Introduction to Data Analysis and Machine Learning in Physics: 3. Machine Learning Basics

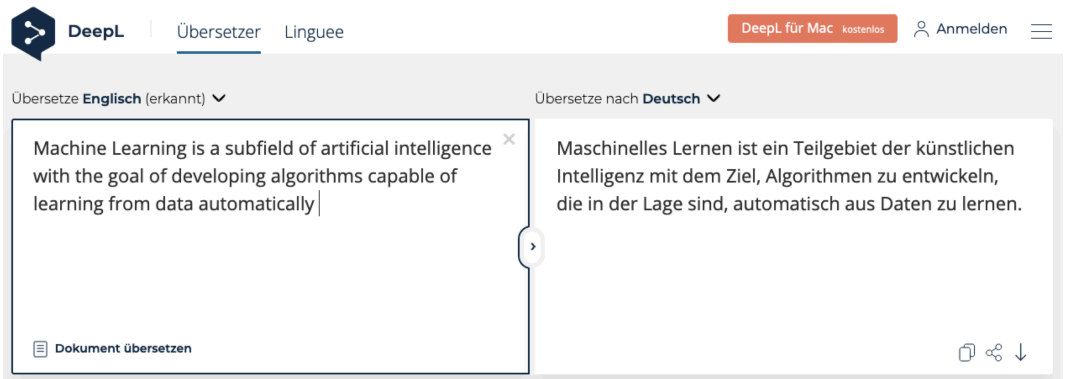
Martino Borsato, Jörg Marks, Klaus Reygers

Studierendentage, 11-14 April 2022

Exercises

- Exercise 1: Air shower classification (MAGIC telescope)
  - Logistic regression
  - `03_ml_basics_ex01_magic.ipynb`
- Exercise 2: Hand-written digit recognition with logistic regression
  - Logistic regression
  - `03_ml_basics_ex02_mnist_softmax_regression.ipynb`
- Exercise 3: Data preprocessing

What is machine learning? (1)

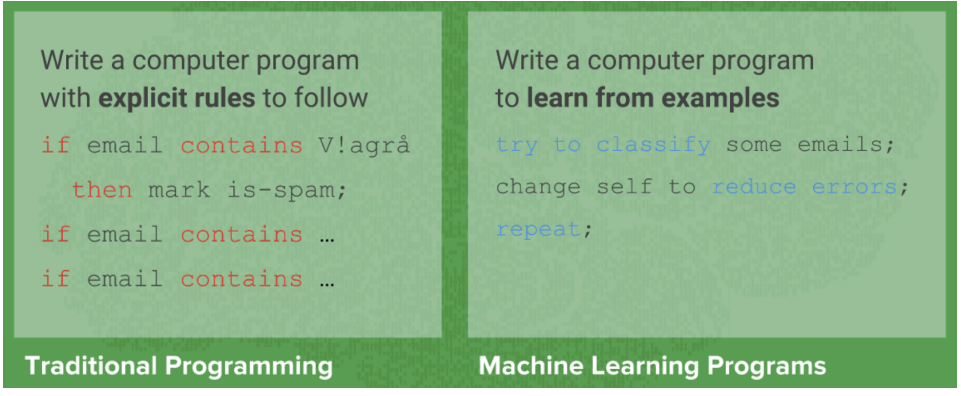


What is machine learning? (2)

"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

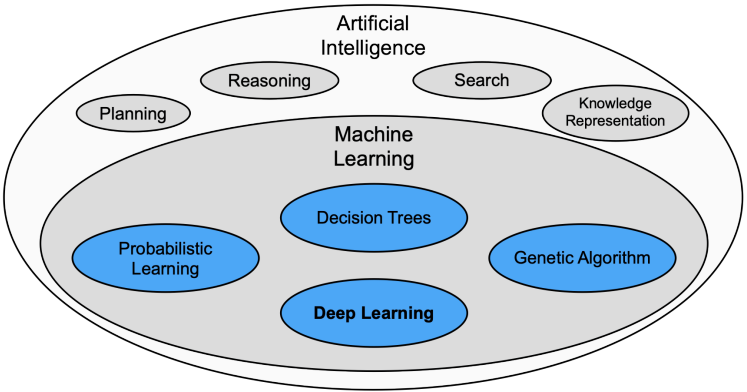


Manual feature engineering vs. automatic feature detection

AI, ML, and DL

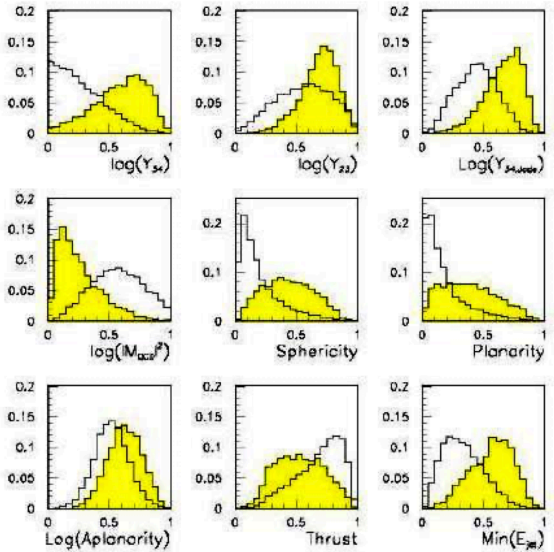
"AI is the study of how to make computers perform things that, at the moment, people do better." Elaine Rich, Artificial intelligence, McGraw-Hill 1983

G. Marcus, E. Davis, Rebooting AI

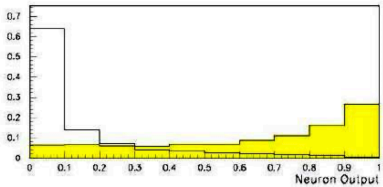


"deep" in deep learning: artificial neural nets with many neurons and multiple layers of nonlinear processing units for feature extraction

Multivariate analysis: An early example from particle physics



- Signal: $e^+e^- \rightarrow W^+W^-$
  - often 4 well separated hadron jets
- Background: $e^+e^- \rightarrow qqgg$
  - 4 less well separated hadron jets
- Input variables based on jet structure, event shape, … none by itself gives much separation.



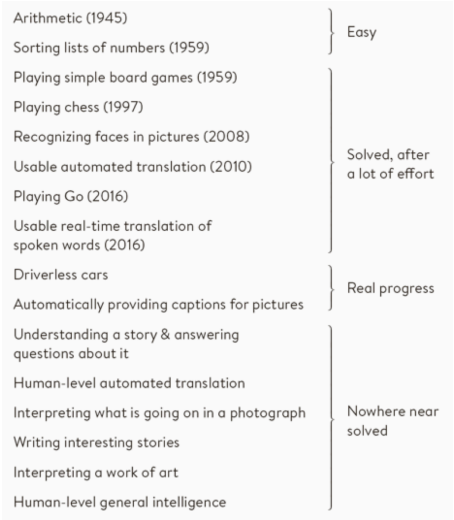
(Garrido, Juste and Martinez, ALEPH 96-144)

Applications of machine learning in physics

- Particle physics: Particle identification / classification
- Astronomy: Galaxy morphology classification
- Chemistry and material science: predict properties of new molecules / materials
- Many-body quantum matter: classification of quantum phases

Machine learning and the physical sciences, arXiv:1903.10563

Some successes and unsolved problems in AI



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?

The deep learning hype – why now?

Artificial neural networks are around for decades. Why did deep learning take off after 2012?

- Improved hardware – graphical processing units [GPUs]
- Large data sets (e.g. images) distributed via the Internet
- Algorithmic advances

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

Machine learning: The “hello world” problem

Recognition of handwritten digits

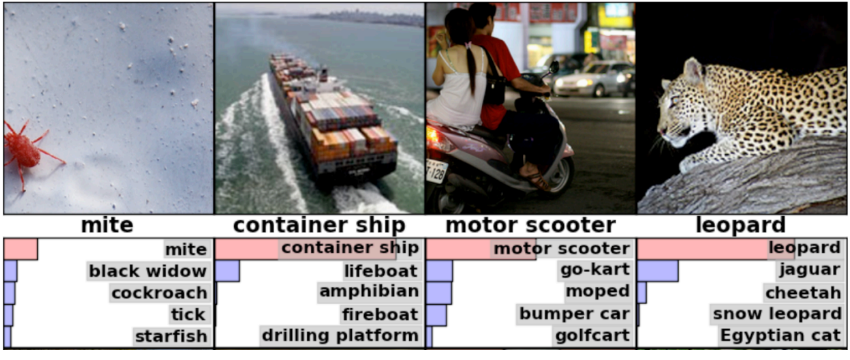
- MNIST database (Modified National Institute of Standards and Technology database)
- 60,000 training images and 10,000 testing images labeled with correct answer
- 28 pixel x 28 pixel
- Algorithms have reached “near-human performance”
- Smallest error rate (2018): 0.18%

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

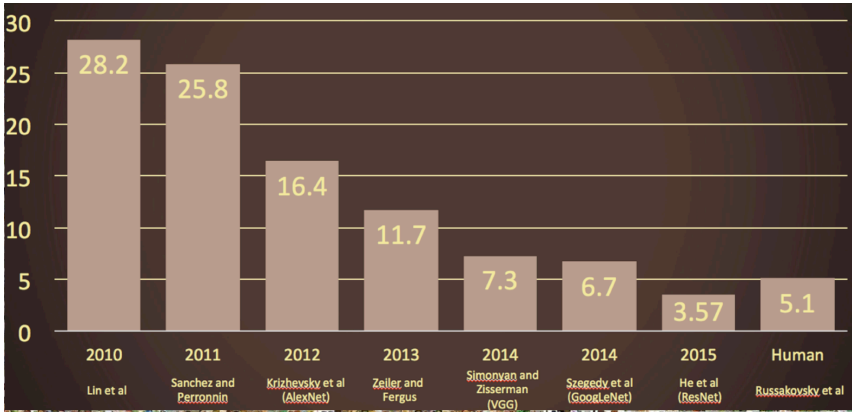
Machine learning: Image recognition

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

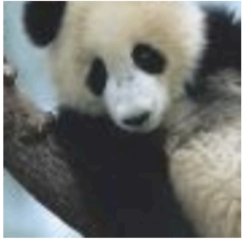


ImageNet: Large Scale Visual Recognition Challenge



O. Russakovsky et al, arXiv:1409.0575

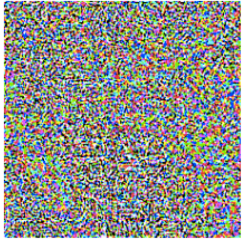
Adversarial attack



$\boldsymbol{x}$

"panda"
57.7% confidence

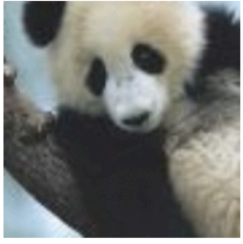
$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$



$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

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

Types of machine learning

Reinforcement learning

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

Supervised learning

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

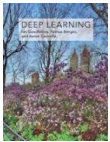
Unsupervised learning

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

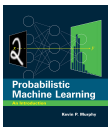
LeCun 2018, Power And Limits of Deep Learning

Books on machine learning (1)

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



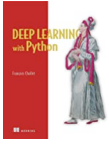
Kevin Murphy, *Probabilistic Machine Learning: An Introduction*, draft pdf version



Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*

Books on machine learning (2)

Francois Chollet, *Deep Learning with Python*



Martin Erdmann, Jonas Glombitza, Gregor Kasieczka, Uwe Klemradt,
Deep Learning for Physics Research

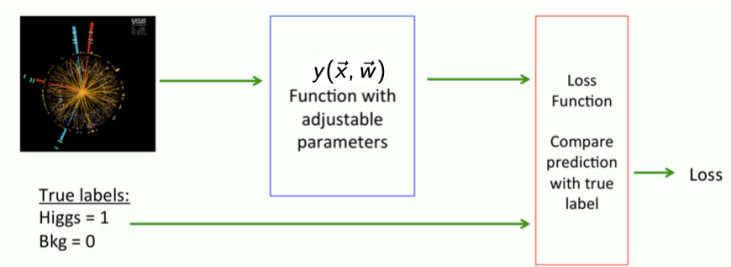
Papers

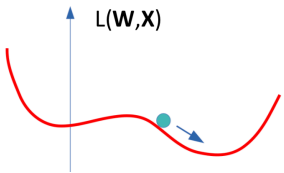
A high-bias, low-variance introduction to Machine Learning for physicists
https://arxiv.org/abs/1803.08823

Machine learning and the physical sciences
https://arxiv.org/abs/1903.10563

Supervised learning in a nutshell

- 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.
- Each event is characterized by n observables: $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ "feature vector"





- Design function $y(\boldsymbol{x}, \boldsymbol{w})$ with adjustable parameters $\boldsymbol{w}$
- Design a loss function
- Find best parameters which minimize loss

Supervised learning: classification and regression

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

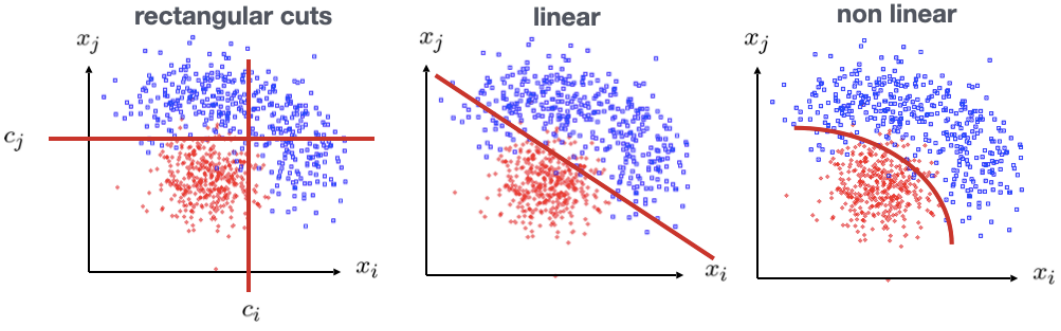
- Y = finite set of labels $\rightarrow$ classification
  - binary classification: $Y = \{0, 1\}$
  - multi-class classification: $Y = \{c_1, c_2, ..., c_n\}$
- Y = real numbers $\rightarrow$ regression

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

J. Pearl, Turing Award Winner 2011

To Build Truly Intelligent Machines, Teach Them Cause and Effect, Quantamagazine

Classification: Learning decision boundaries



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 $\rightarrow$ check for overtraining
- Sign of overtraining: performance on validation sample becomes worse $\rightarrow$ 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 (bias is rather small)

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

Square error loss:

- often used in regression

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

Cross entropy:

- $t \in \{0, 1\}$
- $y(\boldsymbol{x}, \boldsymbol{w})$: predicted probability for outcome $t = 1$
- often used in classification

$$E(y(\boldsymbol{x}, \boldsymbol{w}), t) = -\, t \log y(\boldsymbol{x}, \boldsymbol{w}) \\ -\, (1 - t) \log(1 - y(\boldsymbol{x}, \boldsymbol{w}))$$

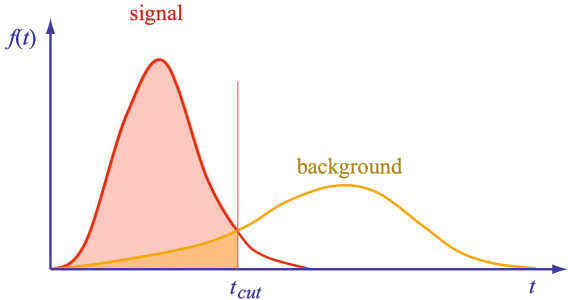
More on entropy

- Self-information of an event x: $I(x) = -\log p(x)$
  - in units of **nats** (1 nat = information gained by observing an event of probability $1/e$)

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

Hypothesis testing



test statistic

- a (usually scalar) variable which is a function of the data alone that can be used to test hypotheses
- example: χ^2 w.r.t. a theory curve

$\epsilon_B \equiv \alpha$: "background efficiency", i.e., prob. to misclassify bckg. as signal

$\epsilon_S \equiv 1 - \beta$: "signal efficiency"

	H_0 is true	H_0 is false (i.e., H_1 is true)
H_0 is rejected	Type I error (α)	Correct decision ($1 - \beta$)
H_0 is not rejected	Correct decision ($1 - \alpha$)	Type II error (β)

Neyman-Pearson Lemma

The likelihood ratio

$$t(\boldsymbol{x}) = \frac{f(\boldsymbol{x}|H_1)}{f(\boldsymbol{x}|H_0)}$$

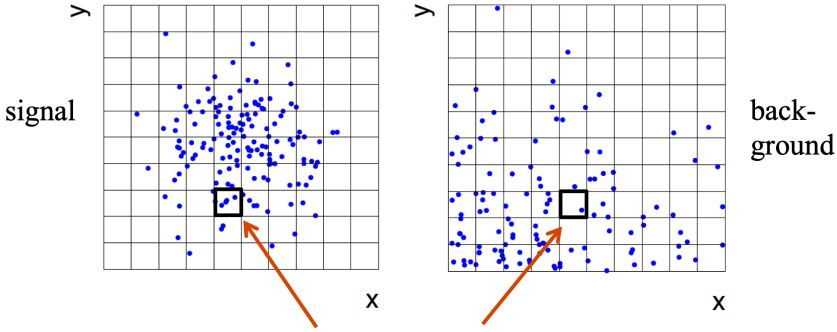
is an optimal test statistic, i.e., it provides highest "signal efficiency" $1 - \beta$ for a given "background efficiency" α. Accept hypothesis if $t(\boldsymbol{x}) > c$.

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?



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

M bins per variable in d dimensions: M^d cells$\rightarrow$ 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"

Naïve Bayesian Classifier (also called “Projected Likelihood Classification”)

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

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

$$\begin{aligned} \text{where} \quad f_1(x_1) &= \int \mathrm{d}x_2 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n) \\ f_2(x_2) &= \int \mathrm{d}x_1 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n) \\ &\vdots \end{aligned}$$

Classification of feature vector x:

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

Performance not optimal if true PDF does not factorize

k-Nearest Neighbor Method (1)

k-NN classifier:

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

Algorithms finds k nearest neighbors:

$$k = k_s + k_b$$

Probability for the event to be of signal type:

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

k-Nearest Neighbor Method (2)

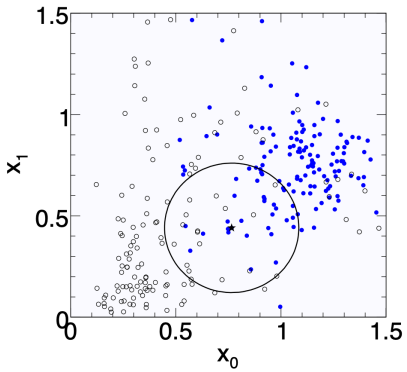
Simplest choice for distance measure in feature space is the Euclidean distance:

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

Better: take correlations between variables into account:

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

$\boldsymbol{V}$ = covariance matrix, R = "Mahalanobis distance"



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

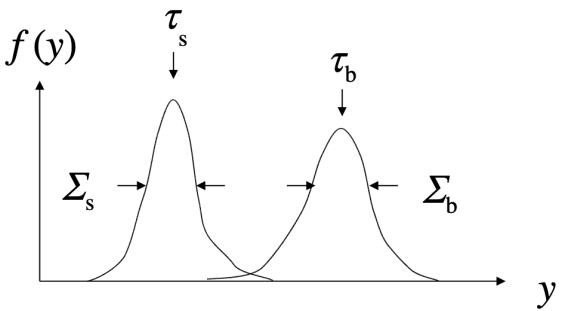
Linear discriminant is simple. Can still be optimal if amount of training data is limited.
Ansatz for test statistic:

$$y(\boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}$$

Choose parameters w_i so that separation between signal and background distribution is maximum.
Need to define “separation”.

Fisher: maximize

$$J(\boldsymbol{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



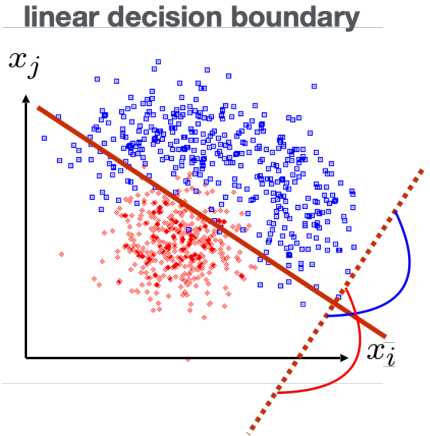
Fisher Linear Discriminant: Determining the Coefficients w_i

Coefficients are obtained from:

$$\frac{\partial J}{\partial w_i} = 0$$

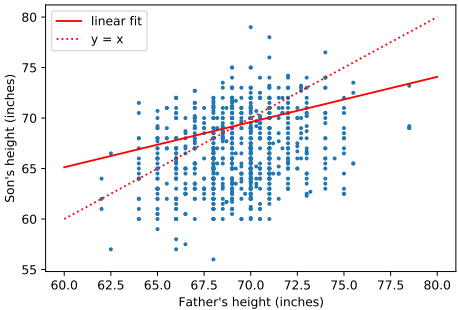
Linear decision boundaries

Weight vector ***w*** can be interpreted as a direction in feature space onto which the events are projected.



Linear regression revisited

”Galton family heights data”:
origin of the term ”regression”



- data: $\{x_i, y_i\}$
- objective: predict $y = f(x)$
- model:
  $f(x; \boldsymbol{\theta}) = mx + b, \quad \boldsymbol{\theta} = (m, b)$
- loss function:
  $J(\theta|x, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2$
- model training: optimal parameters
  $\hat{\boldsymbol{\theta}} = \arg\min J(\boldsymbol{\theta})$

Linear regression

- Data: vectors with p components ("features"): $\boldsymbol{x} = (x_1, ..., x_p)$
- n observations: $\{\boldsymbol{x}_i, y_i\}, \quad i = 1, ..., n$
- Prediction for given vector x:

$$y = w_0 + w_1 x_1 + w_2 x_2 + ... + w_p x_p \equiv \boldsymbol{w}^\mathsf{T} \boldsymbol{x} \quad \text{where } x_0 := 1$$

- Find weights that minimze loss function:

$$\hat{\boldsymbol{w}} = \min_{\boldsymbol{w}} \sum_{i=1}^{n} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i - y_i)^2$$

- In case of linear regression closed-form solution exists:

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^\mathsf{T} \boldsymbol{X})^{-1} \boldsymbol{X}^\mathsf{T} \boldsymbol{y} \quad \text{where } X \in \mathbb{R}^{n \times p}$$

- X is called the design matrix, row i of X is $\boldsymbol{x}_i$

Linear regression with regularization

- Standard loss function

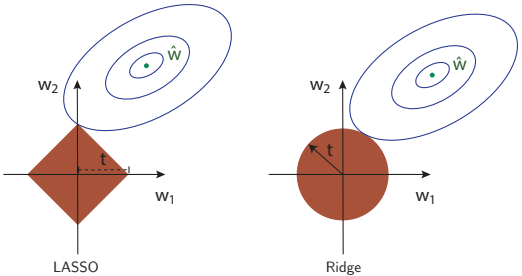
$$C(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2$$

- Ridge regression

$$C(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 + \lambda |\boldsymbol{w}|^2$$

- LASSO regression

$$C(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 + \lambda |\boldsymbol{w}|$$



LASSO regression tends to give sparse solutions (many components $w_j = 0$). This is why LASSO regression is also called sparse regression.

Logistic regression (1)

- Consider binary classification task, e.g., $y_i \in \{0, 1\}$
- Objective: Predict probability for outcome $y = 1$ given an observation $\boldsymbol{x}$
- Starting with linear "score"

$$s = w_0 + w_1 x_1 + w_2 x_2 + ... + w_p x_p \equiv \boldsymbol{w}^\mathsf{T} \boldsymbol{x}$$

- Define function that translates s into a quantity that has the properties of a probability

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

- We would like to determine the optimal weights for a given training data set. They result from the maximum-likelihood principle.

Logistic regression (2)

- Consider feature vector $\boldsymbol{x}$. For a given set of weights $\boldsymbol{w}$ the model predicts
  - a probability $p(1|\boldsymbol{w}) = \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x})$ for outcome $y = 1$
  - a probabiltiy $p(0|\boldsymbol{w}) = 1 - \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x})$ for outcome $y = 0$
- The probability $p(y_i|\boldsymbol{w})$ defines the likelihood $L_i(\boldsymbol{w}) = p(y_i|\boldsymbol{w})$ (the likelihood is a function of the parameters $\boldsymbol{w}$ and the observations y_i are fixed).
- Likelihood for the full data sample (n observations)

$$L(\boldsymbol{w}) = \prod_{i=1}^{n} L_i(\boldsymbol{w}) = \prod_{i=1}^{n} \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x})^{y_i} \left(1 - \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x})\right)^{1-y_i}$$

- Maximizing the log-likelihood $\ln L(\boldsymbol{w})$ corresponds to minimizing the loss function

$$C(\boldsymbol{w}) = -\ln L(\boldsymbol{w}) = \sum_{i=1}^{n} -y_i \ln \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}) - (1 - y_i) \ln(1 - \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}))$$

- This is nothing else but the cross-entropy loss function

scikit-learn

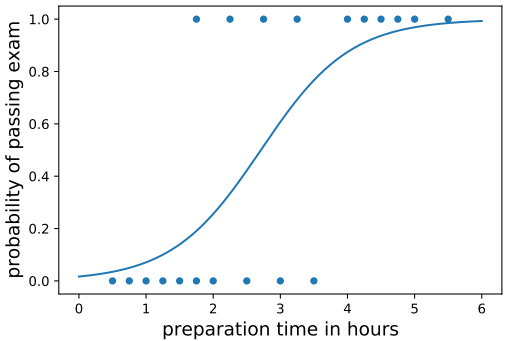
- Free software machine learning library for Python
- Initial release: 2007
- features various classification, regression and clustering algorithms including k-nearest neighbors, multi-layer perceptrons, support vector machines, random forests, gradient boosting, k-means
- Scikit-learn is one of the most popular machine learning libraries on GitHub
- `https://scikit-learn.org/`

Example 1 - Probability of passing an exam (logistic regression) (1)

Objective: predict the probability that someone passes an exam based on the number of hours studying

$$p_{\text{pass}} = \sigma(s) = \frac{1}{1+e^{-s}}, \quad s = w_1 t + w_0, \quad t = \#\text{ hours}$$

- Data set:
  - preparation t time in hours
  - passed / not passes (0/1)
- Parameters need to be determined through numerical minimization
  - $w_0 = -4.0777$
  - $w_1 = 1.5046$



03_ml_basics_logistic_regression.ipynb

Example 1 - Probability of passing an exam (logistic regression) (2)

Read data from file:

```
# data: 1. hours studies, 2. passed (0/1)
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 data:

```
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)
```

Precision and recall

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”

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

Example 2: Heart disease data set (logistic regression) (1)

Read data:

```
filename = "https://www.physi.uni-heidelberg.de/~reygers/lectures/2022/ml/data/heart.csv"
df = pd.read_csv(filename)
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

03_ml_basics_log_regr_heart_disease.ipynb

Example 2: Heart disease data set (logistic regression) (2)

Define array of labels and feature vectors

```
y = df['target'].values
X = df[[col for col in df.columns if col!="target"]]
```

Generate training and test data sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
        = train_test_split(X, y, test_size=0.5, shuffle=True)
```

Fit the model

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty='none',
        fit_intercept=True, max_iter=1000, tol=1E-5)
lr.fit(X_train, y_train)
```

Example 2: Heart disease data set (logistic regression) (3)

Test predictions on test data set:

```
from sklearn.metrics import classification_report
y_pred_lr = lr.predict(X_test)
print(classification_report(y_test, y_pred_lr))
```

Output:

```
              precision    recall  f1-score   support

           0       0.75      0.86      0.80        63
           1       0.89      0.80      0.84        89

    accuracy                           0.82       152
   macro avg       0.82      0.83      0.82       152
weighted avg       0.83      0.82      0.82       152
```

Example 2: Heart disease data set (logistic regression) (4)

Compare to another classifier using the *receiver operating characteristic* (ROC) curve

Let's take the random forest classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=3)
rf.fit(X_train, y_train)
```

Use `roc_curve` from scikit-learn

```
from sklearn.metrics import roc_curve

y_pred_prob_lr = lr.predict_proba(X_test) # predicted probabilities
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_prob_lr[:,1])

y_pred_prob_rf = rf.predict_proba(X_test) # predicted probabilities
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_prob_rf[:,1])
```

Example 2: Heart disease data set (logistic regression) (5)

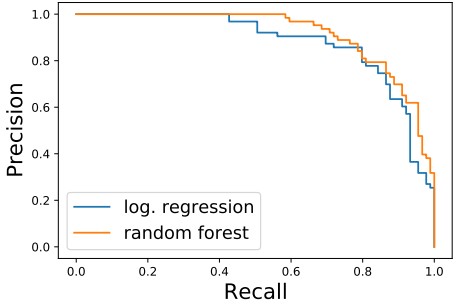
```
plt.plot(tpr_lr, 1-fpr_lr, label="log. regression")
plt.plot(tpr_rf, 1-fpr_rf, label="random forest")
```

Classifiers can be compared with the *area under curve* (AUC) score.

```
from sklearn.metrics import roc_auc_score
auc_lr = roc_auc_score(y_test,y_pred_lr)
auc_rf = roc_auc_score(y_test,y_pred_rf)
print(f"AUC scores: {auc_lr:.2f}, {auc_knn:.2f}")
```

This gives

```
AUC scores: 0.82, 0.83
```



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.

Now consider k classes and let s_i be the score for class i: $\boldsymbol{s} = (s_1, ..., s_k)$

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

$$\sigma(\boldsymbol{s})_i = \frac{e^{s_i}}{\sum_{j=1}^{k} e^{s_j}} \quad \text{for} \quad i = 1, ..., k$$

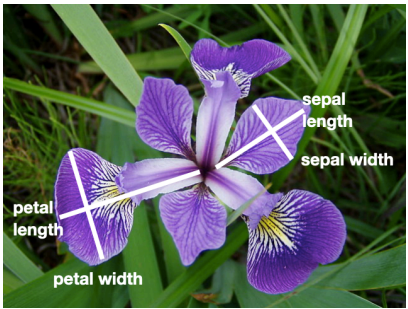
The softmax functions is often used as the last activation function of a neural network in order to predict probabilities in a classification task.

Multinomial logistic regression is also known as softmax regression.

Example 3: Iris data set (softmax regression) (1)

Iris flower data set

- 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



03_ml_basics_iris_softmax_regression.ipynb

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

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

Example 3: Iris data set (softmax regression) (2)

Get data set

```
# 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

# 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)
```

Softmax regression

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

Example 3 : Iris data set (softmax regression) (3)

Accuracy and confusion matrix for different classifiers

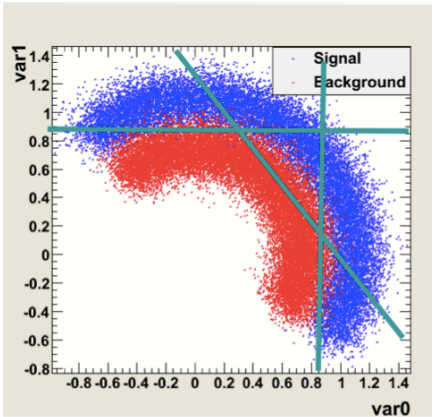
```
for clf in [log_reg, kn_neigh, fisher_ld]:
    y_pred = clf.predict(x_test)
    acc = accuracy_score(y_test, y_pred)
    print(type(clf).__name__)
    print(f"accuracy: {acc:0.2f}")

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

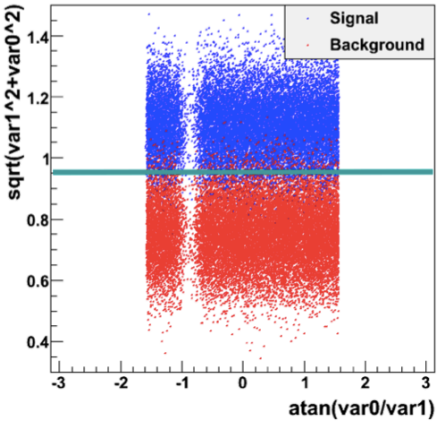
```
LogisticRegression
accuracy: 0.96
[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]

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]]
```

General remarks on multi-variate analyses (MVAs)

- MVA Methods
  - More effective than classic cut-based analyses
  - Take correlations of input variables into account
- Important: find good input variables for MVA methods
  - Good separation power between S and B
  - No strong correlation among variables
  - No correlation with the parameters you try to measure in your signal sample!
- Pre-processing
  - Apply obvious variable transformations and let MVA method do the rest
  - Make use of obvious symmetries: if e.g. a particle production process is symmetric in polar angle θ use $|\cos\theta|$ and not $\cos\theta$ as input variable
  - It is generally useful to bring all input variables to a similar numerical range

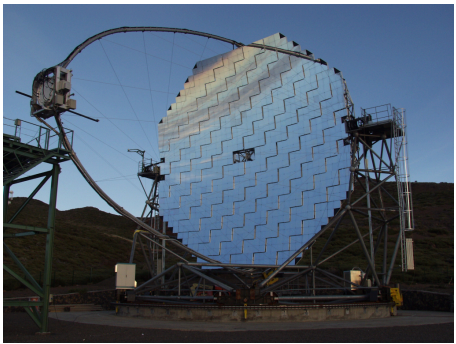
Example of feature transformation

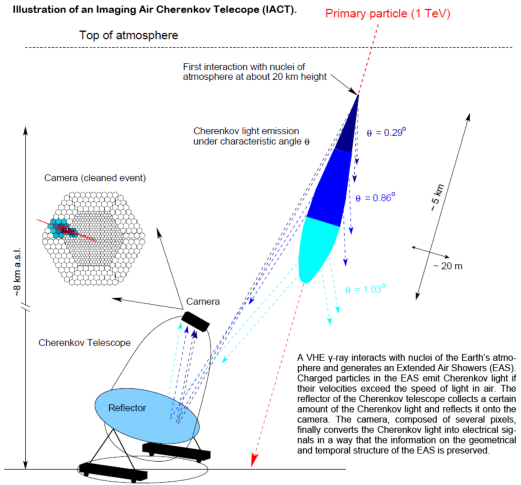




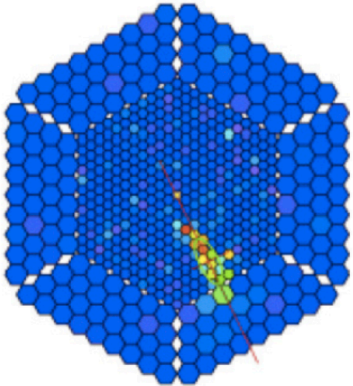
Exercise 1: Classification of air showers measured with the MAGIC telescope

- Cosmic gamma rays (30 GeV - 30 TeV).
- Cherenkov light from air showers
- Background: air showers caused by hadrons.

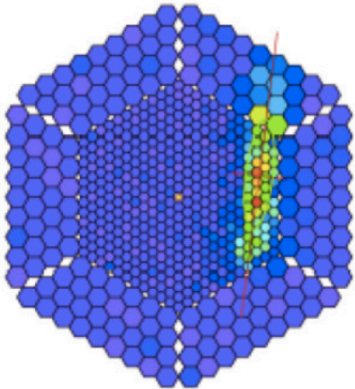




Exercise 1: Classification of air showers measured with the MAGIC telescope

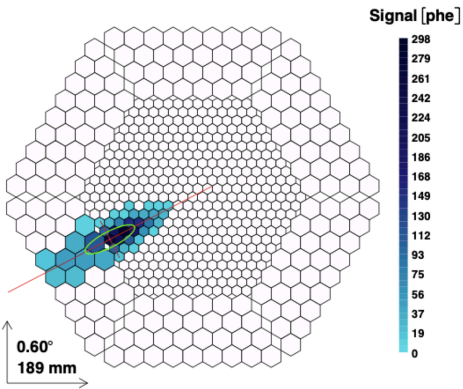


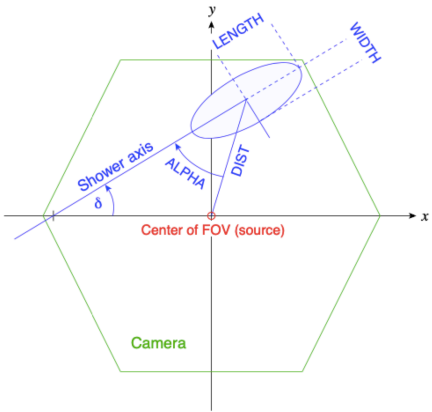
Gamma shower



Hadronic shower

Exercise 1: Classification of air showers measured with the MAGIC telescope





Exercise 1: Classification of air showers measured with the MAGIC telescope

MAGIC data set

https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope

```
1.  fLength:  continuous  # major axis of ellipse [mm]
2.  fWidth:   continuous  # minor axis of ellipse [mm]
3.  fSize:    continuous  # 10-log of sum of content of all pixels [in #phot]
4.  fConc:    continuous  # ratio of sum of two highest pixels over fSize  [ratio]
5.  fConc1:   continuous  # ratio of highest pixel over fSize  [ratio]
6.  fAsym:    continuous  # dist. from highest pixel to center, proj. onto major axis [mm]
7.  fM3Long:  continuous  # 3rd root of third moment along major axis  [mm]
8.  fM3Trans: continuous  # 3rd root of third moment along minor axis  [mm]
9.  fAlpha:   continuous  # angle of major axis with vector to origin [deg]
10. fDist:    continuous  # distance from origin to center of ellipse [mm]
11. class:    g,h         # gamma (signal), hadron (background)

g = gamma (signal):     12332
h = hadron (background): 6688

For technical reasons, the number of h events is underestimated.
In the real data, the h class represents the majority of the events.
```

Exercise 1: Classification of air showers measured with the MAGIC telescope

03_ml_basics_ex_1_magic.ipynb

a) Create for each variable a figure with a plot for gammas and hadrons overlayed.

b) Create training and test data set. The test data should amount to 50% of the total data set.

c) Define the logistic regressor and fit the training data

d) Determine the model accuracy and the AUC score

e) Plot the ROC curve (background rejection vs signal efficiency)

Exercise 2: Hand-written digit recognition with logistic regression

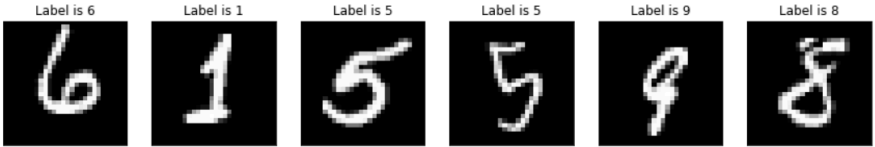
03_ml_basics_ex_2_mnist_softmax_regression.ipynb

a) Define logistic regressor from scikit-learn and fit data

b) Use `classification_report` from scikit-learn to determine precision and recall

c) Read in a hand-written digit and classify it. Print the probabilities for each digit. Determine the digit with the highest probability.

d) (Optional) Create you own hand-written digit with a program like gimp and check what the classifier does



Hint: You can install required packages on the jupyter hub server like so:

```
!pip3 install --user pypng
```

Exercise 3: Data preprocessing

a) Read the description of the `sklearn.preprocessing` package.

b) Start from the example notebook on the logistic regression for the heart disease data set (03_ml_basics_log_regr_heart_disease.ipynb). Pre-process the heart disease data set according to the given example. Does preprocessing make a difference in this case?