Statistical Methods in Particle Physics

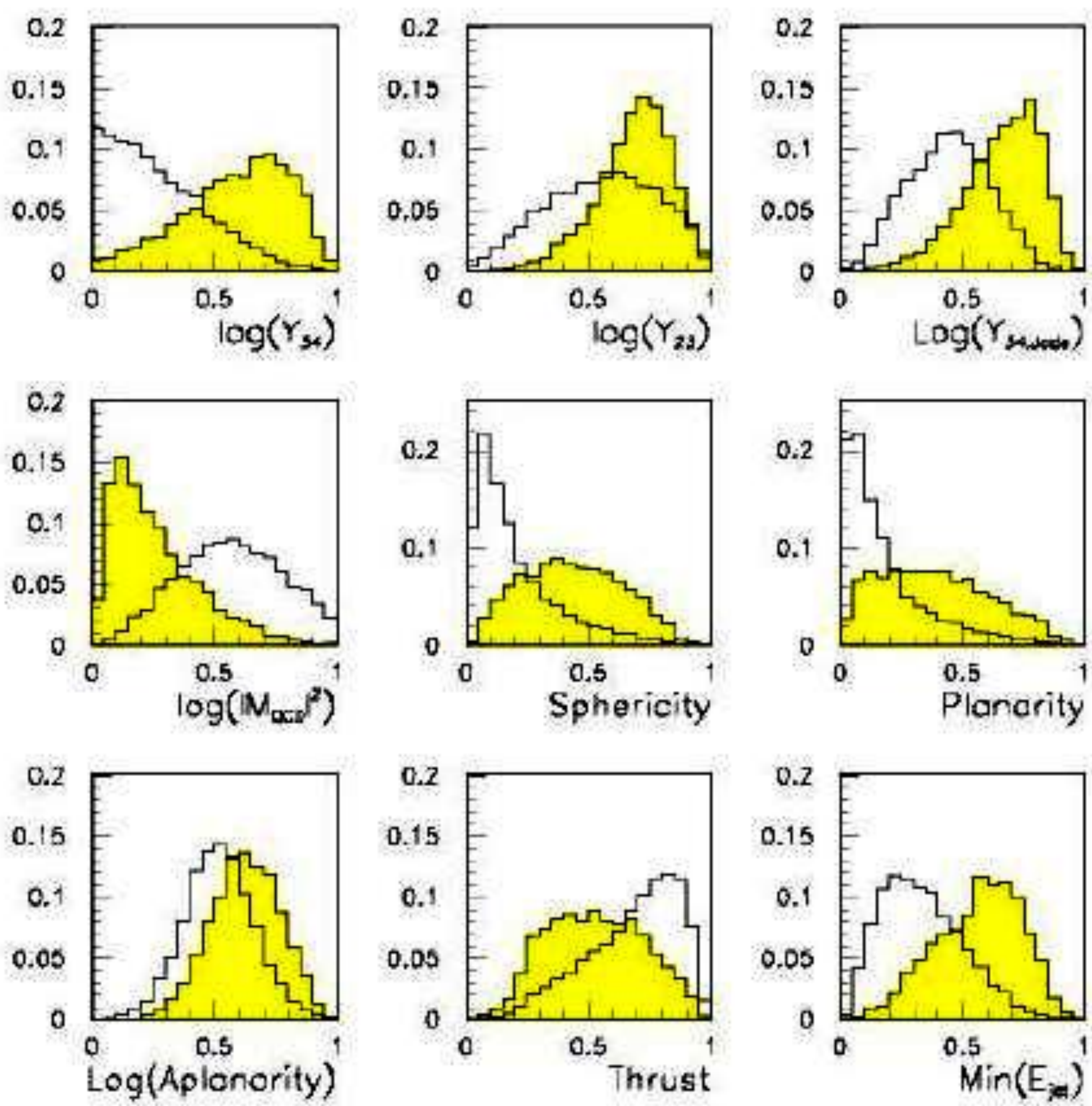
9. Machine Learning

Heidelberg University, WS 2020/21

Klaus Reygers (lectures)
Rainer Stamen, Martin Völkl (tutorials)

Multivariate analysis: An early example from particle physics

G. Cowan, Lecture on Statistical data analysis



(Garrido, Juste and Martinez, ALEPH 96-144)

Signal: $e^+e^- \to W^+W^-$

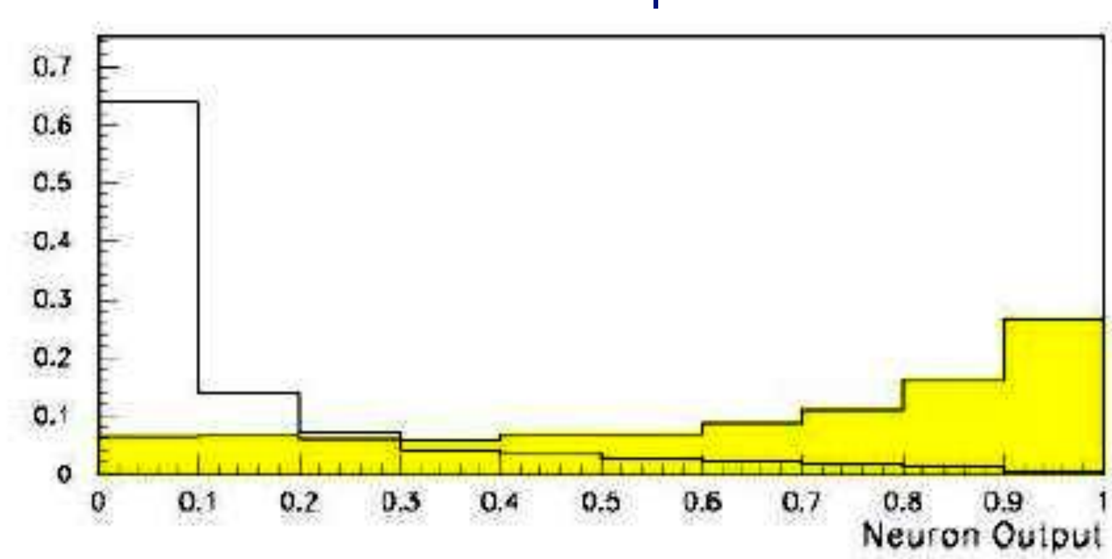
often 4 well separated hadron jets

Background: e+e– → qqgg

4 less well separated hadron jets

← input variables based on jet structure, event shape, ... none by itself gives much separation.

Neural network output:

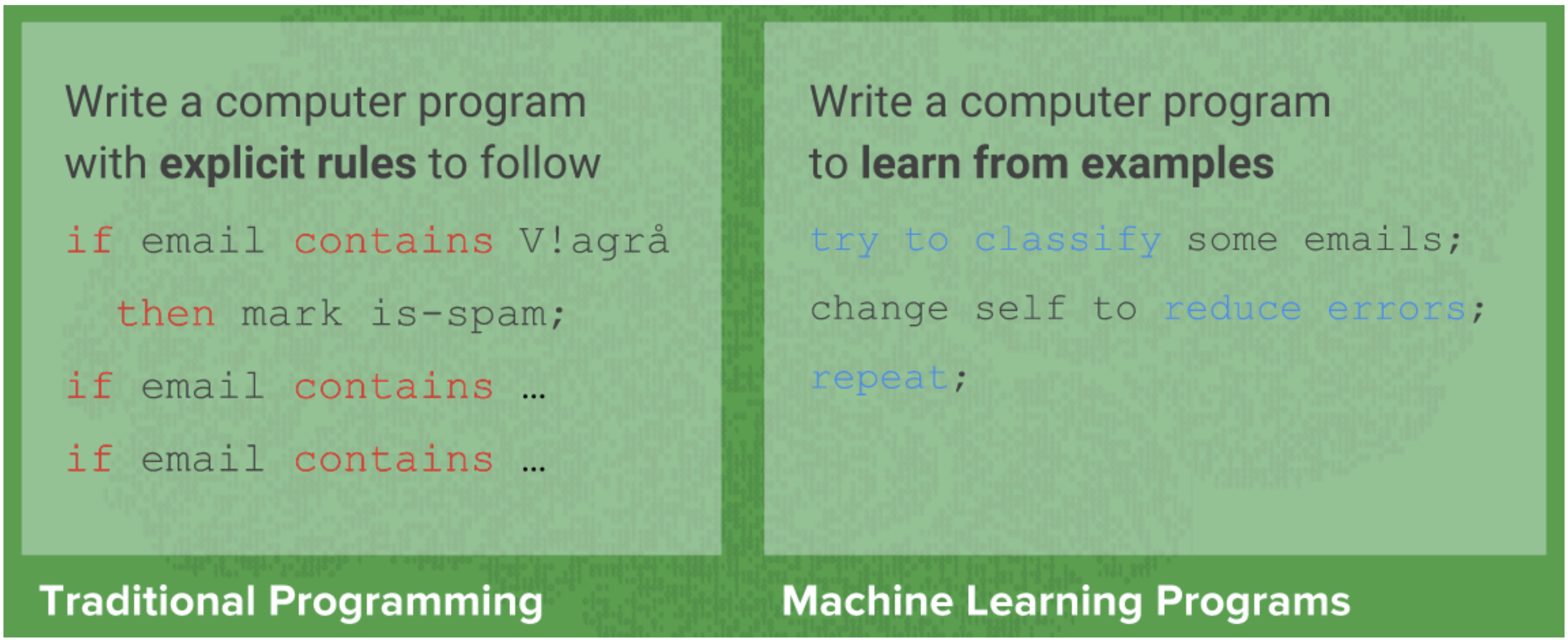


Machine learning

"Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed" – Wikipedia

Example: spam detection

J. Mayes, Machine learning 101

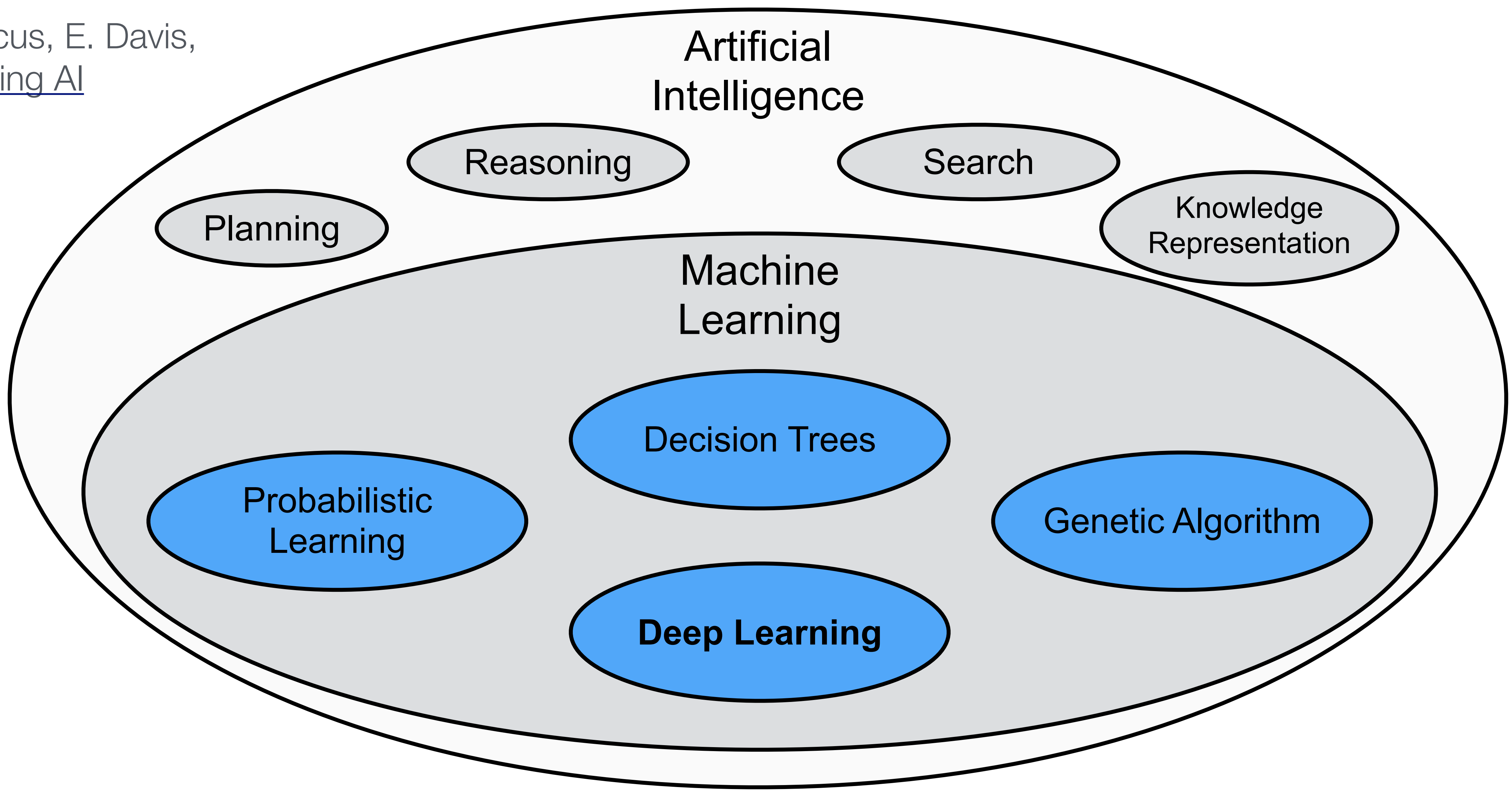


Manual feature engineering vs. automatic feature detection

AI, ML, and DL

"AI is the study of how to make computers perform things that, at the moment, people do better." Elaine Rich, Artificial intelligence, McGraw-Hill 1983

G. Marcus, E. Davis, Rebooting AI



"deep" in deep learning: artificial neural nets with many neurons and multiple layers of nonlinear processing units for feature extraction

Some successes and unsolved problems in AI

Arithmetic (1945)	Easy
Sorting lists of numbers (1959)	
Playing simple board games (1959)	Solved, after a lot of effort
Playing chess (1997)	
Recognizing faces in pictures (2008)	
Usable automated translation (2010)	
Playing Go (2016)	
Usable real-time translation of spoken words (2016)	
Driverless cars	Real progress
Automatically providing captions for pictures	
Understanding a story & answering questions about it	Nowhere near solved
Human-level automated translation	
Interpreting what is going on in a photograph	
Writing interesting stories	
Interpreting a work of art	
Human-level general intelligence	

M. Woolridge,
The Road to Conscious Machines

Impressive progress in certain fields:

- Image recognition
- Speech recognition
- Recommendation systems
- Automated translation
- Analysis of medical data

How can we profit from these developments in physics?

Different modeling approaches

- Simple mathematical representation like linear regression. Favored by statisticians.
- Complex deterministic models based on scientific understanding of the physical process. Favored by physicists.
- Complex algorithms to make predictions that are derived from a huge number of past examples (“machine learning” as developed in the field of computer science). These are often black boxes.
- Regression models that claim to reach causal conclusions. Used by economists.

D. Spiegelhalter, The Art of Statistics – Learning from data

Application of machine learning in experimental particle physics

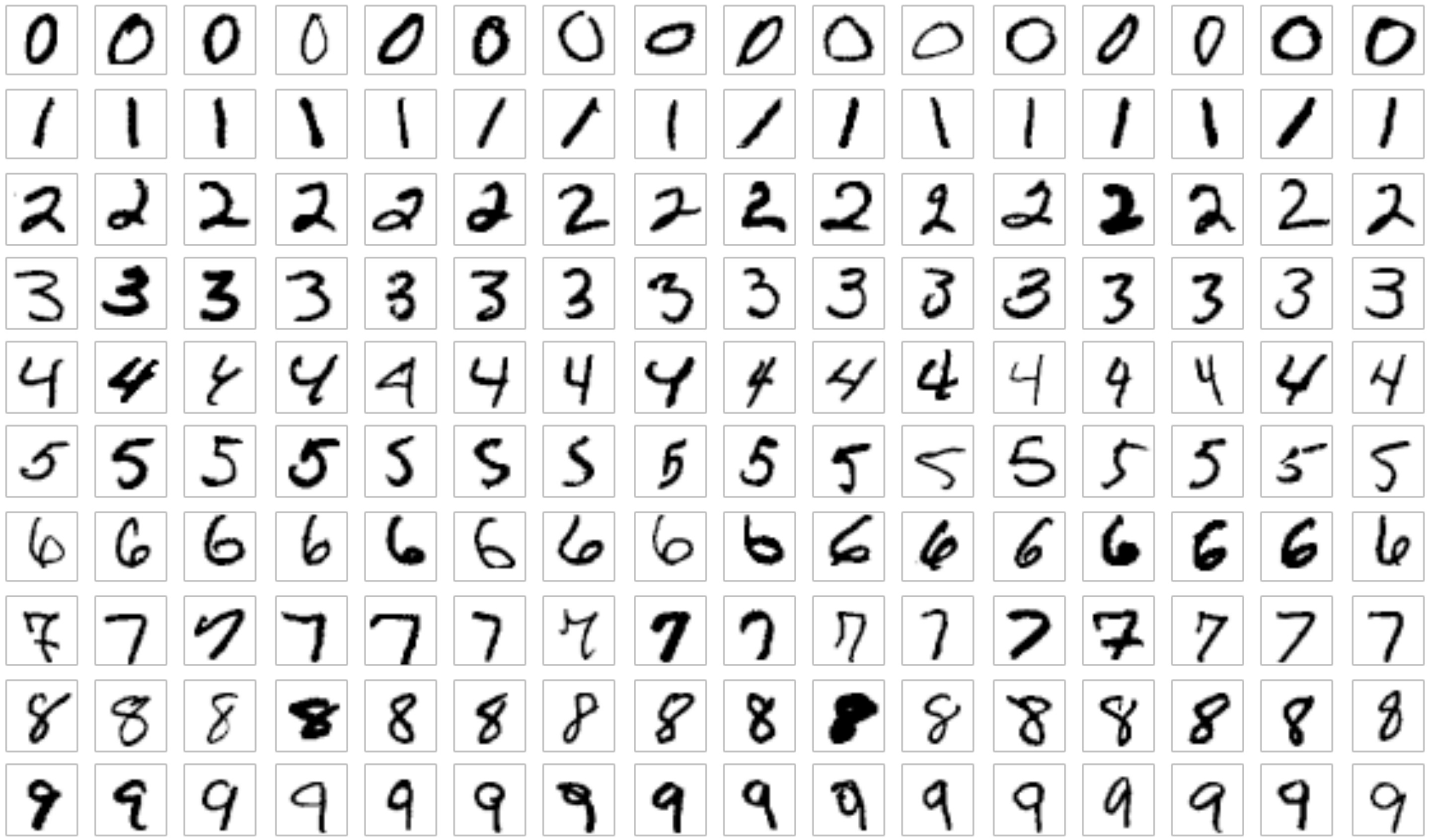
- Monte Carlo simulation
  - use generative models for faster MC event generation
- Event reconstruction and particle identification
- Data acquisition / trigger
  - faster algorithms
- Offline data analysis
  - better algorithms
- Detector monitoring
  - anomaly detection

"Machine Learning in High Energy Physics Community White Paper", arXiv:1807.02876

Machine learning: The "hello world" problem

Recognition of handwritten digits

- MNIST database
(Modified National Institute of Standards and Technology database)
- 60,000 training images and 10,000 testing images labeled with correct answer
- 28 pixel x 28 pixel
- Algorithms have reached "near-human performance"
- Smallest error rate (2018): 0.18%



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

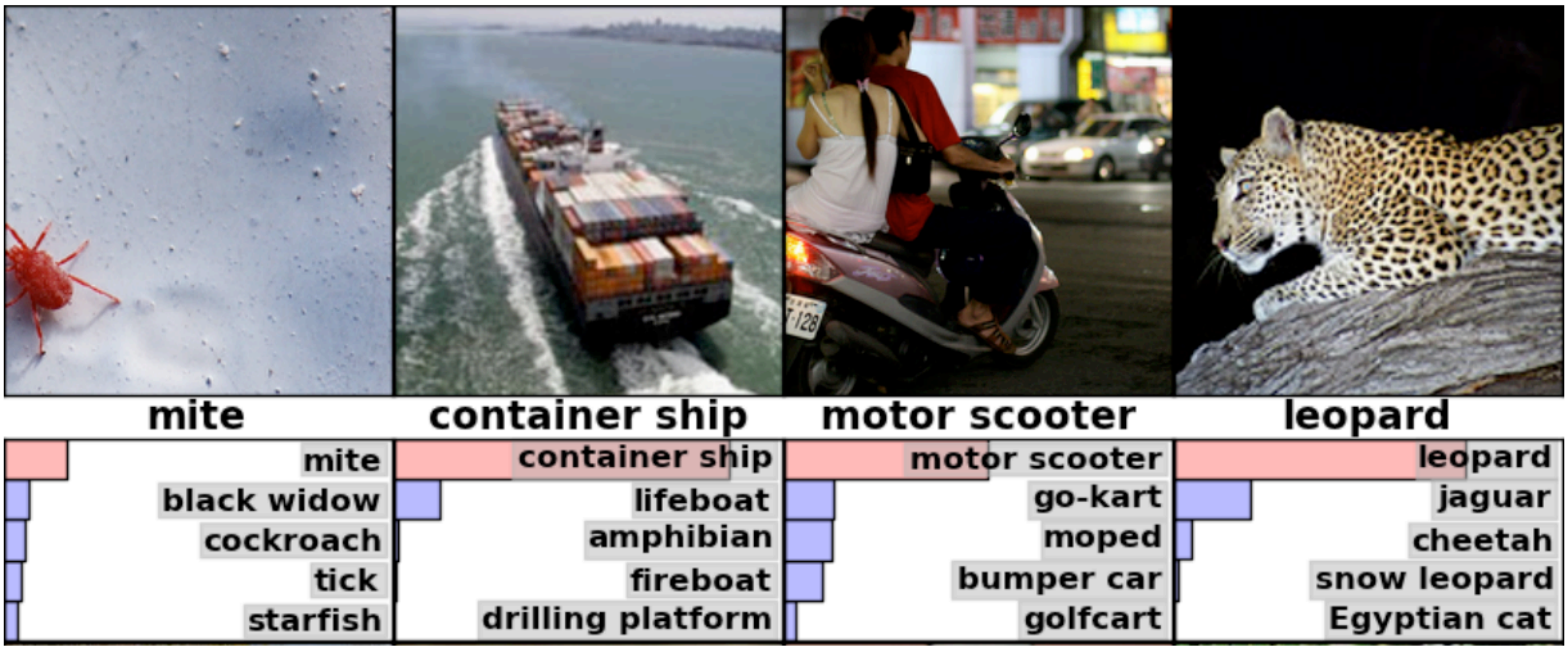
Play with MNIST data set and Keras (Stefan Wunsch, CERN IML Workshop):
https://github.com/stwunsch/iml_tensorflow_keras_workshop

Machine learning: Image recognition

ImageNet database

- 14 million images, 22,000 categories
- Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC): 1.4 million images, 1000 categories
- In 2017, 29 of 38 competing teams got less than 5% wrong

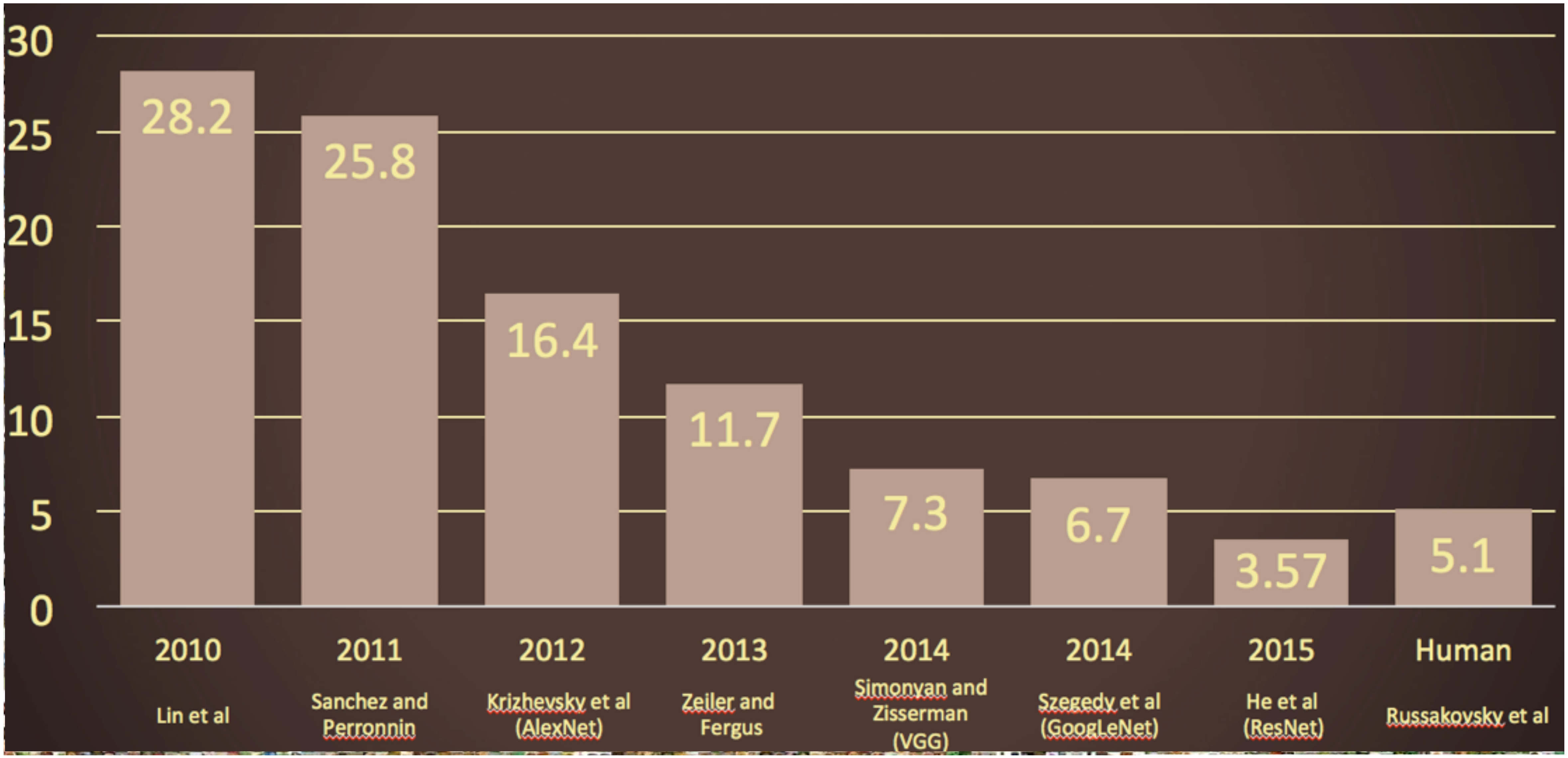
https://en.wikipedia.org/wiki/ImageNet



https://www.tensorflow.org/tutorials/image_recognition

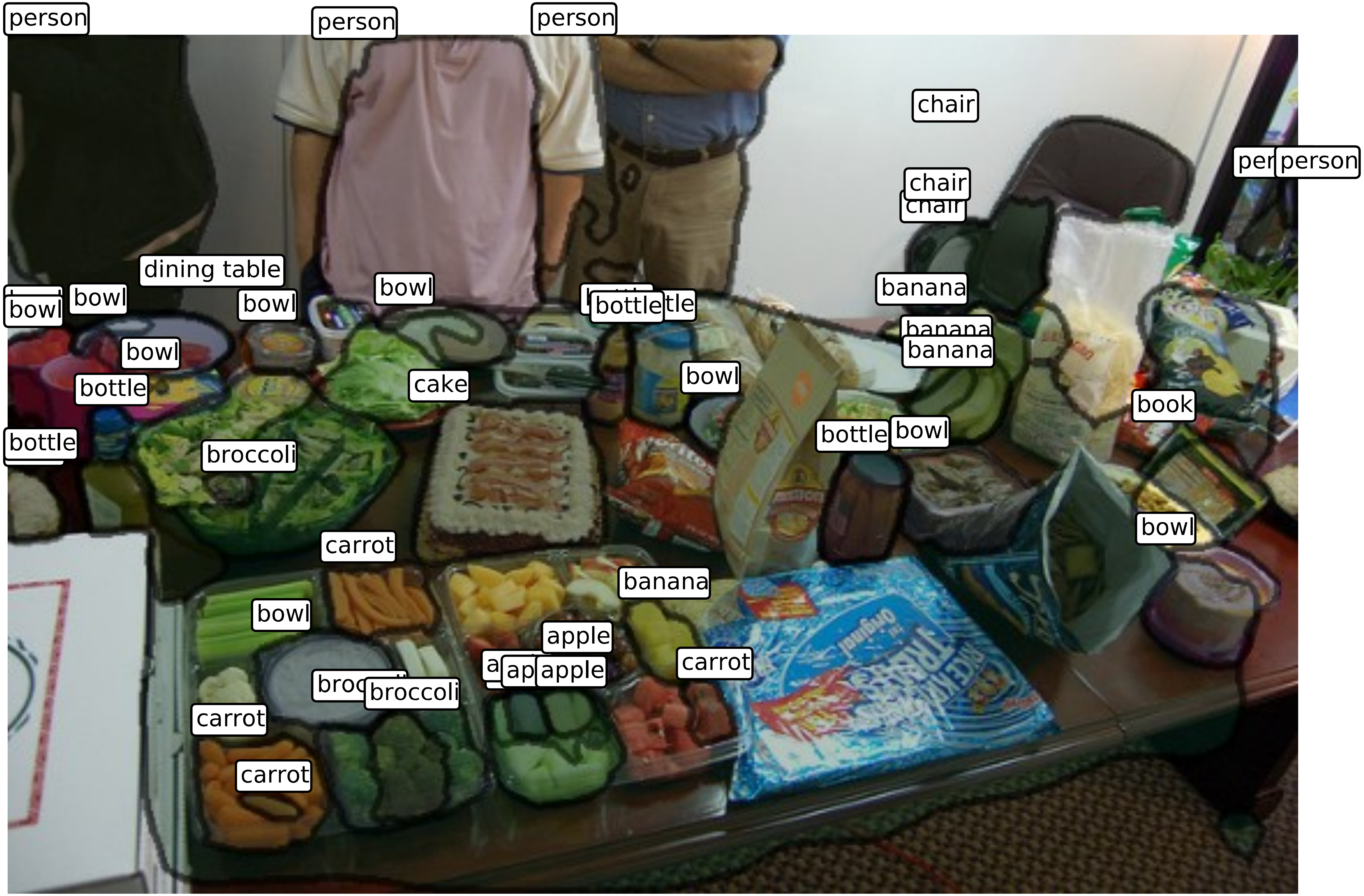
ImageNet: Large Scale Visual Recognition Challenge

Error rate in percent:



O. Russakovsky et al, arXiv:1409.0575

Further examples (1):
Segmenting and localizing objects



Further examples (2): Image captioning

- [Lebret, Pinheiro, Collobert 2015] [Kulkarni 11] [Mitchell 12] [Vinyals 14] [Mao 14]

A man riding skis on a snow covered ski slope.
NP: a man, skis, the snow, a person, a woman, a snow covered slope, a slope, a snowboard, a skier, man.
VP: wearing, riding, holding, standing on, skiing down.
PP: on, in, of, with, down.
A man wearing skis on the snow.

A man is doing skateboard tricks on a ramp.
NP: a skateboard, a man, a trick, his skateboard, the air, a skateboarder, a ramp, a skate board, a person, a woman.
VP: doing, riding, is doing, performing, flying through.
PP: on, of, in, at, with.
A man riding a skateboard on a ramp.

The girl with blue hair stands under the umbrella.
NP: a woman, an umbrella, a man, a person, a girl, umbrellas, that, a little girl, a cell phone.
VP: holding, wearing, is holding, holds, carrying.
PP: with, on, of, in, under.
A woman is holding an umbrella.

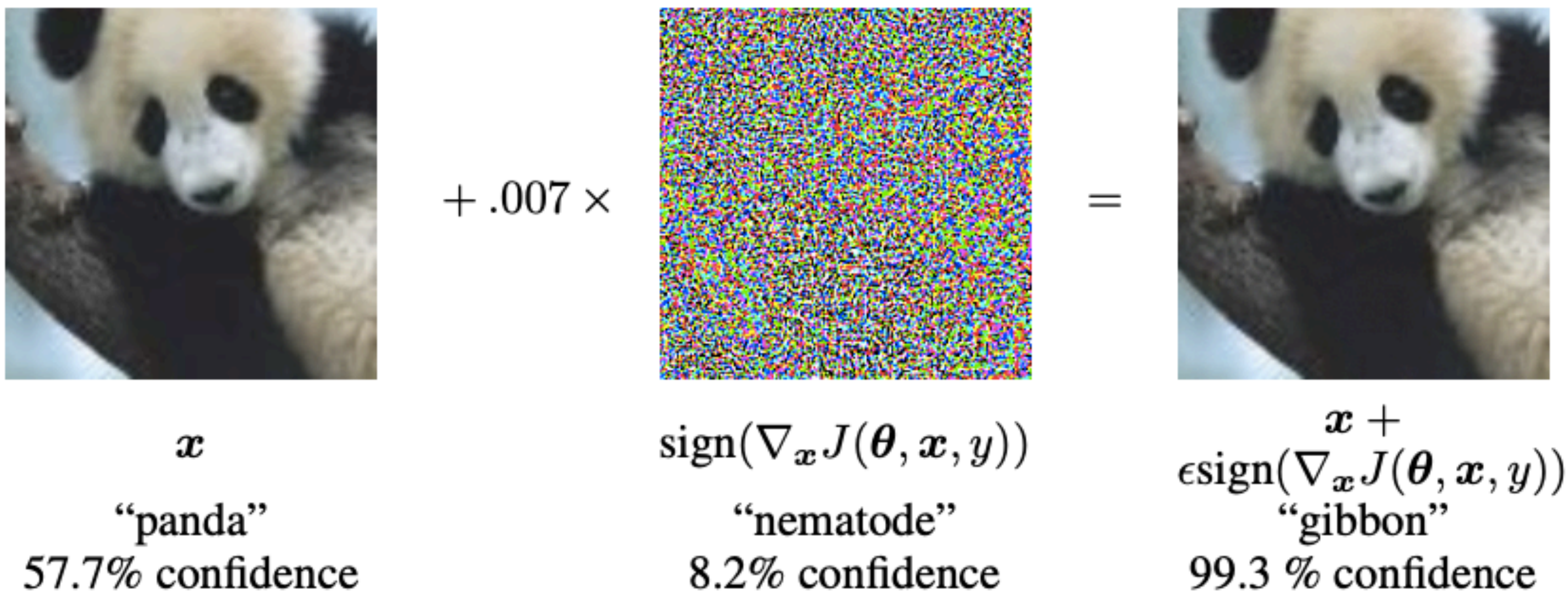
A slice of pizza sitting on top of a white plate.
NP: a plate, a white plate, a table, pizza, it, a pizza, food, a sandwich, top, a close.
VP: topped with, has, is, sitting on, is on.
PP: of, on, with, in, up.
A table with a plate of pizza on a white plate.

A baseball player swinging a bat on a field.
NP: the ball, a game, a baseball player, a man, a tennis court, a ball, home plate, a baseball game, a batter, a field.
VP: swinging, to hit, playing, holding, is swinging.
PP: on, during, in, at, of.
A baseball player swinging a bat on a baseball field.

A bunch of kites flying in the sky on the beach.
NP: the beach, a beach, a kite, kites, the ocean, the water, the sky, people, a sandy beach, a group.
VP: flying, flies, is flying, flying in, are.
PP: on, of, with, in, at.
People flying kites on the beach.

Adversarial examples

Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, arXiv:1412.6572v1

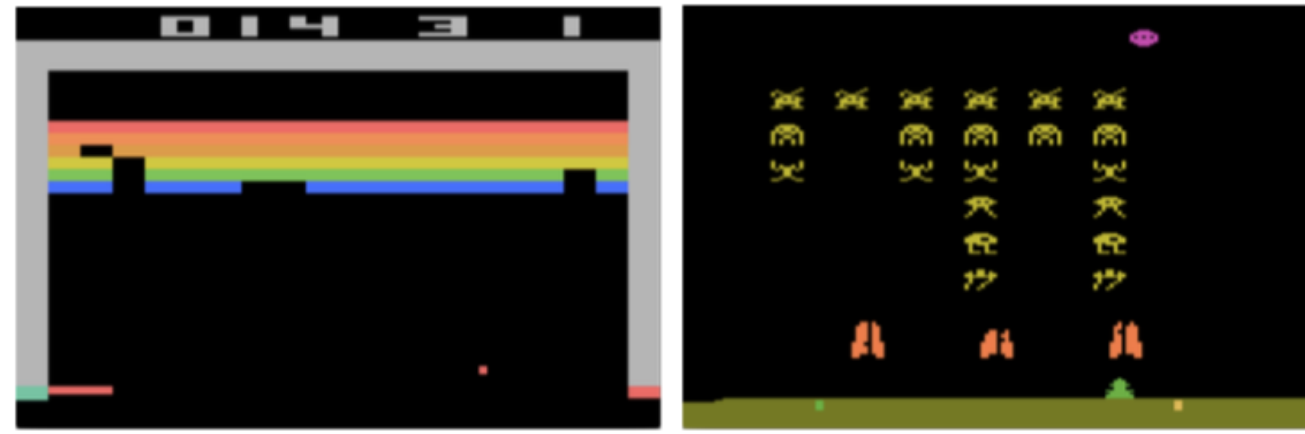


Three types of learning

LeCun 2018, Power And Limits of Deep Learning, https://www.youtube.com/watch?v=0tEhw5t6rhc

Reinforcement learning

- The machine ("the agent") predicts a scalar reward given once in a while
- Weak feedback



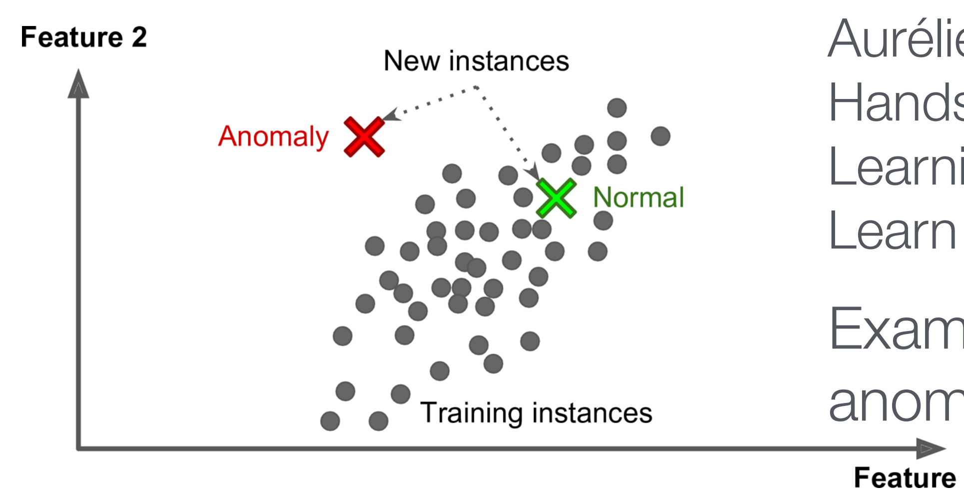
arXiv:1312.5602

Supervised learning

- The machine predicts a category based on labeled training data
- Medium feedback

Unsupervised learning

- Describe/find hidden structure from "unlabeled" data
- Cluster data in different sub-groups with similar properties

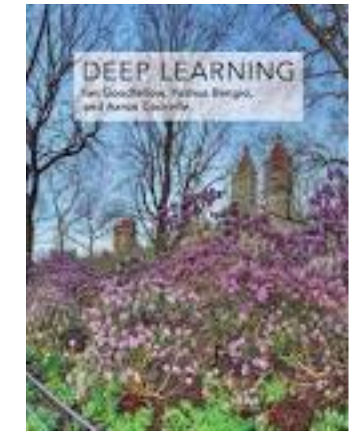


Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

Example: anomaly detection

Books on machine learning

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, free online http://www.deeplearningbook.org/



- Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*

Multivariate classification

Consider events which can be either signal or background events.

Each event is characterized by *n* observables:

$$\vec{x} = (x_1, \ldots, x_n) \qquad \text{"feature vector"}$$

Goal: classify events as signal or background in an optimal way.

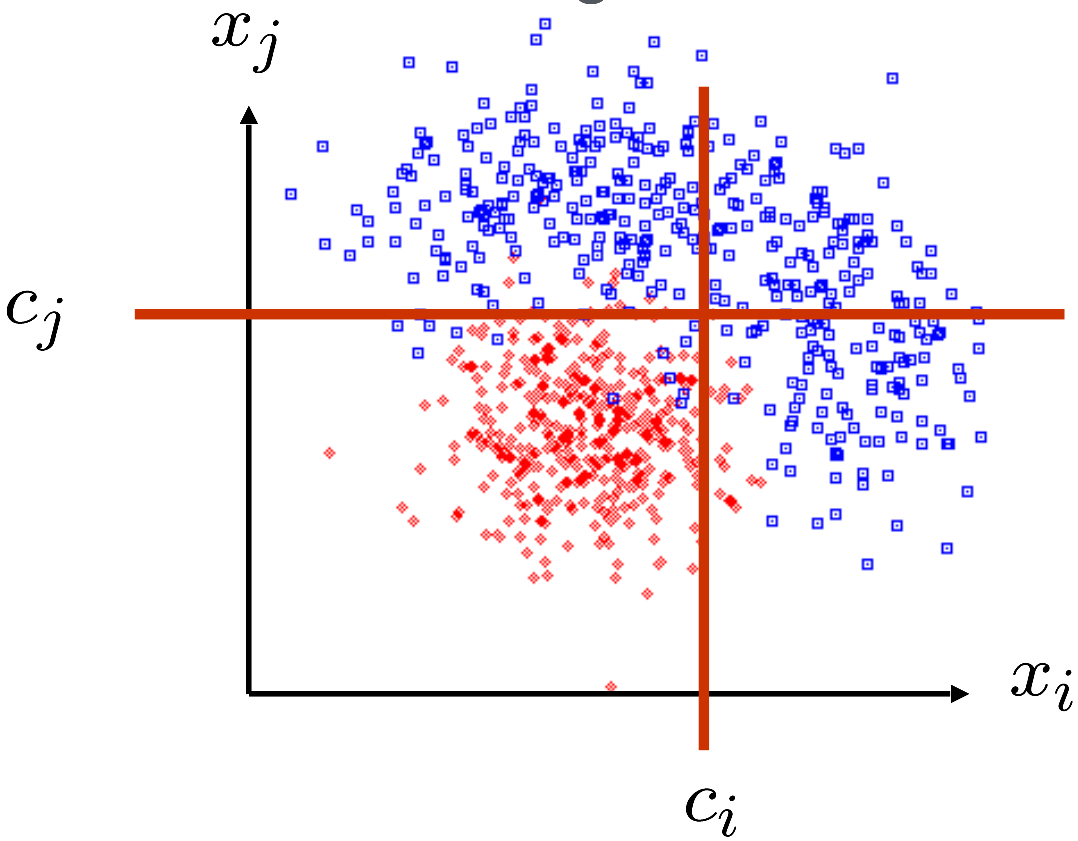
This is usually done by mapping the feature vector to a single variable, i.e., to scalar "test statistic":

$$\mathbb{R}^n \to \mathbb{R}: \quad y(\vec{x})$$

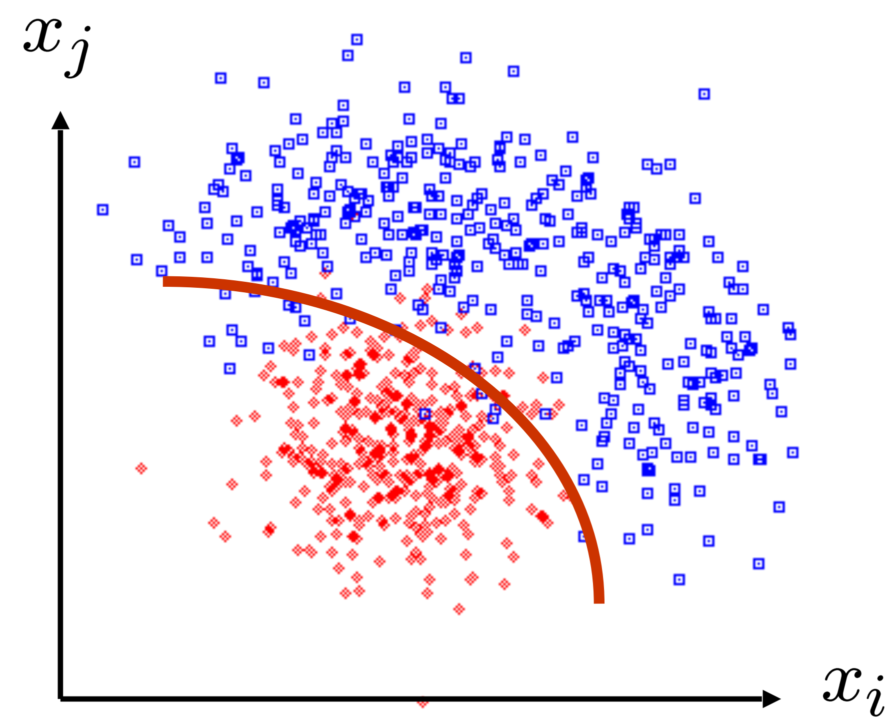
A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the x_i.

Classification: Learning decision boundaries

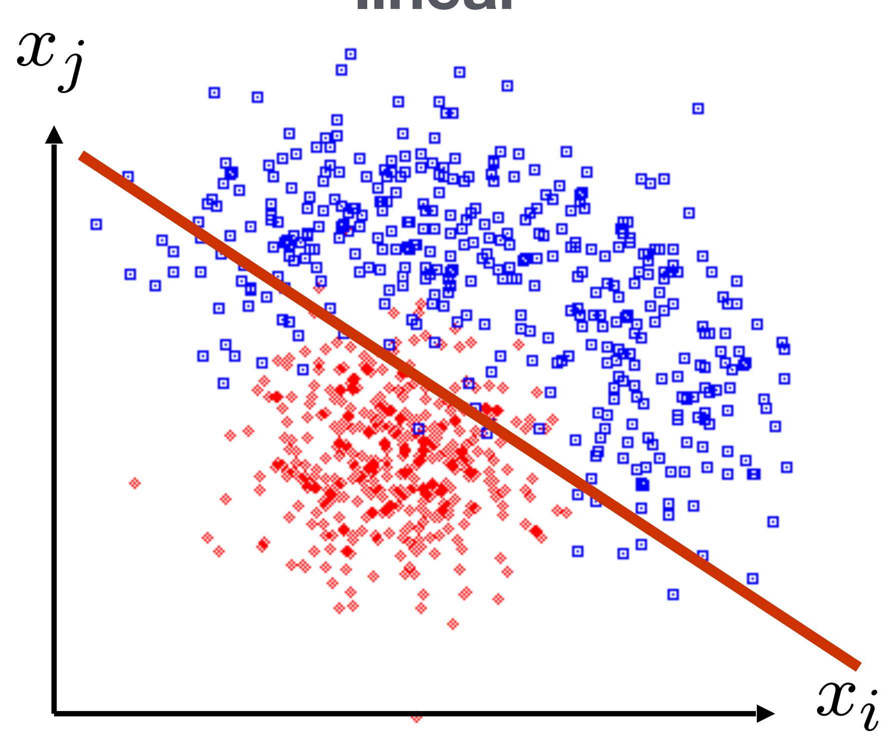
rectangular cuts



non linear



linear



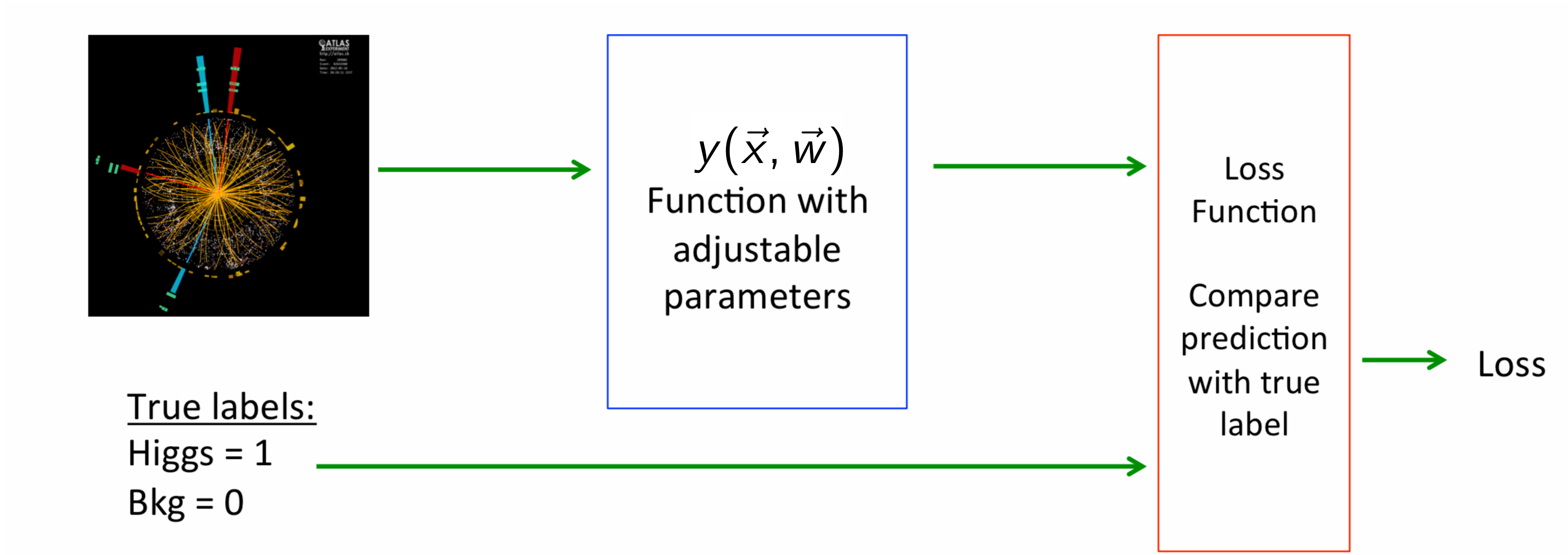
k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines
…

G. Cowan:
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Supervised learning in a nutshell

M. Kagan, https://indico.cern.ch/event/619370/

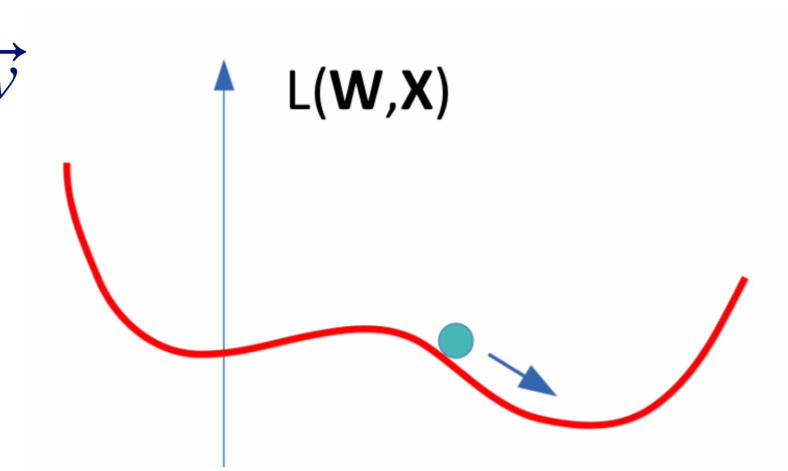
Supervised Machine Learning requires *labeled training data*, i.e., a training sample where for each event it is known whether it is a signal or background event



Design function $y(\vec{x}, \vec{w})$ with ajdustable parameters $\vec{w}$

Design a loss function

Find best parameters which minimize loss



Supervised learning: classification and regression

The codomain Y of the function y: $X \to Y$ can be a set of labels or classes or a continuous domain, e.g., $\mathbb{R}$

Binary classification: $Y = \{0, 1\}$ e.g., signal or background

Multi-class classification $Y = \{c_1, c_2, \ldots, c_n\}$

Labels sometimes represented as "**one-hot vector**" (no ordering btw. labels): $t_a = \{0, 0, \ldots, 1, \ldots, 0\}$

Y = finite set of labels → classification

Y = real numbers → regression

"All the impressive achievements of deep learning amount to just curve fitting"

J. Pearl, Turing Award Winner 2011,
https://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/

Supervised learning: Training, validation, and test sample

- Decision boundary fixed with training sample
- Performance on training sample becomes better with more iterations
- Danger of overtraining:
  Statistical fluctuations of the training sample will be learnt
- Validation sample = independent labeled data set not used for training
  → check for overtraining
- Sign of overtraining: performance on validation sample becomes worse
  → Stop training when signs of overtraining are observed ("early stopping")
- Performance: apply classifier to independent test sample
- Often: test sample = validation sample (only small bias)

Supervised learning: Cross validation

Rule of thumb if training data not expensive

- Training sample: 50%
- Validation sample: 25%
- Test sample: 25%

often test sample = validation sample,
i.e., training : validation/test = 50:50

Cross validation (efficient use of scarce training data)

- Split training sample in k independent subset T_k of the full sample T
- Train on $T \setminus T_k$ resulting in k different classifiers
- For each training event there is one classifier that didn't use this event for training
- Validation results are then combined

Often used loss functions

predicted label **true label**

Square error loss:

- often used in regression

$$E(y(\vec{x}, \vec{w}), t) = (y(\vec{x}, \vec{w}) - t)^2$$

predicted "probability" for outcome t = 1

Cross entropy:

- $t \in \{0, 1\}$
- Often used in classification

$$\begin{aligned} E(y(\vec{x}, \vec{w}), t) = &- t \log y(\vec{x}, \vec{w}) \\ &- (1 - t) \log(1 - y(\vec{x}, \vec{w})) \end{aligned}$$

More on entropy

Self-information of an event x: $I(x) = -\log p(x)$

Shannon entropy: $H(P) = -\sum p_i \log p_i$

- Expected amount of information in an event drawn from a distribution *P*.
- Measure of the minimum of amount of bits needed on average to encode symbols drawn from a distribution

Cross entropy: $H(P, Q) = -E[\log q_i] = -\sum p_i \log q_i$

- Can be interpreted as a measure of the amount of bits needed when a wrong distribution Q is assumed while the data actually follows a distribution P
- Measure of dissimilarity between distributions P and Q (i.e, a measure of how well the model Q describes the true distribution P)

Cross-entropy error function for logistic regression

Let $Y \in \{0,1\}$ be a random variable; outcome of experiment *i*: y_i

Consider one event with feature vector $\vec{x}$ and label $y \in \{0,1\}$

Predicted probability q_1 for outcome $Y = 1$:

By construction the right property for predicting a probability

$$q_1 \equiv q(Y=1) = \sigma(\vec{x}; \vec{w}) \equiv \sigma(w_0 + \sum_{i=1}^{n} w_i x_i), \qquad \sigma: \mathbb{R} \mapsto [0,1], \ \ \sigma(z) = \frac{1}{1+e^{-z}}$$

$$q_0 \equiv q(Y=0) = 1 - q(Y=1)$$

logistic function

The true probabilities p_i are either 0 or 1, so we can write
$p_1 \equiv p(Y=1) = y, \ \ p_0 = 1 - p_1 \equiv 1 - y$. With this the cross entropy is:

$$H(p,q) \ = -\sum_{k=0}^{1} p_k \log q_k \ = -y \ \log \sigma(\vec{x}, \vec{w}) - (1-y) \ \log(1 - \sigma(\vec{x}, \vec{w}))$$

Loss function from sum over entire data set:

$$E(\vec{w}) = -\sum_{i=1}^{n_\text{samples}} y_i \ \log \sigma(\vec{x}_i, \vec{w}) + (1-y_i) \ \log(1 - \sigma(\vec{x}_i, \vec{w}))$$

Logistic regression: loss function from maximum likelihood

We can write the two predicted probabilities q_0 and q_1 in the following way:

$$q(Y = y) = \sigma(\vec{x}; \vec{w})^y \cdot (1 - \sigma(\vec{x}; \vec{w}))^{1-y}$$

With this the likelihood can be written as

$$L(\vec{w}) = \prod_{i=1}^{n_{\text{samples}}} q(Y = y_i)$$

$$= \prod_{i=1}^{n_{\text{samples}}} \sigma(\vec{x}; \vec{w})^{y_i} \cdot (1 - \sigma(\vec{x}; \vec{w}))^{1-y_i}$$

The corresponding log-likelihood function is

$$\log L(\vec{w}) = \sum_{i=1}^{n_{\text{samples}}} y_i \log \sigma(\vec{x}_i; \vec{w}) + (1 - y_i) \log(1 - \sigma(\vec{x}_i; \vec{w}))$$

Thus, minimizing the cross entropy loss function corresponds to finding the maximum likelihood estimate.

Multinomial logistic regression: Softmax function

In the previous example we considered two classes (0, 1). For multi-class classification, the logistic function can generalized to the softmax function.

Consider K classes and let z_i be the score for class *i*, $\vec{z} = (z_1, \ldots, z_K)$

A probability for class *i* can be predicted with the softmax function:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for} \quad i = 1,\ldots,K$$

The softmax functions is often used as the last activation function of a neural network in order to predict probabilities in a classification task.

Multinomial logistic regression is also known as softmax regression.

Simple example of logistic regression with scikit-learn (1)

Read data

https://scikit-learn.org

Data are from the wikipedia article on logistic regression

```
# data: 1. hours studies, 2. passed (0/1)
filename = "data/exam.txt"
df = pd.read_csv(filename, engine='python', sep='\s+')
```

```
x_tmp = df['hours_studied'].values
x = np.reshape(x_tmp, (-1, 1))
y = df['passed'].values
```

Fit the model

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='none', fit_intercept=True)
clf.fit(x, y);
```

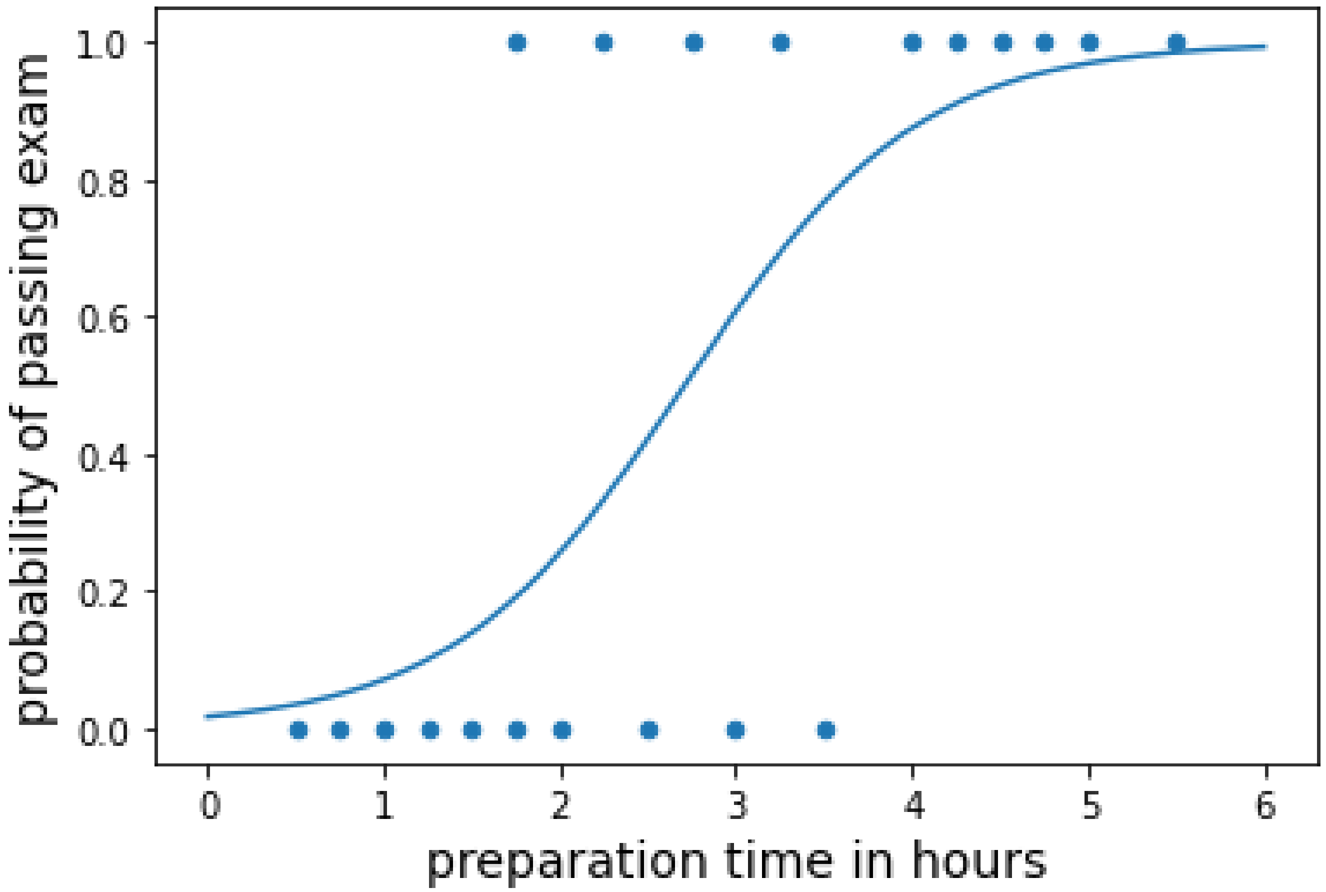
Calculate predictions

```
hours_studied_tmp = np.linspace(0., 6., 1000)
hours_studied = np.reshape(hours_studied_tmp, (-1, 1))
y_pred = clf.predict_proba(hours_studied)
```

Simple example of logistic regression with scikit-learn (2)

Plot result

```
df.plot.scatter(x='hours_studied', y='passed')
plt.plot(hours_studied, y_pred[:,1])
plt.xlabel("preparation time in hours", fontsize=14)
plt.ylabel("probability of passing exam", fontsize=14)
plt.savefig("logistic_regression.pdf")
```



Reminder: Neyman–Pearson lemma

The likelihood ratio

$$t(\vec{x}) = \frac{f(\vec{x}|H_1)}{f(\vec{x}|H_0)} \qquad \begin{aligned} H_1 &: \text{ signal hypothesis} \\ H_0 &: \text{ background hypothesis} \end{aligned}$$

is an optimal test statistic, i.e., it provides highest "signal efficiency" 1 – β for a given "background efficiency" α.

Accept hypothesis if $\quad t(\vec{x}) = \frac{f(\vec{x}|H_1)}{f(\vec{x}|H_0)} > c$

Problem: the underlying pdf's are almost never known explicitly.

Two approaches:

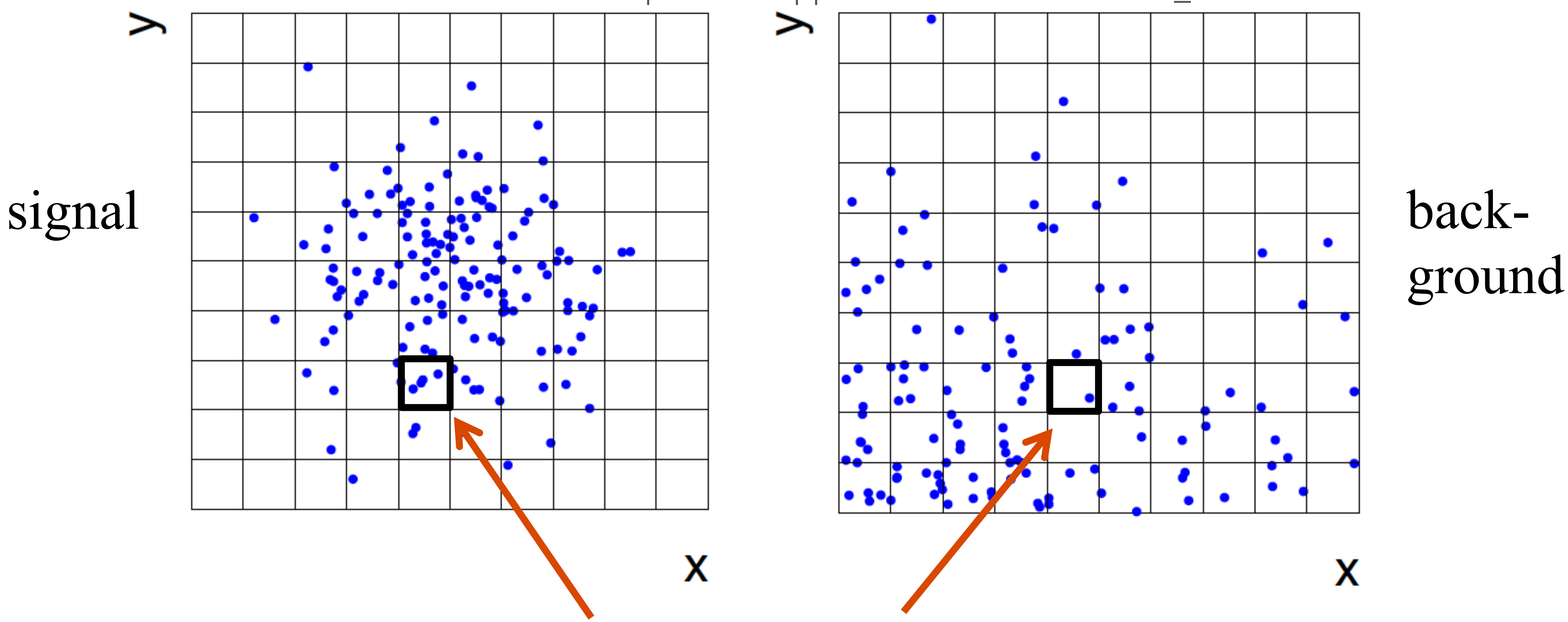
1. Estimate signal and background pdf's and construct test statistic based on Neyman-Pearson lemma

2. Decision boundaries determined directly without approximating the pdf's (linear discriminants, decision trees, neural networks, …)

Estimating PDFs from histograms?

Consider 2d example:

G. Cowan': https://www.pp.rhul.ac.uk/~cowan/stat_course.html



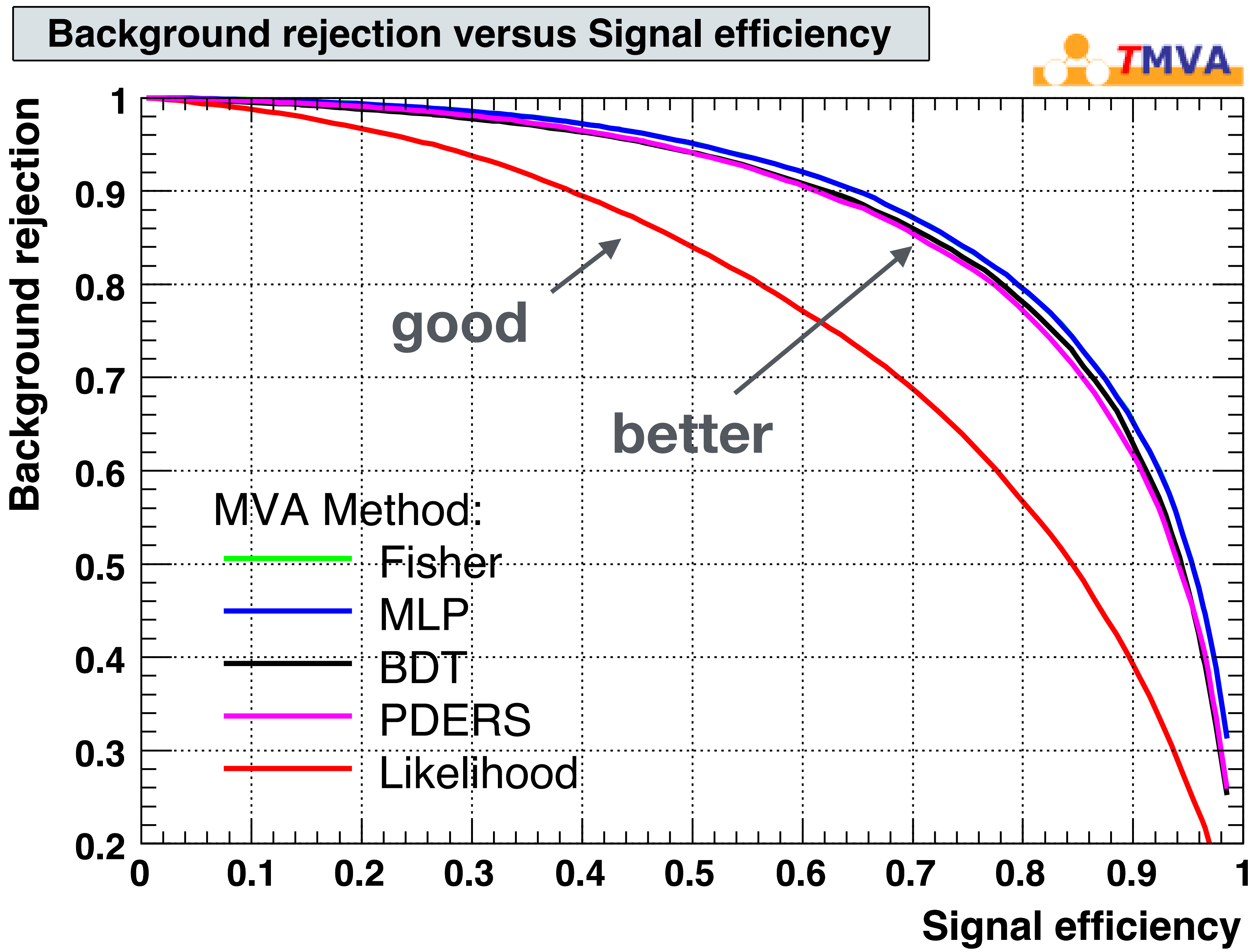
approximate PDF by $N(x, y|S)$ and $N(x, y|B)$

M bins per variable in *d* dimensions: M^d cells
→ hard to generate enough training data (often not practical for $d > 1$)

In general in machine learning, problems related to a large number of dimensions of the feature space are referred to as the "**curse of dimensionality**"

ROC Curve

Quality of the classification can be characterized by the *receiver operating characteristic* (ROC curve)



$1 - \varepsilon_B$

ε_B: background efficiency

Naïve Bayesian classifier (also called "projected likelihood classification")

Application of the Neyman-Pearson lemma (ignoring correlations between the x_i):

$$f(x_1, x_2, ..., x_n) \quad \text{approximated as} \quad L = f_1(x_1) \cdot f_2(x_2) \cdot ... \cdot f_n(x_n)$$

$$\text{where} \quad f_1(x_1) = \int \mathrm{d}x_2 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n)$$

$$f_2(x_2) = \int \mathrm{d}x_1 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n)$$

$$\vdots$$

Classification of feature vector $\vec{x}$:

$$y(\vec{x}) = \frac{L_\mathrm{s}(\vec{x})}{L_\mathrm{s}(\vec{x}) + L_\mathrm{b}(\vec{x})} = \frac{1}{1 + L_\mathrm{b}(\vec{x})/L_\mathrm{s}(\vec{x})}$$

Performance not optimal if true PDF does not factorize

k-nearest neighbor method (1)

k-NN classifier

- Estimates probability density around the input vector
- $p(\vec{x}|S)$ and $p(\vec{x}|B)$ are approximated by the number of signal and background events in the training sample that lie in a small volume around the point $\vec{x}$

Algorithms finds k nearest neighbors:

$$k = k_s + k_b$$

Probability for the event to be of signal type:

$$p_s(\vec{x}) = \frac{k_s(\vec{x})}{k_s(\vec{x}) + k_b(\vec{x})}$$

k-nearest neighbor method (2)

Simplest choice for distance measure in feature space is the Euclidean distance:

$$R = |\vec{x} - \vec{y}|$$

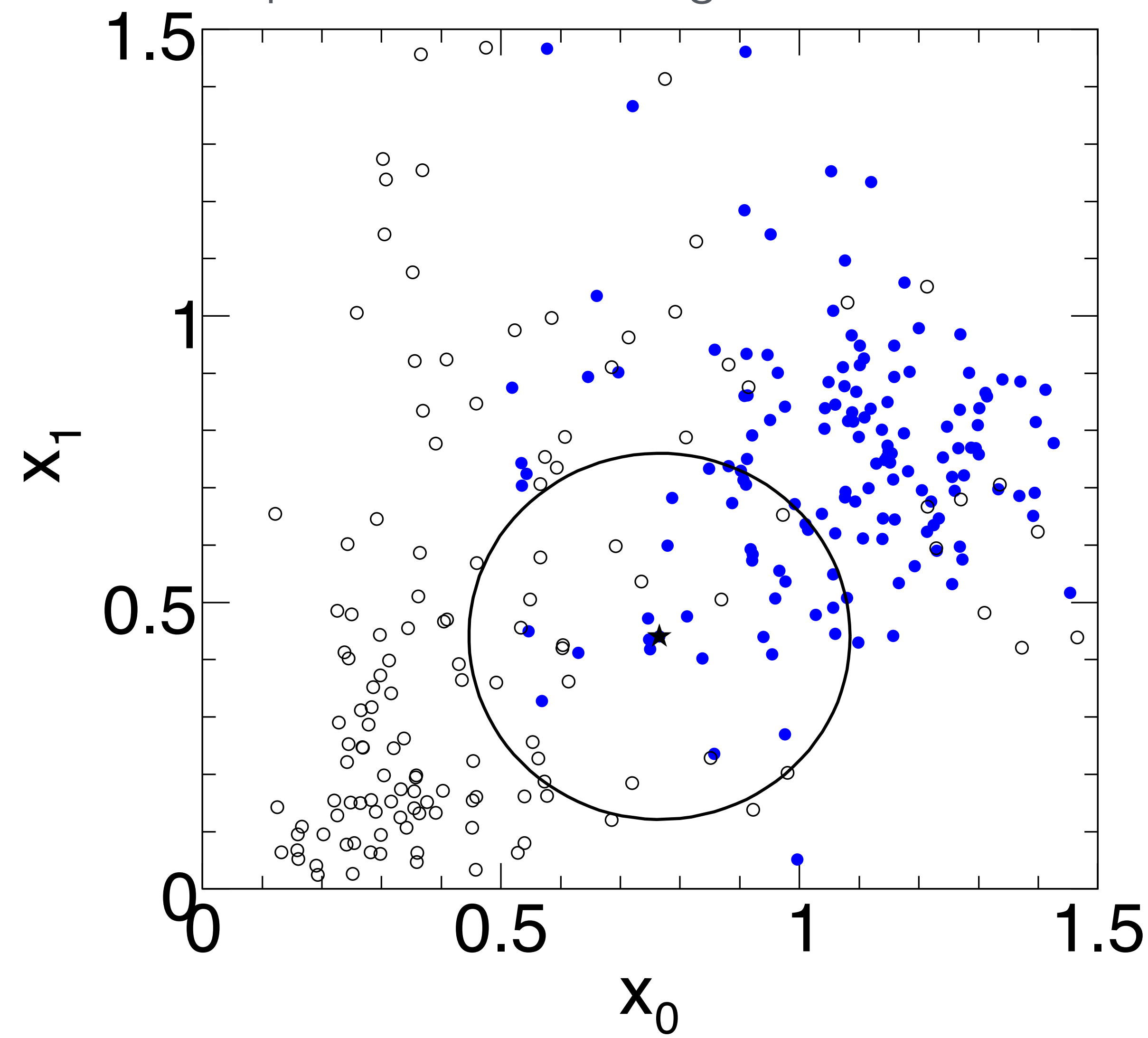
Better: take correlations between variables into account:

$$R = \sqrt{(\vec{x} - \vec{y})^T V^{-1} (\vec{x} - \vec{y})}$$

$$V = \text{covariance matrix}$$

"Mahalanobis distance"

TMVA manual
https://root.cern.ch/guides/tmva-manual



The k-NN classifier has best performance when the boundary that separates signal and background events has irregular features that cannot be easily approximated by parametric learning methods.

Fisher linear discriminant

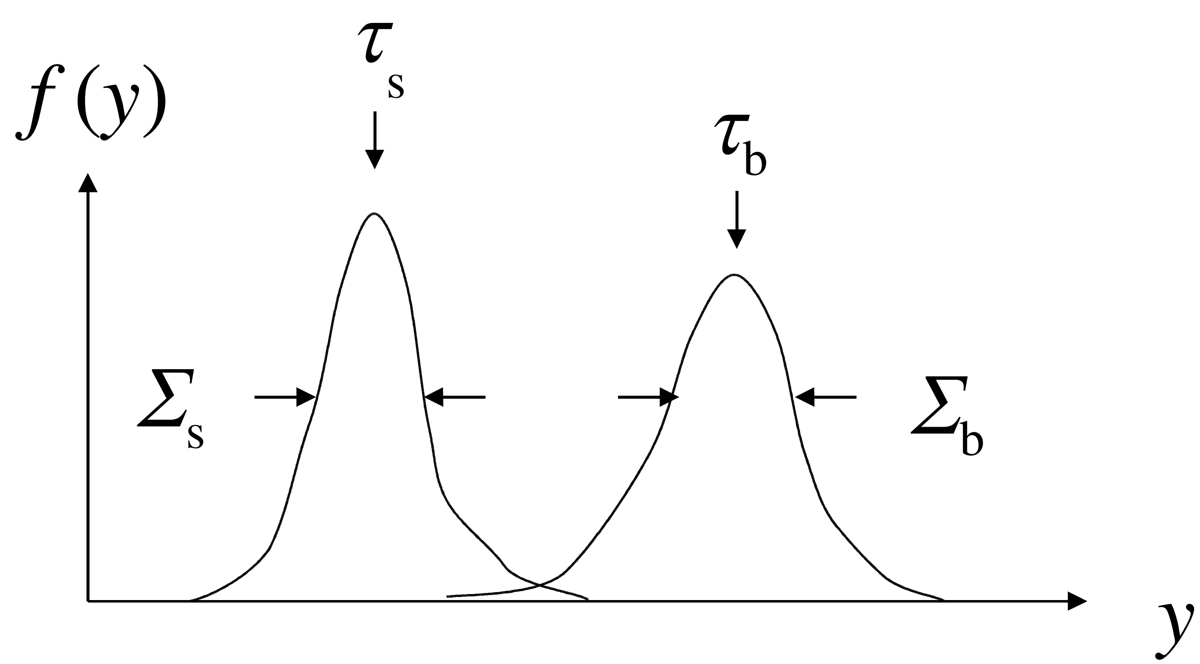
Linear discriminant is simple. Can still be optimal if amount of training data is limited.

Ansatz for test statistic: $$y(\vec{x}) = \sum_{i=1}^{n} w_i x_i = \vec{w}^{\mathsf{T}} \vec{x}$$

Choose parameters w_i so that separation between signal and background distribution is maximum.

Need to define "separation".

Fisher: maximize $$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher linear discriminant: Variable definitions

Mean and covariance for signal and background:

$$\mu_i^{\mathrm{s,b}} = \int x_i \, f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x}$$

$$V_{ij}^{\mathrm{s,b}} = \int (x_i - \mu_i^{\mathrm{s,b}})(x_j - \mu_j^{\mathrm{s,b}}) \, f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x}$$

Mean and variance of $y(\vec{x})$ for signal and background:

$$\tau_{\mathrm{s,b}} = \int y(\vec{x}) f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x} = \vec{w}^{\mathsf{T}} \vec{\mu}_{\mathrm{s,b}}$$

$$\Sigma_{\mathrm{s,b}}^2 = \int (y(\vec{x}) - \tau_{\mathrm{s,b}})^2 f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x} = \vec{w}^{\mathsf{T}} V_{\mathrm{s,b}} \vec{w}$$

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher linear discriminant: Determining the coefficients w_i

Numerator of $J(\vec{w})$:

$$(\tau_\mathrm{s} - \tau_\mathrm{b})^2 = \left(\sum_{i=1}^{n} w_i (\mu_i^\mathrm{s} - \mu_i^\mathrm{b}) \right)^2 = \sum_{i,j=1}^{n} w_i w_j (\mu_i^\mathrm{s} - \mu_i^\mathrm{b})(\mu_j^\mathrm{s} - \mu_j^\mathrm{b})$$

$$\equiv \sum_{i,j=1}^{n} w_i w_j B_{ij} = \vec{w}^\mathrm{T} B \vec{w}$$

Denominator of $J(\vec{w})$:

$$\Sigma_\mathrm{s}^2 + \Sigma_\mathrm{b}^2 = \sum_{i,j=1}^{n} w_i w_j \,(V^\mathrm{s} + V^\mathrm{b})_{ij} \equiv \vec{w}^\mathrm{T} W \vec{w}$$

Maximize:

$$J(\vec{w}) = \frac{\vec{w}^\mathrm{T} B \vec{w}}{\vec{w}^\mathrm{T} W \vec{w}} = \frac{\text{separation between classes}}{\text{separation within classes}}$$

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher linear discriminant: Determining the coefficients w_i

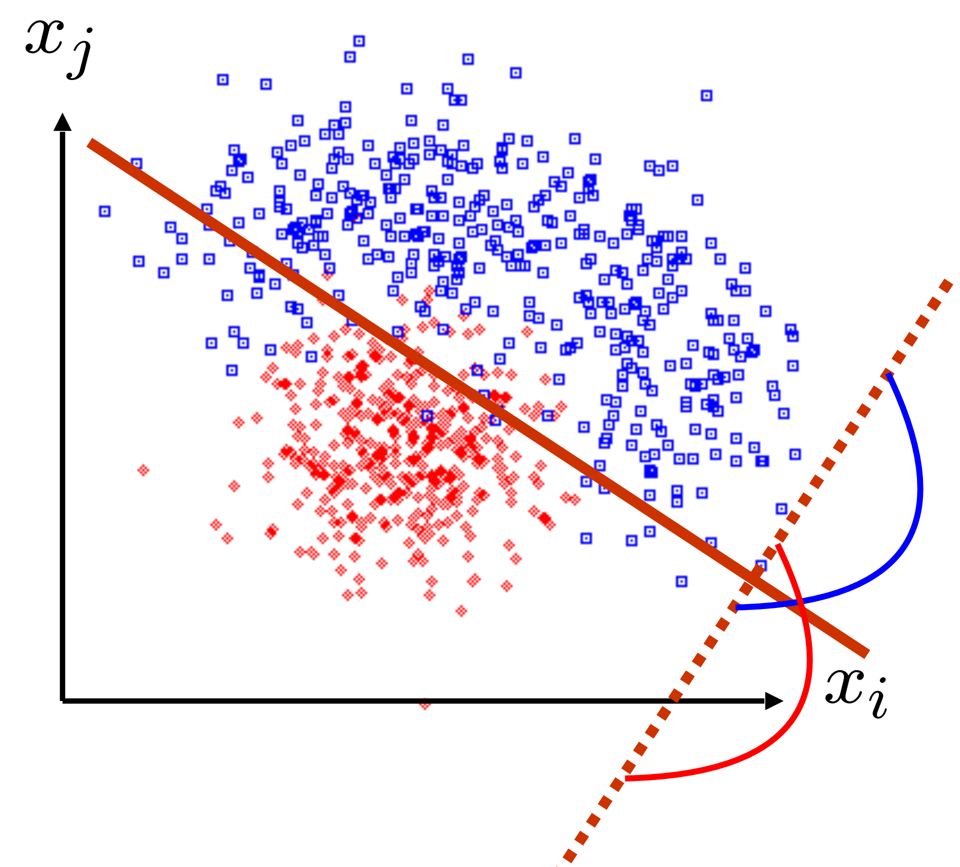
Setting $\dfrac{\partial J}{\partial w_i} = 0$ gives:

$$\boxed{y(\vec{x}) = \vec{w}^{\mathsf{T}} \vec{x} \qquad \text{with} \quad \vec{w} \propto W^{-1}(\vec{\mu}_{\mathrm{s}} - \vec{\mu}_{\mathrm{b}})}$$

We obtain linear decision boundaries.

Weight vector $\vec{w}$ can be interpreted as a direction in feature space on which the events are projected.

linear decision boundary



G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher linear discriminant: Remarks

In case the signal and background pdfs $f(\vec{x}|H_\mathrm{s})$ and $f(\vec{x}|H_\mathrm{b})$ are both multivariate Gaussian with the same covariance but different means, the Fisher discriminant is

$$y(\vec{x}) \propto \ln \frac{f(\vec{x}|H_\mathrm{s})}{f(\vec{x}|H_\mathrm{b})}$$

That is, in this case the Fisher discriminant is an optimal classifier according to the Neyman-Pearson lemma (as $y(\vec{x})$ is a monotonic function of the likelihood ratio)

Test statistic can be written as

$$y(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i$$

where events with $y > 0$ are classified as signal. Same functional form as for the **perceptron** (prototype of neural networks).

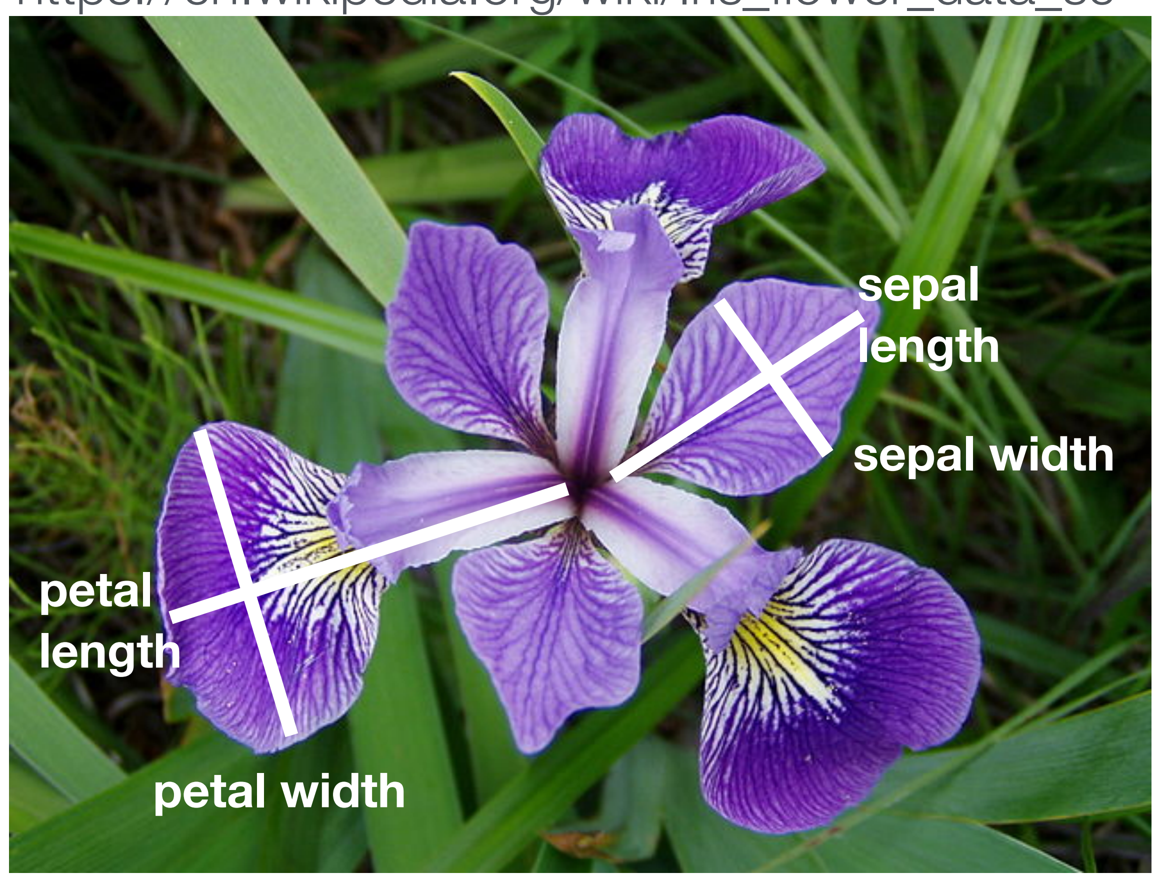
Example: Classification with scikit-learn (1)

Iris flower data set

https://archive.ics.uci.edu/ml/datasets/Iris

- Introduced 1936 in a paper by Ronald Fisher
- Task: classify flowers
- Three species: iris setosa, iris virginica and iris versicolor
- Four features: petal width and length, sepal width/length, in centimeters

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



Example: Classification with scikit-learn (2)

```
# import some data to play with
# columns: Sepal Length, Sepal Width, Petal Length and Petal Width
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
# just to create a nice table
df = pd.DataFrame({"Sepal Length (cm)": X[:,0], "Sepal Width (cm)": X[:,1],
                   'Petal Length (cm)': X[:,2], 'Petal Width (cm)': X[:,3],
                   'category': y})
df.head()
```

	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	category
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
list(iris.target_names)
```

```
['setosa', 'versicolor', 'virginica']
```

```
# split data into training and test data sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

Example: Classification with scikit-learn (3)

Softmax regression

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(multi_class='multinomial', penalty='none')
log_reg.fit(x_train, y_train);
```

k-nearest neighbor

```
from sklearn.neighbors import KNeighborsClassifier
kn_neigh = KNeighborsClassifier(n_neighbors=5)
kn_neigh.fit(x_train, y_train);
```

Fisher linear discriminant

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
fisher_ld = LinearDiscriminantAnalysis()
fisher_ld.fit(x_train, y_train);
```

Classification accuracy

```
for clf in [log_reg, kn_neigh, fisher_ld]:
    y_pred = clf.predict(x_test)
    acc = accuracy_score(y_test, y_pred)
    print(type(clf).__name__)
    print(f"accuracy: {acc:0.2f}")

    # confusion matrix: columns: true class, row: predicted class
    print(confusion_matrix(y_test, y_pred),"\n")
```

Output:

```
LogisticRegression
accuracy: 0.96
[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]

KNeighborsClassifier
accuracy: 0.95
[[29  0  0]
 [ 0 23  0]
 [ 0  4 19]]

LinearDiscriminantAnalysis
accuracy: 0.99
[[29  0  0]
 [ 0 23  0]
 [ 0  1 22]]
```

With scikit-learn it is extremely simple to test and apply different classification methods

Precision and recall

TP: true positives
FP: false positives
FN: false negatives

Precision:

Fraction of correctly classified instances among all instances that obtain a certain class label.

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

“purity”

Recall:

Fraction of positive instances that are correctly classified.

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

“efficiency”

Iris classification example:
precision and recall for softmax classification

see sklearn.metrics.classification_report

```
y_pred = log_reg.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        29
           1       0.88      1.00      0.94        23
           2       1.00      0.87      0.93        23

    accuracy                           0.96        75
   macro avg       0.96      0.96      0.96        75
weighted avg       0.96      0.96      0.96        75
```