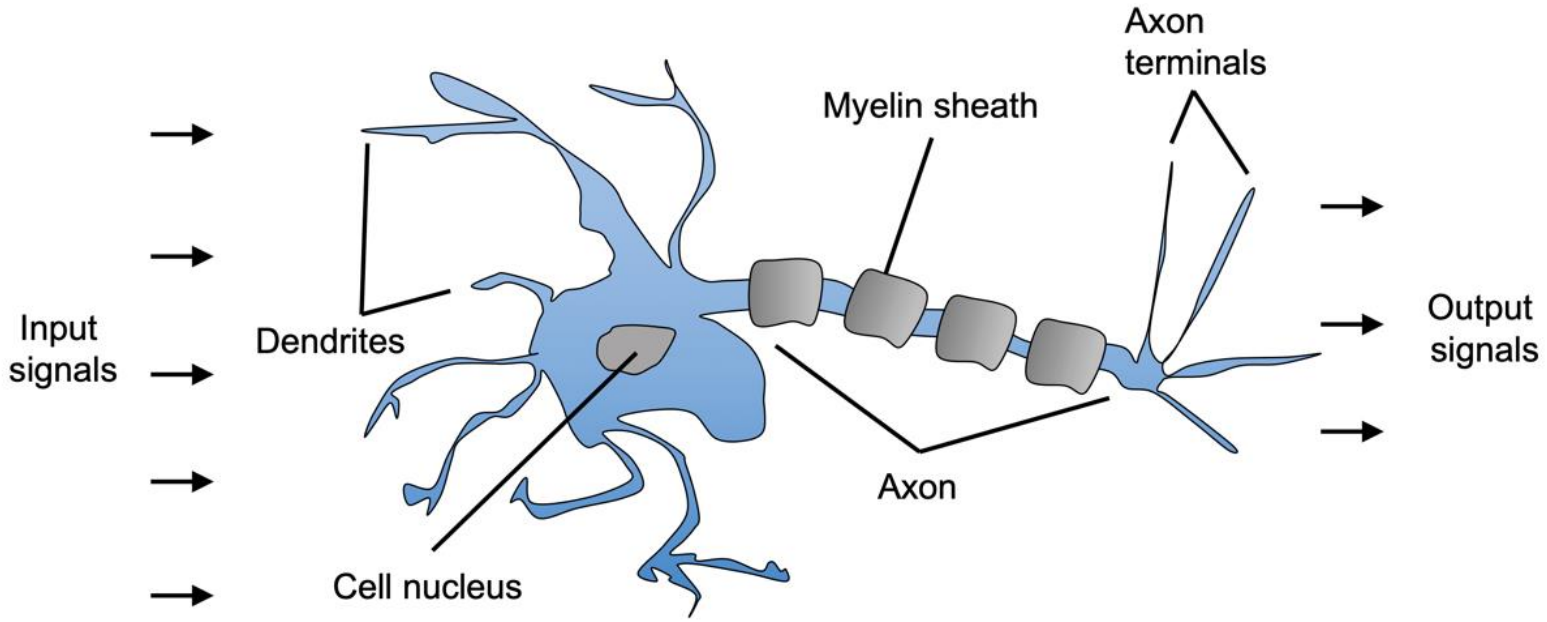


Longlong Wu, Shinjae Yoo, Ana F. Suzana, Tadesse A. Assefa, Jiecheng Diao, Ross J. Harder, Wonsuk Cha & Ian K. Robinson, *Three-dimensional coherent X-ray diffraction imaging via deep convolutional neural networks*

https://www.nature.com/articles/s41524-021-00644-z

Brain structure and McCulloch-Pitts neuron

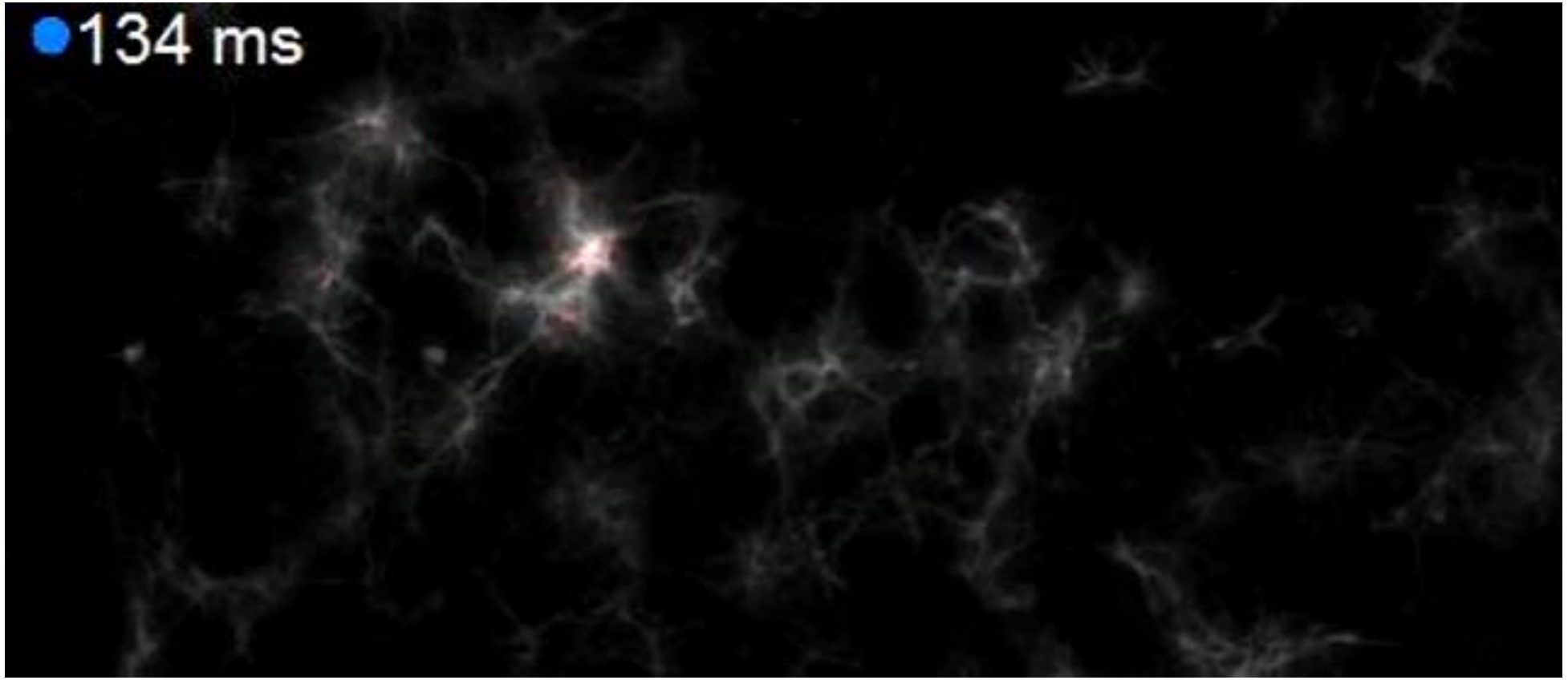
https://www.kenhub.com/en/library/anatomy/histology-of-neurons



A Logical Calculus of the Ideas Immanent in Nervous Activity by *W. S. McCulloch* and *W. Pitts*, *Bulletin of Mathematical Biophysics*, 5(4): 115-133, 1943).

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Neurons in Action

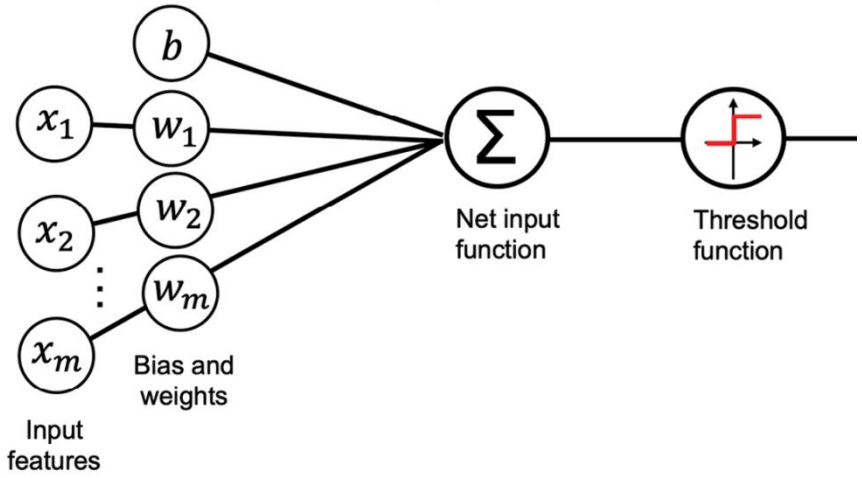


https://news.harvard.edu/wp-content/uploads/2018/02/imaging-neuronal-activity-video-eurekalert-science-news.mp4

Building Linear Neuron

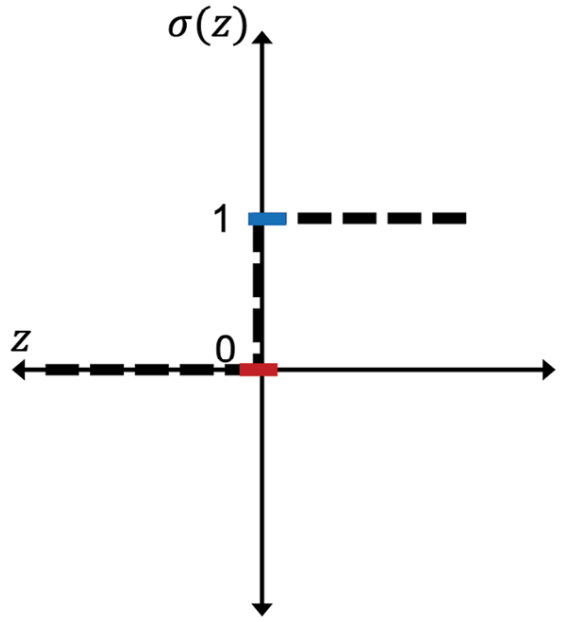
Input: $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

Weights: $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$



Linear transform: $\boldsymbol{z = w_1 x_1 + \ldots + w_m x_m + b = w^T x + b}$

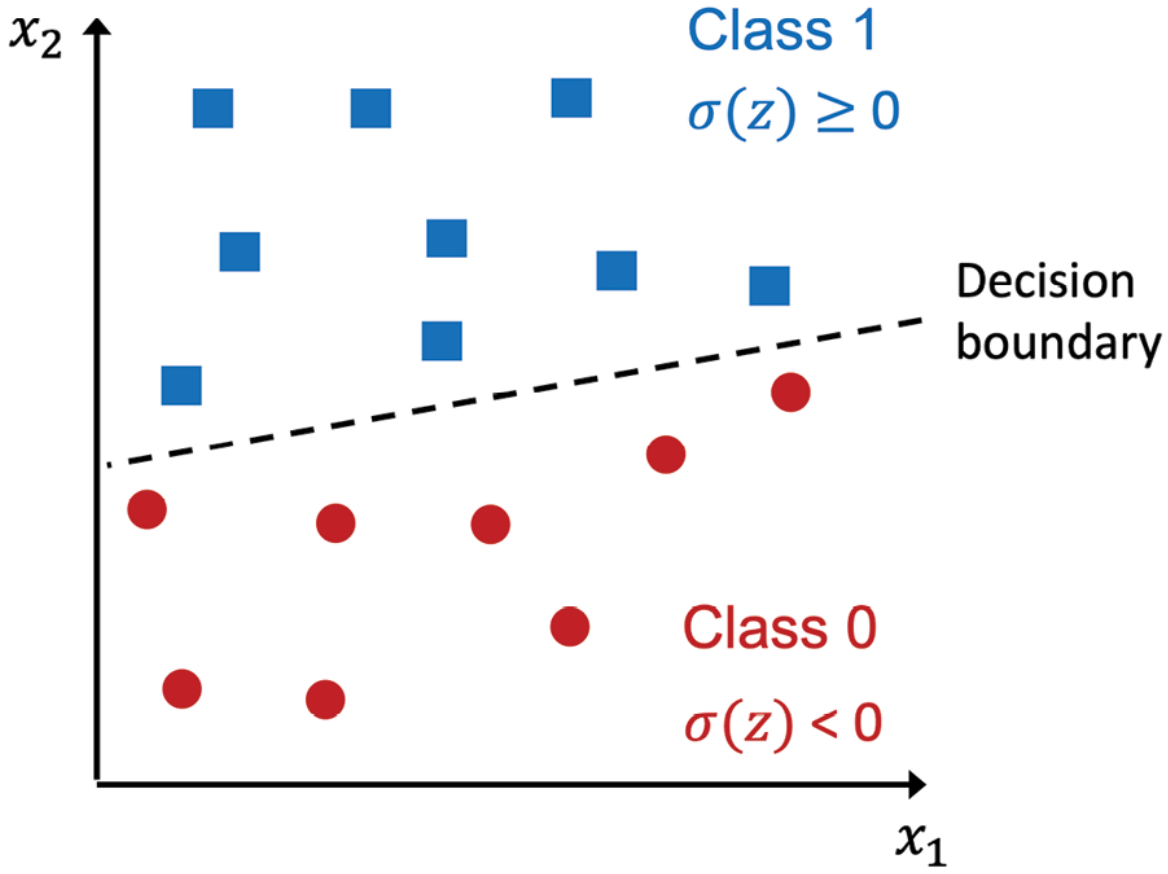
Output: $\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$



where $z = \boldsymbol{w^T x + b}$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Linear Neuron in 2D



Linear transform: $z = w_1 x_1 + w_2 x_2 + b$

Line: $x_2 = - w_1 / w_2 \, x_1 - b / w_2$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Training Linear Neuron

- Initialize the weights and bias unit to 0 or small random numbers
- For each training example, ***x(i)***:
- Compute the output value, $\boldsymbol{y(i) = w^T x(i) + b}$
- Update the weights and bias unit: $w_j := w_j + \Delta w_j$ and $b := b + \Delta b$
- Where $\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$ and $\Delta b = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)$

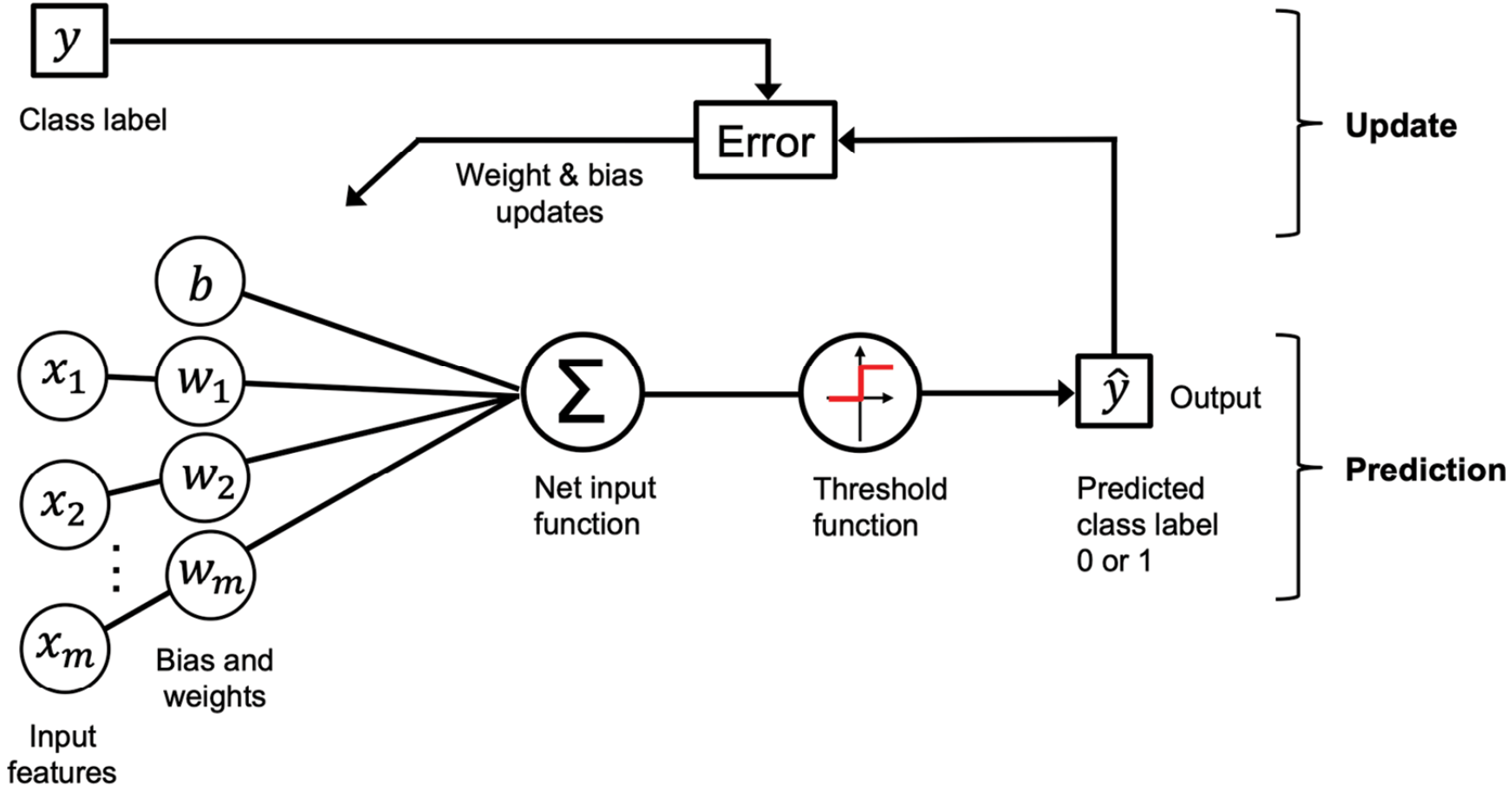
Each weight, w_j, corresponds to a feature, x_j, in the dataset,

η is the **learning rate** (typically a constant between 0.0 and 1.0),

$y^{(i)}$ is the **true class label** of the *i*th training example,

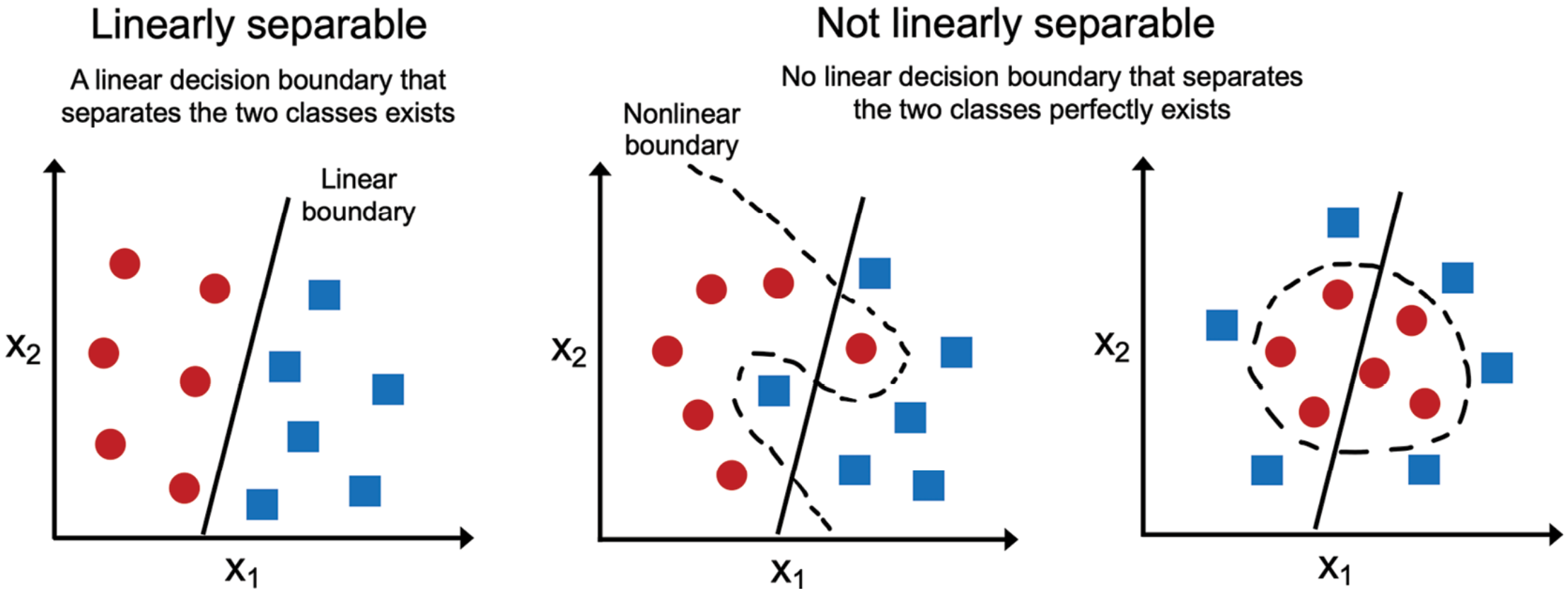
$\hat{y}^{(i)}$ is the **predicted class label**

Training Linear Neuron



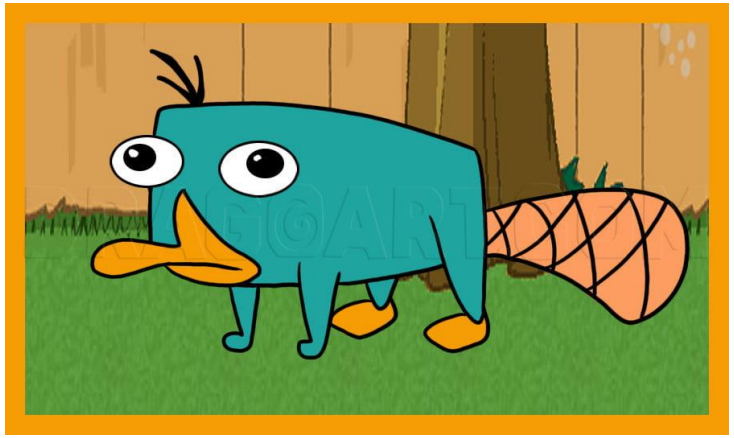
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

What problems can perceptron solve?

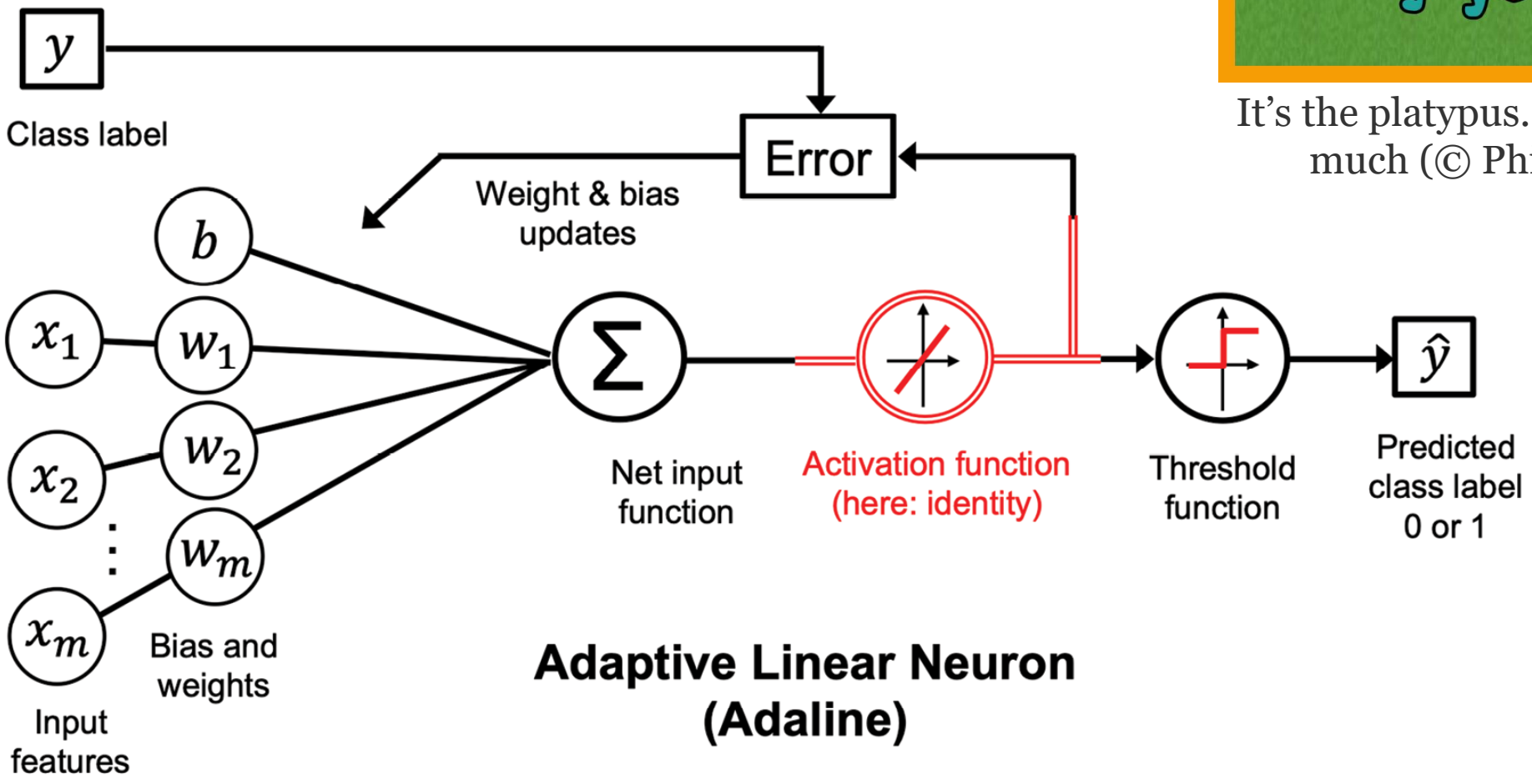


From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaline



It's the platypus. They do not do much (© Phineas and Ferb)

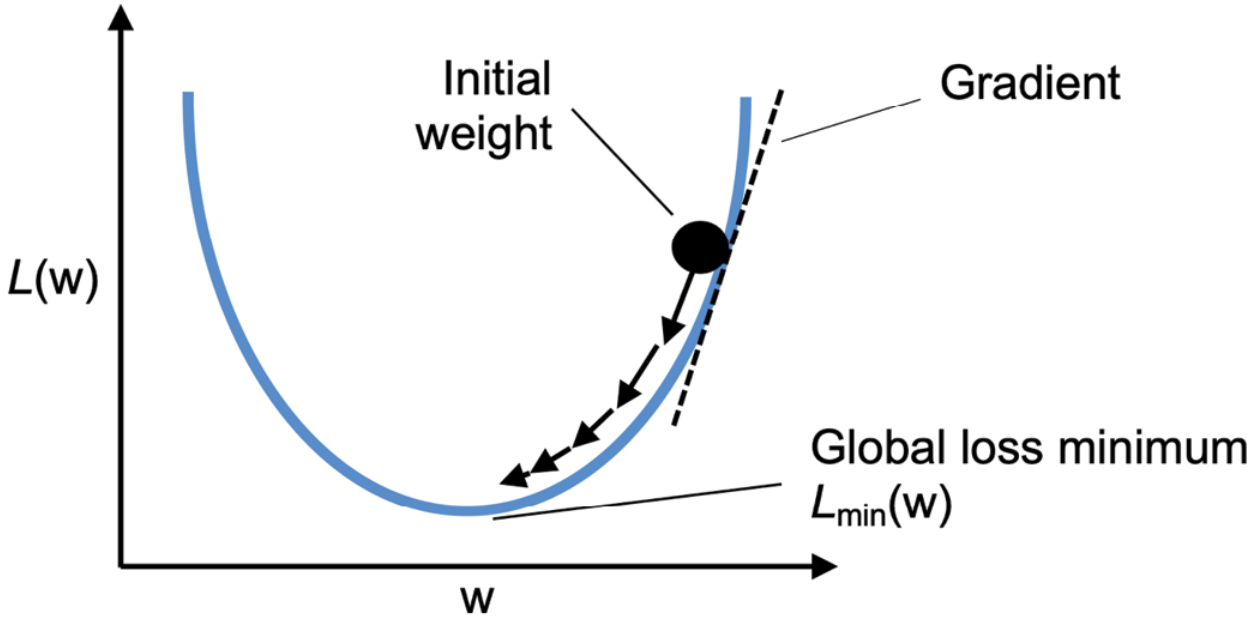


From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaline training

Loss function:

$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$



Weights update:

$$\boldsymbol{w} := \boldsymbol{w} + \Delta\boldsymbol{w}, \quad b := b + \Delta b$$

Learning rule:

$$\Delta\boldsymbol{w} = -\eta\nabla_w L(\boldsymbol{w}, b), \quad \Delta b = -\eta\nabla_b L(\boldsymbol{w}, b)$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

How does it look like?

$$\frac{\partial L}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2 = \frac{1}{n}\frac{\partial}{\partial w_j}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$

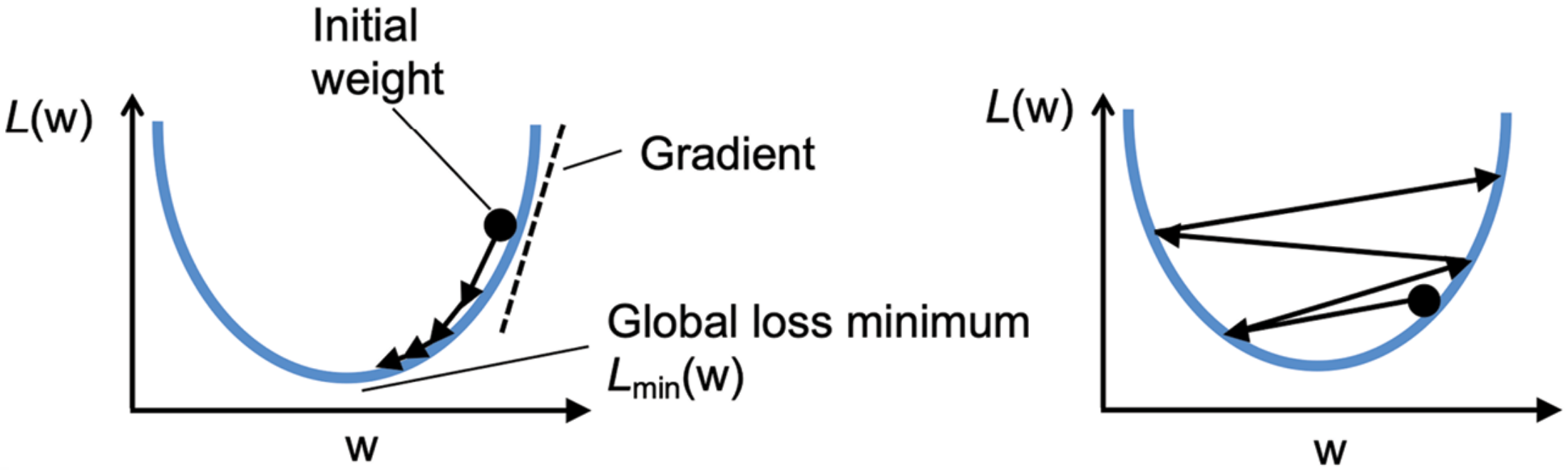
$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j \left(w_j x_j^{(i)} + b\right)\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\left(-x_j^{(i)}\right) = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)x_j^{(i)}$$

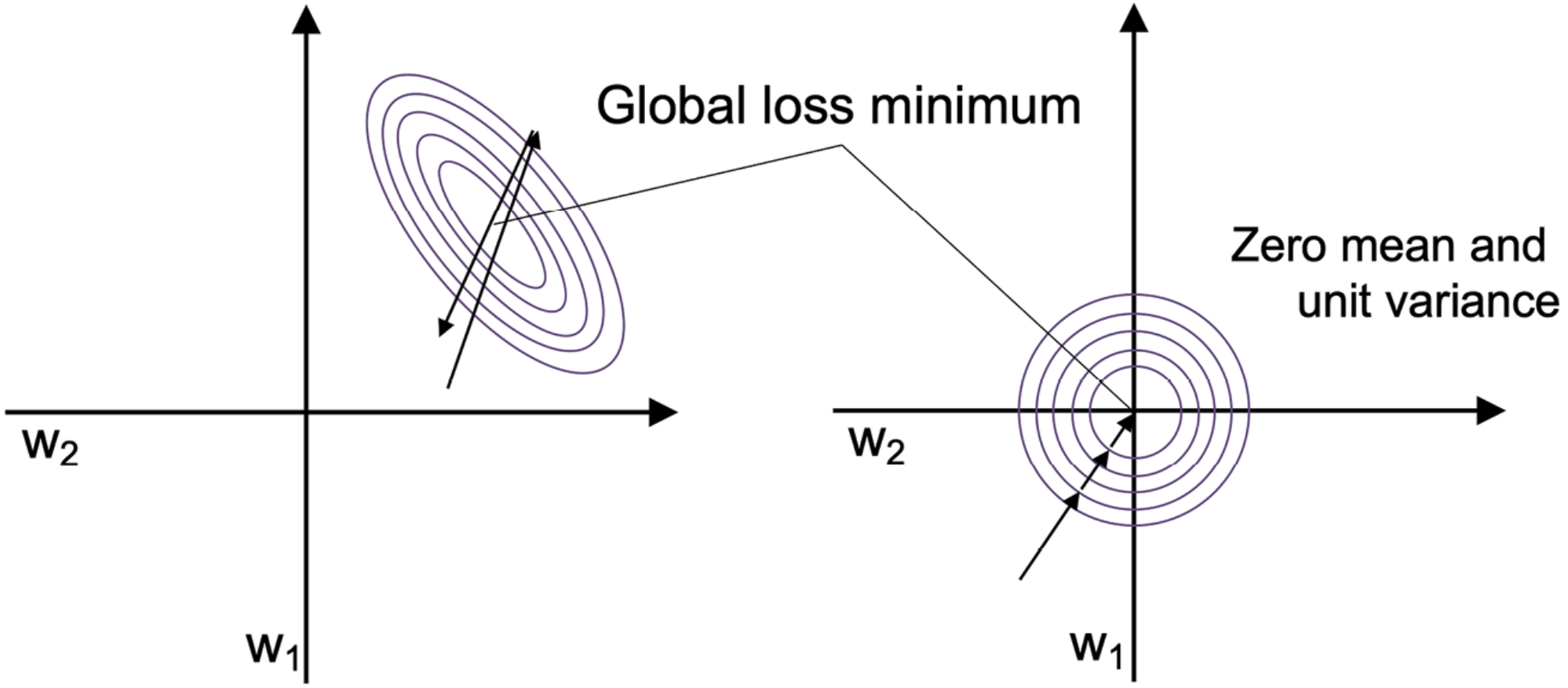
- Adaline learning rule looks identical to the perceptron rule
- However, $\sigma\left(z^{(i)}\right)$ where $z^{(i)} = \boldsymbol{w}^T\boldsymbol{x}^{(i)} + b$ is a real number and not an integer class label.
- Also, weight update is calculated based on all examples in the training dataset (instead of updating the parameters incrementally after each training example): **batch gradient descent**.

Role of learning rate



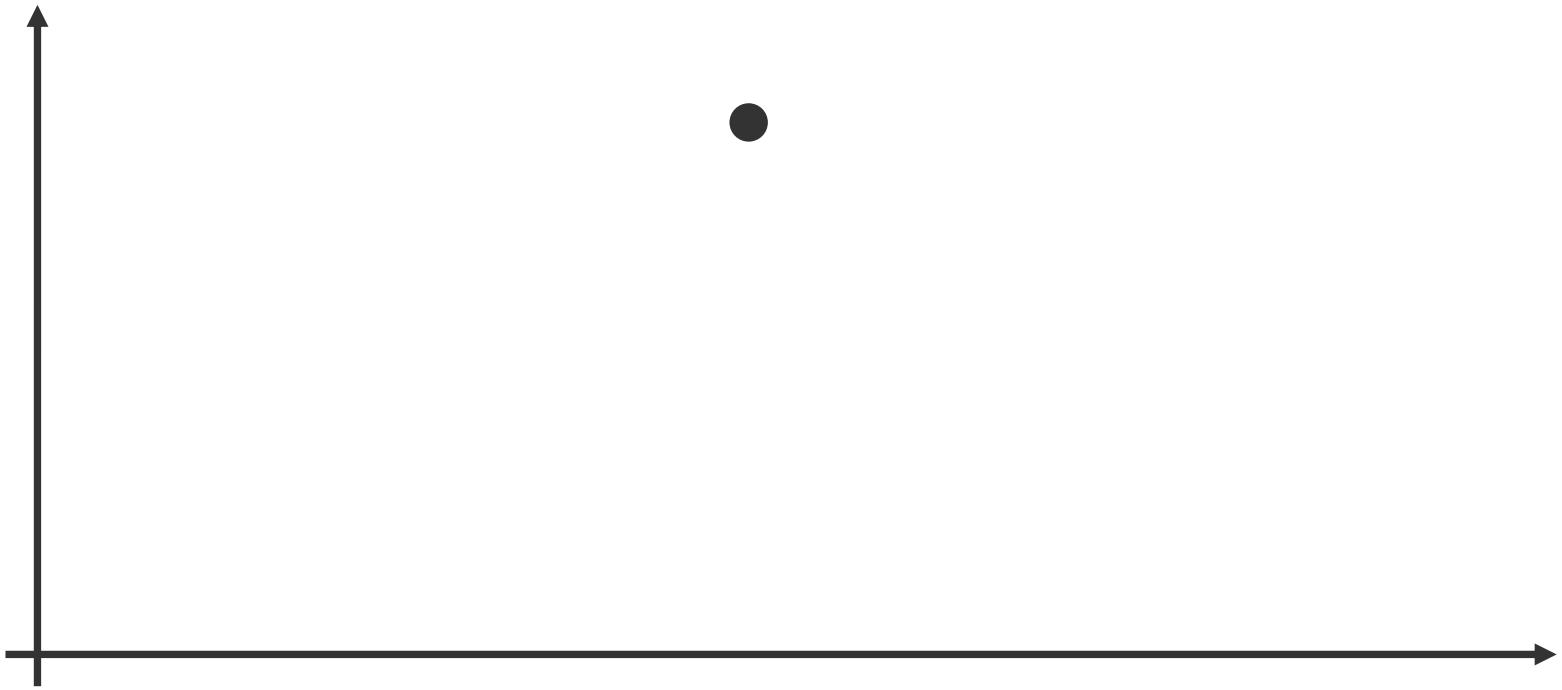
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Why do we standardize input features?

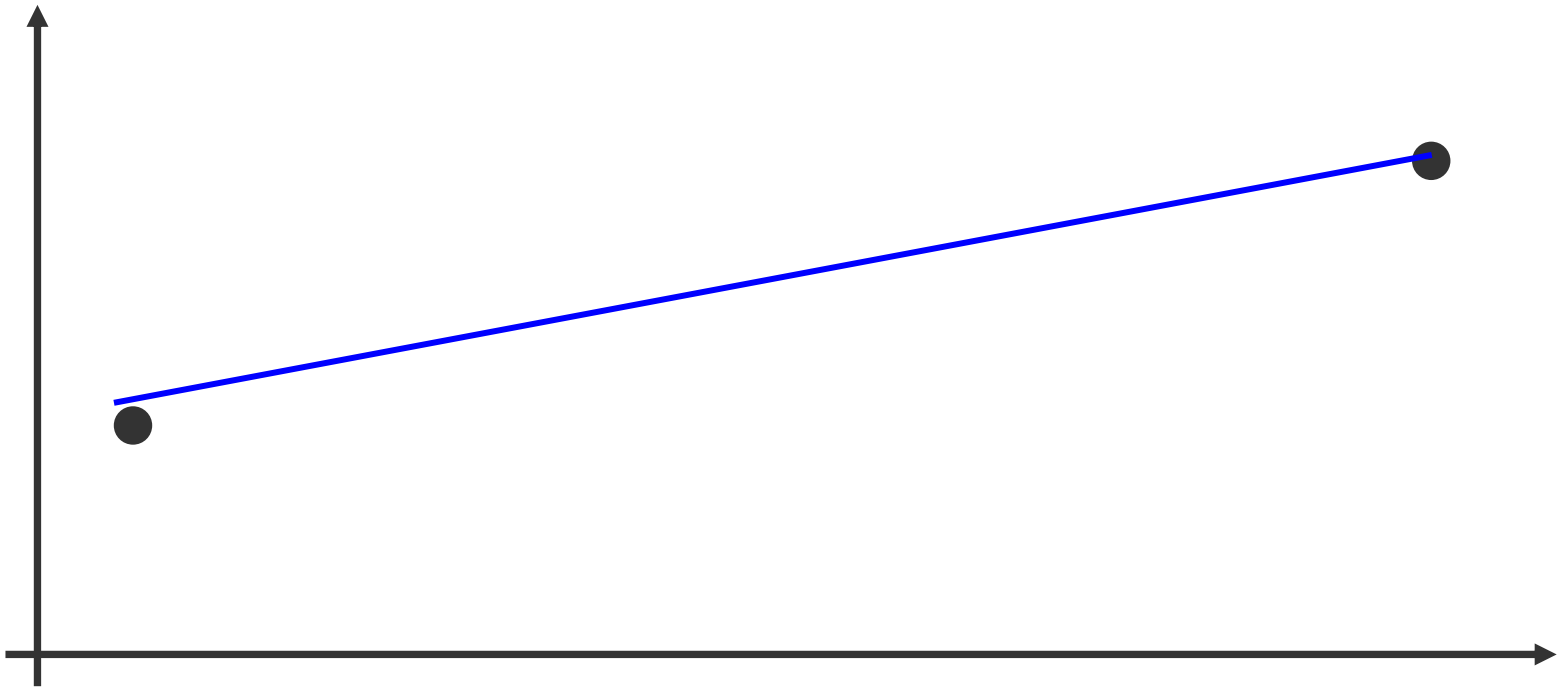


From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Physics vs. data science

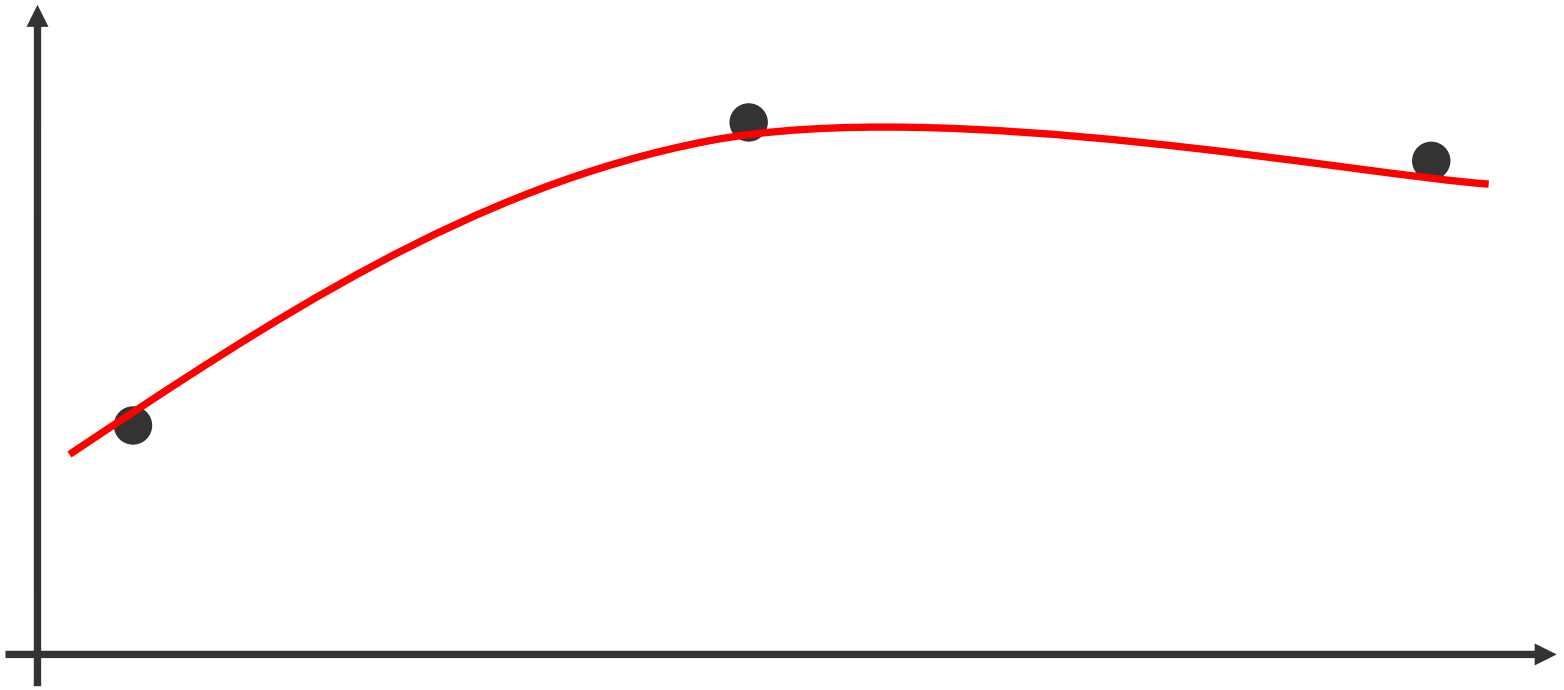


Physics vs. data science



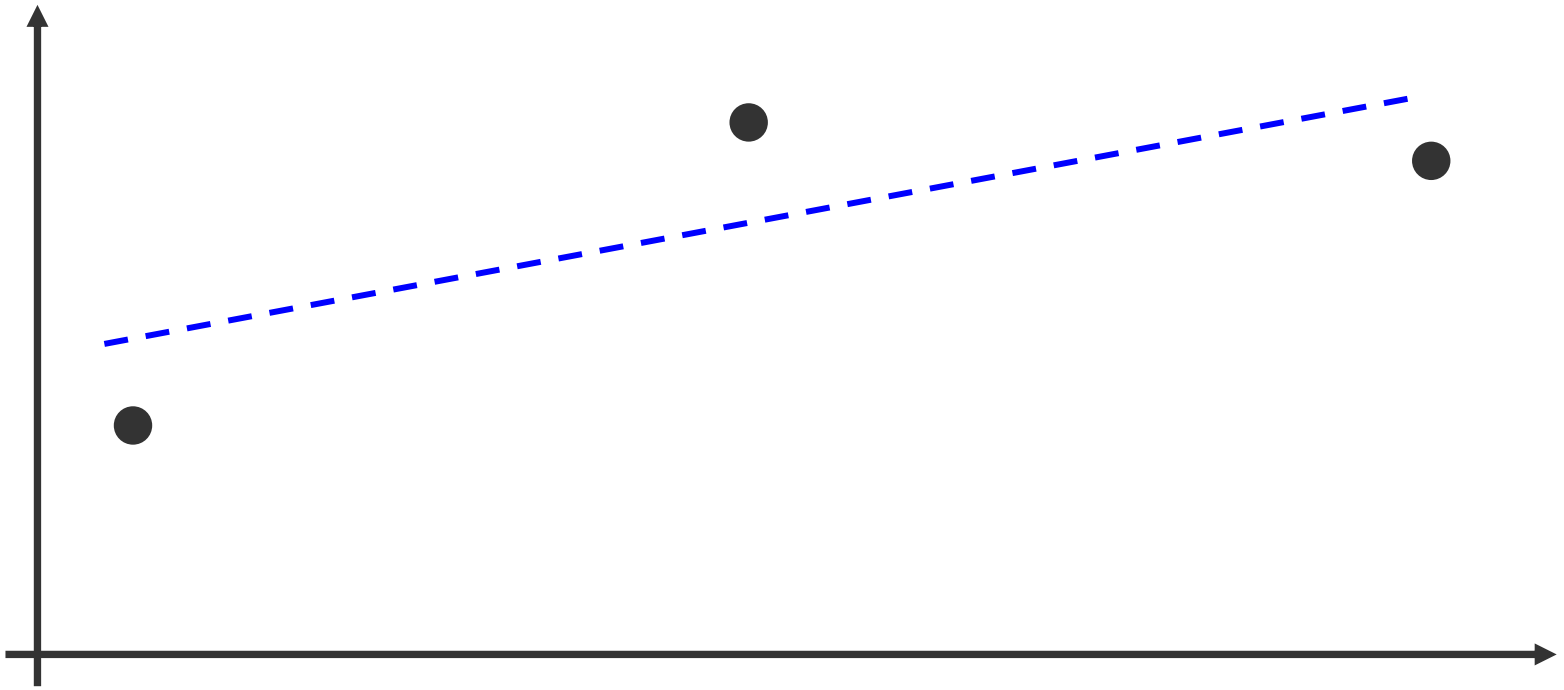
- If we have 2 data points, we “naturally” use linear model
- What should we use if we have three data points? Parabola or linear?
- What if we have one data point?

Physics vs. data science



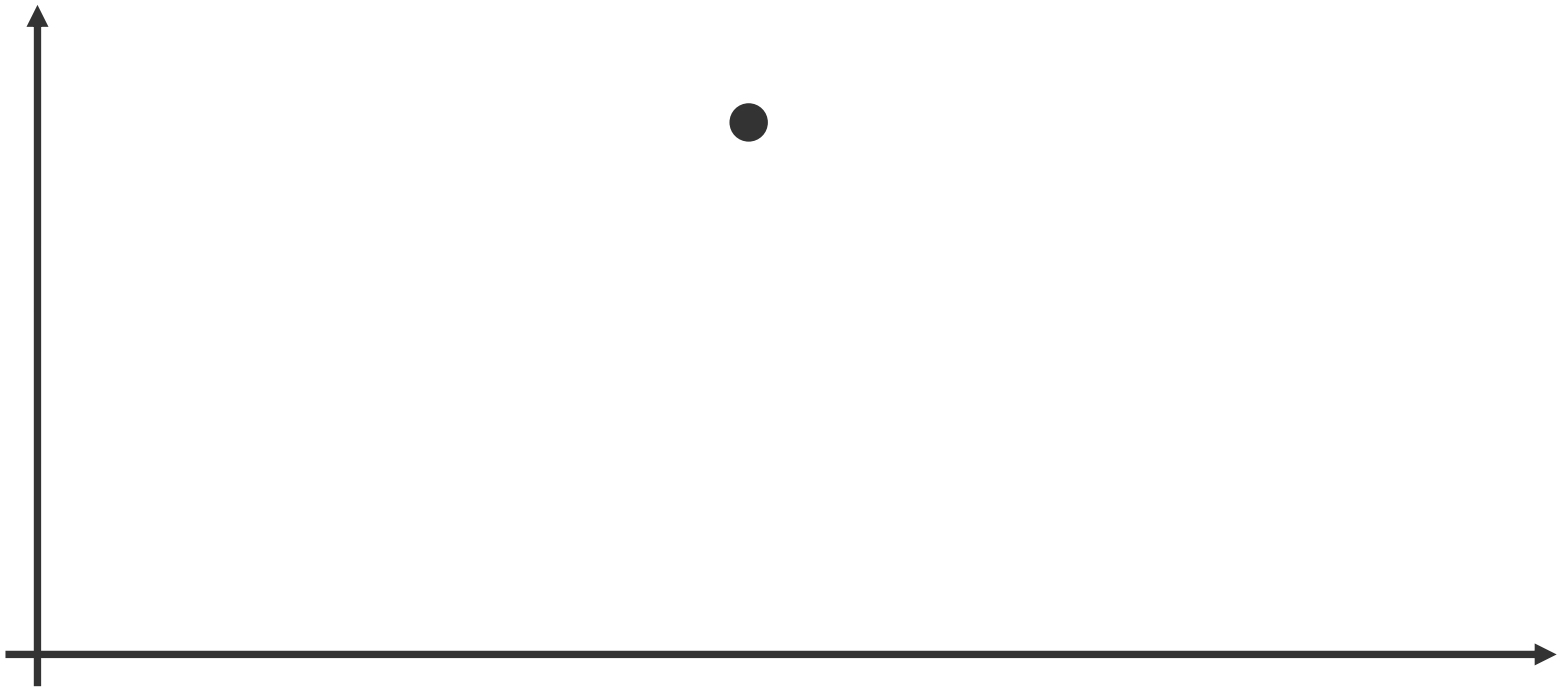
- If we have 2 data points, we “naturally” use linear model
- What should we use if we have three data points? Parabola or linear?
- What if we have one data point?

Physics vs. data science



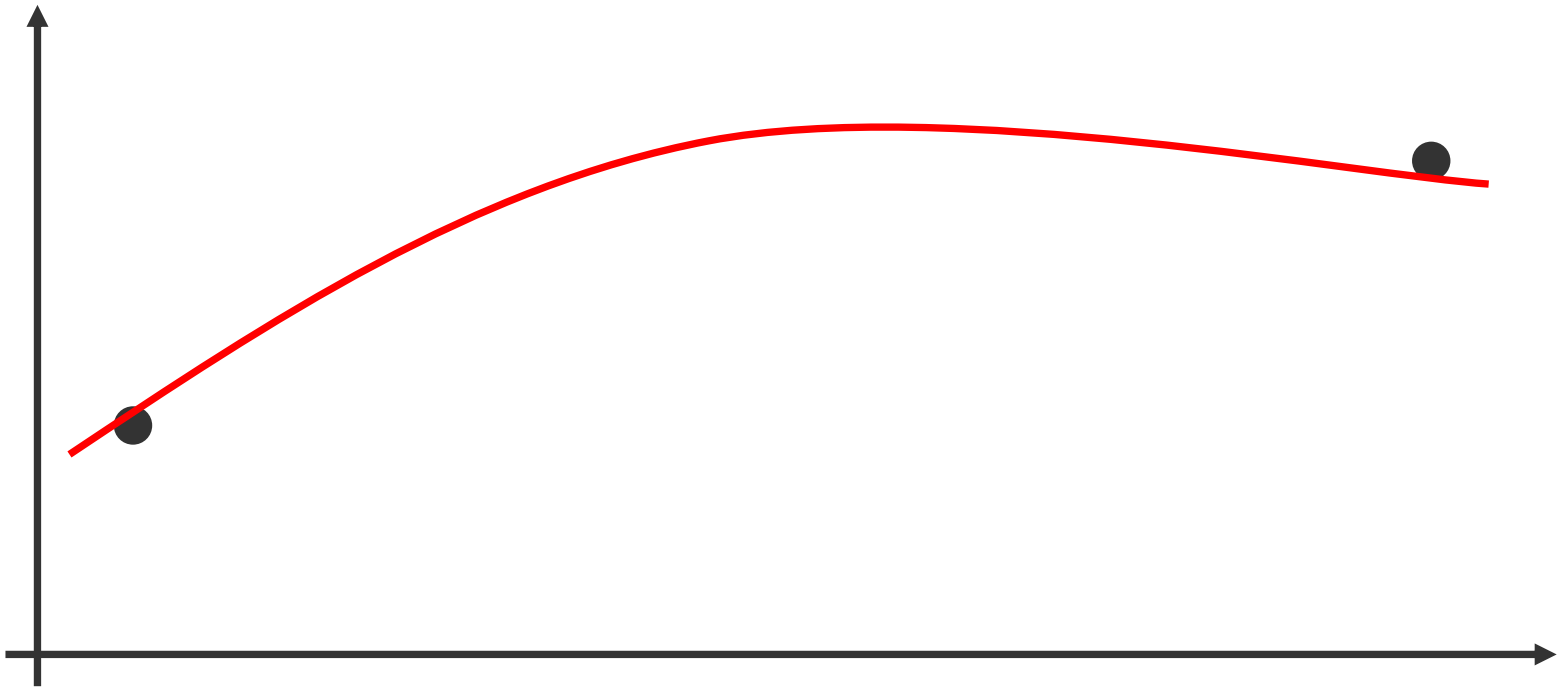
- If we have 2 data points, we “naturally” use linear model
- What should we use if we have three data points? Parabola or linear?
- What if we have one data point?

Physics vs. data science



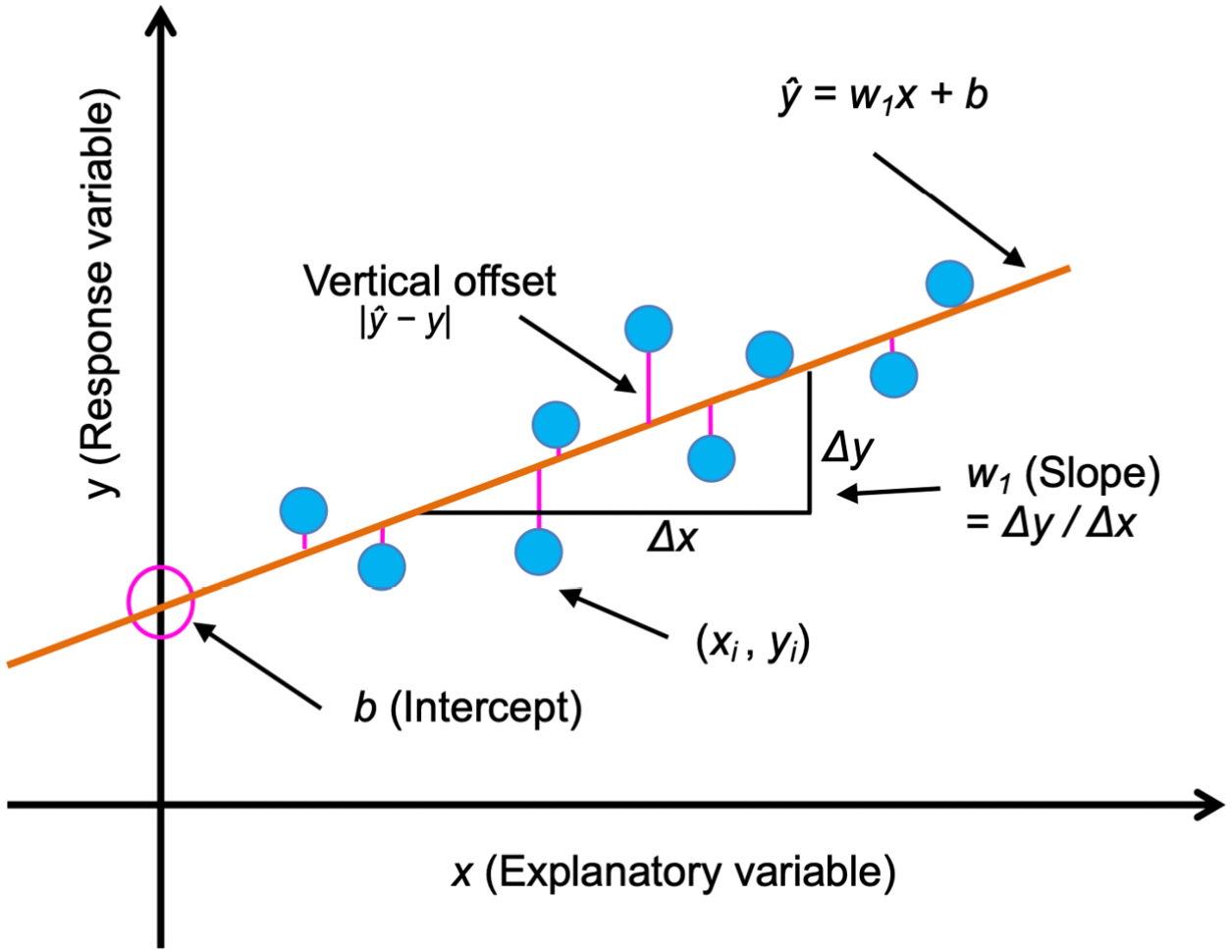
- If we have 2 data points, we “naturally” use linear model
- What should we use if we have three data points? Parabola or linear?
- What if we have one data point?

Physics vs. data science



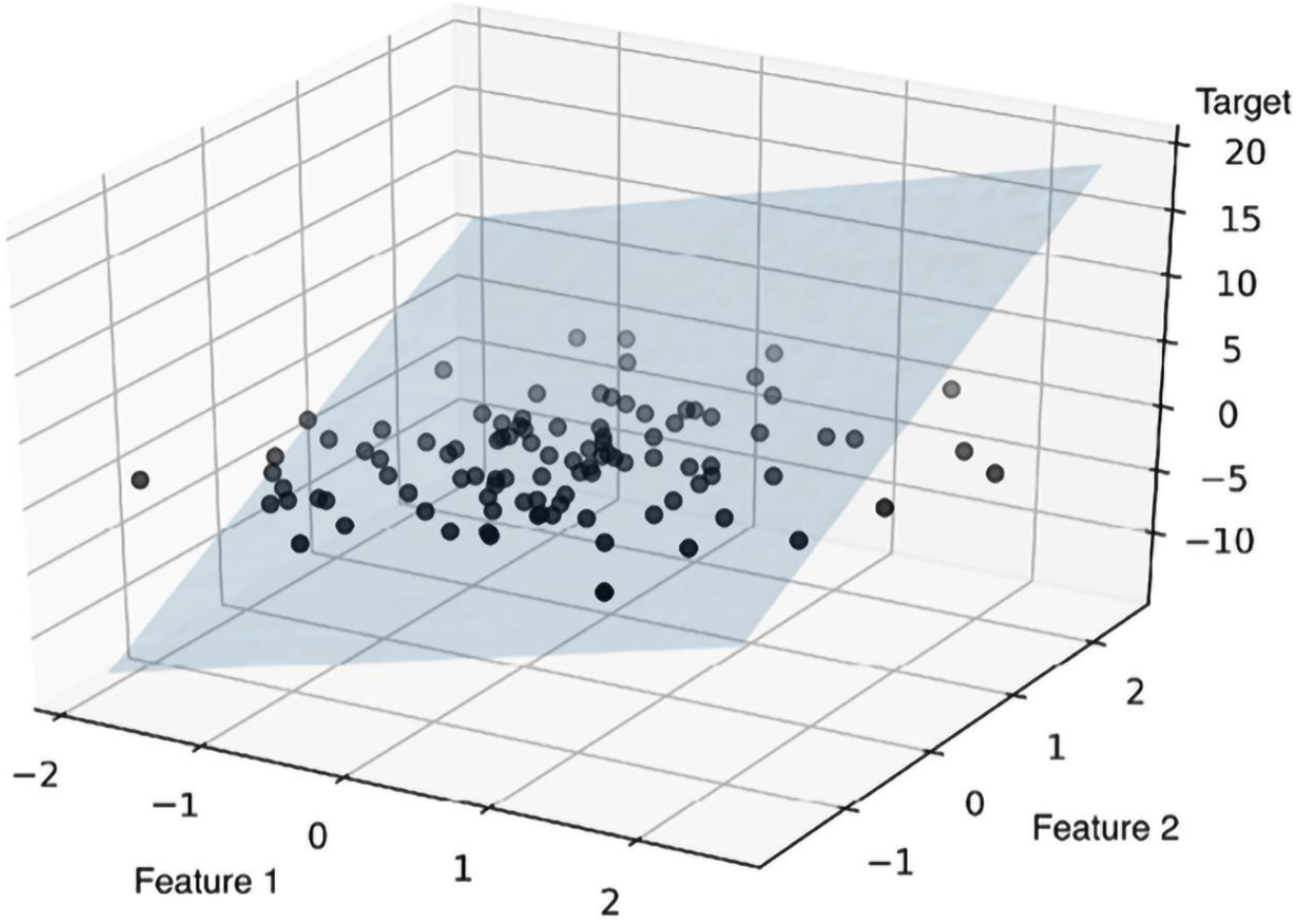
- If we have 2 data points, we “naturally” use linear model
- What should we use if we have three data points? Parabola or linear?
- What if we have one data point?

Linear Regression in 1D



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Linear Regression in 2D



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

RANSAC: Random Sample Consensus

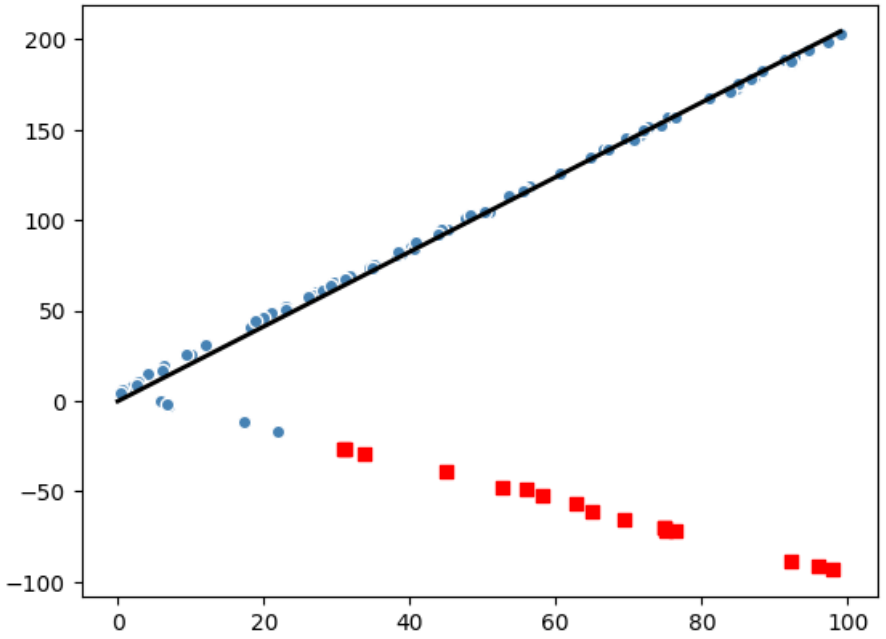
1. Select a random number of examples to be inliers and fit the model.

2. Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers.

3. Refit the model using all inliers.

4. Estimate the error of the fitted model versus the inliers.

5. Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations was reached; go back to *step 1* otherwise.



Regularized linear regression

Linear regression can become nontrivial if **x** in y = lin(**x**) has high D

1. Ridge regression:

$$L(\boldsymbol{w})_{Ridge} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda ||\boldsymbol{w}||_2^2$$

2. Least absolute shrinkage and selection operator (LASSO):

$$L(\boldsymbol{w})_{Lasso} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda ||\boldsymbol{w}||_1$$

3. Elastic net

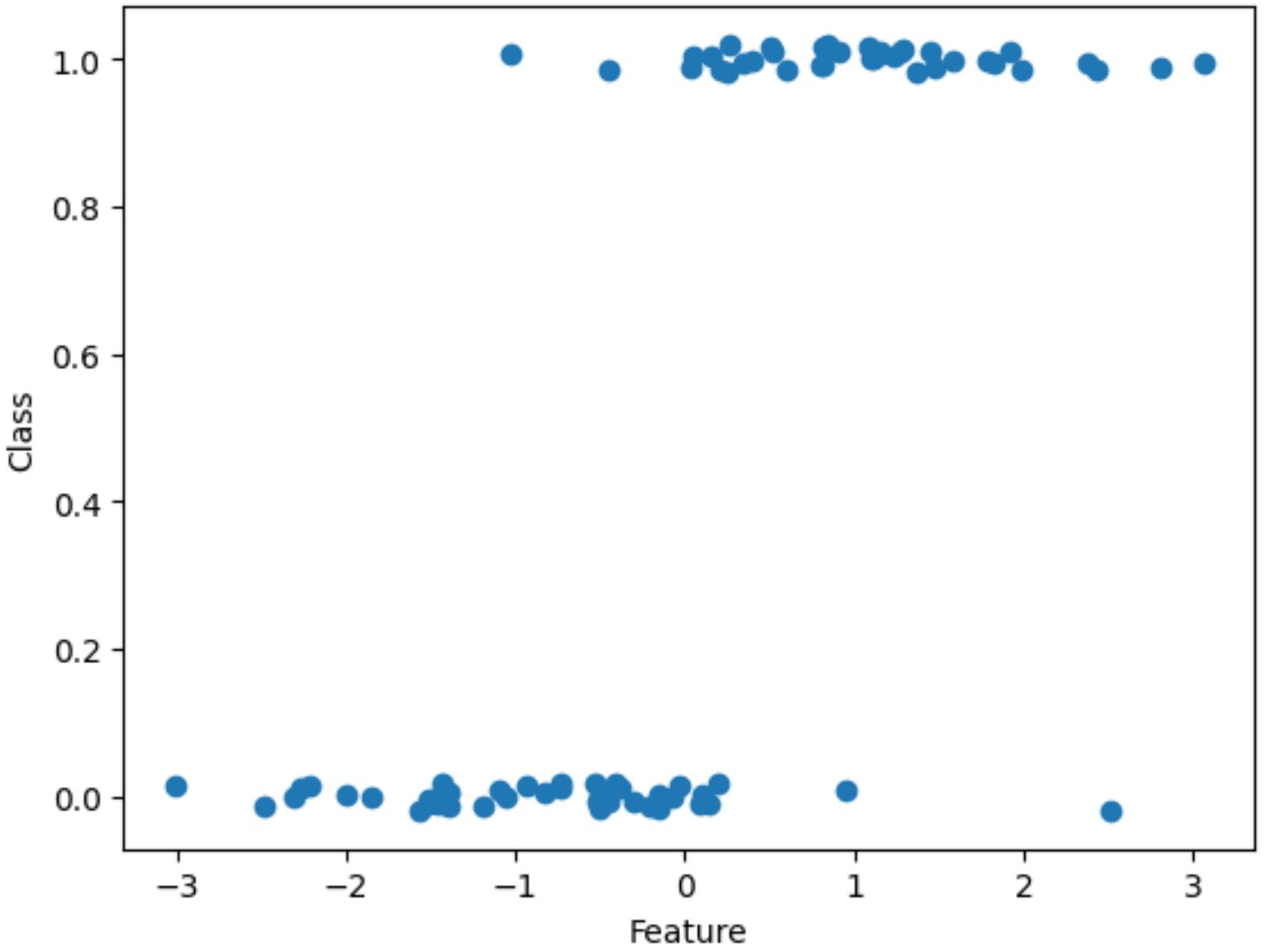
$$L(\boldsymbol{w})_{Elastic\ Net} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda_2 ||\boldsymbol{w}||_2^2 + \lambda_1 ||\boldsymbol{w}||_1$$

Practically: need careful consideration of what x is. Dependent on physics, better approach can be DCNNs, causal methods, etc.

Logistic regression

https://www.weknowpets.com.au/blogs/news/are-guinea-pigs-the-right-pet-for-your-family

When do we need logistic regression?



Logistic regression

Probability of event: p

Odds: $\frac{p}{(1-p)}$

Logit: $\text{logit}(p) = \log \frac{p}{(1-p)}$

May the odds ever be in your favor!

Logistic model assumes that there is a linear relationship between the weighted inputs and the log-odds

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Logistic regression

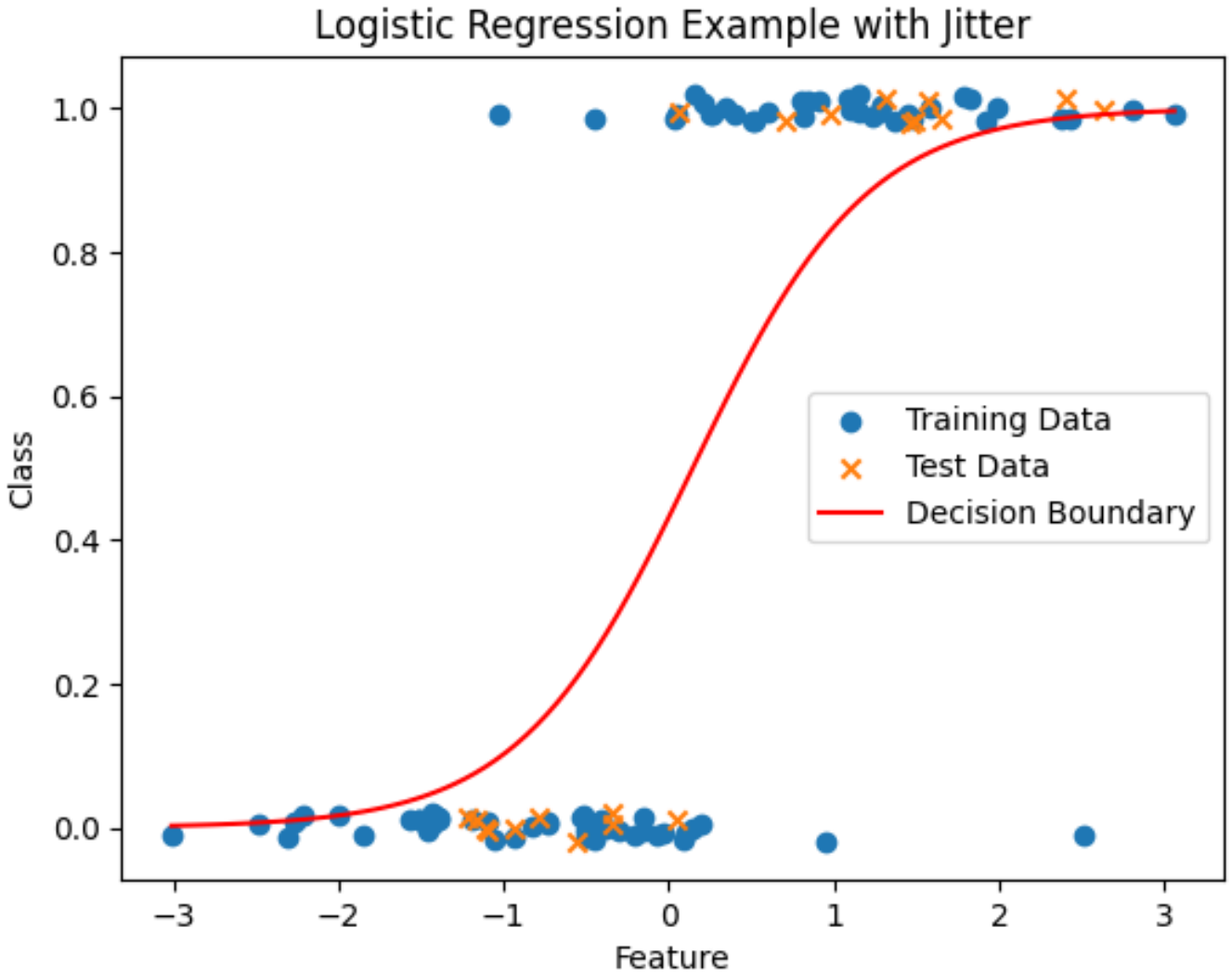
Logistic model: $$\ln \frac{p}{1-p} = W^T x + b$$

Logistic function: $$\sigma(z) = \frac{1}{1+e^{-z}}$$

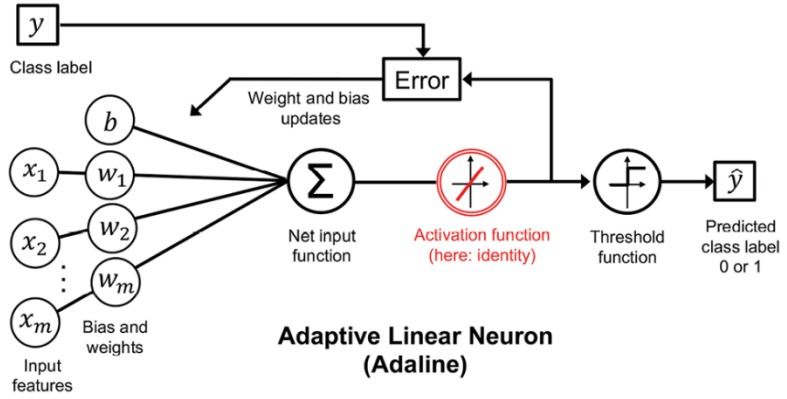
$$z = W^T x + b$$

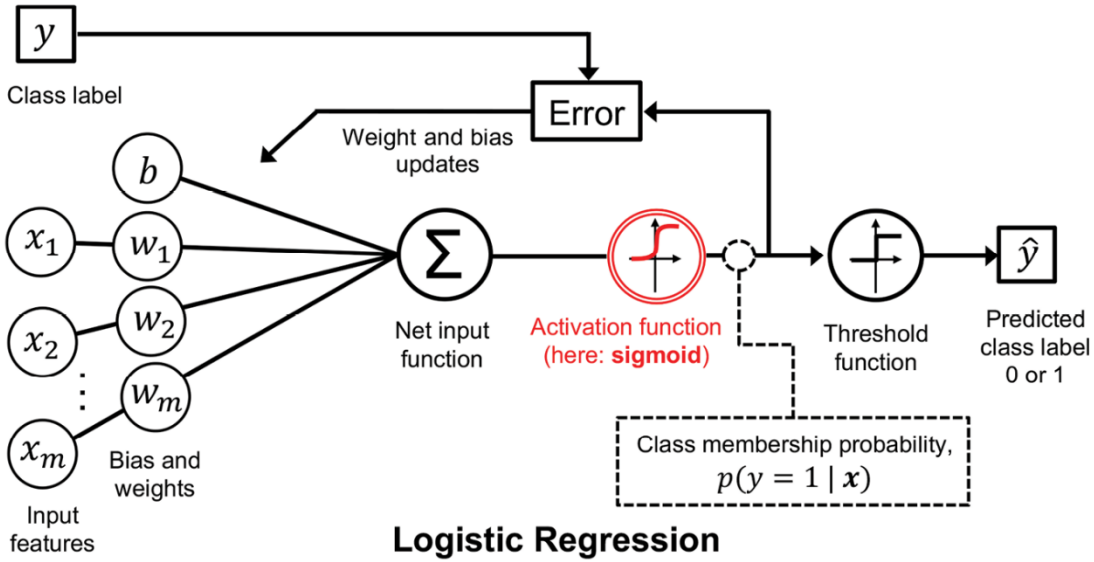
Goal of logistic regression: Predict the "true" proportion of success, p, at any value of the predictor.

Logistic regression prediction



Logistic regression vs. Adaline



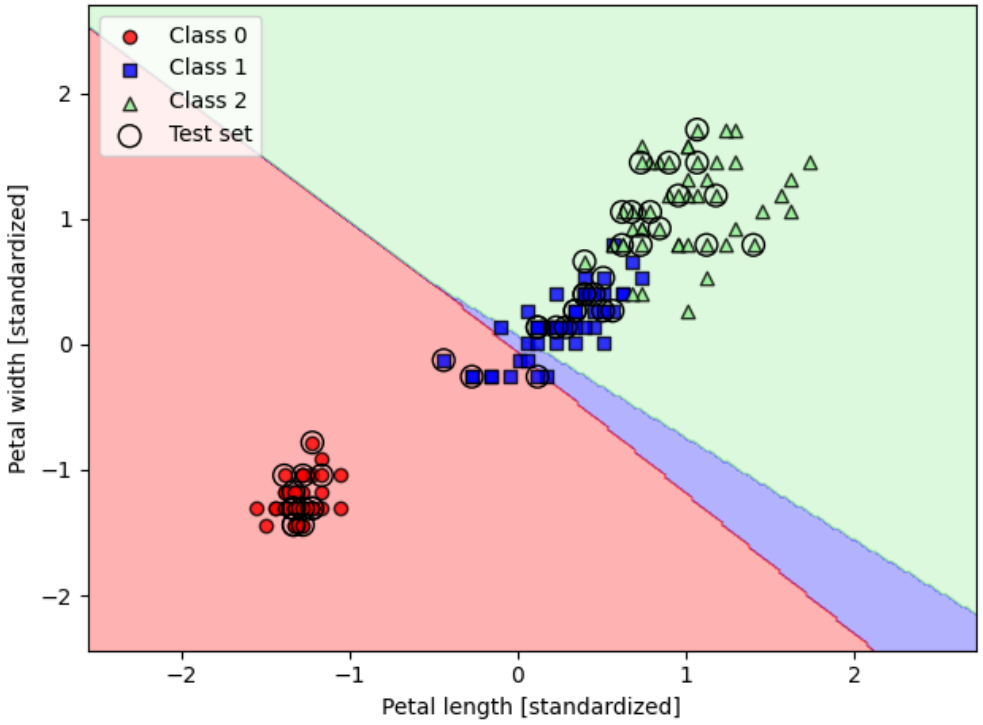


Similar to Adaline, but uses sigmoid as activation function

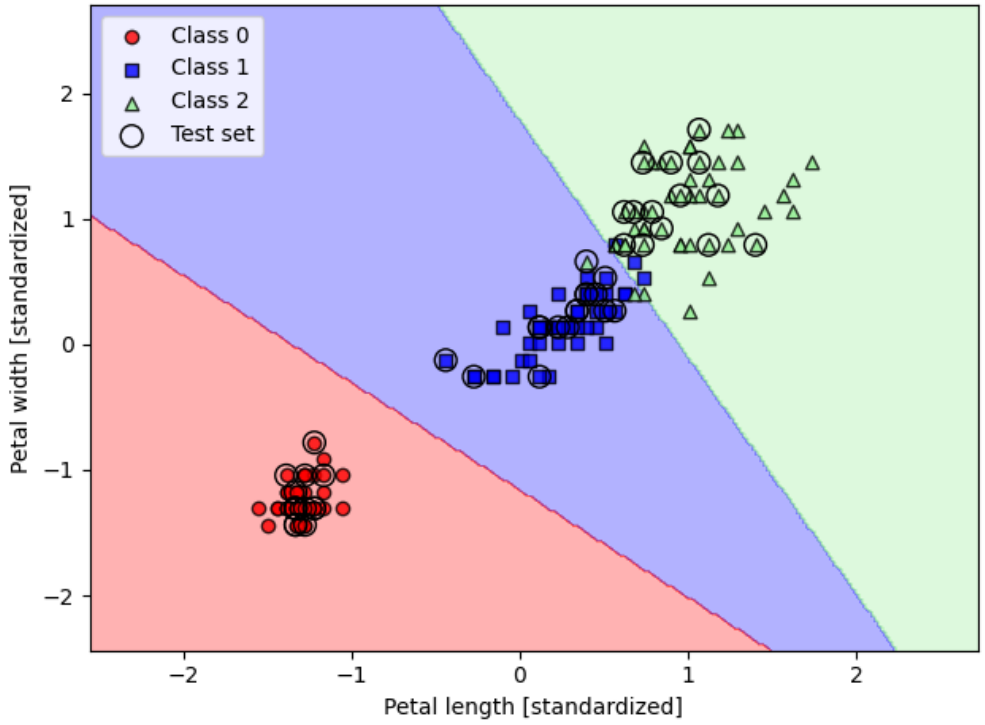
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Regularization in logistic regression

C = 0.001



C = 100



1/C

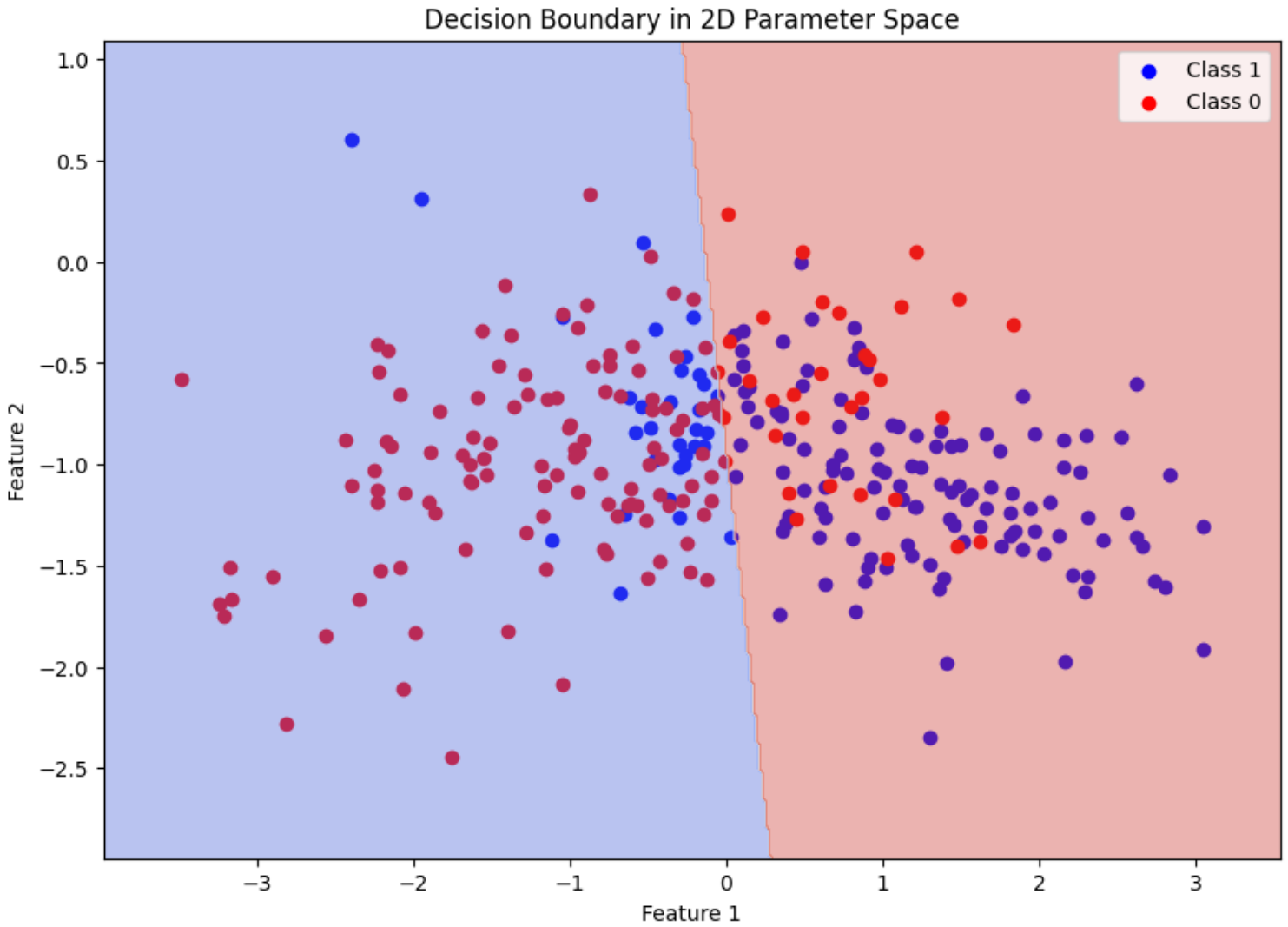
$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\left[-y^{(i)}\log\left(\sigma\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - \sigma\left(z^{(i)}\right)\right)\right] + \frac{\lambda}{2n}\|\boldsymbol{w}\|^2$$

We have (already) several tools in the toolbox

- Classifier and Regression Trees (CART), Random Forest
- Perceptron
- Adaline
- Logistic regression

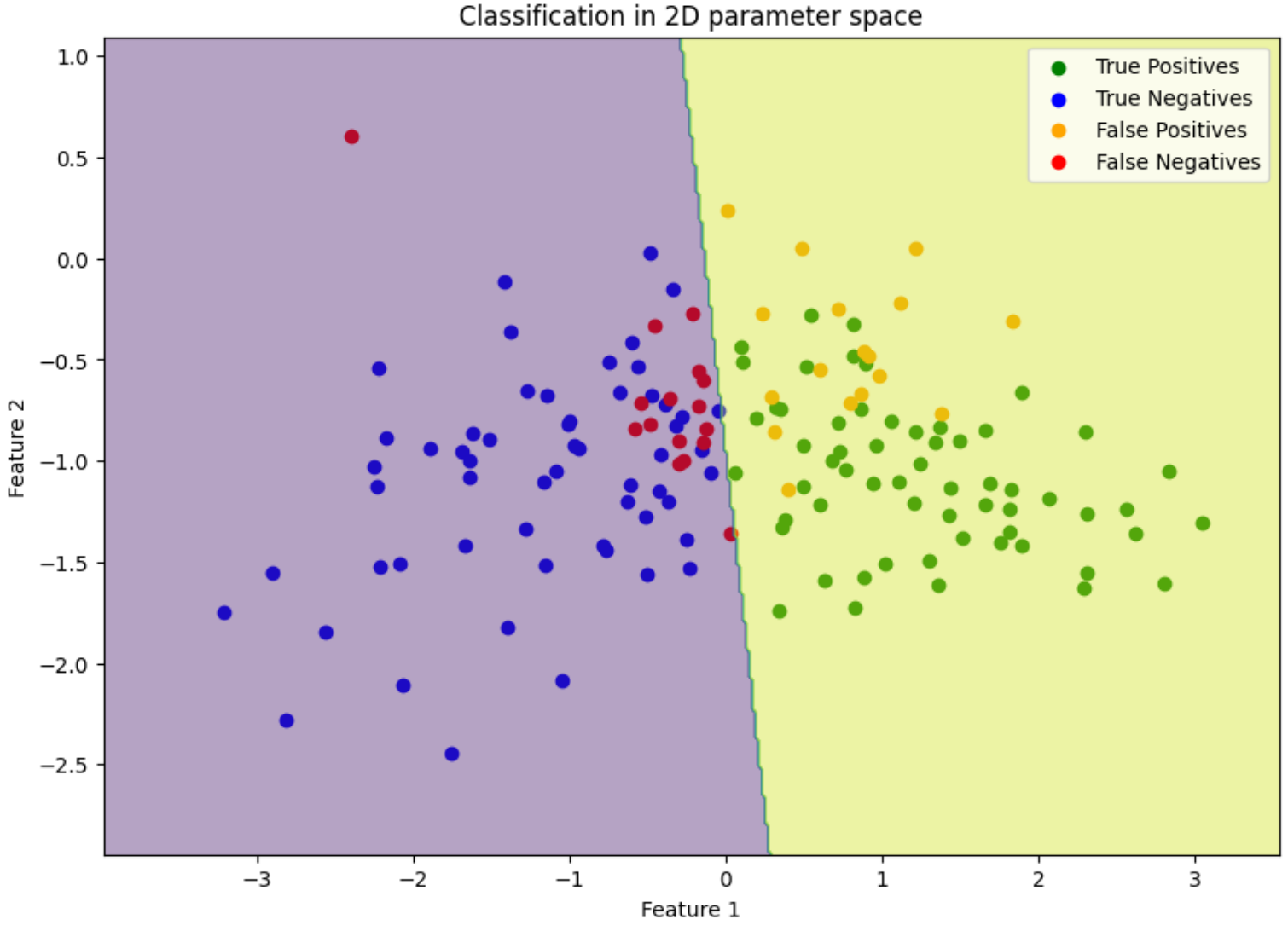
There are more classifiers in scikit-learn – and all of them use the same Pythonic methods .fit(), .predict(), and (many) yield probabilities

Let’s consider the binary classifier



- Classification is not perfect
- We have examples of Class 1 identified as 0, and Class 0 as 1

Let’s consider the binary classifier



- Does the mis-identification matter?

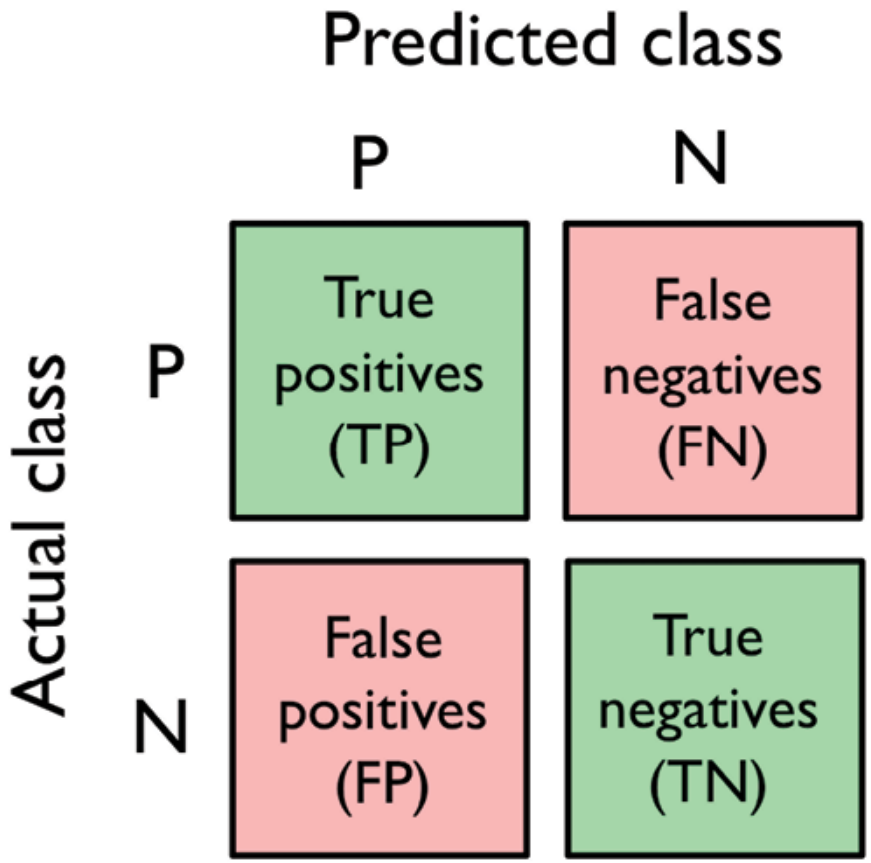
It depends on the problem!

- Molecular prediction: toxicity
- Medical tests: tumor, COVID, flu, etc…
- Finance: fraud detection
- Metal detector: sensitivity threshold
- Object detection: face identification
- Quality control: ripe/rotten

Standard assumptions

- Standard Cost Model
  - correct classification costs 0
  - cost of misclassification depends only on the class, not on the individual example
  - over a set of examples costs are additive
- Costs or Class Distributions:
  - are not known precisely at evaluation time
  - may vary with time
  - may depend on where the classifier is deployed
- True FP and TP do not vary with time or location, and are accurately estimated.

Basic definitions



Error:

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}$$

Accuracy:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Same definition – different names

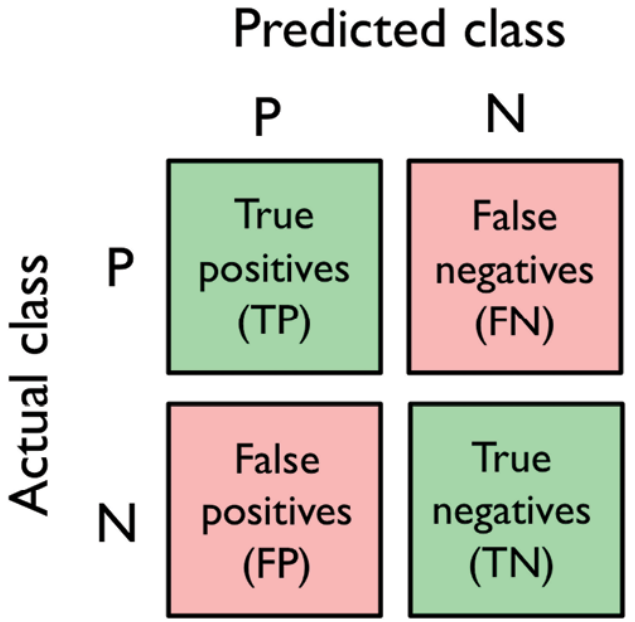
Event happens?	Forecast says event will happen?	
	Yes	No
Yes	Hit	Miss
No	False Alarm	Correct Rejection

From Simon J. Mason, International Research Institute for Climate Prediction, Columbia University

Basic definitions

False positive rate: $$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

True positive rate: $$TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$



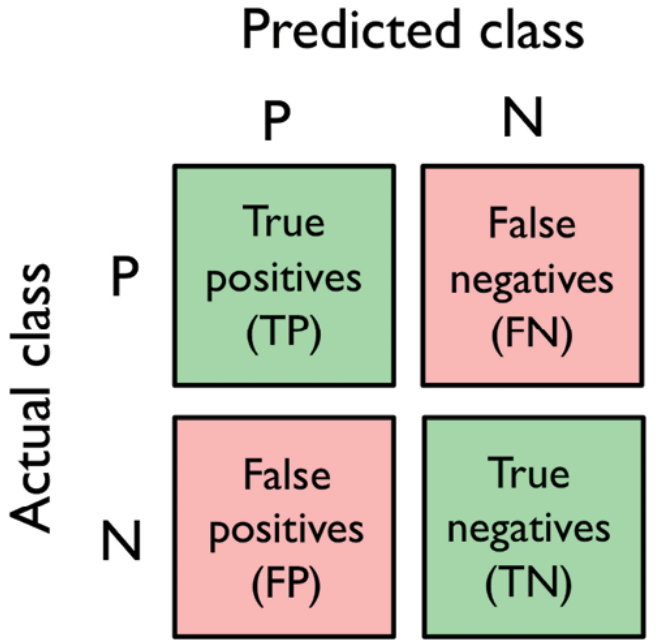
Useful for imbalanced class problems

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Basic definitions

Recall: $$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

Precision: $$PRE = \frac{TP}{TP + FP}$$



For medicine (cancer detection):

Optimizing for recall helps with minimizing the chance of not detecting a malignant tumor. However, this comes at the cost of predicting malignant tumors in patients although the patients are healthy (a high number of FPs).

Optimize for precision, on the other hand, we emphasize correctness if we predict that a patient has a malignant tumor. However, this comes at the cost of missing malignant tumors more frequently (a high number of FNs).

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

There is always more….

F1 Score:

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

David M. W. Power *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*, https://arxiv.org/abs/2010.16061.

Matthews correlation coefficient:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

D. Chicco and G. Jurman, *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation* by *BMC Genomics*. pp. 281-305, 2012, https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

… and more

		Predicted condition		Sources: [17][18][19][20][21][22][23][24][25] view · talk · edit	
	Total population $= P + N$	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP)**, hit	**False negative (FN)**, type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	**False positive (FP)**, type I error, false alarm, overestimation	**True negative (TN)**, correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{PN}$ $= 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Limitation of the scalar measures

- **A scalar does not tell the whole story.**
  - There are fundamentally two numbers of interest (FP and TP), a single number invariably loses some information.
  - How are errors distributed across the classes ?
  - How will each classifier perform in different **testing** conditions (costs or class ratios other than those measured in the experiment) ?

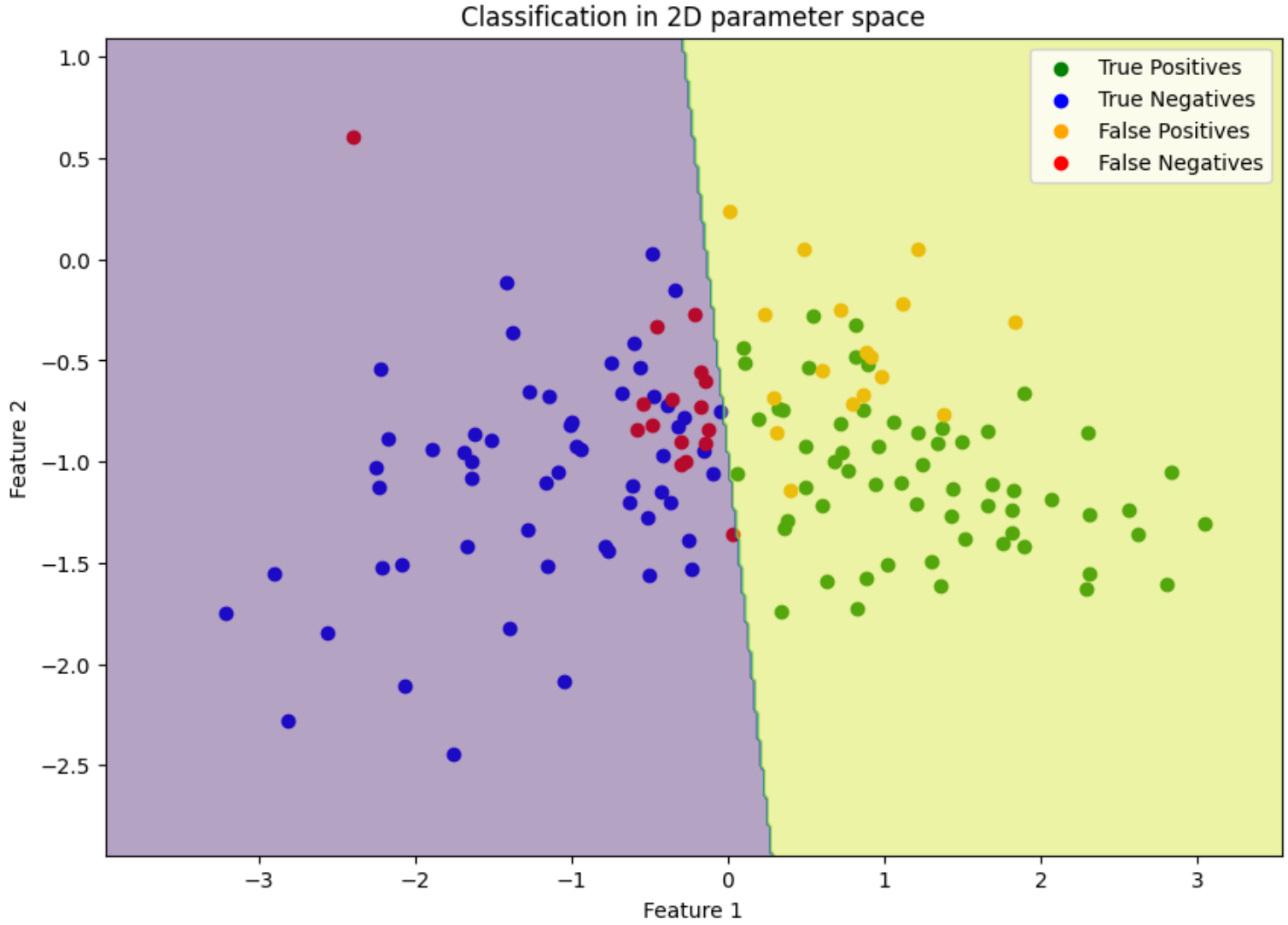
- **A scalar imposes a linear ordering on classifiers.**
  - what we want is to identify the conditions under which each is better.

Limitations of Scalar Measures

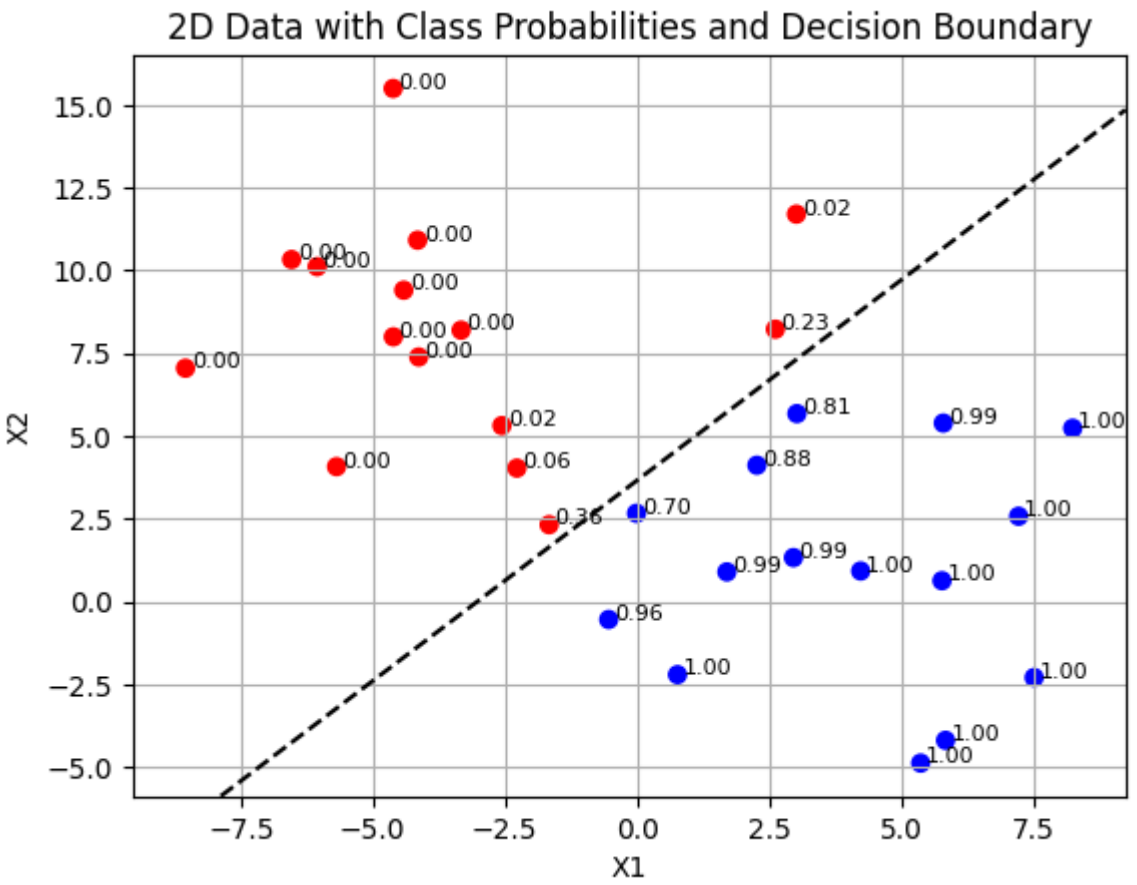
- **A table of scalars is just a mass of numbers.**
  - No immediate impact
  - Poor way to present results in a paper
  - Equally poor way for an experimenter to analyze results

- **Some scalars (accuracy, expected cost) require precise knowledge of costs and class distributions.**
  - Often these are not known precisely and might vary with time or location of deployment.

Can we tune the classifier?



- Depending on context, cost of FP and FN errors can be different
- Can we tune the classifiers to match the business case?



1.Train the Classifier on train data

2.Get Class Probabilities for test dataset. This will give the probability of each instance belonging to the positive class.

3.Order Probabilities: Sort by the predicted probabilities of belonging to the positive class.

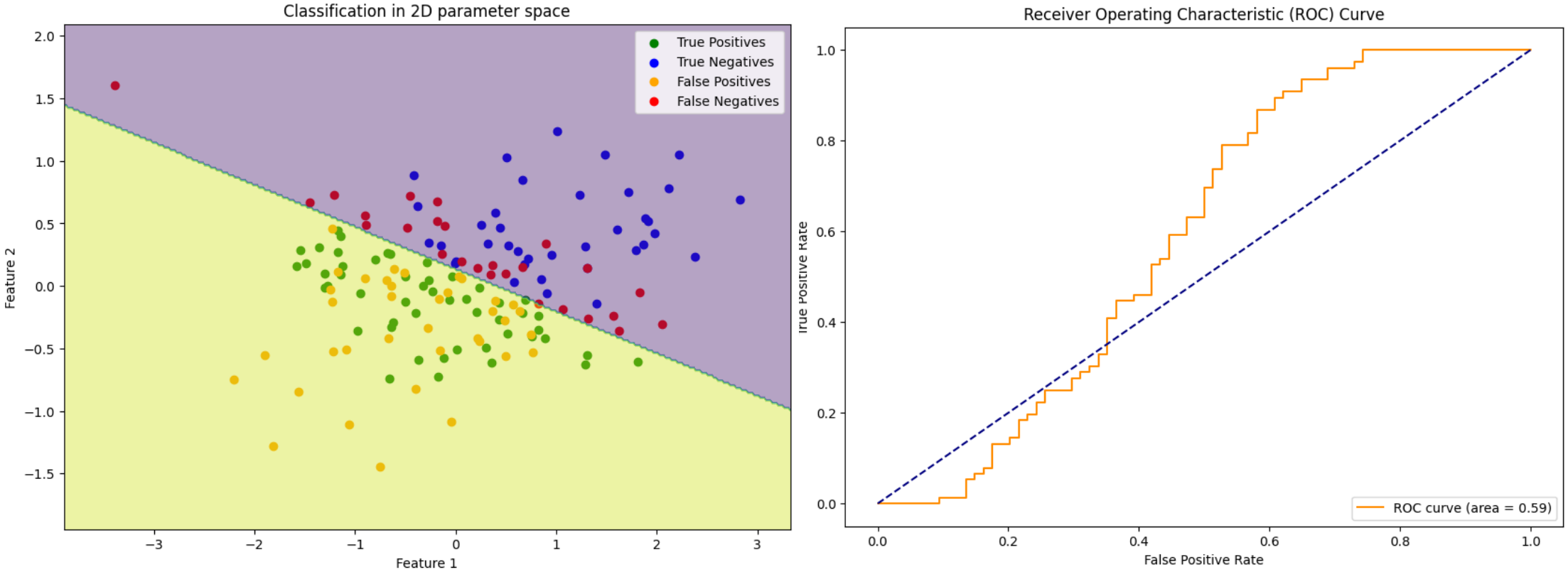
4.Calculate TPR and FPR for Each Threshold: For every unique probability value, treat it as a threshold. Instances with probabilities above (or equal to) the threshold are classified as positive, and those below are classified as negative. For each threshold, calculate TPR and FPR.

5.Plot ROC Curve: Plot TPR vs. FPR for each threshold, resulting in the ROC curve.

6.Calculate AUC: Compute the Area Under the Curve (AUC)

Receiver-Operator Characteristics (ROC)

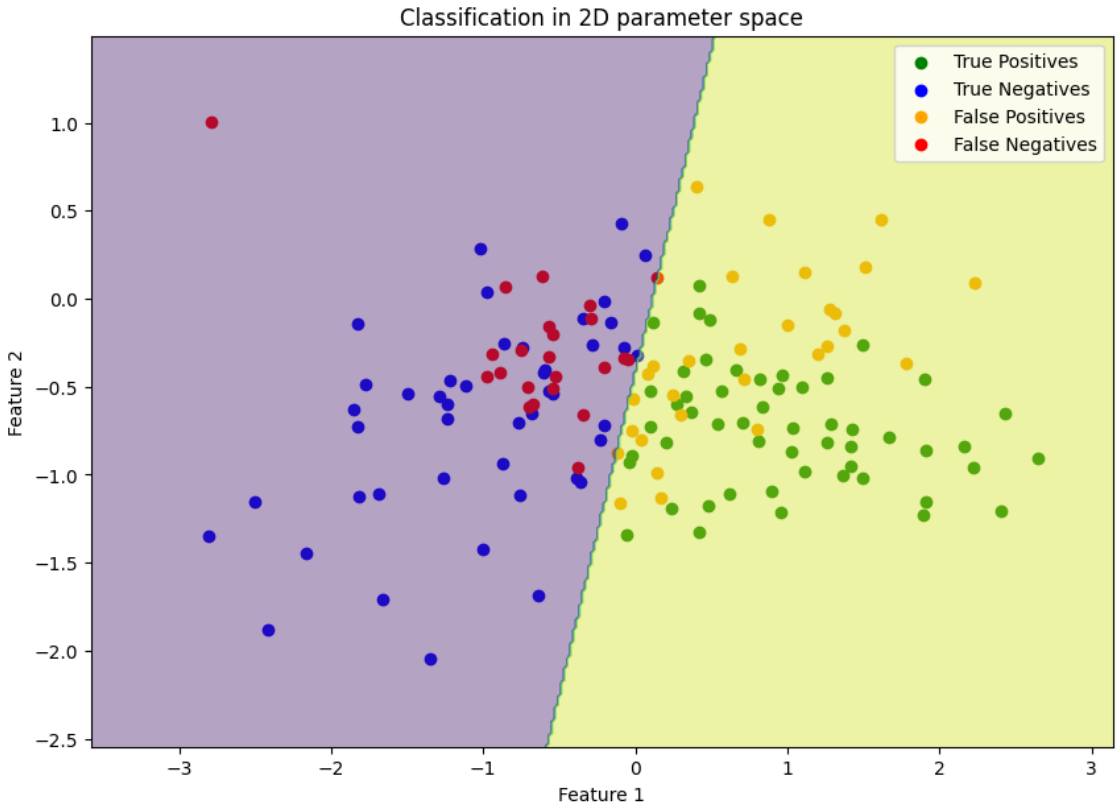
Class separation 0

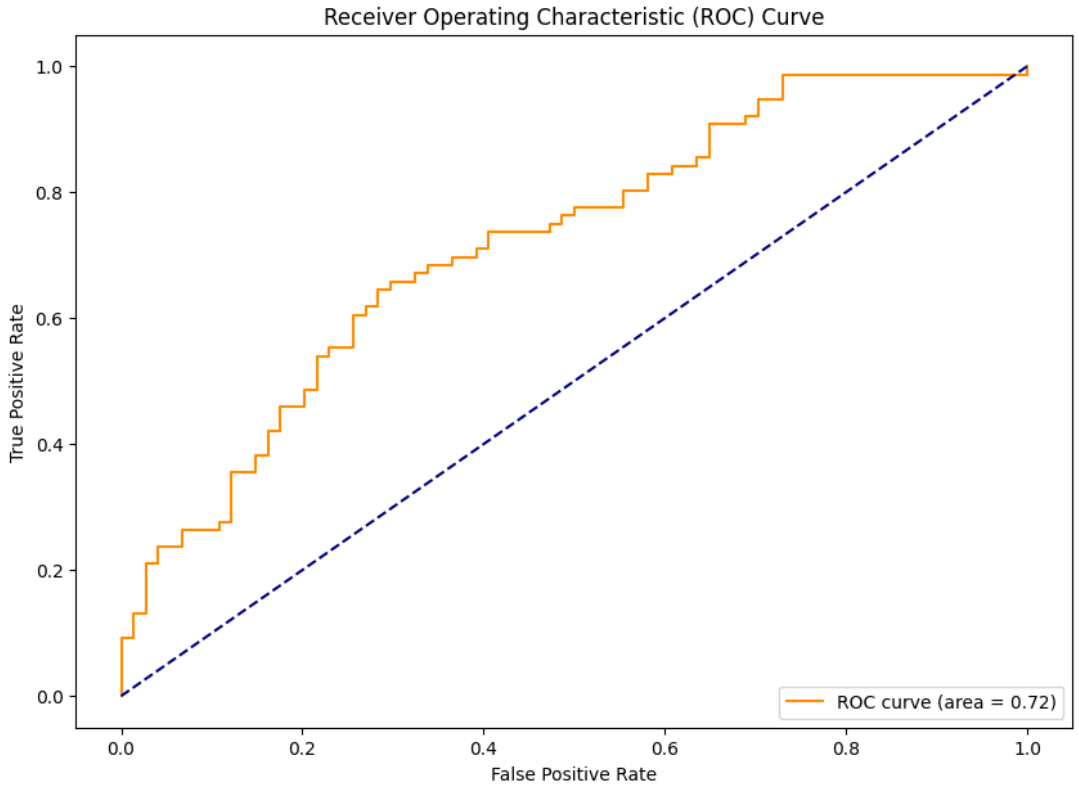


The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 0.6

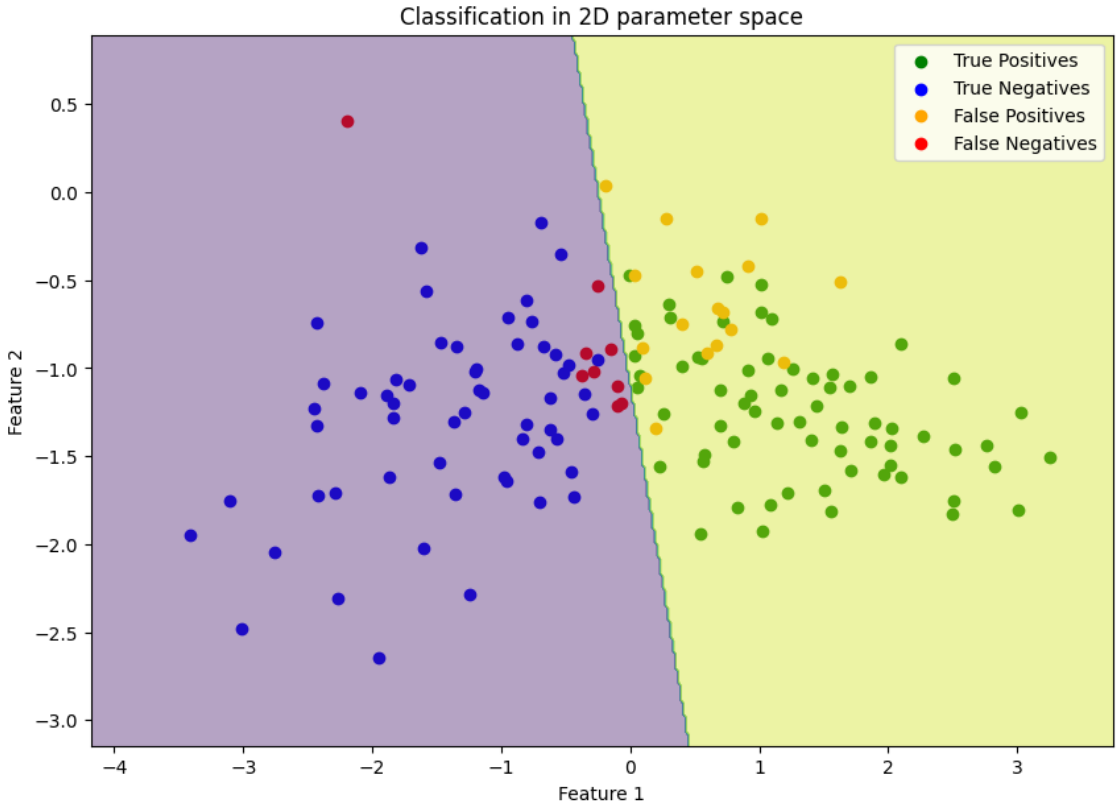


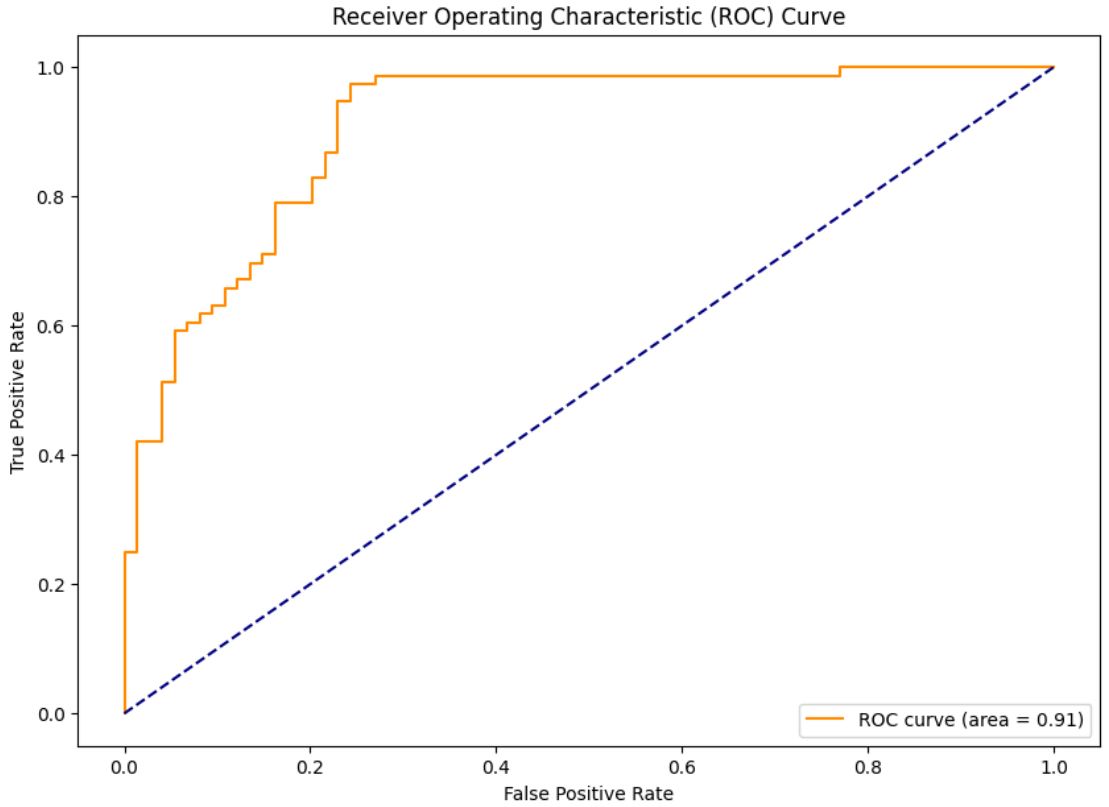


The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 1.2



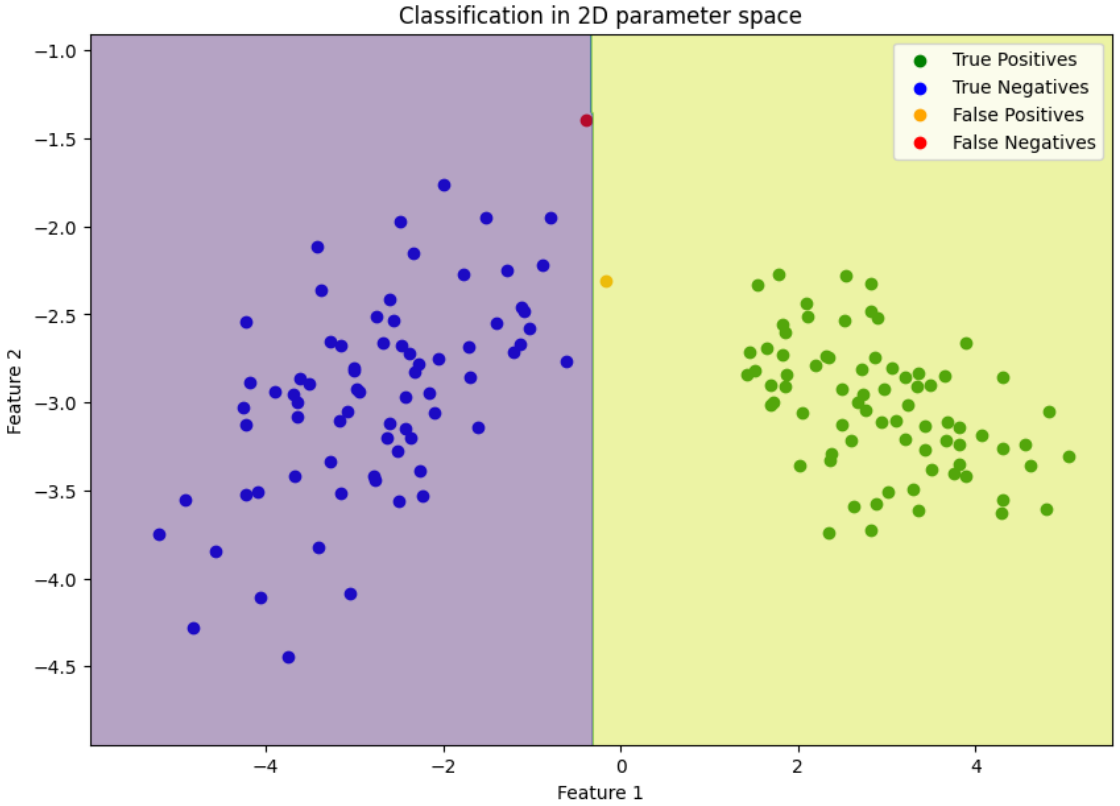


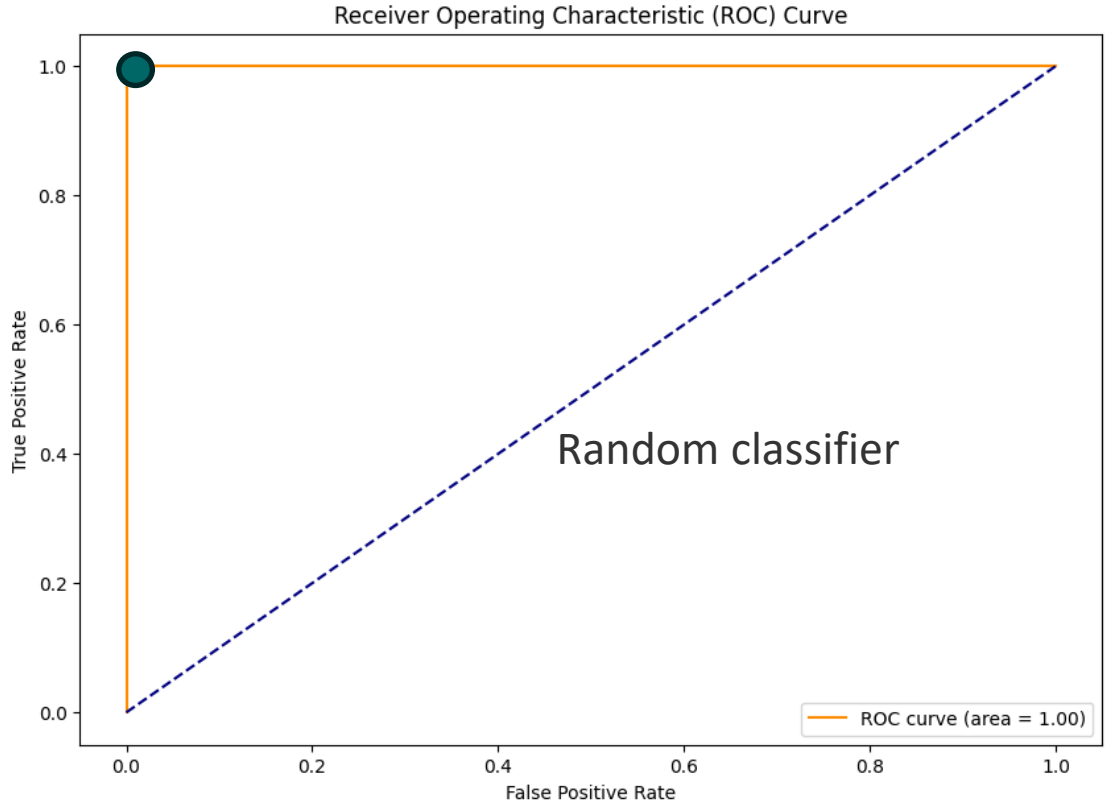
The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 3

Ideal classifier





The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Advantage of ROC approach:

- A classifier produces a single ROC point.
- If the classifier has a "sensitivity" parameter, varying it produces a series of ROC points (confusion matrices).
- Alternatively, if the classifier is produced by a learning algorithm, a series of ROC points can be generated by varying the class ratio in the training set.

- ROC curves allow:
  - Visualization of all tradeoffs a model can make
  - Comparison of models across types of tradeoffs
- AUC – an aggregate measure of quality across tradeoffs
- Operating points are the tradeoff selected for specific application

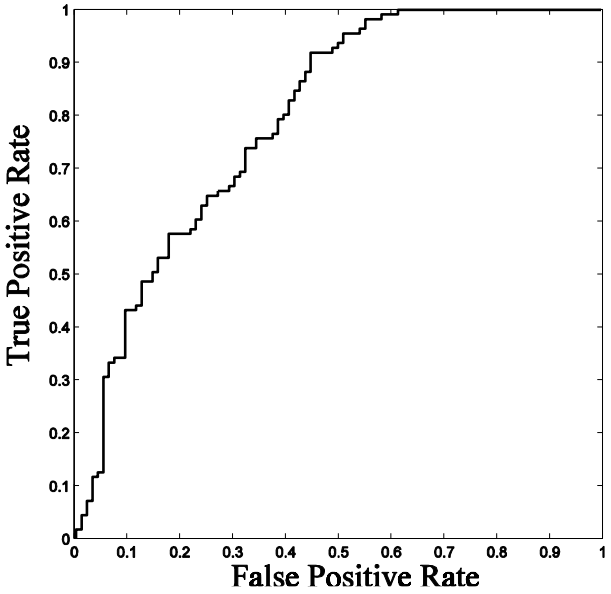
From ROC curves to Business models

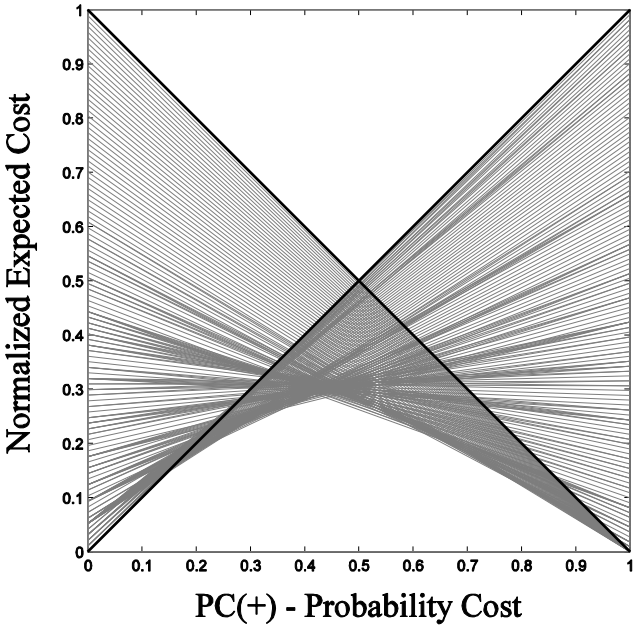
$Y = FN \bullet X + FP \bullet (1-X)$

So far, $X = p(+)$, making Y = error rate

$$X = \frac{p(+) \bullet C(-|+)}{p(+) \bullet C(-|+) + (1-p(+)) \bullet C(+|-)}$$

Y = expected cost normalized to [0,1]





Cost curves enable easy visualization of

- Average performance (expected cost)
- Operating range
- Confidence intervals on performance
- Difference in performance and its significance

From slides by Robert Holte, Computing Science Dept., University of Alberta **Cost-Sensitive Classifier Evaluation**

What if we have multiple classes?

1. One-vs-One (OvO) or One-vs-Rest (OvR) approach: This involves decomposing the multi-class classification problem into multiple binary classification problems and plotting a ROC curve for each one.

1. **One-vs-Rest (OvR)**: For K classes, you train K separate binary classifiers. For each classifier, one class is treated as positive while all others are treated as negative. We will have K separate ROC curves.

2. **One-vs-One (OvO)**: For K classes, we train a binary classifier for every pair of classes. This results in $K(K-1)/2$ classifiers and ROC curves.

For multi-class classification, we can compute the AUC for each class individually and then average them, providing a single scalar value that summarizes the performance.

What if we have multiple classes?

1. **Macro-averaging and Micro-averaging**:

    1. **Macro-averaging**: Calculate the ROC curve for each class and then average them to get a single ROC curve.

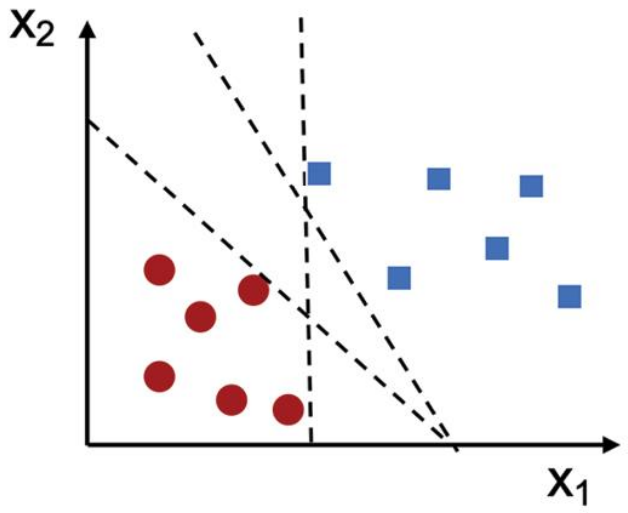
    2. **Micro-averaging**: Aggregate the outcomes of individual classifications (over all classes) to compute the ROC curve. This means you'll combine all the true positives, false positives, true negatives, and false negatives across all classes and then compute the TPR and FPR.

What if we have imbalanced classes?

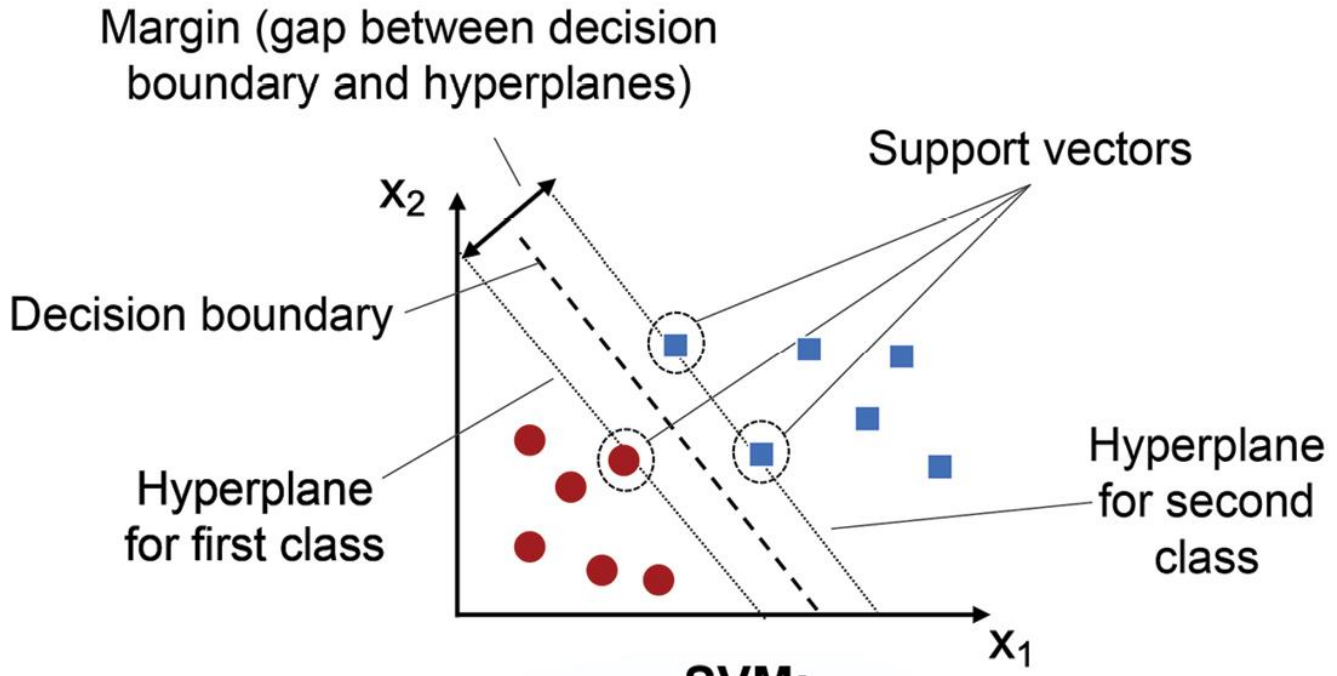
- Assign a larger penalty to wrong predictions on the minority class. Via scikit-learn, adjusting such a penalty is as convenient as setting the *class_weight* parameter to *class_weight*='balanced', which is implemented for most classifiers
- Upsampling the minority class or downsampling the majority class
- Generation of synthetic training examples

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Support Vector Machines



Which hyperplane?



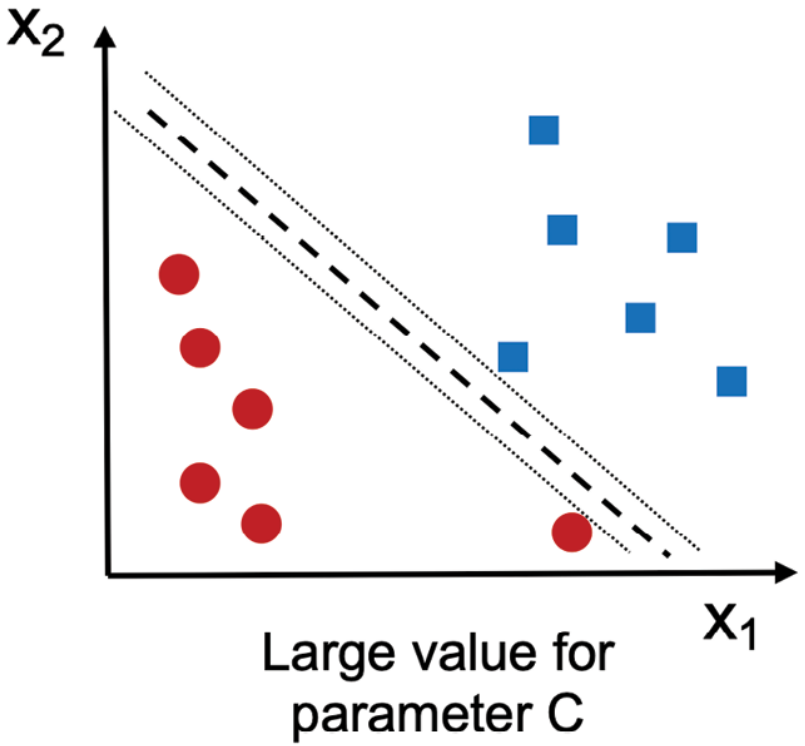
SVM:
Maximize the margin

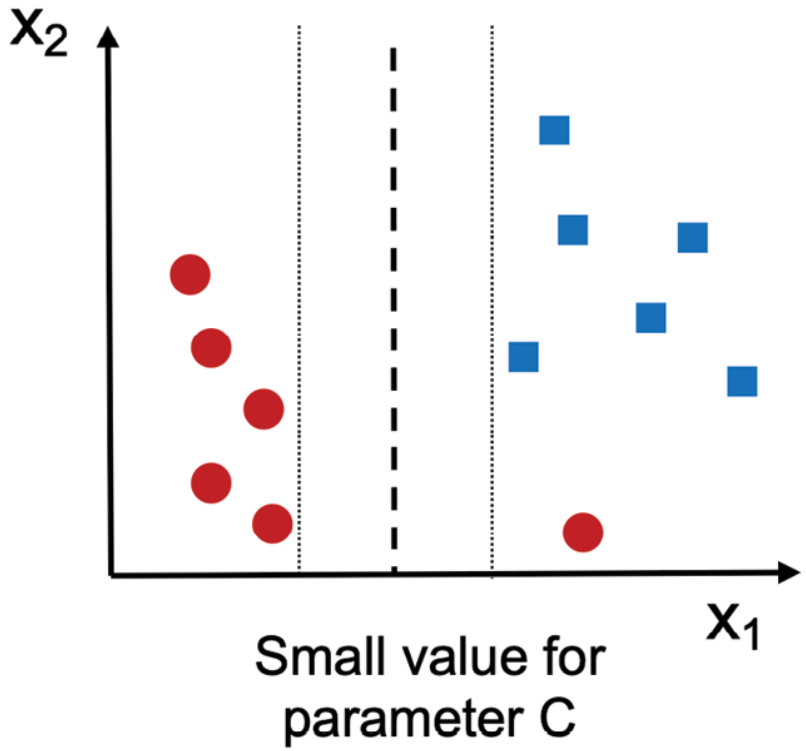
Perceptron: minimize misclassification errors.
SVM: maximize the margin.

The margin is the distance between the separating hyperplane (decision boundary) and the training examples that are closest to this hyperplane, which are called **support vectors**.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Regularization in SVMs

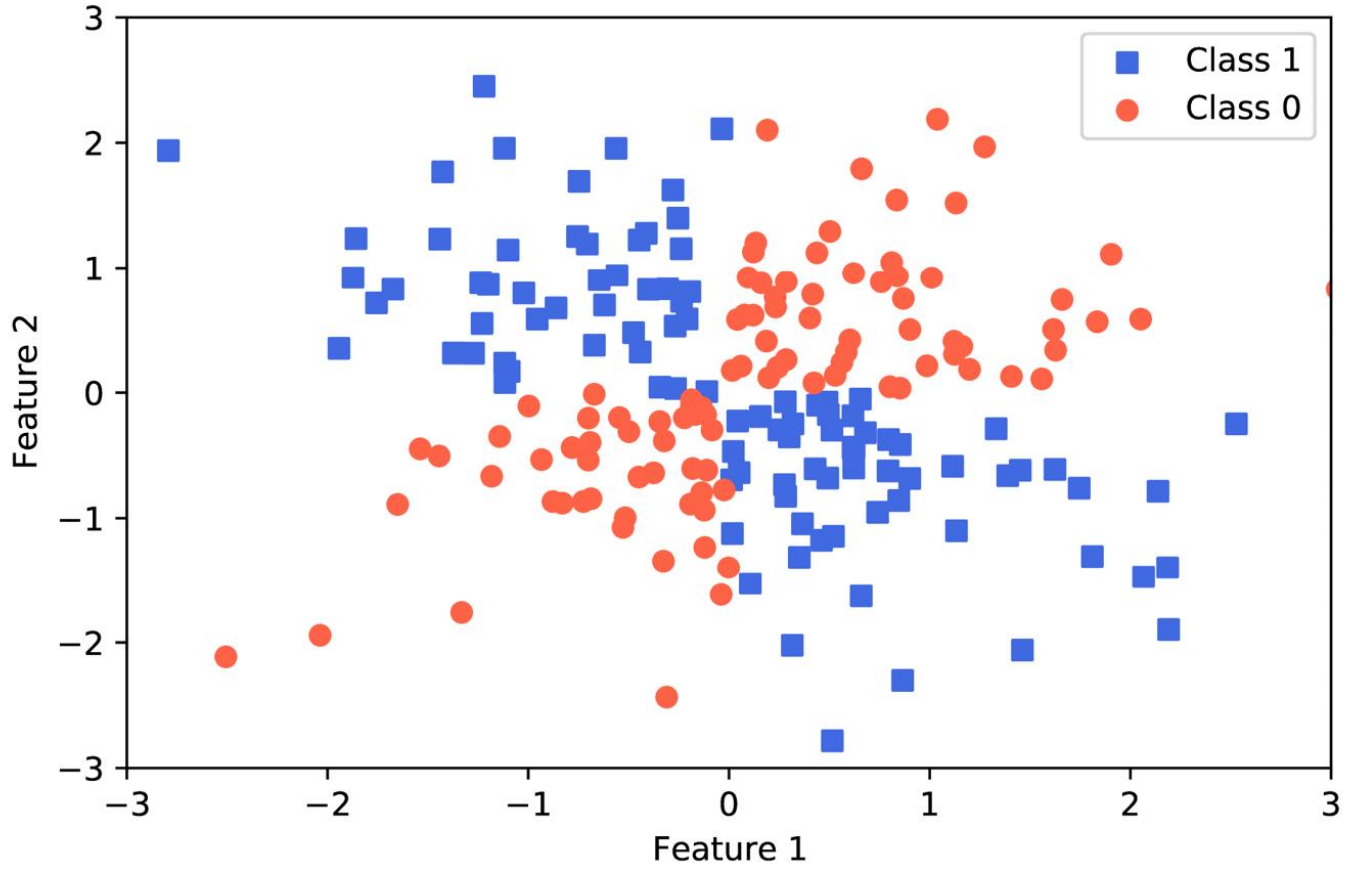




Large values of *C* correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for *C*.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

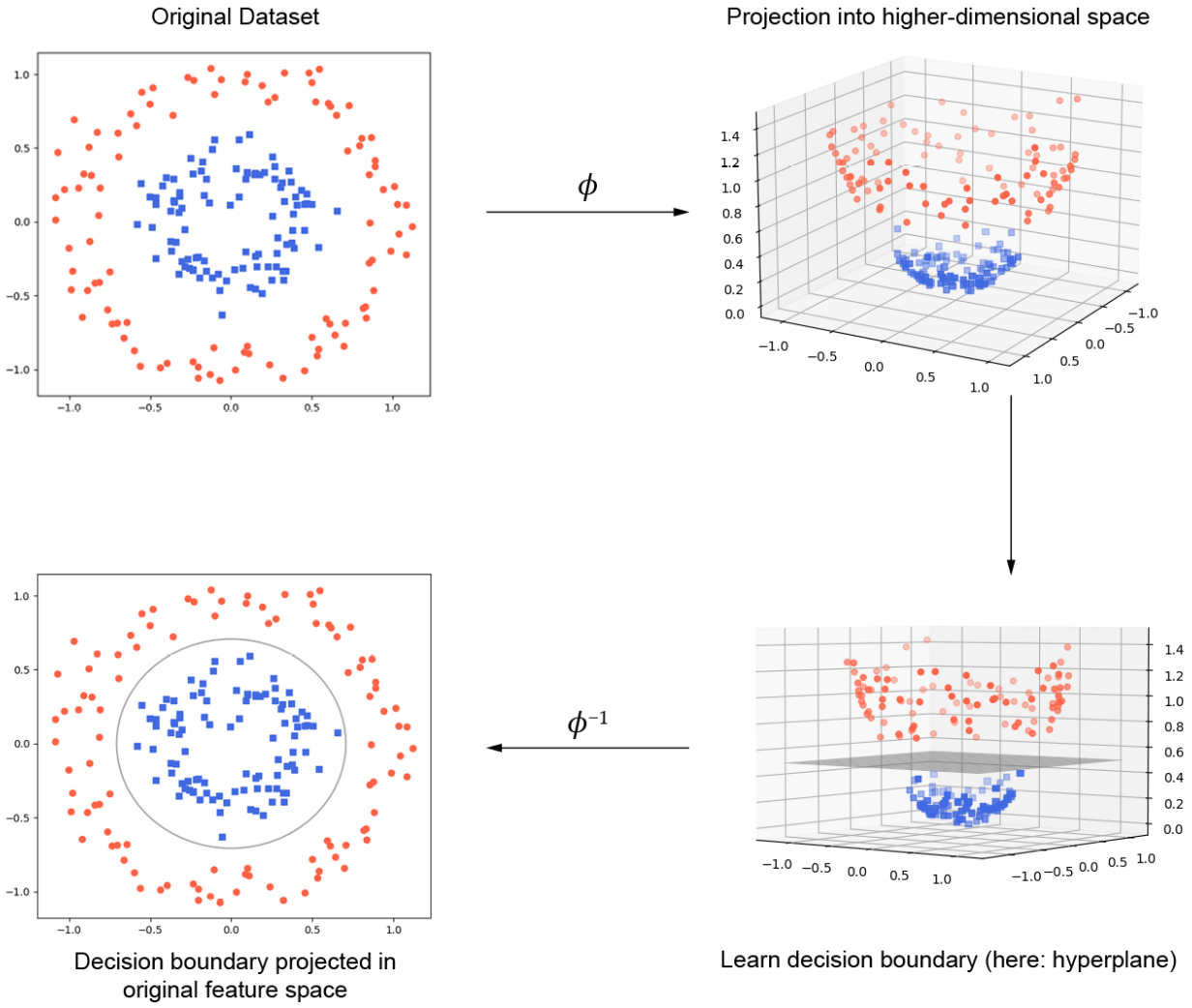
Kernel SVM: Motivation



Non-linearly separable problem

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Kernel SVM



The idea behind **kernel methods** for dealing with linearly inseparable data is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping function, ϕ , where the data becomes linearly separable.

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Kernel SVM

- To train an SVM, in practice, we need to replace the dot product $\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$ by $\phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$.

- To save the expensive step of calculating this dot product between two points explicitly, we define a **kernel function** $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$
-
- One of the most widely used kernels is the **radial basis function** (**RBF**) kernel, or the **Gaussian kernel**:

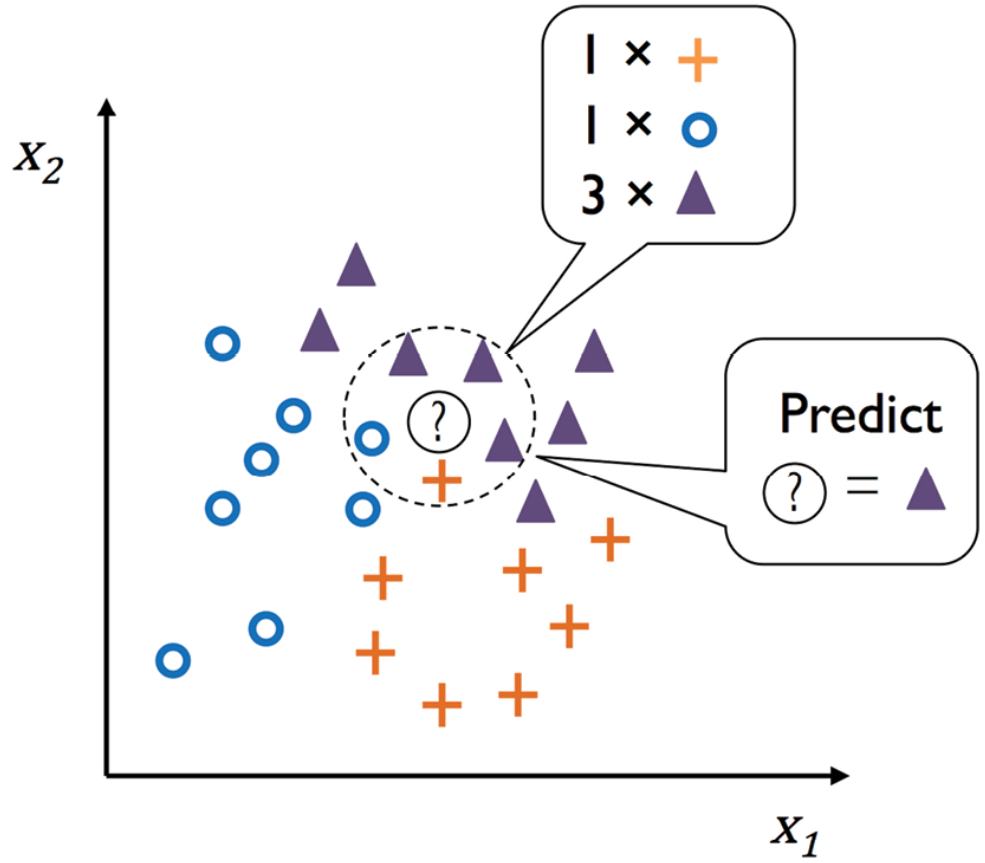
$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\gamma \left\| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \right\|^2\right)$$

γ is a free parameter to be optimized.

- Roughly speaking, the term "kernel" can be interpreted as a **similarity function** between a pair of examples.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

k Nearest Neighbours (kNN) Classifier



1. Choose the number of *k* and a distance metric

2. Find the *k*-nearest neighbors of the data record that we want to classify

3. Assign the class label by majority vote

KNN is an example of a **lazy learner**. It is called "lazy" not because of its apparent simplicity, but because it doesn't learn a discriminative function from the training data but memorizes the training dataset instead.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Parametric vs. Non-parametric methods

Parametric models: we estimate parameters from the training dataset to learn a function that can classify new data points without requiring the original training dataset anymore. Typical examples of parametric models are the perceptron, logistic regression, and the linear SVM.

Non-parametric models can't be characterized by a fixed set of parameters, and the number of parameters changes with the amount of training data. Two examples of non-parametric models are the decision tree classifier/random forest and the kernel (but not linear) SVM.

KNN belongs to a subcategory of non-parametric models described as **instance-based learning**. Models based on instance-based learning are characterized by memorizing the training dataset, and lazy learning is a special case of instance-based learning that is associated with no (zero) cost during the learning process.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Pros and Cons of Memory Based Approaches

- The main advantage of memory-based approach is that the classifier immediately adapts as we collect new training data

- The downside is that the computational complexity for classifying new examples grows linearly with the number of examples in the training dataset in the worst-case scenario—unless the dataset has very few dimensions (features) and the algorithm has been implemented using efficient data structures for querying the training data more effectively.

- Such data structures include k-d tree (https://en.wikipedia.org/wiki/K-d_tree) and ball tree (https://en.wikipedia.org/wiki/Ball_tree), which are both supported in scikit-learn. Furthermore, next to computational costs for querying data, large datasets can also be problematic in terms of limited storage capacities.

- However, in many cases when we are working with relatively small to medium-sized datasets, memory-based methods can provide good predictive and computational performance and are thus a good choice for approaching many real-world problems.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Distances and Neighbours

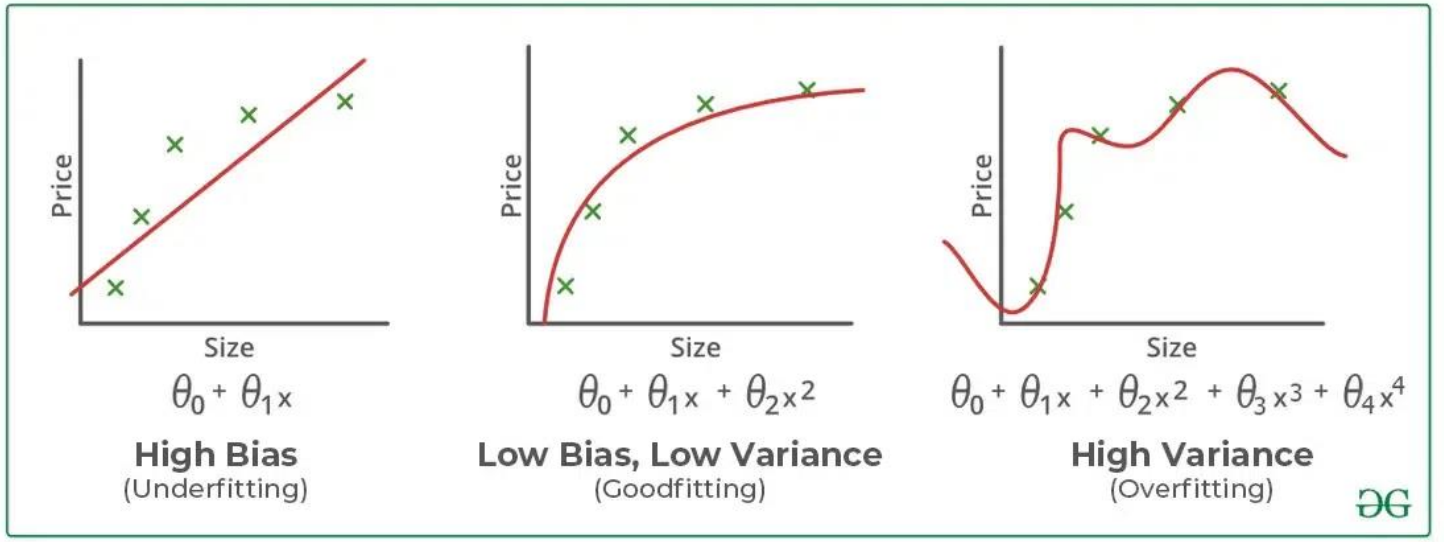
- We need to choose a distance metric that is appropriate for the features in the dataset.
- Usual choice: Minkowski distance, a generalization of the Euclidean (p = 2) and Manhattan (p = 1) distance:

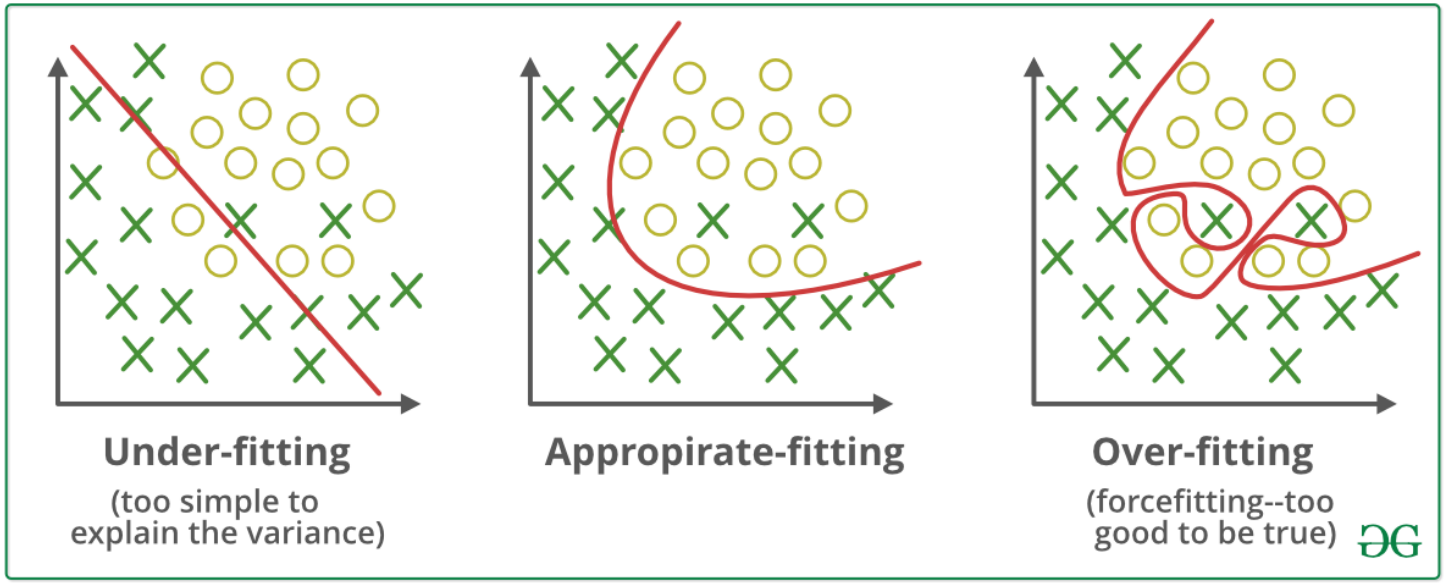
$$d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \sqrt[p]{\sum_k \left| x_k^{(i)} - x_k^{(j)} \right|^p}$$

- Many other distance metrics are available in scikit-learn and can be provided to the metric parameter. They are listed at https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html.
- KNN is very susceptible to overfitting due to the **curse of dimensionality**, where feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset.
- For regression and SVM, we use regularization to avoid this problem.
- We cannot use regularization for decision trees and KNN. Instead, we can use feature selection and dimensionality reduction techniques

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

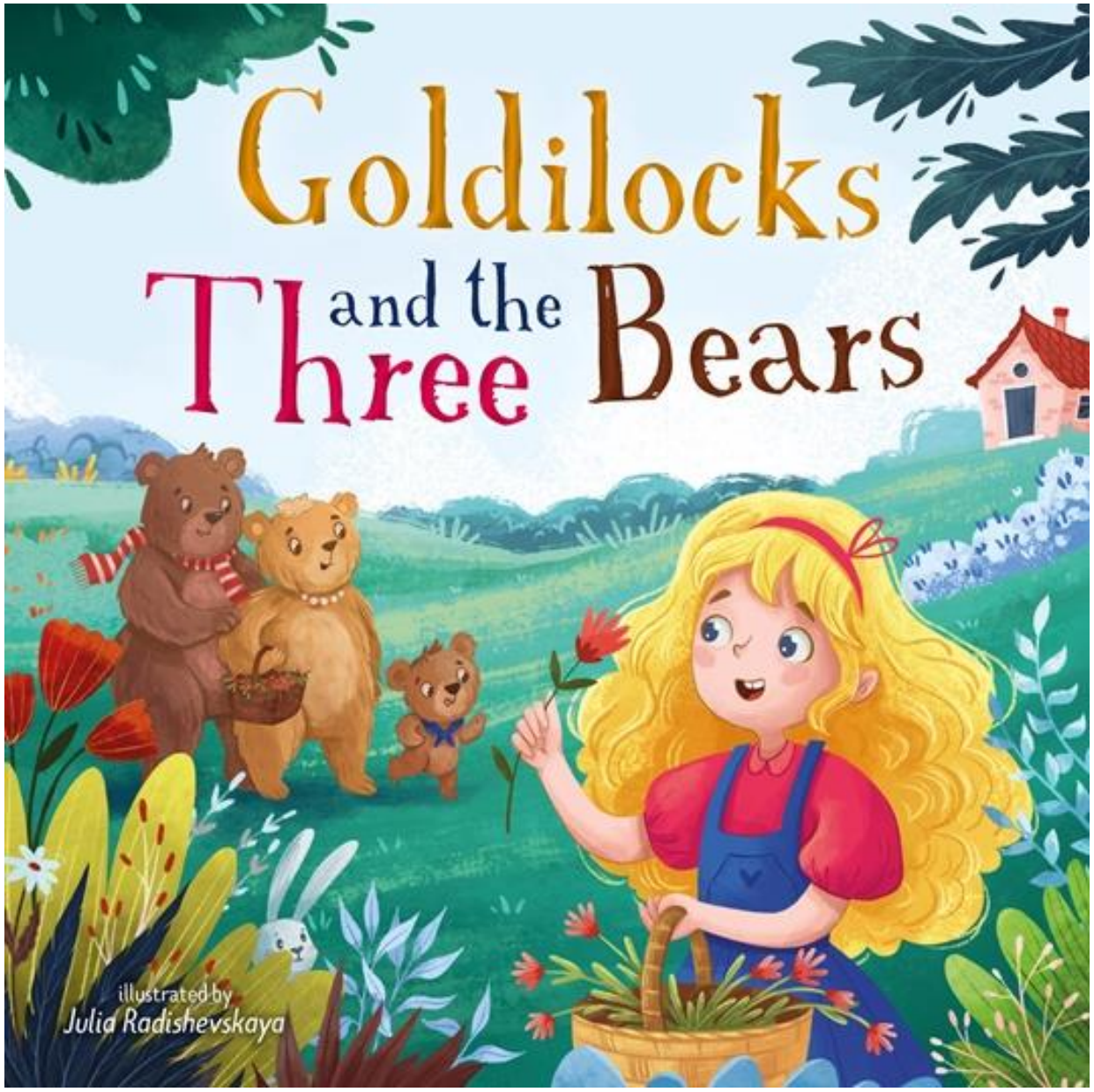
Overfitting and Underfitting





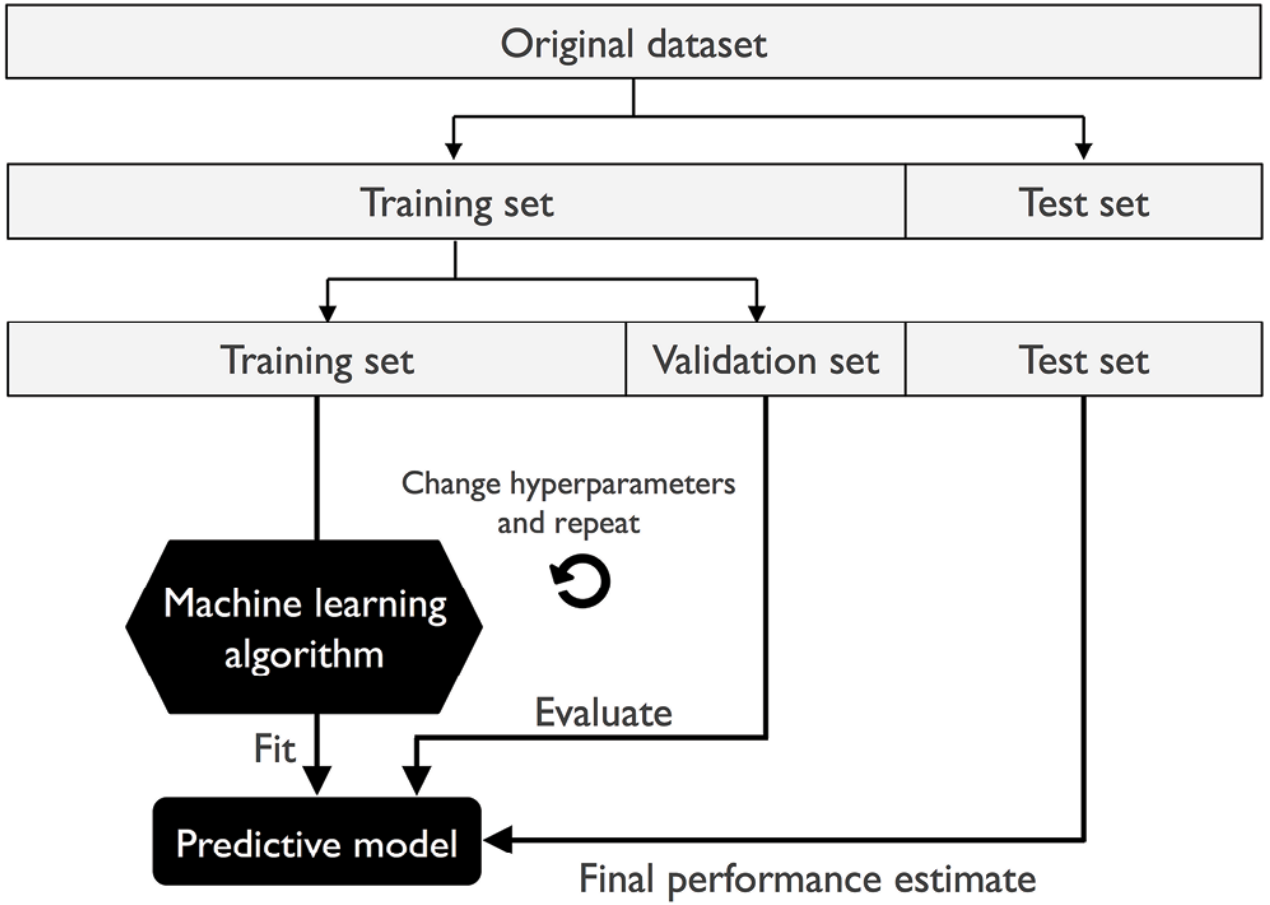
https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Overfitting and Underfitting



Training, testing, and validating

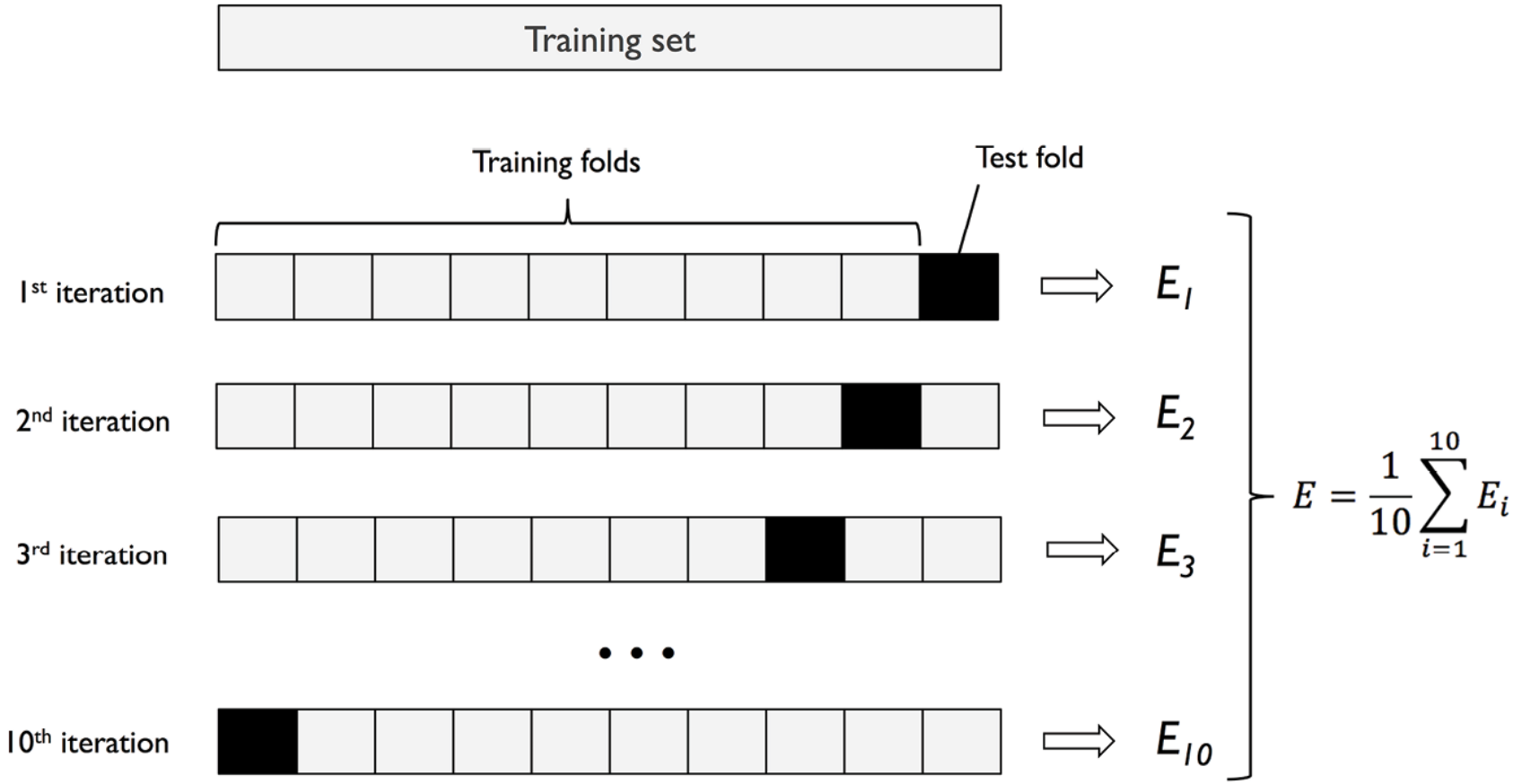
How can we be certain that model that is trained on data we have will perform well in production?



- In real world, we make decisions based on:
  - prior knowledge,
  - intuition,
  - simulations,
  - data,
  - advice
  - ....
- Now imagine we have data only!

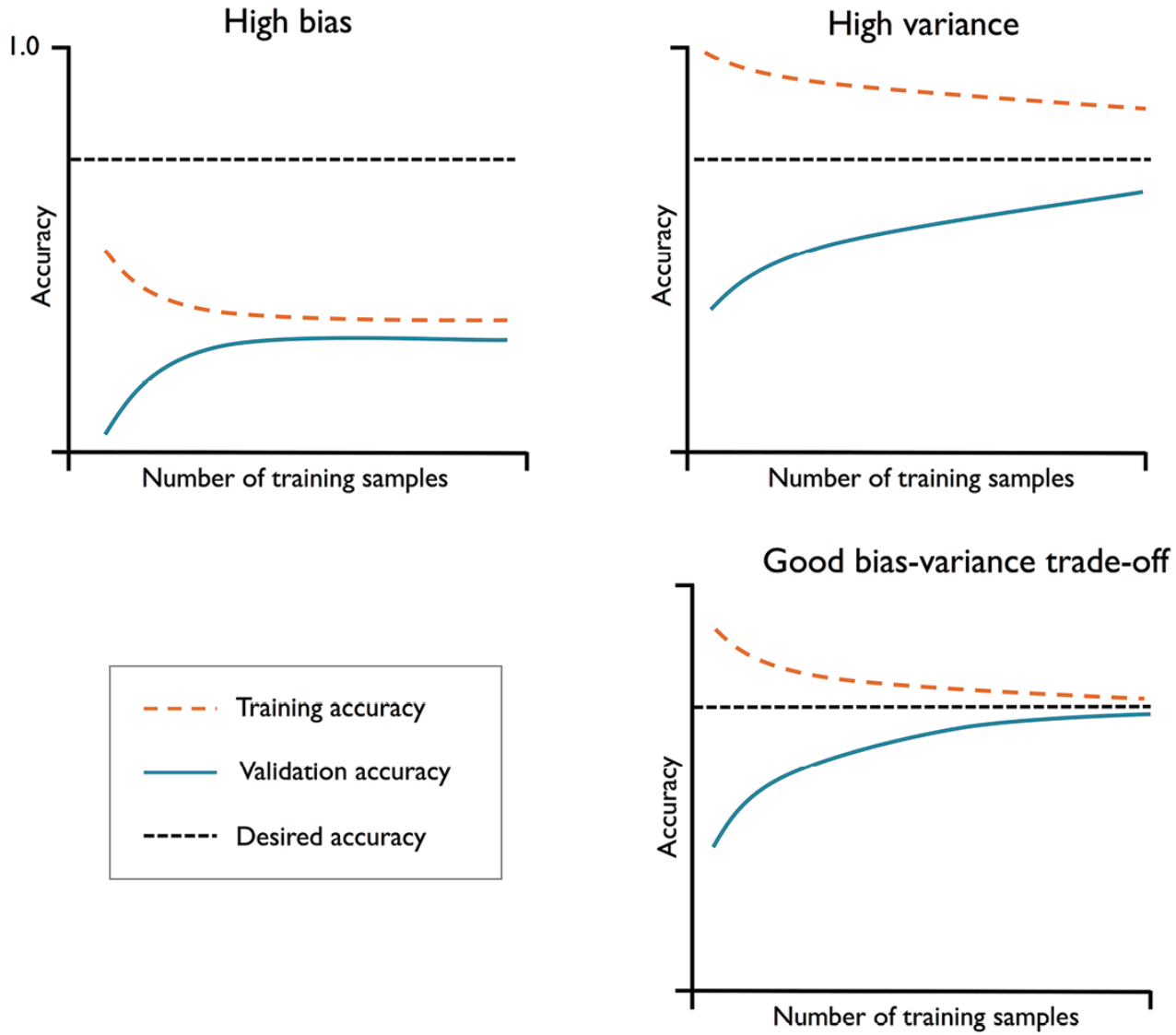
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

k-Fold cross-validation



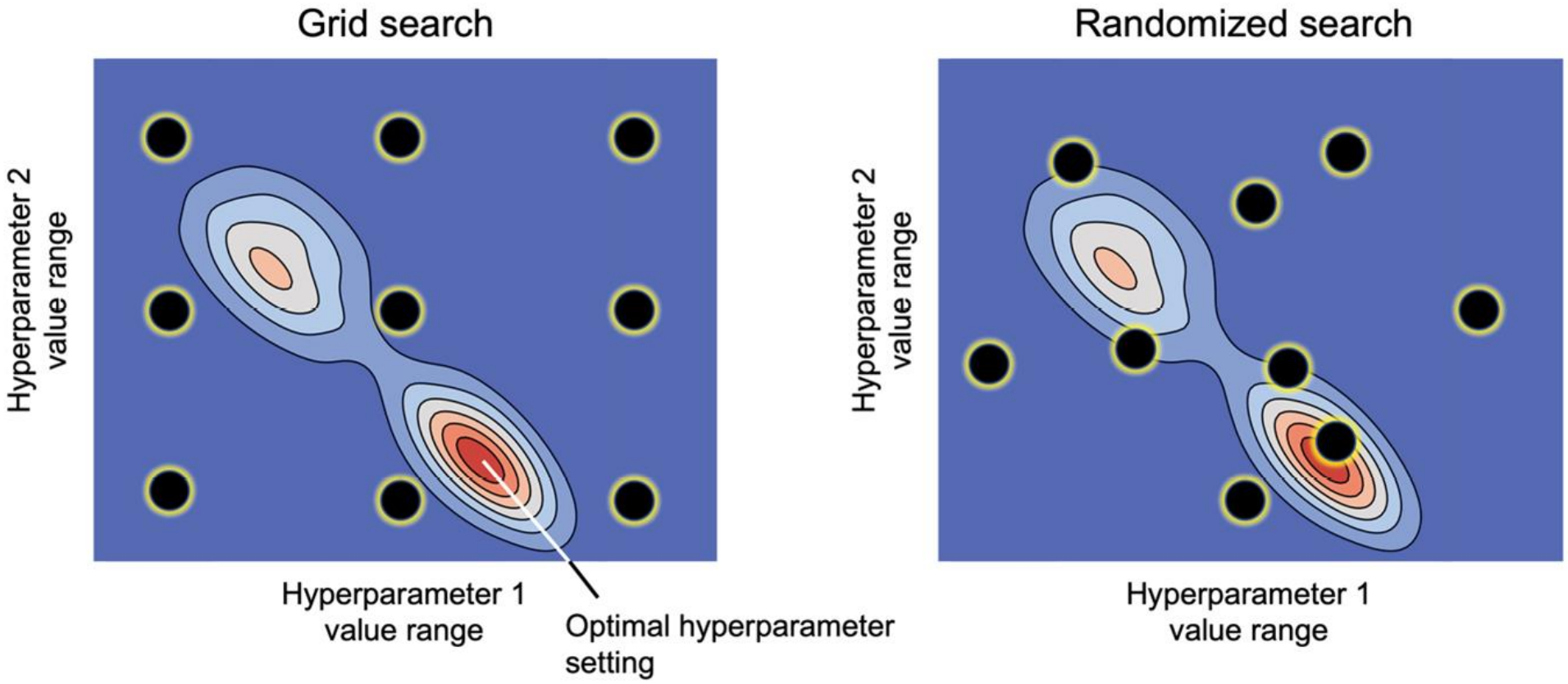
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Bias-variance trade-off



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Finding the right hyperparameters



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Automated tuning hyperparameters

Parameter scape halving:

1.Draw a large set of candidate configurations via random sampling
2. Train the models with limited resources, for example, a small subset of the training data (as opposed to using the entire training set)
3. Discard the bottom 50 percent based on predictive performance
4. Go back to *step 2* with an increased amount of available resources

Hyperopt:

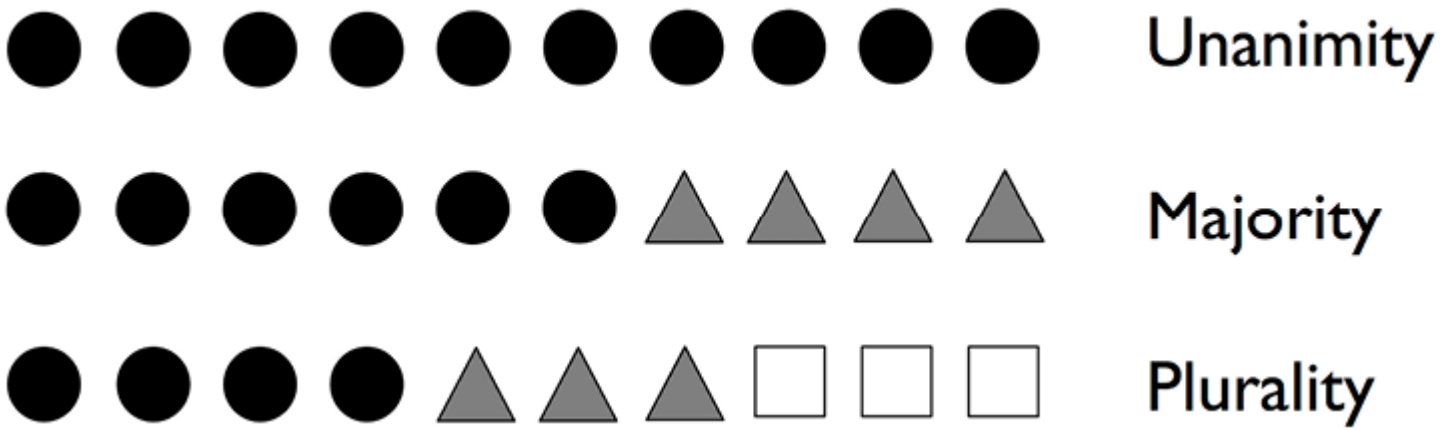
Hyperopt (https://github.com/hyperopt/hyperopt), implements several different methods for hyperparameter optimization, including randomized search and the **Tree-structured Parzen Estimators** (**TPE**) method. TPE is a Bayesian optimization method based on a probabilistic model that is continuously updated based on past hyperparameter evaluations and the associated performance scores instead of regarding these evaluations as independent events.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Combining weak learners

https://en.wikipedia.org/wiki/Lemmings_(video_game)

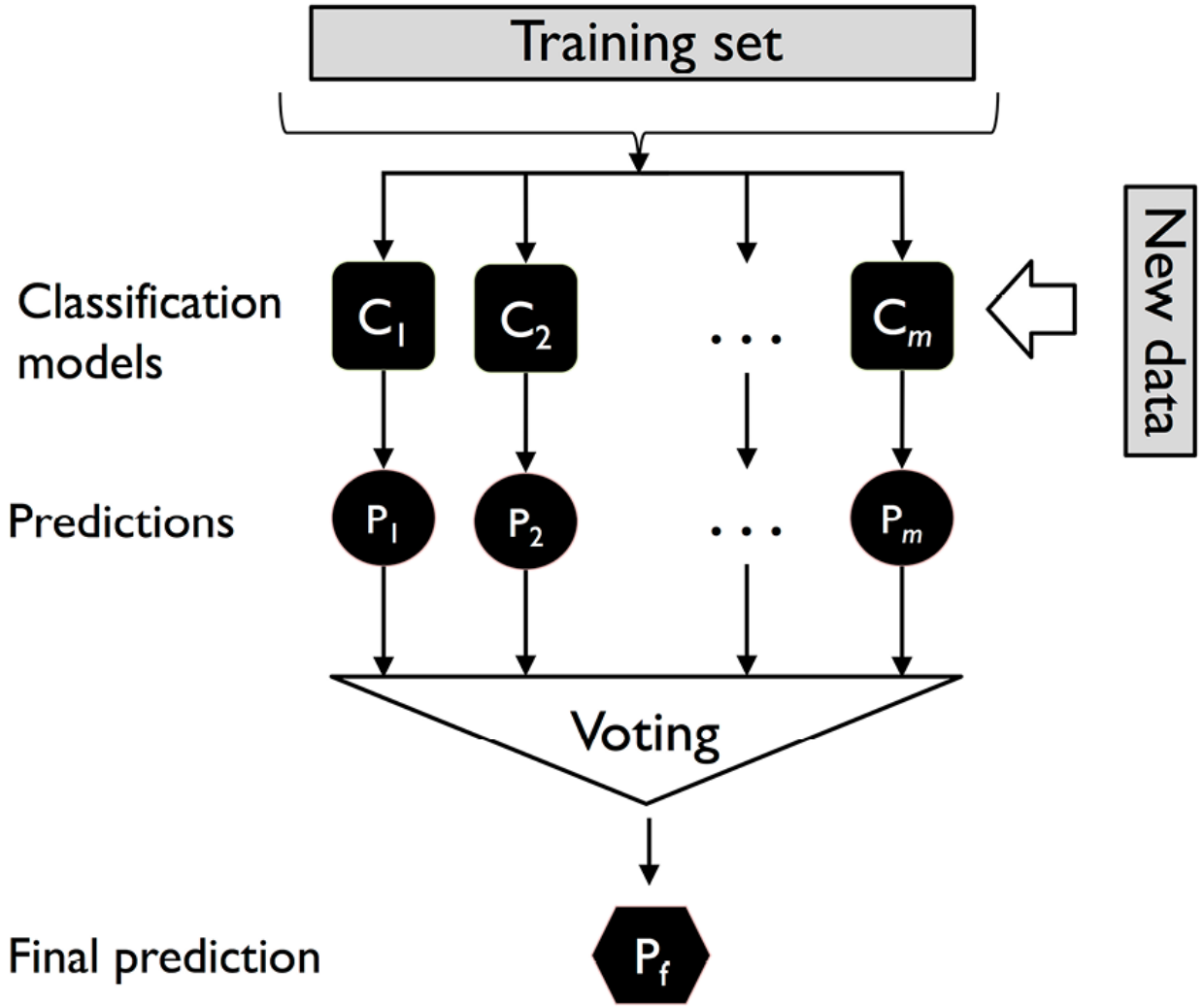
Combining weak learners



We can combine multiple weak learners into a strong learner

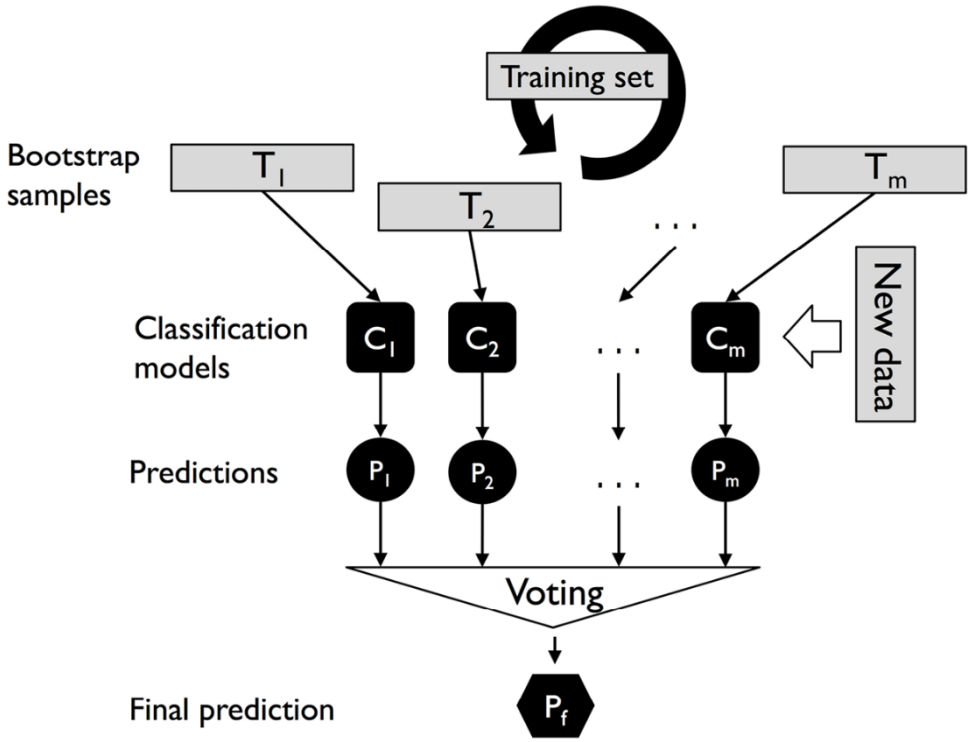
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Combining weak learners



From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

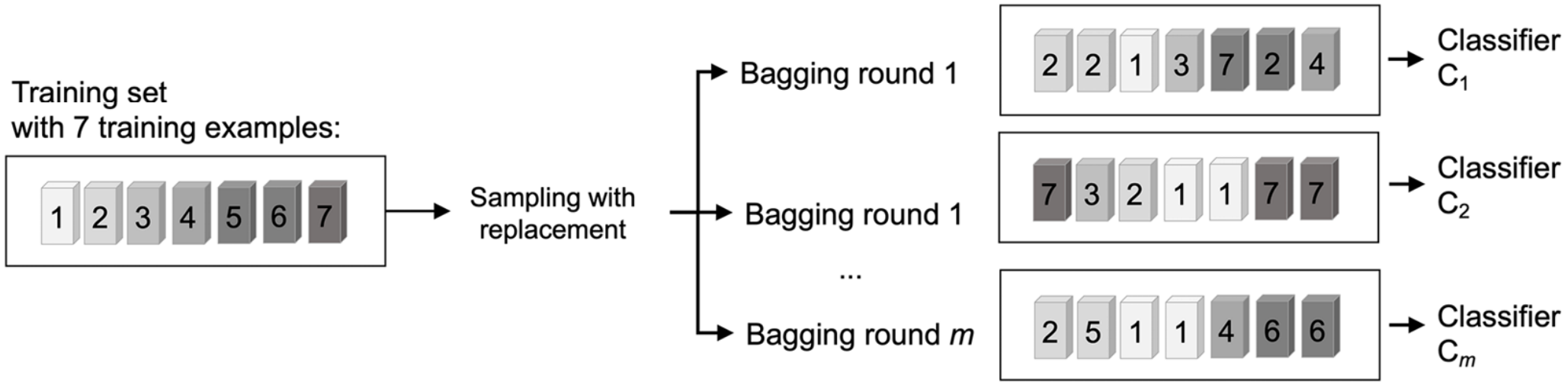
Bagging



- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class
- The multiple versions are formed by making bootstrap replicates (samples of the data, with repetition) of the learning set and using these as new learning sets.
- Bagging can give substantial gains in accuracy.
- Bagging can improve accuracy if prediction is unstable

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

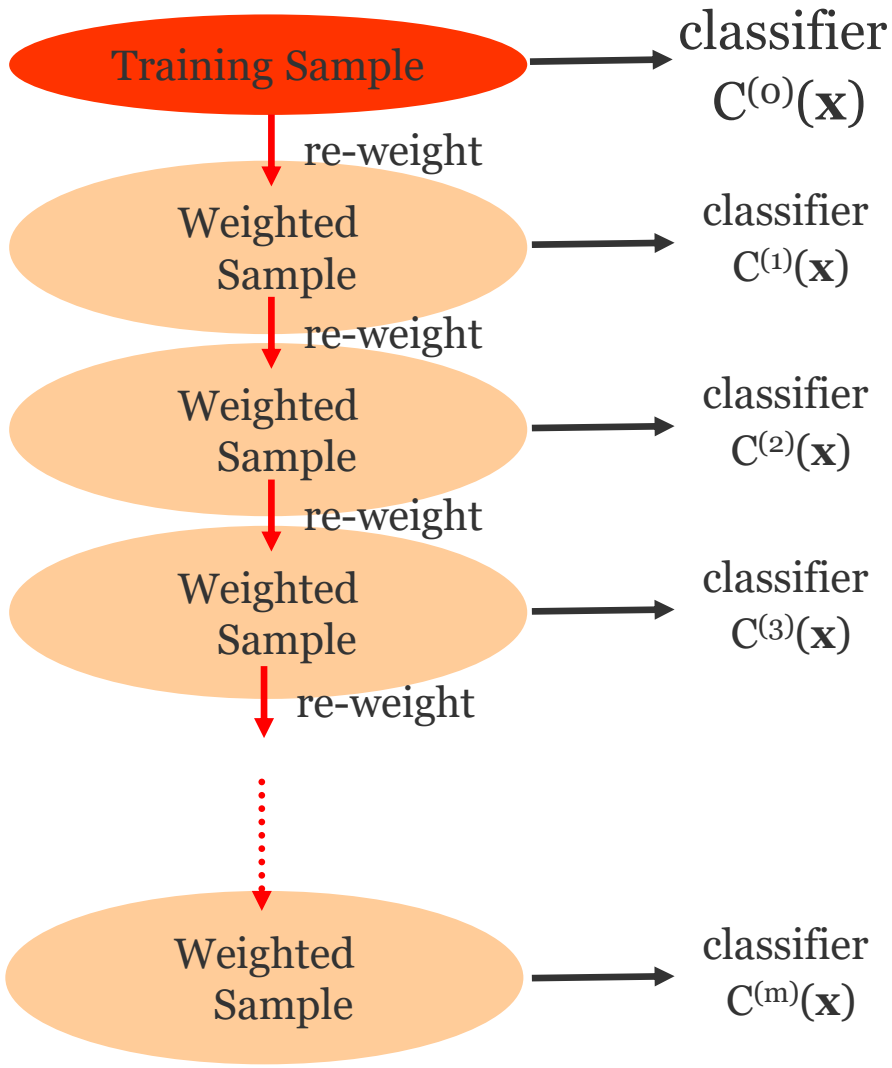
Bagging



- Draw 100 bootstrap samples of data
- Train trees on each sample → 100 trees
- Average prediction of trees on out-of-bag samples

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaptive Boosting (AdaBoost)



AdaBoost re-weights events misclassified by previous classifier by:

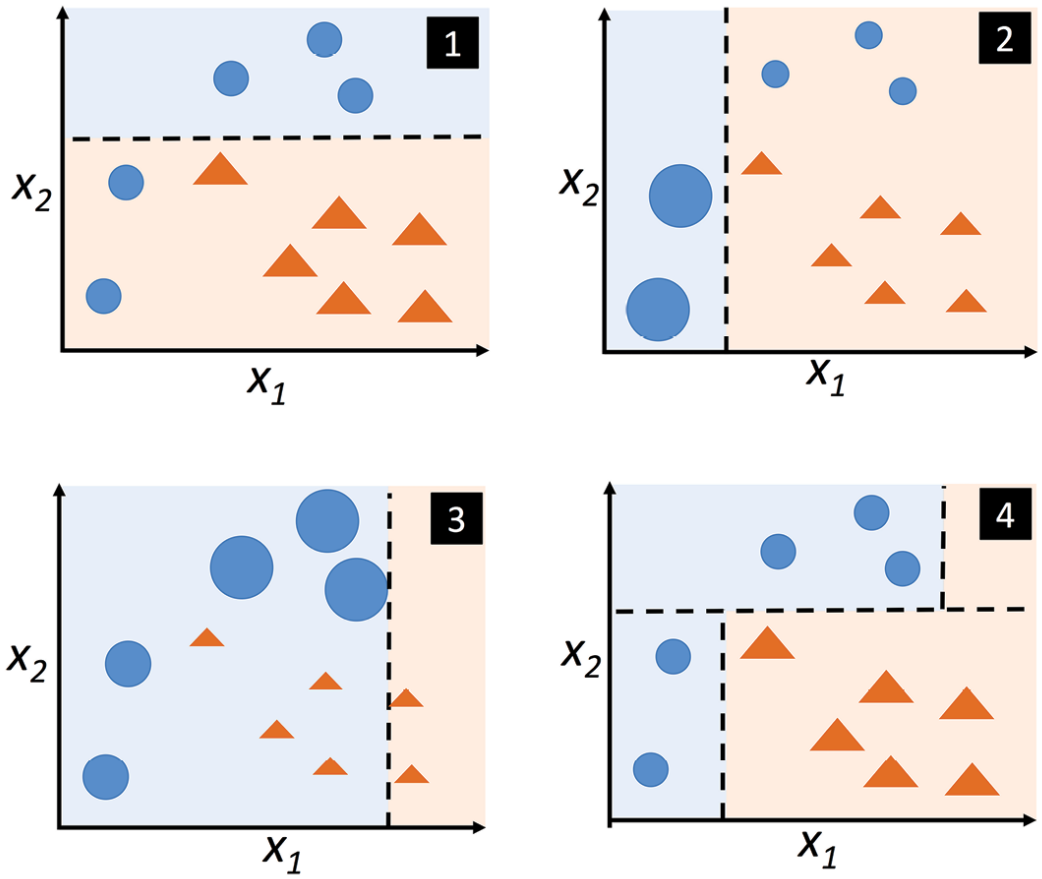
$$\frac{1-f_{err}}{f_{err}} \quad \text{with :}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

AdaBoost



1.Draw a random subset (sample) of training examples, d_1, without replacement from the training dataset, D, to train a weak learner, C_1.

2.Draw a second random training subset, d_2, without replacement from the training dataset and add 50 percent of the examples that were previously misclassified to train a weak learner, C_2.

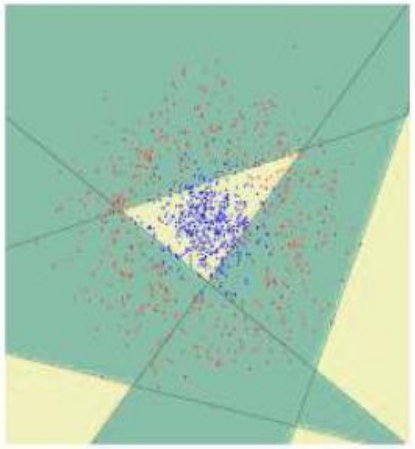
3.Find the training examples, d_3, in the training dataset, D, which C_1 and C_2 disagree upon, to train a third weak learner, C_3.

4.Combine the weak learners C_1, C_2, and C_3 via majority voting.

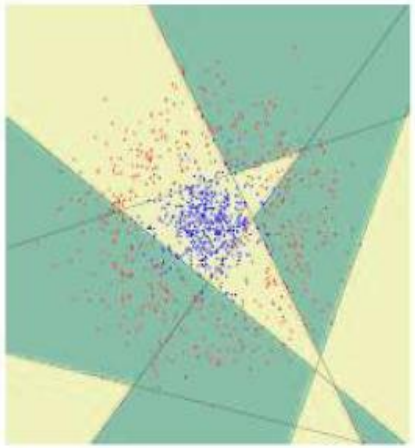
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

AdaBoost On a linear Classifier

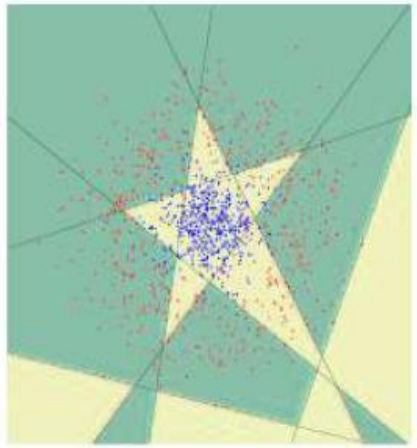
$t = 5$



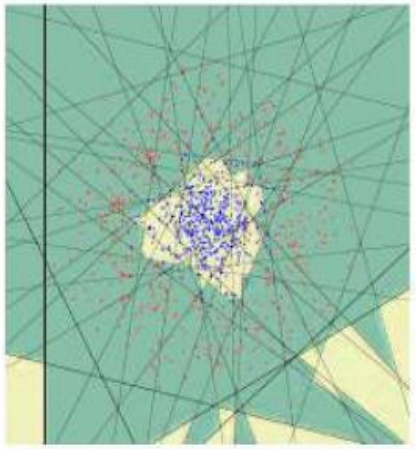
$t = 6$

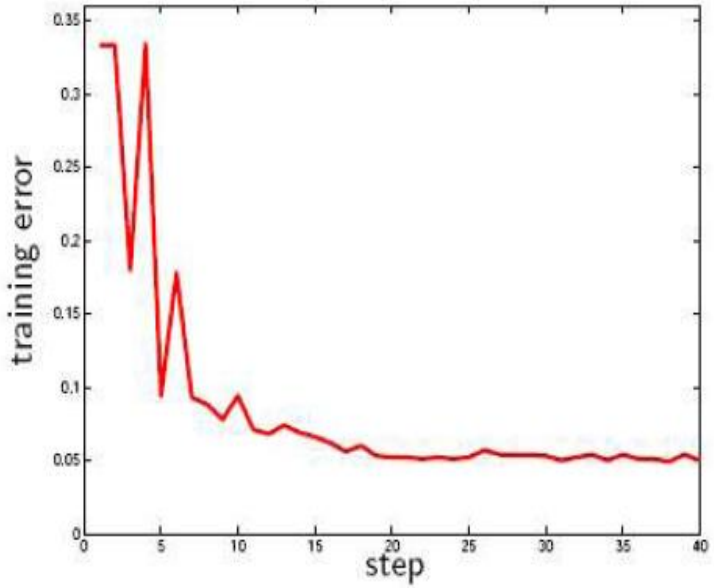


$t = 7$



$t = 40$





Boosted classifiers

1. Give events that are "difficult to categorize" more weight and average afterwards the results of all classifiers that were obtained with different weights

2. See each Tree as a "basis function" of a possible classifier:
   1. boosting or bagging is just a mean to generate a set of "basis functions"
   2. linear combination of basis functions gives final classifier

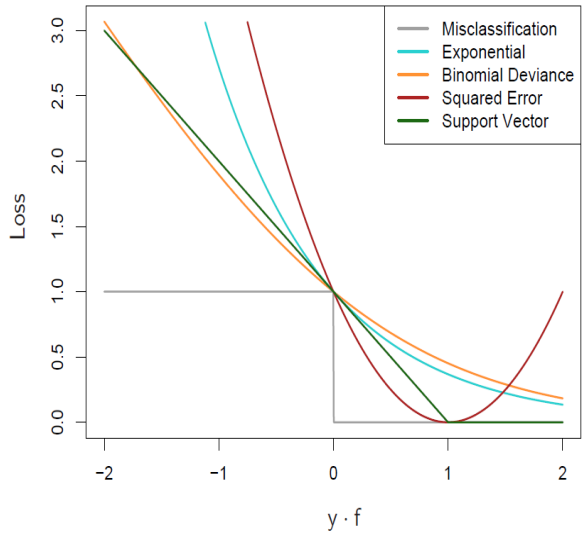
3. Every "boosting" algorithm can be interpreted as optimizing the loss function in a "greedy stagewise" manner, *i.e.* from the current point in the optimization – *e.g. building of the decision tree forest*- chooses the parameters for the next boost step (weights) such that one moves a long the steepest gradient of the loss function

AdaBoost: Exponential loss: $\exp(-y_0 y(\alpha,x))$

- theoretically sensitive to outliers

Binomial log-likelihood loss: $\ln(1 + \exp(-2y_0 y(\alpha,x))$

- more well-behaved loss function



Note of warning: you cannot do better then data

How materials are discovered?

Corning Ware glass was accidentally discovered via a furnace mishap

"The temperature gauge was stuck on 900 degrees,
and I thought I had ruined the furnace …

I grabbed some tongs to get it out as fast as I could,
but the glass slipped out of the tongs and fell to the floor.

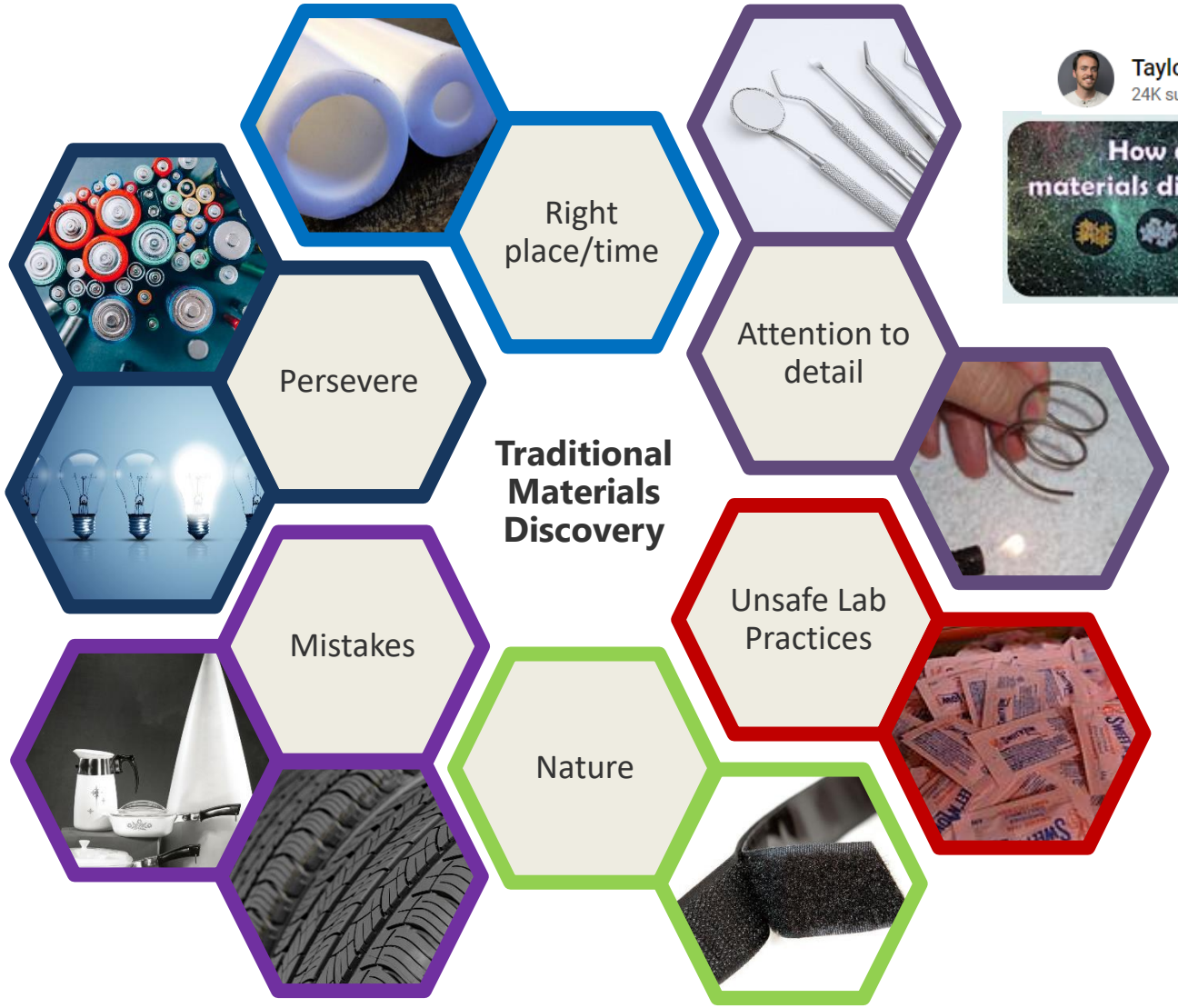
The thing bounced and didn't break."

Donald Stookey (1915-2014)

https://cen.acs.org/articles/92/web/2014/12/Donald-StookeyGuy-Gave-Us-Corning.html
https://www.nytimes.com/2014/11/07/business/s-donald-stookey-inventor-of-corningware-dies-at-99.html

Slide by Sterling Baird

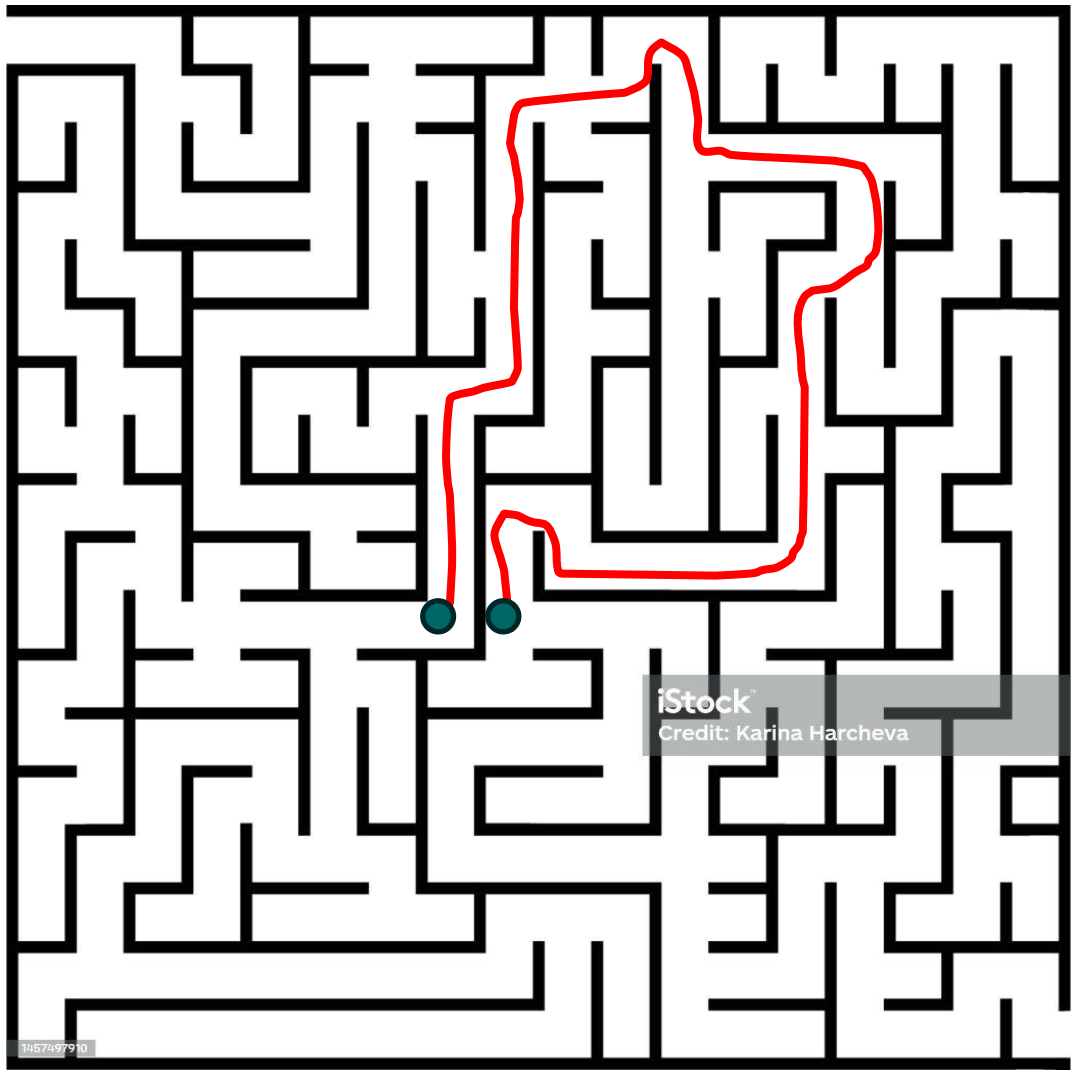
How materials are discovered?



"How are materials discovered?" by Taylor Sparks: https://www.youtube.com/watch?v=RRNcqJSJ6vc

Slide by Sterling Baird

How materials are discovered?



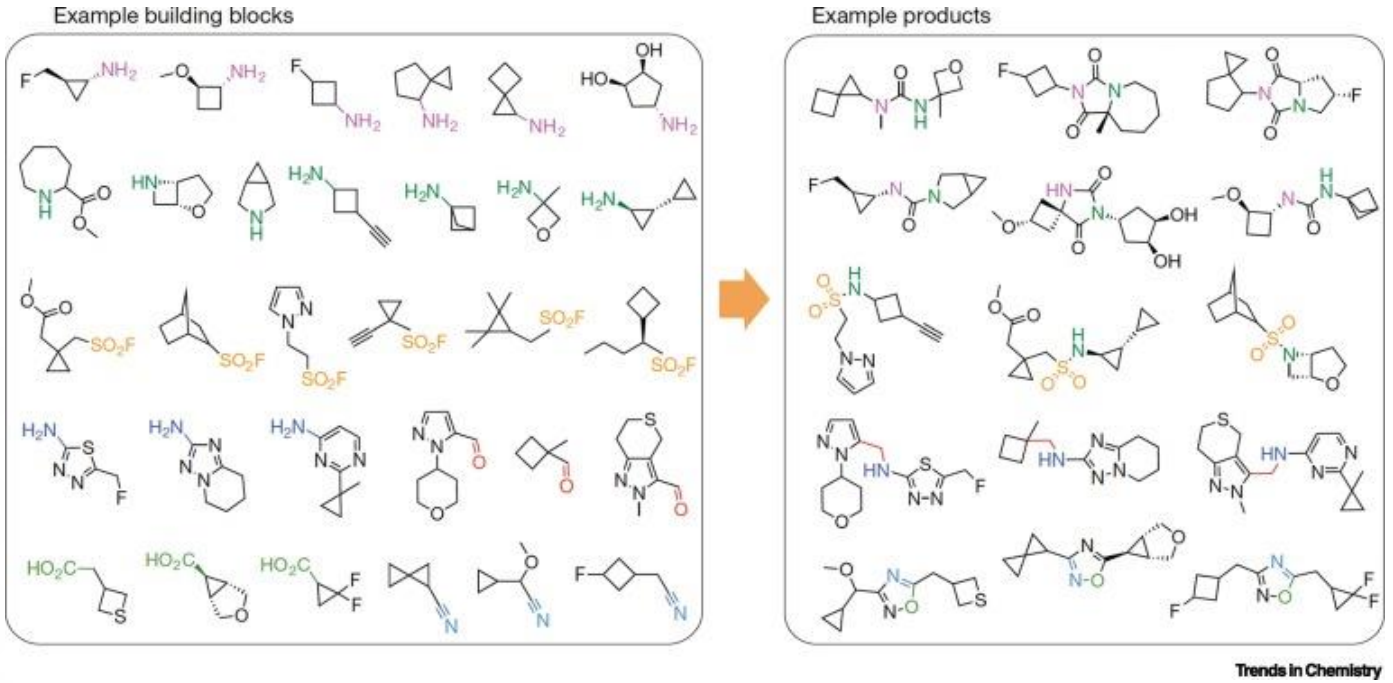
https://en.wikipedia.org/wiki/LK-99

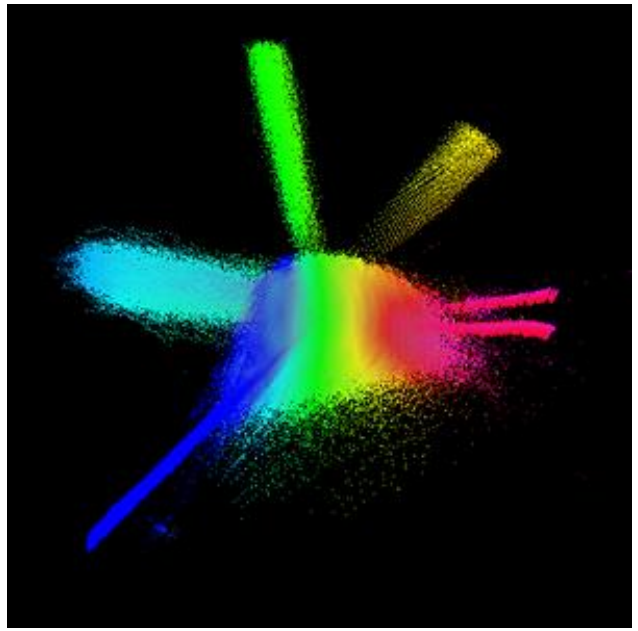
1986 – $YBa_2Cu_3O_7$. Gave rise to multiple families of Cu and Hg superconductors

2001 – MgB_2. Point compound

2006 - Layered iron pnictides. Gave rise to multiple families of superconductors

How many molecules are there?



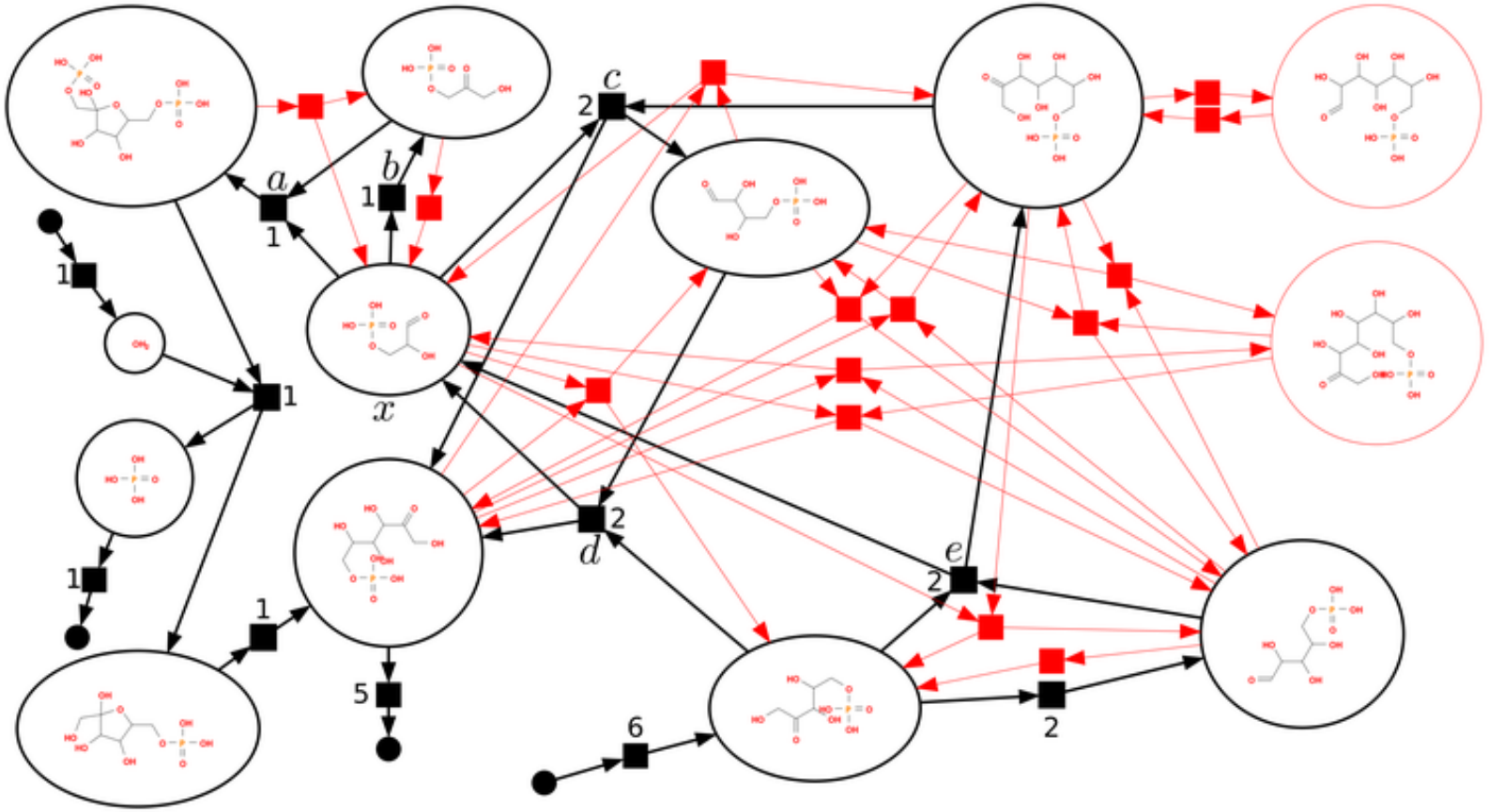


A chemical space often referred to in cheminformatics is that of potential biologically active molecules. Its size is estimated to be in the order of 10^{60} molecules. The estimate restricts the chemical elements used to be C, H, O, N and S. It further makes the assumption of a maximum of 30 atoms to stay below 500 Daltons, allows for branching and a maximum of 4 rings and arrives at an estimate of 10^{63}.

https://www.cell.com/trends/chemistry/fulltext/S2589-5974%2820%2930288-4

https://en.wikipedia.org/wiki/Chemical_space

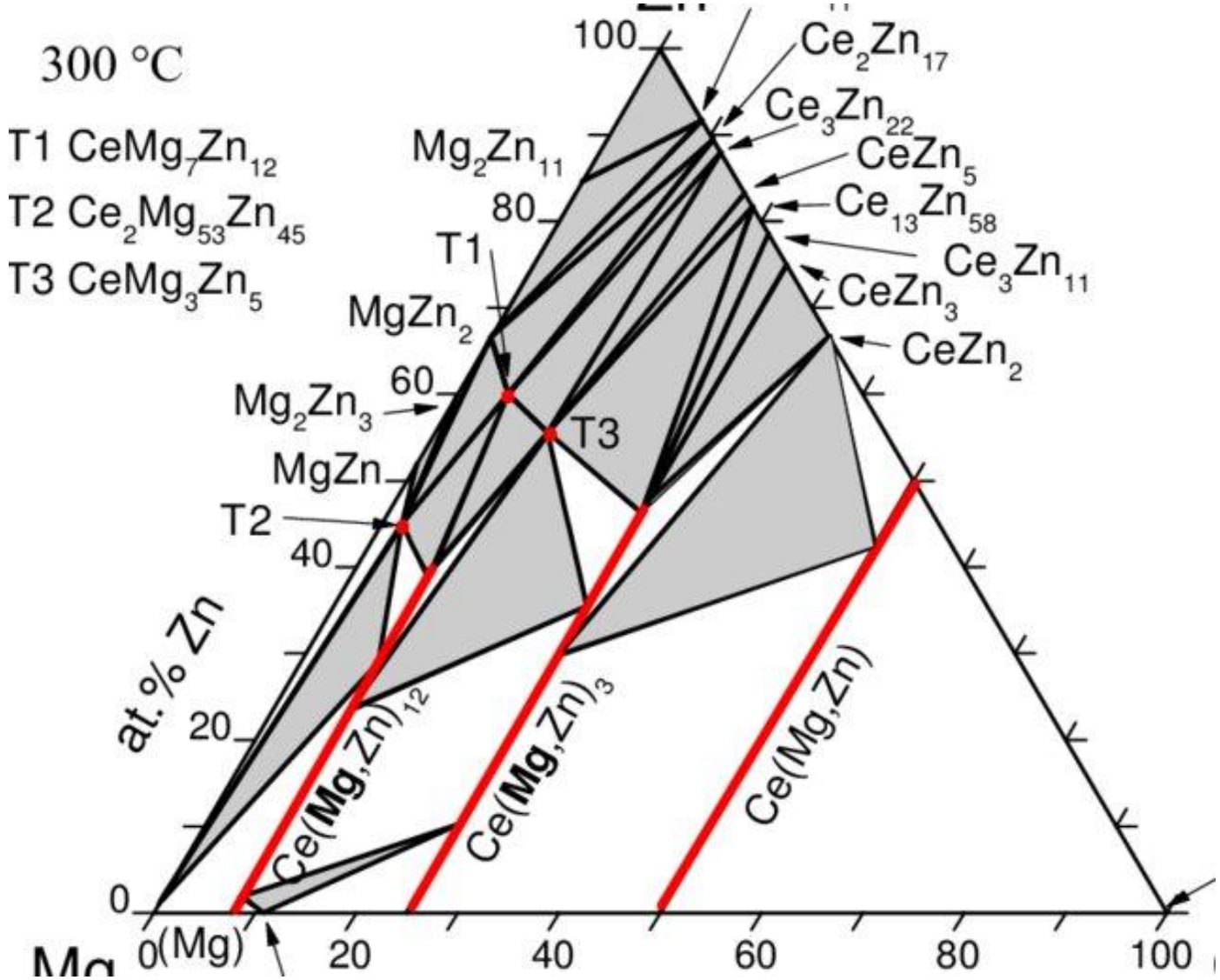
Chemical reactions networks:



- Molecular property predictions: are they **likely** to be useful?
- Synthesizability scores: what would it **probably** take to make them
- Reaction network mining and retrosynthesis: can we identify **possible** synthetic pathways?
- Optimization of specific reaction conditions and pathways: myopic and non-myopic

https://www.mis.mpg.de/stadler/research/chemical-reaction-networks.html

Why dimensionality is a problem?



Let's think about it as a search problem:

- **Alloying:** need maintain composition ~1%
- **Doping:** need maintain composition ~ 10^{-6}
- Grid search is out for D > 3 (experiment)

Why dimensionality is a problem?

- Suppose that we have data for 1000 students' performance (discretized scores of 0; 25; 50; 75; 100)% in 2 courses c1 and c2. Then in total there are 5 x 5 = 25 different grade combinations.

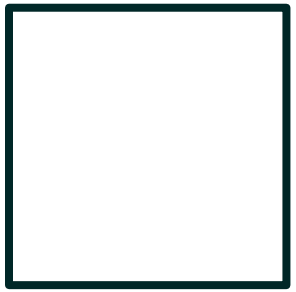
- If the 1000 students are randomly distributed among each grade combination, then on average there are 40 students with each possible grade combination, which is a good enough sample to draw conclusions such as if, for a student, grade(c1) 50 and grade(c2) 75, then that student is likely to be a Math major.

- Now suppose there are 4 courses, then the number of possible grades combination is 54 = 625, and an average number of students per combination is 1:6. For 10 courses, this number reduces to 0:0001024. This means that almost all possible combinations are never observed.

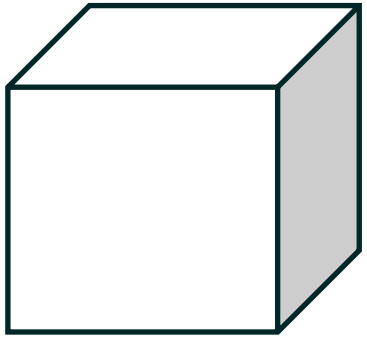
From "Curse of Dimensionality", Lecture Notes for Big Data Analytics, Faizad Ullah, February 2019

Why dimensionality is a problem?

- Suppose n points in X are chosen uniformly at random from $[0; 1]^m$ (m-cube). For the query point q grow a hypercube around q to contain f fraction of points (k = f n) in X. This cube (the search space for q) grows very large (covering almost the whole input space) in large dimension.

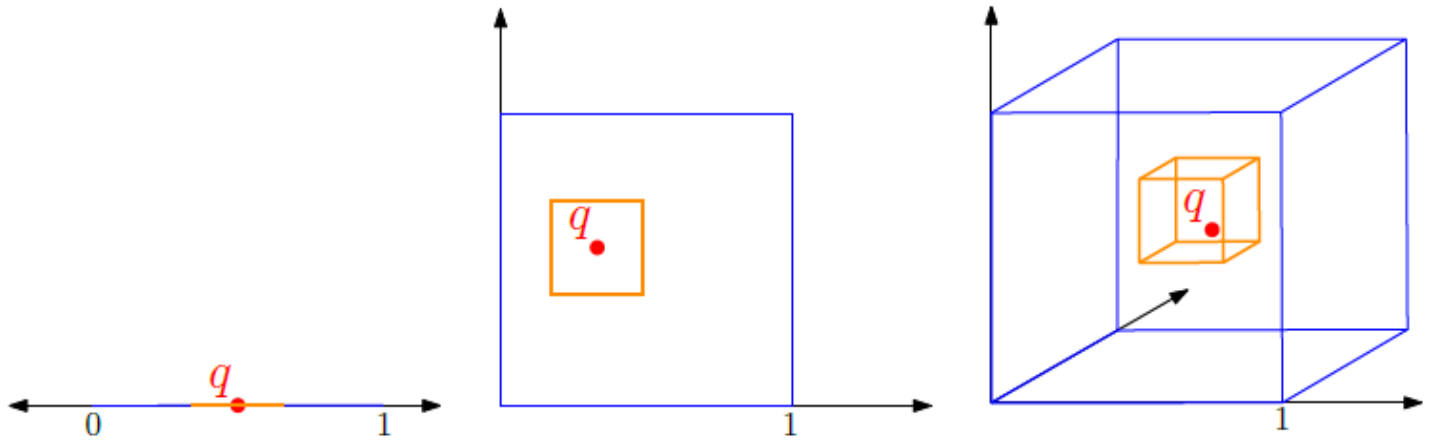


- The expected length of the edge of the search cube $E_m(f) = f^{1/m}$, i.e. in 10d to get 10% points around q need cube with edge length 0.8 (which is 80% of the whole cube, the input space). Similarly, to get only 1% points one needs to extend the search cube by 0:63 units along each dimension



From “Curse of Dimensionality”, Lecture Notes for Big Data Analytics, Faizad Ullah, February 2019

Why dimensionality is a problem?

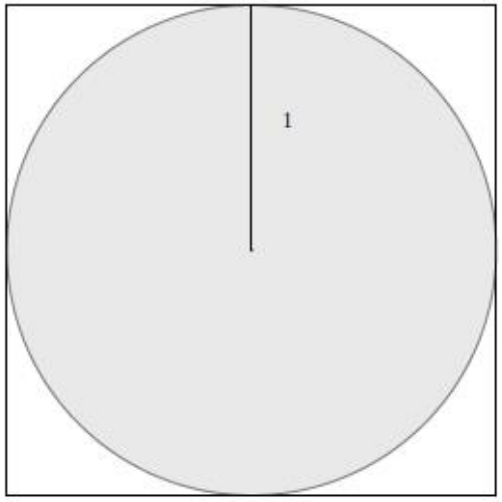


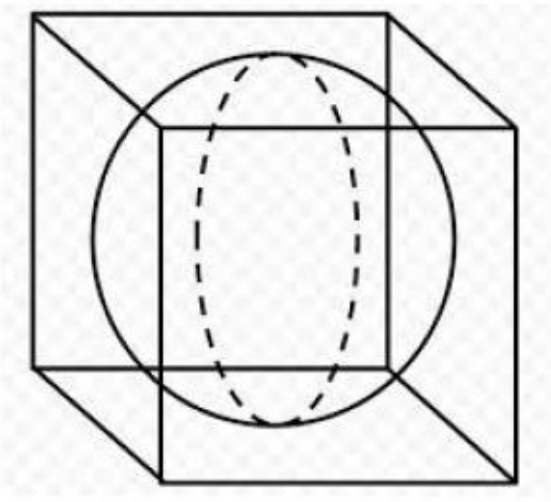
Suppose we have 5000 points:

- In 1d we have to explore 0.001 on average to capture 5 NN
- In 2d, on average we must explore 0:031 units along both dimensions to get 5 nearest neighbors points (about 3% of the whole cube).
- In 3d, on average we must go 10% of the total (unit) length in each of the 3 dimensions
- In 4d, we must explore 17:7% of unit length
- In 10d, we must go 50.1% of unit length along each dimension

From “Curse of Dimensionality”, Lecture Notes for Big Data Analytics, Faizad Ullah, February 2019

Why dimensionality is a problem?





dim m	volume of m-ball	volume of m-cube	ratio
2	π	2^2	~ 0.785
3	$4/3\pi$	2^3	~ 0.523
4	$\pi^2/2$	2^4	~ 0.308
6	$\pi^3/6$	2^6	~ 0.080
m	$\frac{\pi^{m/2}}{m/2!}$	2^m	$\to 0$

From “Curse of Dimensionality”, Lecture Notes for Big Data Analytics, Faizad Ullah, February 2019

Why dimensionality is a problem?

However if a dataset exhibit this phenomenon that the issue has be overcome by getting a larger training set (exponential in m). One way to look at this is as follows.

To cover $[-1,1]^m$ with $B_{m,1}$'s, the number of balls n must be

$$n \geq \frac{2^m}{V_m(1)} = \frac{2^m}{\pi^{m/2}/_{m/2!}} = \frac{m/2!\, 2^m}{\pi^{m/2}} \overset{m\to\infty}{\sim} \sqrt{m\pi}\left(\frac{m2^{m/2}}{2\pi e}\right)^{m/2}$$

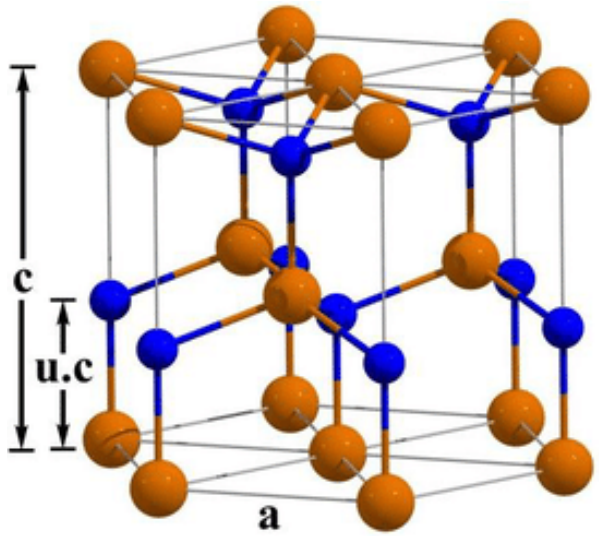
For $m = 16$ (a very small number) this n is substantially larger than 2^{58}

- In higher dimensions all the volume is in `corners'
- Points in high dimensional spaces are isolated (empty surrounding)
- The probability that a randomly generated point is within r radius of q approaches 0 as dimensionality increases
- The probability of a close nearest neighbor in a data set is very small

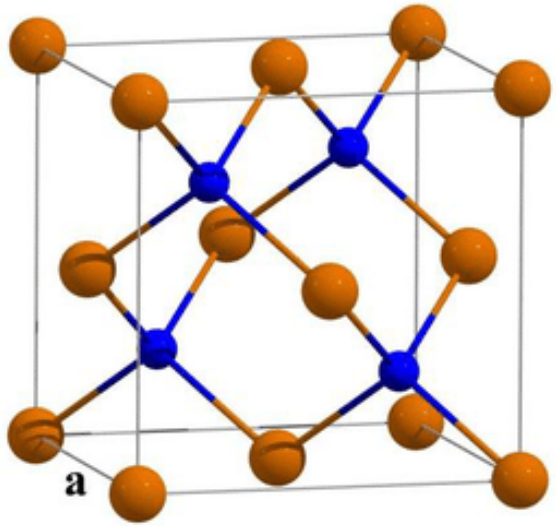
From “Curse of Dimensionality”, Lecture Notes for Big Data Analytics, Faizad Ullah, February 2019

Binary Octet Compounds

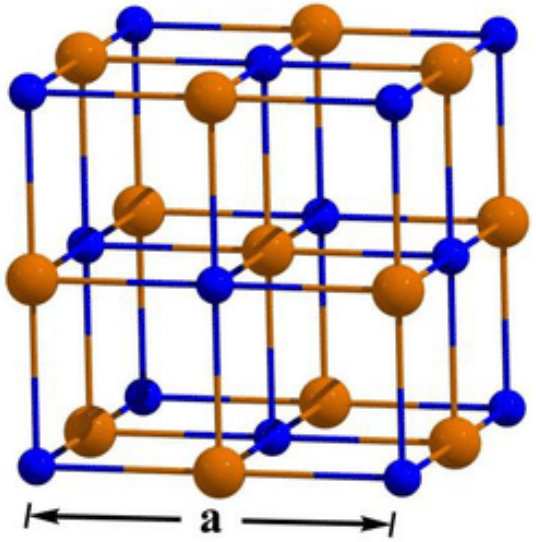
- NaCl, LiI, BeO, AlN, ….
- Can exist in zincblende (ZB), wurtzite (WZ), rocksalt (RS), cesium chloride (CsCl), and diamond cubic (DC) crystal structures



(a) wurtzite



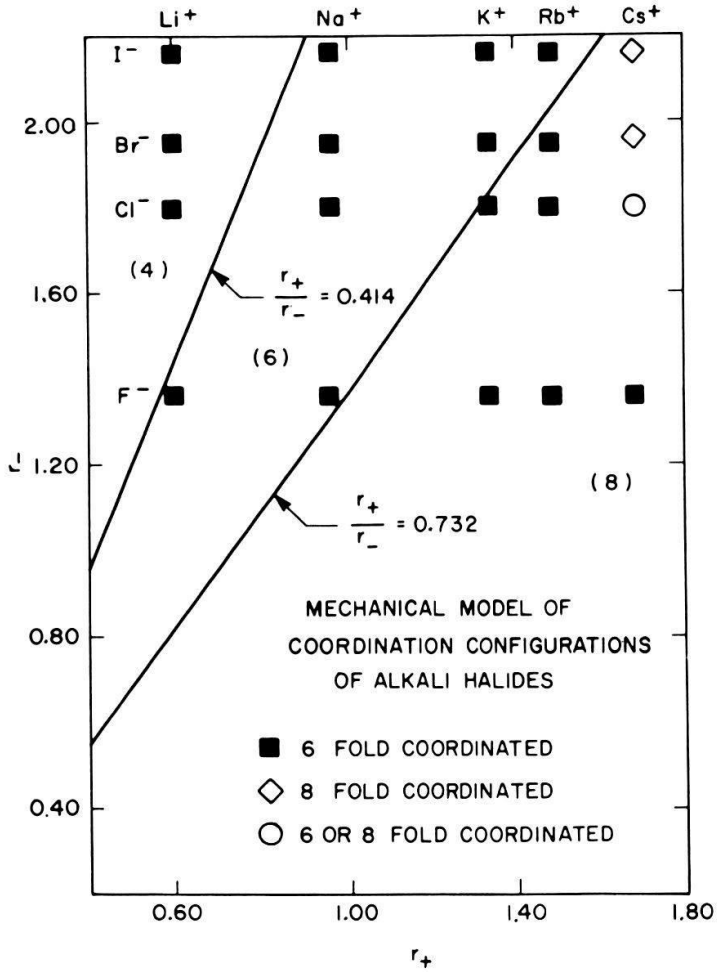
(b) zinc-blende



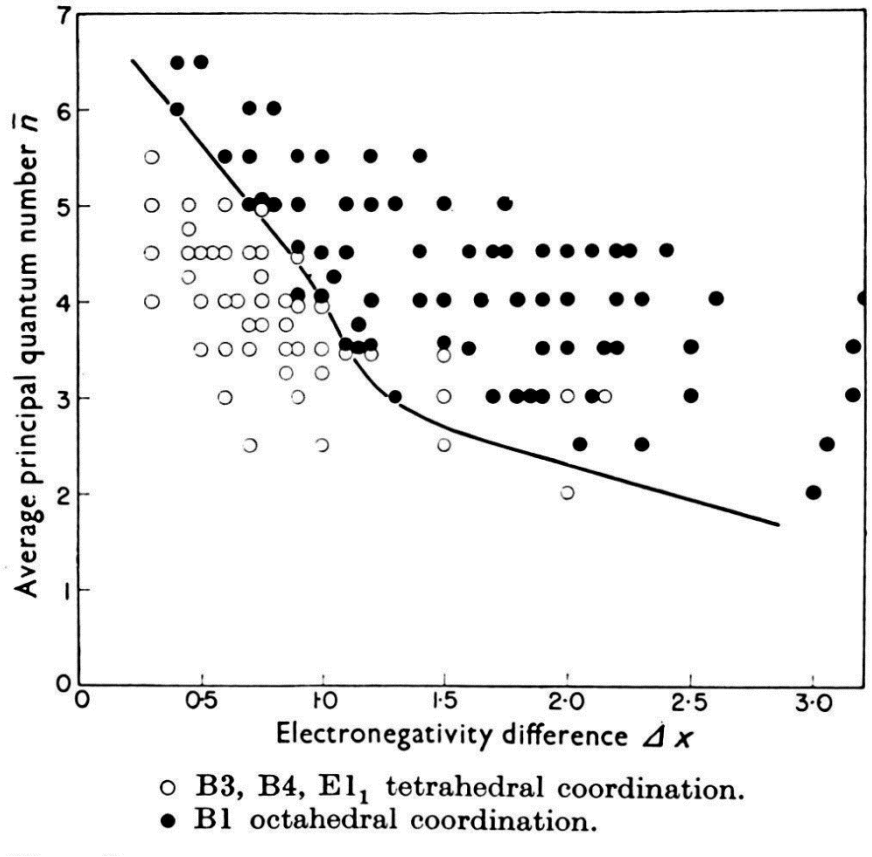
(c) rock-salt

T. Wonglakhon and D. Zahn, Interaction Potentials for modelling GaN precipitation and solid state polymorphism, Journal of Physics Condensed Matter 32(20), DOI:10.1088/1361-648X/ab6cbe

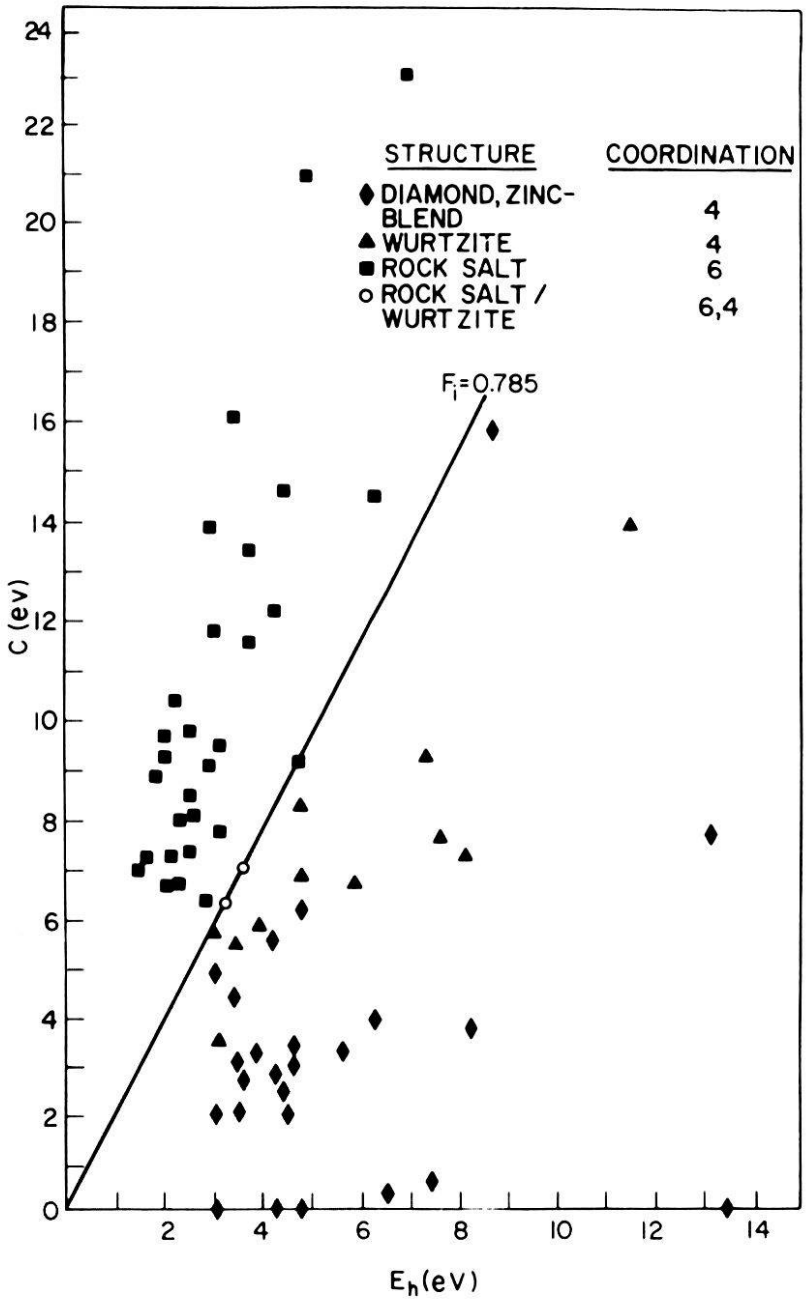
Can we predict the structure from composition?



Mooser-Pearson plots, 1959



J. C. Phillips, Structure and Properties: Mooser-Pearson plots, Helvetica Physica Acta, Vol. 58 (1985)



This average energy gap E_g was separated into covalent and ionic components, E_h and C respectively, by a Hückel relation $E_g^2 = E_h^2 + C^2$. One could then determine E_h and C separately by scaling the former with the bond length d and obtain E_g and C from ϵ. In this model the transformation from tetrahedral to octahedral coordination depends on the fraction of ionic character in the chemical bond given by $f_i = C^2/E_g^2$.

The Phillips-Van Vechten plot for AB valence compounds utilizing 'symmetric' energy-gap coordinates Eh and C. The use of quantum-mechanically defined coordinates, together with the restriction to valence compounds and exclusion of transition-metal compounds, leads to an exact separation with a straight line corresponding to constant critical ionicity.

J. C. Phillips, Structure and Properties: Mooser-Pearson plots, Helvetica Physica Acta, Vol. 58 (1985)

Zunger diagrams

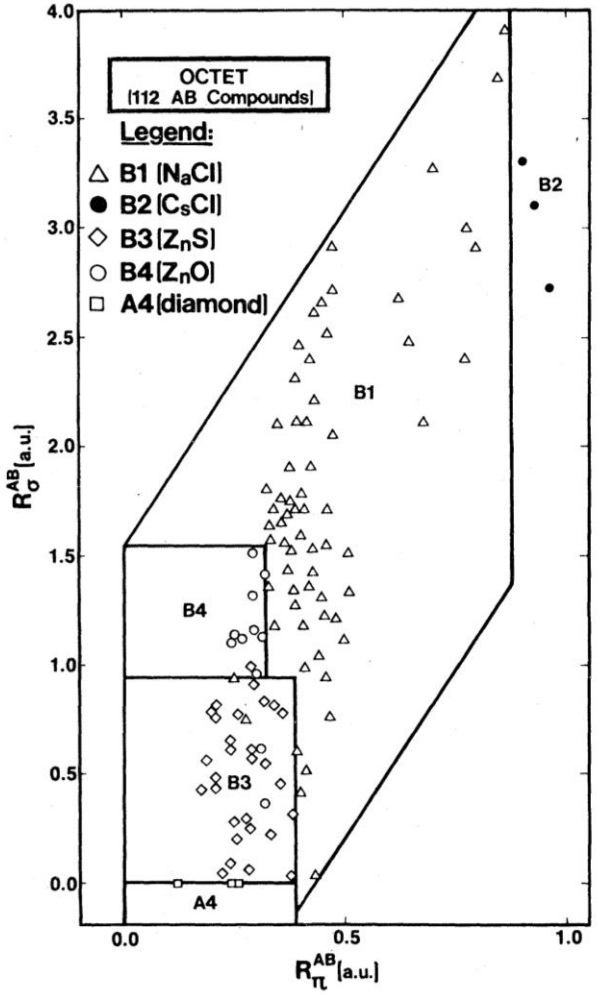


FIG. 19. Structural separation plot for the 112 binary octet compounds $A^N B^{(8-N)}$, obtained with the density-functional orbital radii, with

$$R_\sigma^{AB} = |\,(r_p^A + r_s^A) - (r_p^B + r_s^B)\,|,$$

$$R_\pi^{AB} = |\,r_p^A - r_s^A\,| + |\,r_p^B - r_s^B\,|.$$

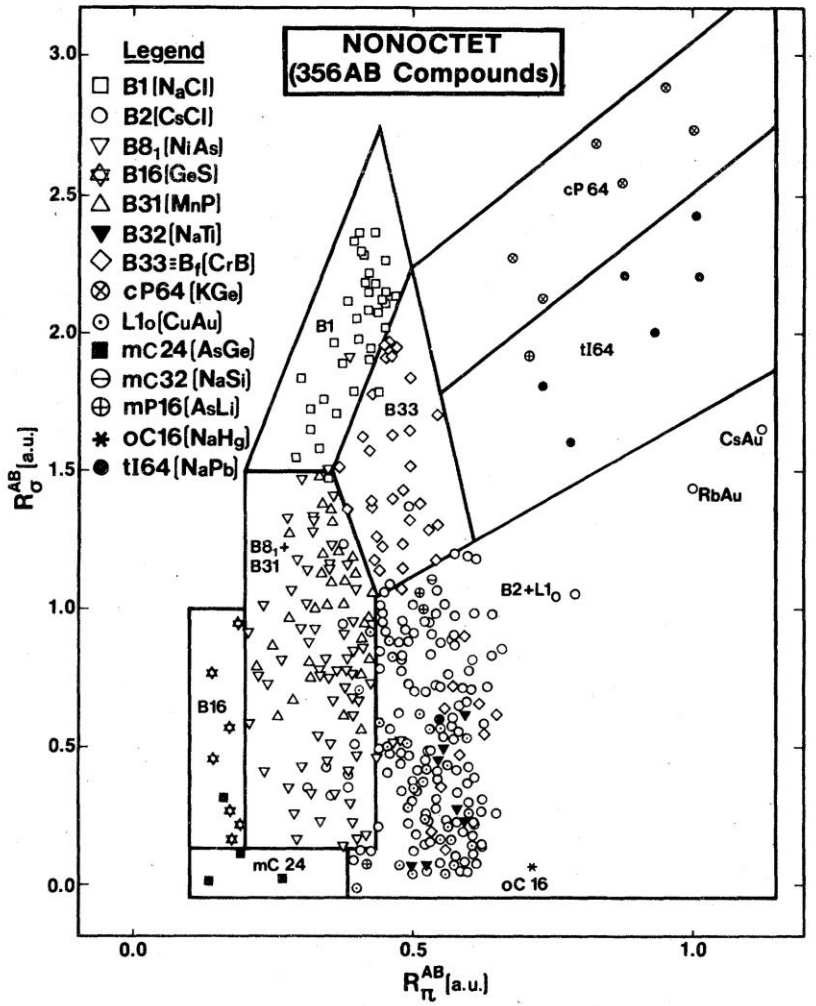


FIG. 20. Structural separation plot for the 356 binary nonoctet compounds, obtained with the density-functional orbital radii, with $R_\sigma^{AB} = |\,(r_p^A + r_s^A) - (r_p^B + r_s^B)\,|$, $R_\pi^{AB} = |\,r_p^A - r_s^A\,| + |\,r_p^B - r_s^B\,|$.

A. Zunger, Systematization of the stable crystal structure of all AB-type binary compounds: A pseudopotential orbital-radii approach, Phys. Rev. B 8, 15 (1980).

Villars diagrams

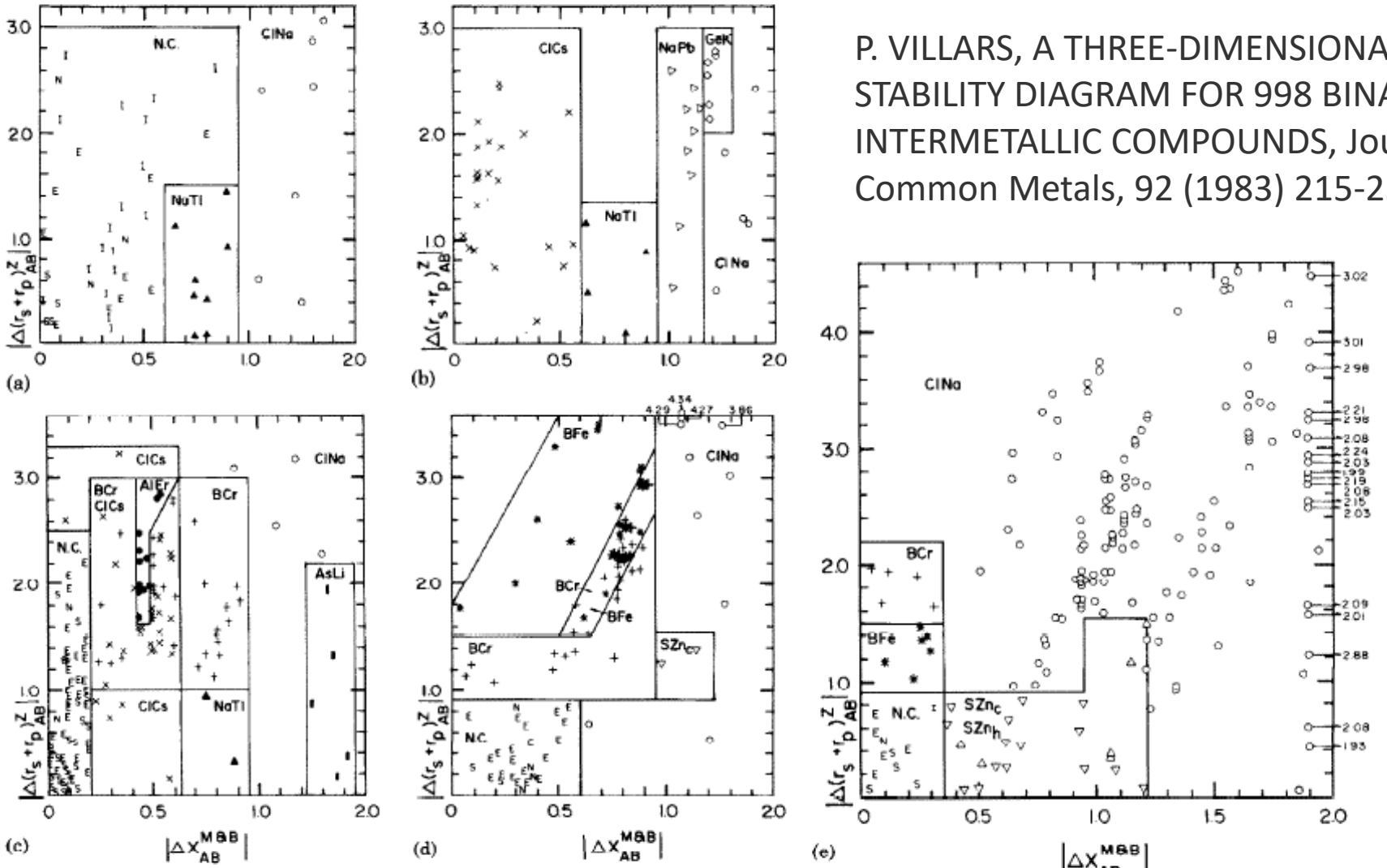
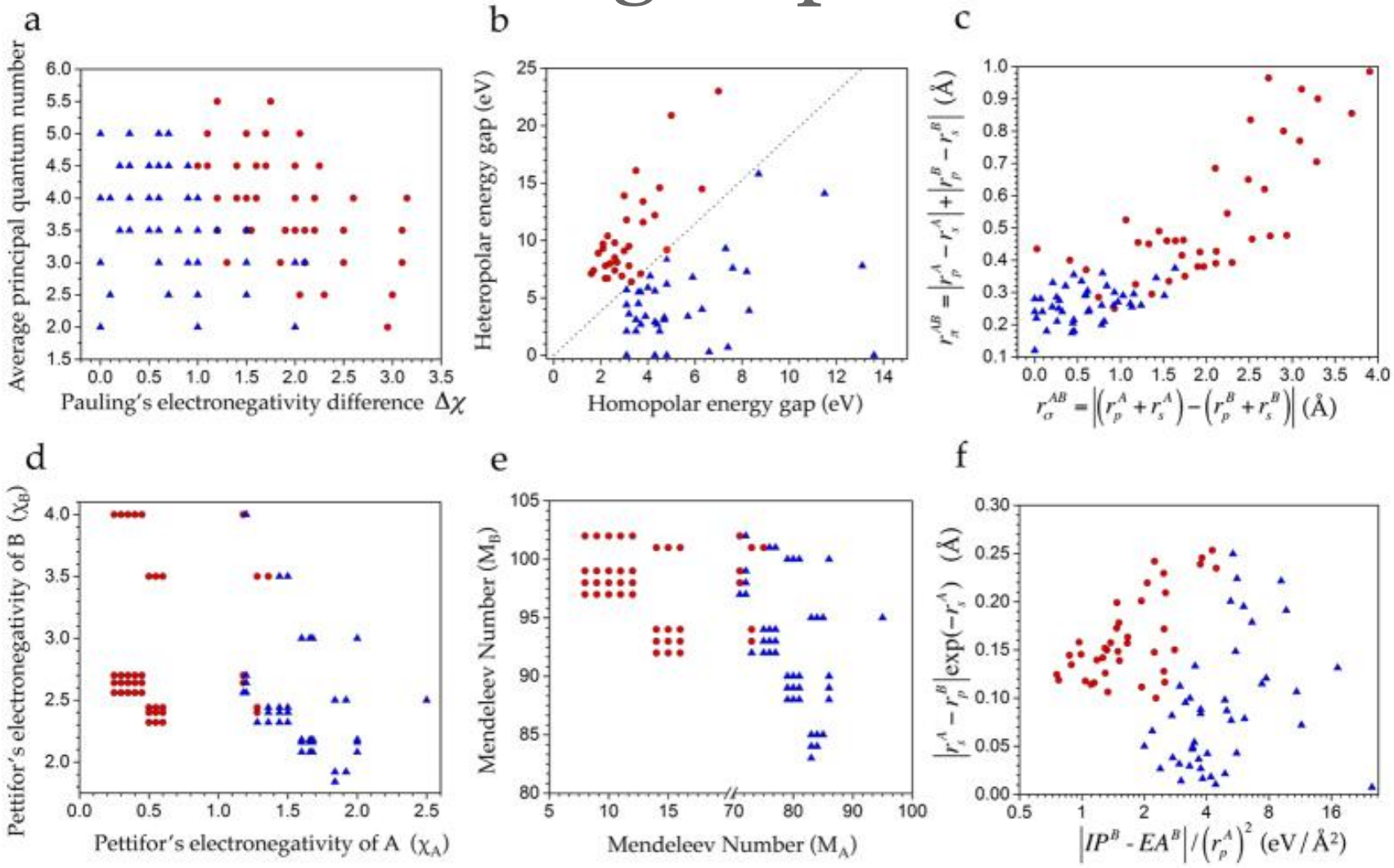


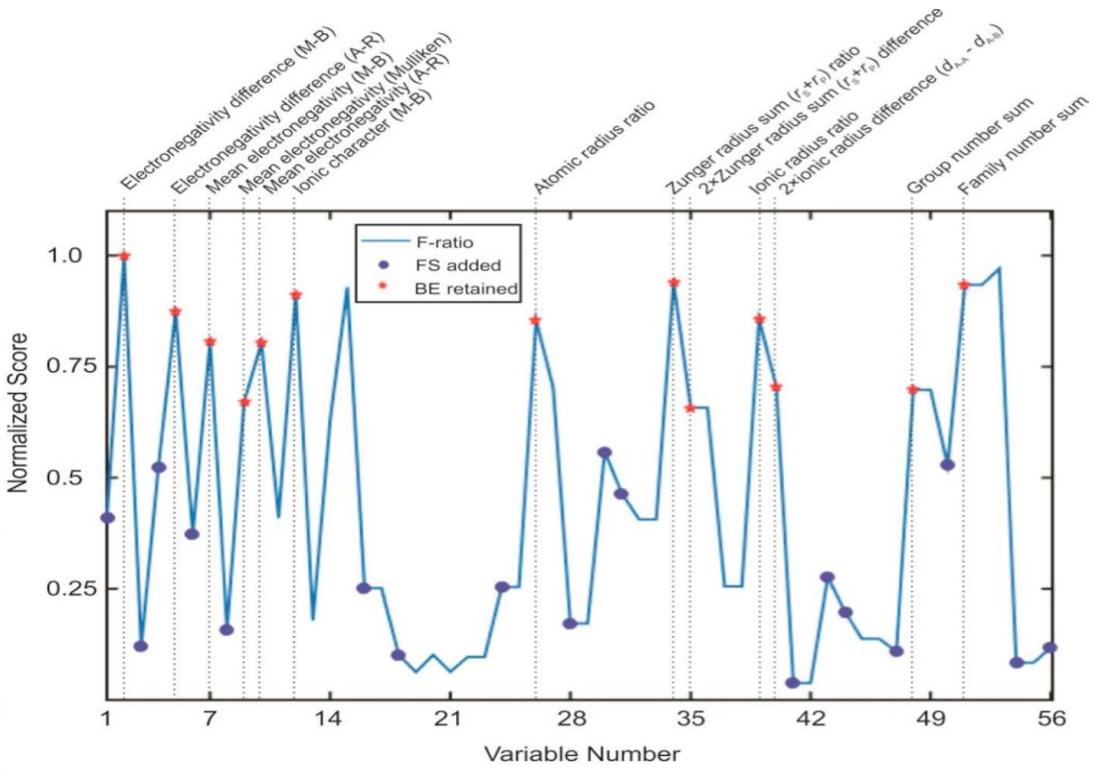
Fig. 3 *(continued).*

P. VILLARS, A THREE-DIMENSIONAL STRUCTURAL STABILITY DIAGRAM FOR 998 BINARY AB INTERMETALLIC COMPOUNDS, Journal of the Less-Common Metals, 92 (1983) 215-238 215

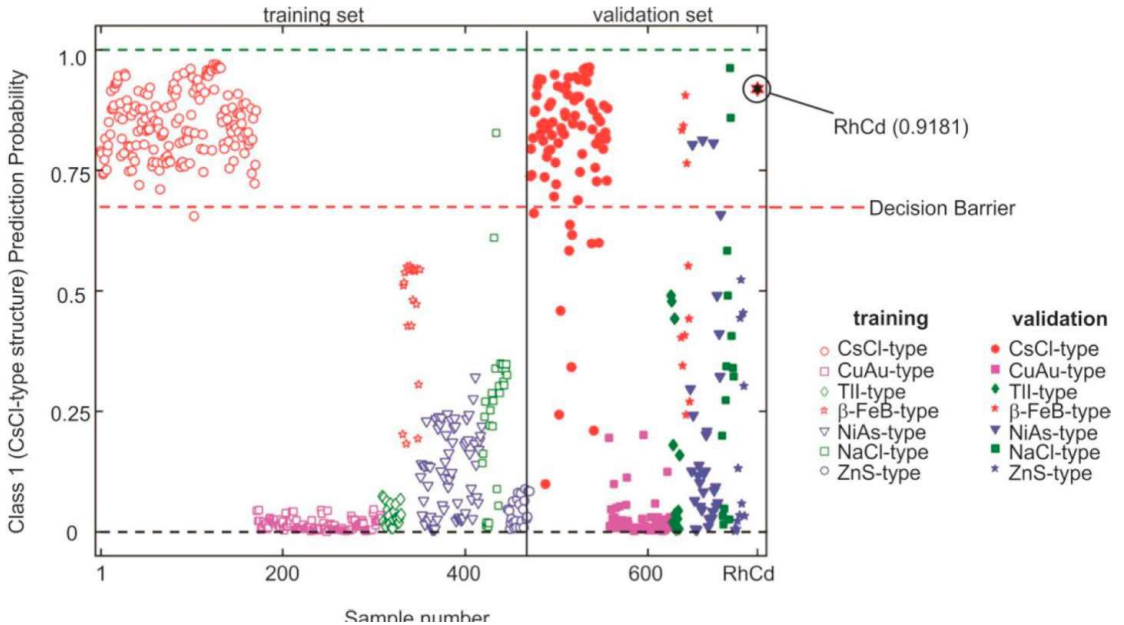
Can machine learning help?



G. Pilania, J. E. Gubernatis, and T. Lookman, Classification of octet AB-type binary compounds using dynamical charges: A materials informatics perspective, Sci Rep. 2015; 5: 17504.



1. ●Electronegativity difference (Pauling scale)	13. Ionic character (Gordy scale)	37. Ionic radius sum (d_{A-B})
2. ★Electronegativity difference (Martynov-Batsanov scale)	14. Ionic character (Mulliken scale)	38. Mean ionic radius
3. ●Electronegativity difference (Gordy scale)	15. Ionic character (Allred-Rochow scale)	39. ★Ionic radius ratio
4. ●Electronegativity difference (Mulliken scale)	16. ●Sum of valence electrons	40. ★2×ionic radius difference (d_{A-A} - d_{A-B})
5. ★Electronegativity difference (Allred-Rochow scale)	17. Mean number of electrons	41. ●Crystal radius sum (d_{A-B})
6. ●Mean electronegativity (Pauling scale)	18. ●Atomic number sum	42. Mean crystal radius
7. ★Mean electronegativity (Martynov-Batsanov scale)	19. Atomic number difference	43. ●Crystal radius ratio
8. ●Mean electronegativity (Gordy scale)	20. Mean atomic number	44. ●2×crystal radius difference (d_{A-A} - d_{A-B})
9. ★Mean electronegativity (Mulliken scale)	21. Atomic weight difference	45. Period number sum
10. ★Mean electronegativity (Allred-Rochow scale)	22. Mean atomic weight	46. Mean period number
11. Ionic character (Pauling scale)	23. Atomic weight sum	47. ●Period number difference
12. ★Ionic character (Martynov-Batsanov scale)	24. ●Atomic radius sum (d_{A-B})	48. ★Group number sum
	25. Mean atomic radius	49. Mean group number
	26. ★Atomic radius ratio	50. ●Group number difference
	27. 2×atomic radius difference (d_{A-A} - d_{A-B})	51. ★Family number sum
	28. ●Covalent radius sum (d_{A-B})	52. Mean Family number
	29. Mean covalent radius	53. Family number difference
	30. ●Covalent radius ratio	54. ●Quantum number (l) sum
	31. ●2×covalent radius difference (d_{A-A} - d_{A-B})	55. Mean quantum number (l) mean
	32. Zunger radius sum (r_S+r_P) sum	56. ●Quantum number (l) difference
	33. Mean Zunger radius sum (r_S+r_P)	
	34. ★Zunger radius sum (r_S+r_P) ratio	
	35. ★2×Zunger radius sum (r_S+r_P) difference	
	36. Zunger radius sum (r_S+r_P) difference	



A. O. Oliynyk, L.A. Adutwum, J.J. Harynuk, and A. Mar, Classifying Crystal Structures of Binary Compounds AB through Cluster Resolution Feature Selection and Support Vector Machine Analysis, Chem. Mater. 2016, 28, 18, 6672–6681 (2016)

Feature engineering with machine learning

For instance, the starting point $\mathbf{\Phi}_0$ may comprise readily available and relevant properties, such as atomic radii, ionization energies, valences, bond distances, and so on. The operators set is defined as

$$\hat{\boldsymbol{H}}^{(\mathrm{m})} \equiv \{I, +, -, \times, /, \exp, \log, |-|, \sqrt{}, ^{-1}, ^{2}, ^{3}\}[\phi_1, \phi_2],$$

- Start with available physical descriptors
- Create dimensionally-consistent combinations via allowed operations
- Choose the ones that give best classification

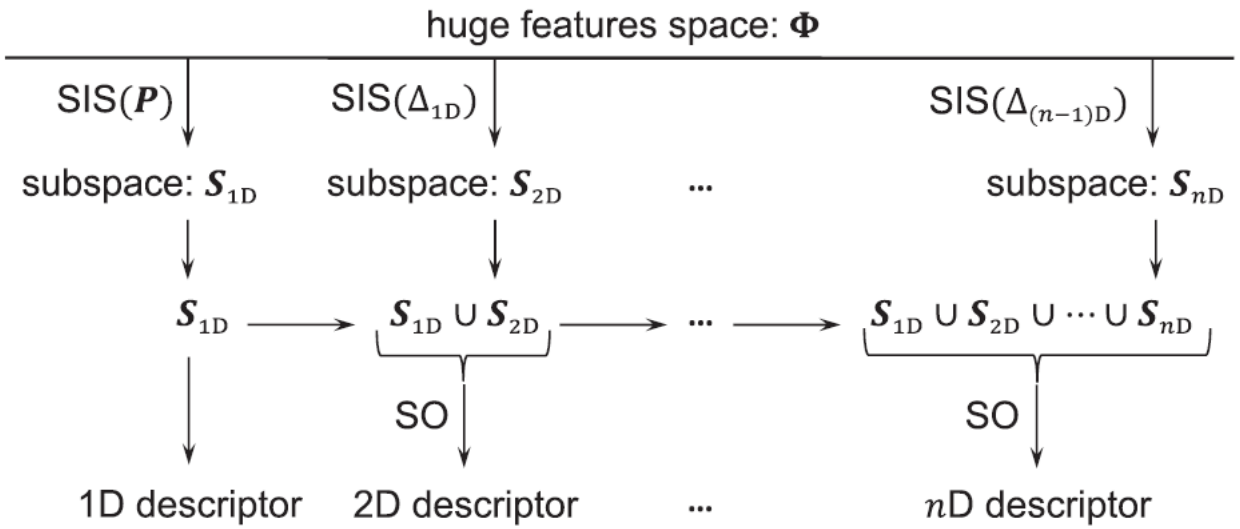


FIG. 1. The method SISSO combines unified subspaces having the largest correlation with residual errors $\boldsymbol{\Delta}$ (or $\boldsymbol{P}$) generated by sure independence screening (SIS) with sparsifying operator (SO) to further extract the best descriptor.

R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L.M. Ghiringhelli, SISSO: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates, PHYSICAL REVIEW MATERIALS 2, 083802 (2018)

Feature engineering with machine learning

TABLE I. Dependence of the metal-insulator classification descriptors on the prototypes of training binary materials.

prototypes	#materials	primary features	descriptor	classification accuracy
NaCl	132	IE_A, IE_B, χ_A, χ_B, r_{covA}, r_{covB}, EA_A, EA_B, v_A, v_B, d_{AB}	$d_1 := \frac{IE_A IE_B (d_{AB} - r_{\text{covA}})}{\exp(\chi_A)\sqrt{r_{\text{covB}}}}$	100%
NaCl, CsCl, ZnS, CaF_2, Cr_3Si	217	IE_A, IE_B, χ_A, χ_B, r_{covA}, r_{covB}, d_{AB}, CN_A, CN_B	$d_1 := \frac{IE_B d_{AB}^2}{\chi_A r_{\text{covA}}^2 \sqrt{CN_B}}$, $d_2 := \frac{IE_A^2 r_{\text{covB}} \log(IE_A) \lvert r_{\text{covA}} - r_{\text{covB}} \rvert}{CN_B}$	100%
NaCl, CsCl, ZnS, CaF_2, Cr_3Si, SiC, TiO_2, ZnO, FeAs, NiAs	260	IE_A, IE_B, χ_A, χ_B, r_{covA}, r_{covB}, d_{AB}, CN_A, CN_B	$d_1 := \frac{d_{AB}/r_{\text{covA}} - \chi_A/\chi_B}{\exp(CN_B/IE_B)}$, $d_2 := \frac{r_{\text{covA}}^3 d_{AB} IE_B}{\lvert \chi_B/\chi_A - \lvert CN_B - CN_A \rvert \rvert}$	99.6%[a]
NaCl, CsCl, ZnS, CaF_2, Cr_3Si, SiC, TiO_2, ZnO, FeAs, NiAs	260	IE_A, IE_B, χ_A, χ_B, x_A, x_B, $V_{\text{cell}}/\sum V_{\text{atom}}$	$d_1 := \frac{V_{\text{cell}}}{\sum V_{\text{atom}}} \frac{\sqrt{\chi_B}}{\chi_A}$, $d_2 := \frac{IE_A IE_B}{\exp(V_{\text{cell}}/\sum V_{\text{atom}})}$	99.6%[a]
NaCl, CsCl, ZnS, CaF_2, Cr_3Si, SiC, TiO_2, ZnO, FeAs, NiAs, Al_2O_3, La_2O_3, Th_3P_4, ReO_3, ThH_2	299	IE_A, IE_B, χ_A, χ_B, x_A, x_B, $V_{\text{cell}}/\sum V_{\text{atom}}$	$d_1 := \frac{x_B}{\sum V_{\text{atom}}/V_{\text{cell}}} \frac{IE_B \sqrt{\chi_B}}{\chi_A}$, $d_2 := \chi_A^2 \lvert \lvert 1 - 2x_A \rvert - x_A^2 \frac{\chi_B}{\chi_A} \rvert$	99.0%[b]

[a]One entry misclassified: YP compound in NaCl prototype.
[b]Three entry misclassified: YP compound in NaCl prototype; Th_3As_4 and La_3Te_4 compounds in Th_3P_4 prototype.

R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L.M. Ghiringhelli, SISSO: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates, PHYSICAL REVIEW MATERIALS 2, 083802 (2018)

Feature engineering with machine learning

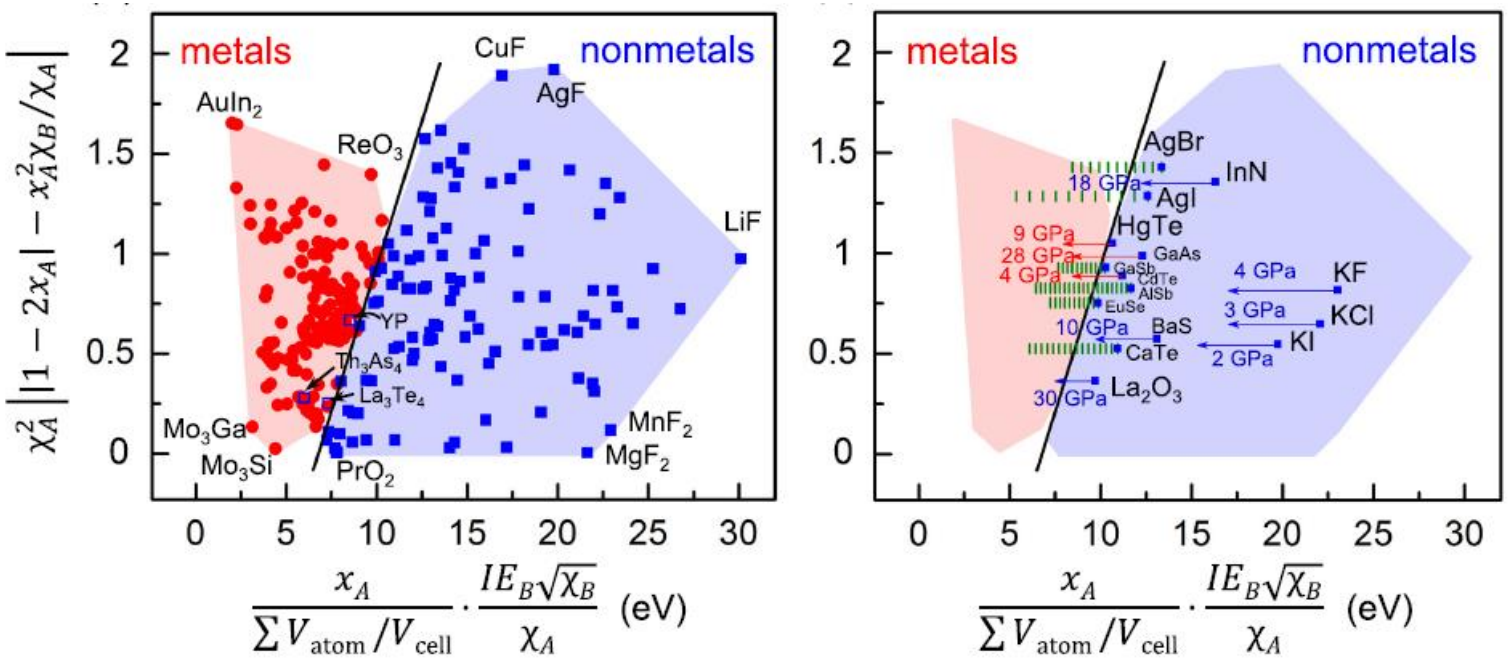


FIG. 4. SISSO for classification. (a) An almost perfect classification (99%) of metal/nonmetal for 299 materials. Symbols: χ, Pauling electronegativity; IE, ionization energy; x, atomic composition; $\sum V_{\text{atom}}/V_{\text{cell}}$, packing fraction. Red circles, blue squares, and open blue squares represent metals, nonmetals, and the three erroneously characterized nonmetals, respectively. (c) Reproduction of pressure-induced insulatormetals transitions (red arrows), of materials that remain insulators upon compression (blue arrows), and computational predictions at step of 1 GPa (green bars).

R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L.M. Ghiringhelli, SISSO: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates, PHYSICAL REVIEW MATERIALS 2, 083802 (2018)