Introduction to Machine Learning: Part II

Prof. Sean Dobbs[1] & Daniel Lersch[2]

April 16, 2020

[1] (sdobbs@fsu.edu)
[2] (dlersch@jlab.org)

About this Lecture

- **Part I:**
  - Introduction to DataFrames
  - Basic concepts of machine learning
    (with focus on feedforward neural networks)
- **Part II:**
  - Machine learning in (physics) data analysis
  - Performance evaluation
- **Part III:**
  - Algorithm tuning
  - Hyper parameter optimization
- **Part IV:**
  - Custom neural networks with Tensorflow
  - Transition to Deep Learning

The individual contents might be subject to change

This Lecture will...

... NOT turn you into a machine learning specialist

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

... utilize the **scikit-learn** library

This Lecture will...

... NOT turn you into a machine learning specialist

... NOT cover all aspects of machine learning

... give a (very) brief overview only (i.e. further reading is definitely required)

... introduce a few machine learning algorithms

... utilize the **scikit-learn** library

... most likely contain several errors ($\rightarrow$ Please send a mail to `dlersch@jlab.org`)

Homework and Literature

- Machine learning can be learned best by simply doing it!

Homework and Literature

- Machine learning can be learned best by simply doing it!
- Homework (most likely posted on Thursday) aims to perform a simple analysis and getting familiar with machine learning

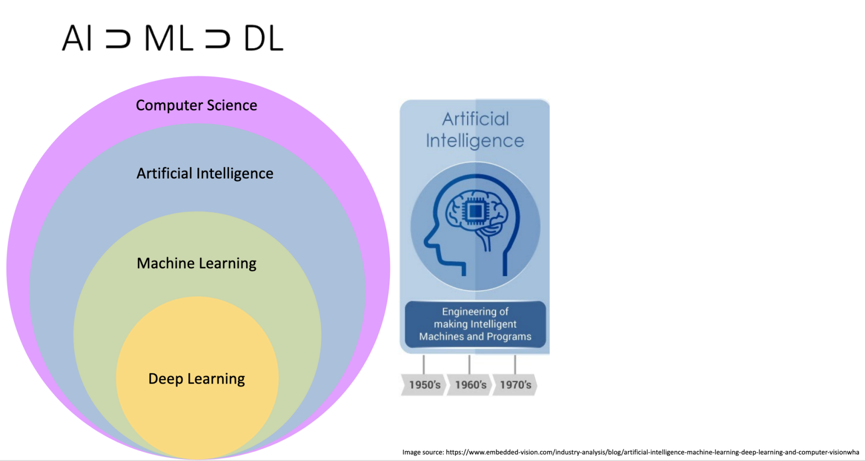
Homework and Literature

- Machine learning can be learned best by simply doing it!
- Homework (most likely posted on Thursday) aims to perform a simple analysis and getting familiar with machine learning
- Helpful literature:
  - The **scikit-learn** documentation
  - Talks from the **deep learning for science school 2019**[3]
  - "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow", by Aurélien Géron
  - The internet is full of good (but also very bad!) literature[4] $\rightarrow$ browse with caution
  - The slides of the lecture are available at: `http://hadron.physics.fsu.edu/~dlersch/ml_slides/`

[3] Very good and detailed explanation of (deep) neural networks

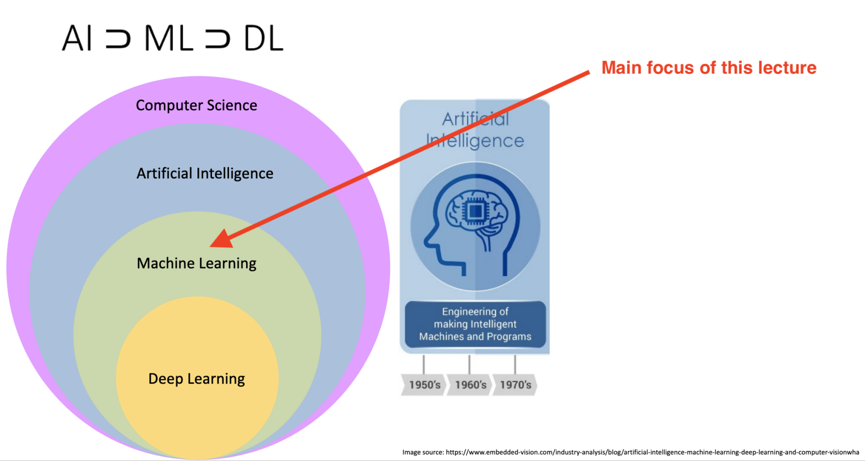
[4] Any document claiming that there is a quick way to understand machine learning without any theory / math is considered as bad

AI, ML and DL



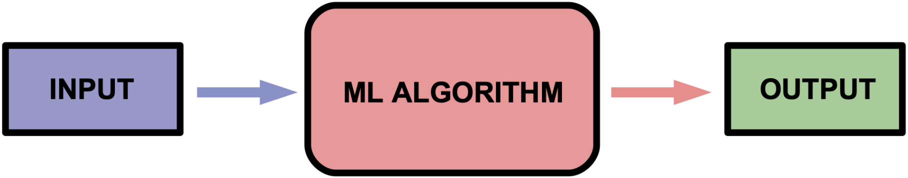
Slide taken from Brenda Ngs introductory talk at the: **deep learning for science school 2019**

AI, ML and DL

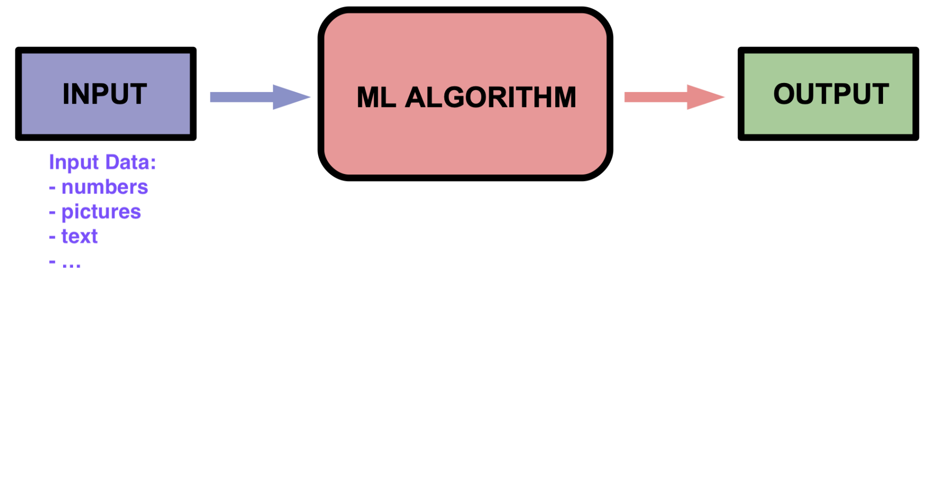


Slide taken from Brenda Ngs introductory talk at the: **deep learning for science school 2019**

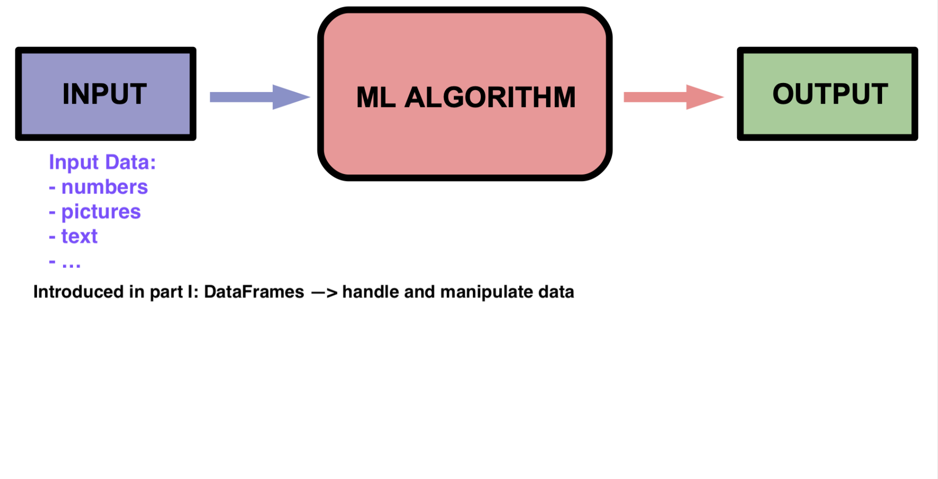
Basic Components of Machine Learning



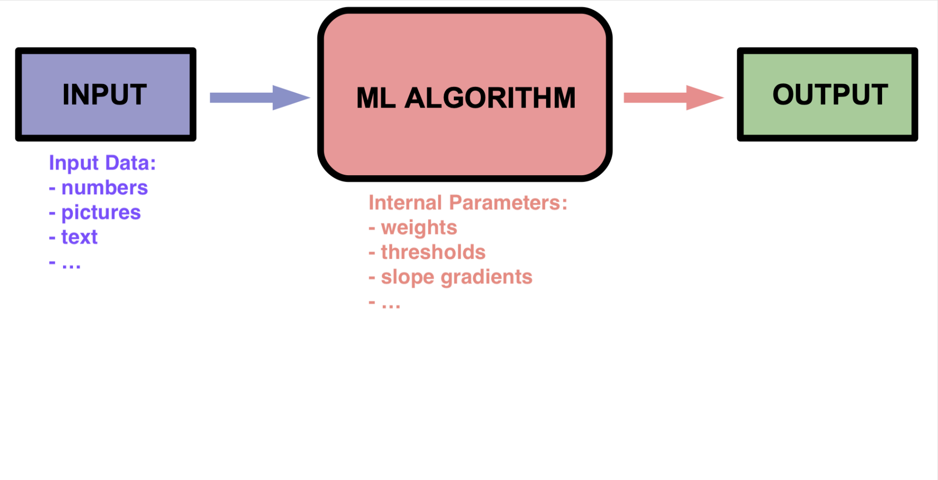
Basic Components of Machine Learning



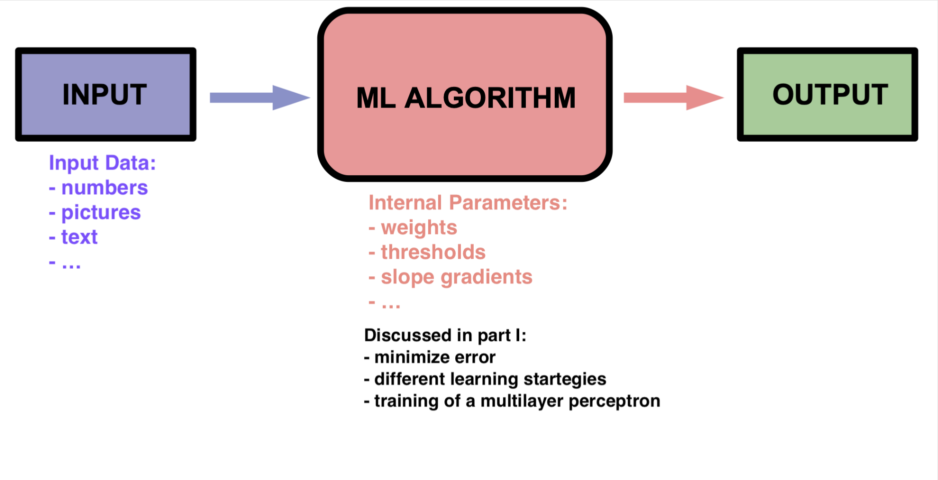
Basic Components of Machine Learning



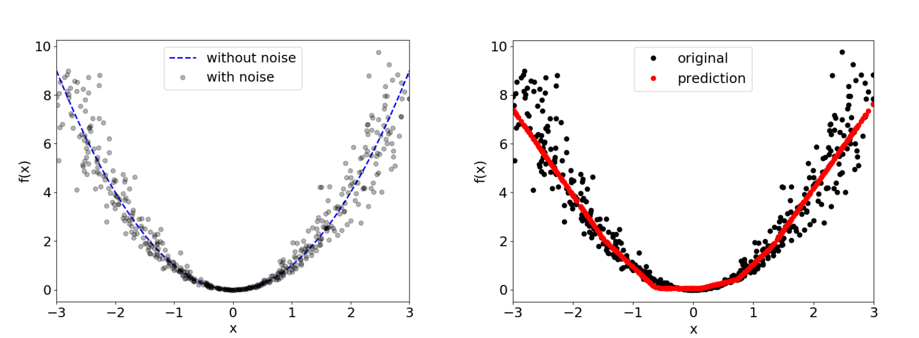
Basic Components of Machine Learning



Basic Components of Machine Learning

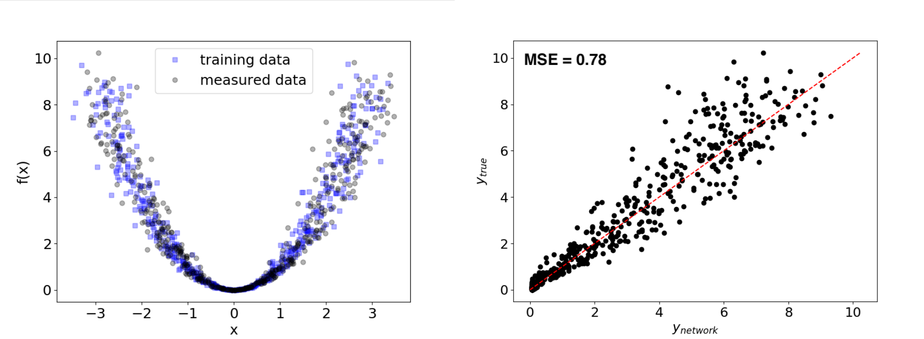


Learning a quadratic Function with Noise



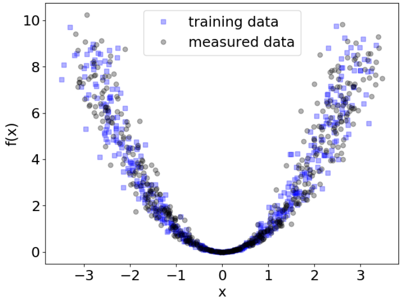
- Like in part I: Try to learn a quadratic function, but you know your measured data will be noisy $\rightarrow$ implement noise in your training data
- Train again mlp, similar to the one used in part I

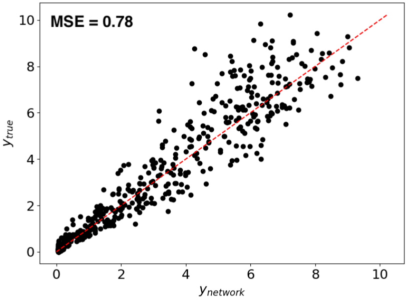
Learning a quadratic Function with Noise



- Like in part I: Try to learn a quadratic function, but you know your measured data will be noisy → implement noise in your training data
- Train again mlp, similar to the one used in part I
- Feed in measured data which is slightly different to the training data
- MSE $= \frac{1}{N}\sum_i (y_{true,i} - y_{network,i})^2$

Learning a quadratic Function with Noise





- Like in part I: Try to learn a quadratic function, but you know your measured data will be noisy → implement noise in your training data
- Train again mlp, similar to the one used in part I
- Feed in measured data which is slightly different to the training data
- $\mathrm{MSE} = \frac{1}{N}\sum_i (y_{true,i} - y_{network,i})^2$
- Not surprising, because network only reflects what it has been trained on

Validation Data

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)

Picture taken from Brenda Ngs introductory talk at the: deep learning for science school 2019

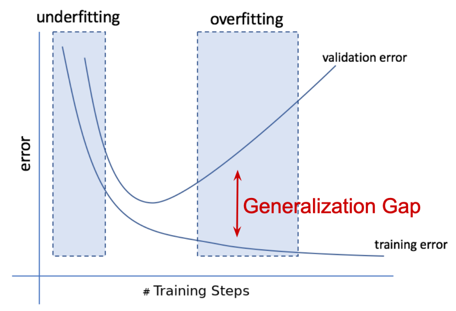
Validation Data

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[5], but used to determine when training is complete

[5] This data is "unseen" by the algorithm during the training stage

Validation Data

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[5], but used to determine when training is complete

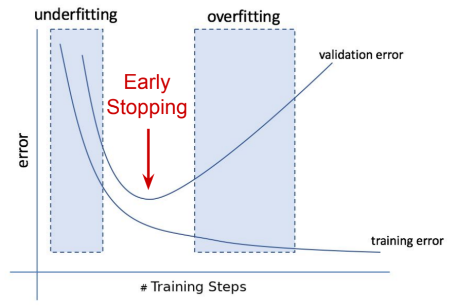


Picture taken from Mustafa Mustafas talk at the: deep learning for science school 2019

[5] This data is "unseen" by the algorithm during the training stage

Validation Data

- Want to enable network to abstract / generalize on unknown data AND avoid overfitting (i.e. avoid that network reproduces features from training data only)
- **Validation Data:** Part of training data that is NOT used to update internal parameters[5], but used to determine when training is complete



Picture taken from Mustafa Mustafas talk at the: deep learning for science school 2019

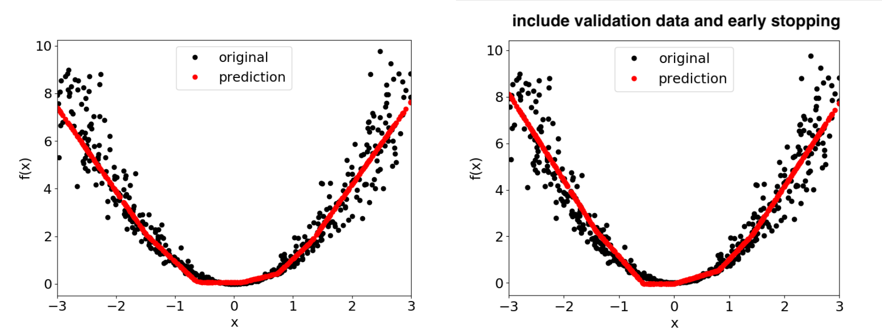
[5] This data is "unseen" by the algorithm during the training stage

Implementing Early Stopping and Validation Data in the scikit MLPRegressor

```
my_mlp =   MLPRegressor(
             hidden_layer_sizes=(10),
             activation='relu',
             solver='sgd',
             warm_start=True,
             max_iter = 1000,
             shuffle=True,
             tol=1e-6,
             validation_fraction=0.5, #---> Define the percentage of
             #training data that shall be kept aside
             early_stopping=True, #---> Enable early stopping
             random_state=0,
             learning_rate_init = 0.05
      )
```

Learning a quadratic Function with Noise

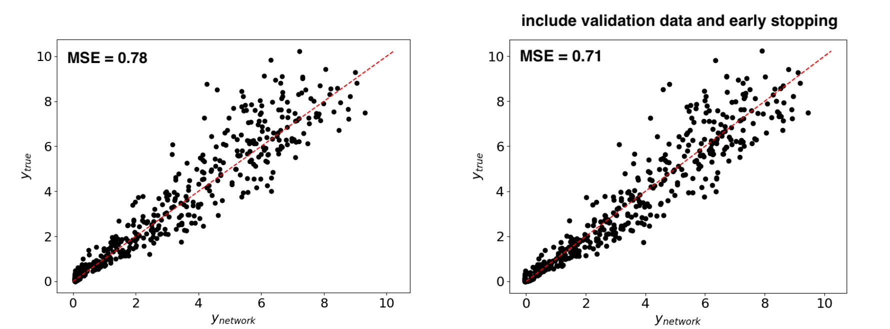
Include Validation Data



- **Left:** No validation data used
- **Right:** Validation data + early stopping

Learning a quadratic Function with Noise

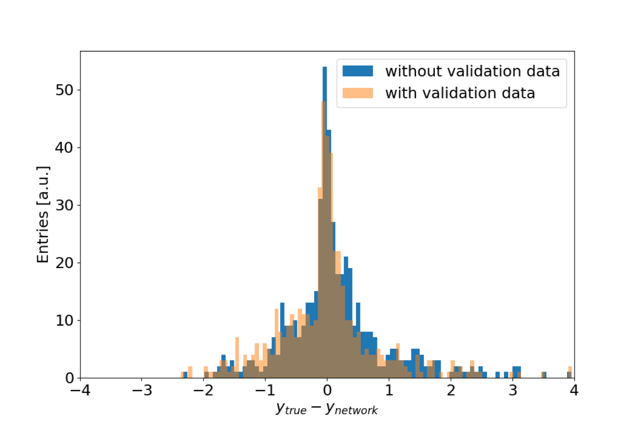
Include Validation Data



- **Left:** No validation data used
- **Right:** Validation data + early stopping
- $\sim$ 9% difference in performance

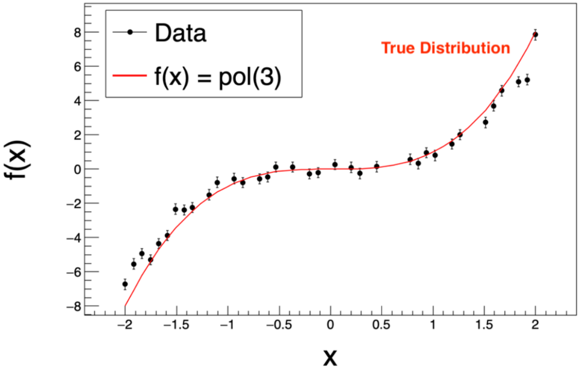
Learning a quadratic Function with Noise

Include Validation Data



- **Left:** No validation data used
- **Right:** Validation data + early stopping
- $\sim$ 9% difference in performance
- **Note:** Probably not the best example to advertise validation data

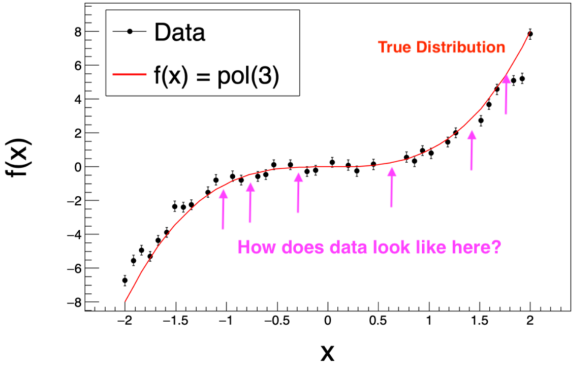
Gaussian Processors: Introduction



- **Goal(s):**
  - i) Describe data shown above, <u>without</u> knowledge of the true distribution
  - ii) Identify missing points
  - iii) Make a prediction/extrapolation?
- Put at least assumptions/information into fit as possible

$\Rightarrow$ **Use Gaussian processors**

Gaussian Processors: Introduction

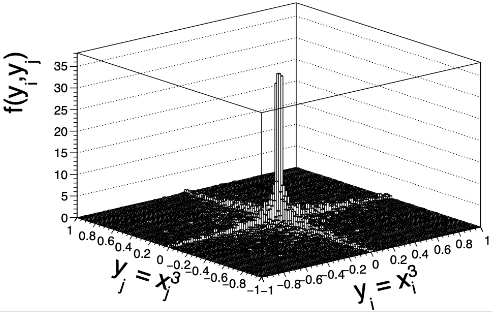


- **Goal(s):**
  i) Describe data shown above, <u>without</u> knowledge of the true distribution
  ii) Identify missing points
  iii) Make a prediction/extrapolation?
- Put at least assumptions/information into fit as possible

$\Rightarrow$ **Use Gaussian processors**

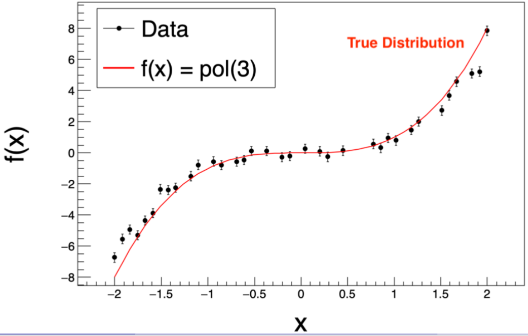
Gaussian Processors: General Idea

- Two points y_i and y_j with correlation: ρ_{ij}
- **Assumption:** y_i, y_j can be sampled from:
  $f(y_i, y_j) = \frac{1}{2\pi\sqrt{1-\rho_{ij}^2}} \times \exp\left\{ -\frac{1}{2(1-\rho_{ij}^2)}[y_i^2 - 2\rho_{ij}y_iy_j + y_j^2]\right\}$, with mean at 0
- Generate data set: $\mathbf{y} = \{y_1, y_2, .., y_n\}$, if ρ_{ij} is known:
  $f(\mathbf{y}) = \frac{1}{2\pi|\rho|} \times \exp[-0.5\mathbf{y}^T\rho^{-1}\mathbf{y}]$
- **Idea:** Parameterize ρ and fit above equation to your data



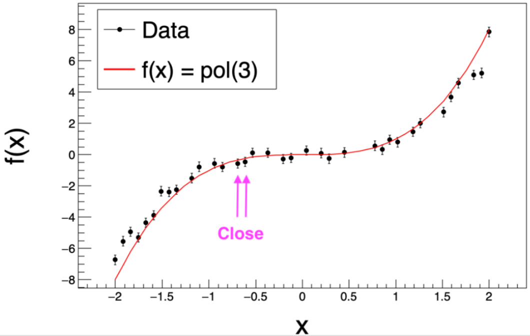
Gaussian Processors: Kernel Functions

- One of the most common functions:
  Exponential Squared: $k(x_i, x_j) = \sum_m \left[\sigma^2_{f,m} \exp\left\{ -\frac{1}{2}\frac{(x_i - x_j)^2}{\Delta^2_m} \right\}\right] + \sigma^2_n \delta(x_i, x_j)$
- Features of this function:
  - $\lim\limits_{|x_i - x_j| \to 0} k(x_i, x_j) \to \sum_m [\sigma^2_{f,m}] + \sigma^2_n$
  - $\lim\limits_{|x_i - x_j| \to \infty} k(x_i, x_j) \to 0$



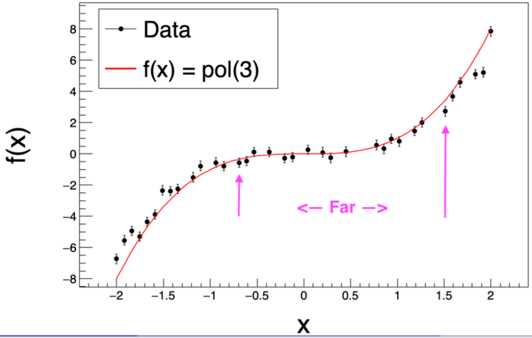
Gaussian Processors: Kernel Functions

- One of the most common functions:
  Exponential Squared: $k(x_i, x_j) = \sum_m \left[\sigma_{f,m}^2 \exp\left\{-\frac{1}{2}\frac{(x_i - x_j)^2}{\Delta_m^2}\right\}\right] + \sigma_n^2 \delta(x_i, x_j)$
- Features of this function:
  - $\lim_{|x_i - x_j| \to 0} k(x_i, x_j) \to \sum_m [\sigma_{f,m}^2] + \sigma_n^2 \to$ Close points share similar features
  - $\lim_{|x_i - x_j| \to \infty} k(x_i, x_j) \to 0$



Gaussian Processors: Kernel Functions

- One of the most common functions:
  Exponential Squared: $k(x_i, x_j) = \sum_m \left[\sigma_{f,m}^2 \exp\left\{ -\frac{1}{2}\frac{(x_i - x_j)^2}{\Delta_m^2} \right\}\right] + \sigma_n^2 \delta(x_i, x_j)$
- Features of this function:
  - $\lim_{|x_i - x_j| \to 0} k(x_i, x_j) \to \sum_m [\sigma_{f,m}^2] + \sigma_n^2$
  - $\lim_{|x_i - x_j| \to \infty} k(x_i, x_j) \to 0$ → Distant points do not "know" each other



Gaussian Processors: K-Matrices

- $k(x_i, x_j) = \left[\sigma_f^2 \exp\left\{-\frac{1}{2}\frac{(x_i - x_j)^2}{\Delta^2}\right\}\right] + \sigma_n^2 \delta(x_i, x_j)$
- Matrices needed for future calculation(s):

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ \vdots & & & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{pmatrix}$$

$$K^* = \begin{pmatrix} k(x_1^*, x_1) & k(x_1^*, x_2) & \dots & k(x_1^*, x_n) \\ \vdots & & & \vdots \\ k(x_n^*, x_1) & k(x_n^*, x_2) & \dots & k(x_n^*, x_n) \end{pmatrix}$$

$$K^{**} = \begin{pmatrix} k(x_1^*, x_1^*) & k(x_1^*, x_2^*) & \dots & k(x_1^*, x_n^*) \\ \vdots & & & \vdots \\ k(x_n^*, x_1^*) & k(x_n^*, x_2^*) & \dots & k(x_n^*, x_n^*) \end{pmatrix}$$

- x_i: x-Value from known data points: (x_i, y_i)
- x_i^*: x-Value from unknown data points: (x_i^*, y_i^*)
- K-matrices pick up the correlations

Gaussian Processors: Parameter Estimation

1.) Using Bayes theorem, minimize:

$-\log[P(\boldsymbol{x}, \boldsymbol{y}, \sigma_f, \Delta, \sigma_n)] \propto \boldsymbol{y}^T K^{-1} \boldsymbol{y} + \log|K|$ with respect to $\sigma_f, \Delta, \sigma_n$

2.) Use parameters $\sigma_f, \Delta, \sigma_n$ found during minimization and calculate

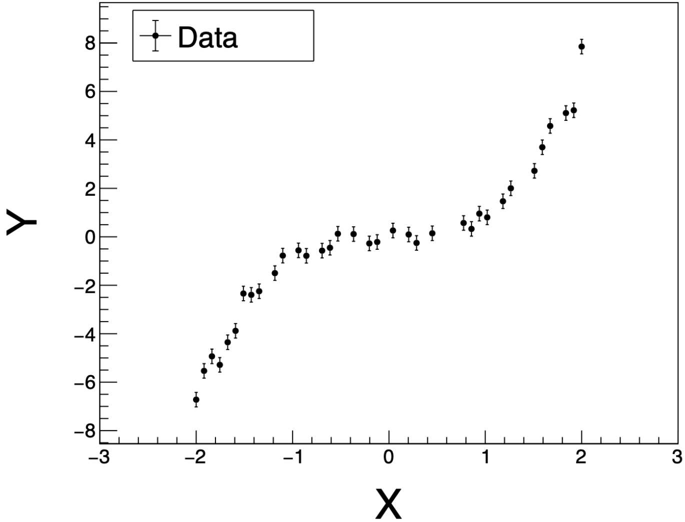
$\boldsymbol{y}^* = K^* K^{-1} \boldsymbol{y}$

$\boldsymbol{\Delta y^*} = K^{**} - K^* K^{-1} K^{*T}$

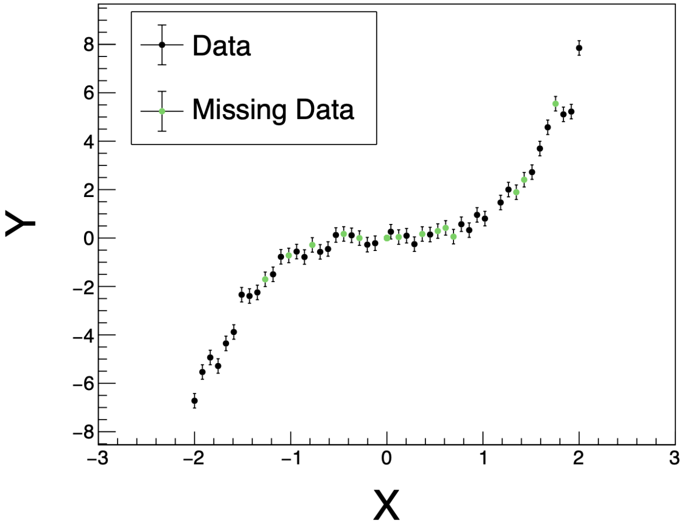
Note:

The expressions for: $\boldsymbol{y}^T K^{-1} \boldsymbol{y} + \log|K|$, $\boldsymbol{y}^*$ and $\boldsymbol{\Delta y^*}$ correspond to a multidimensional Gaussian distribution

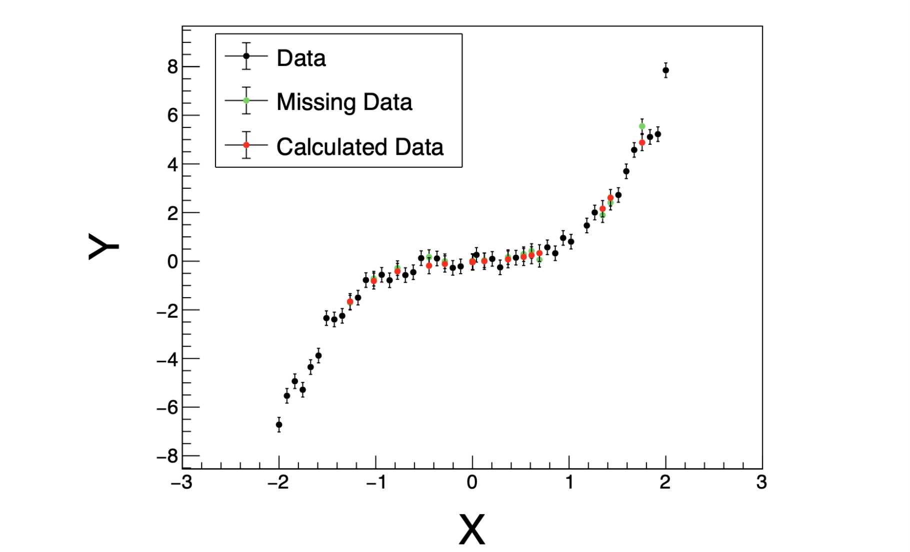
Gaussian Processors: Application on $f(x) = x^3$



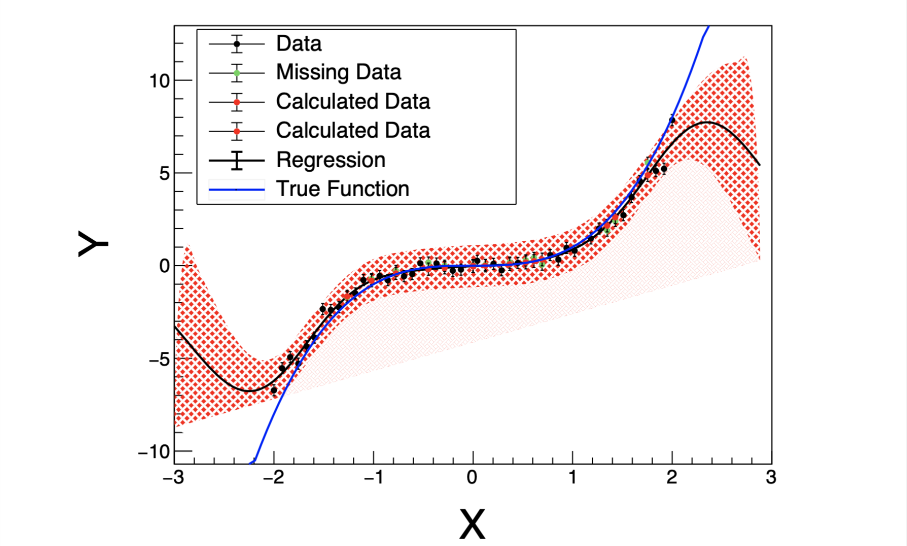
Gaussian Processors: Application on $f(x) = x^3$



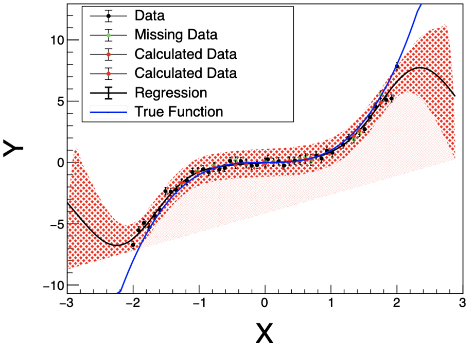
Gaussian Processors: Application on $f(x) = x^3$



Gaussian Processors: Application on $f(x) = x^3$

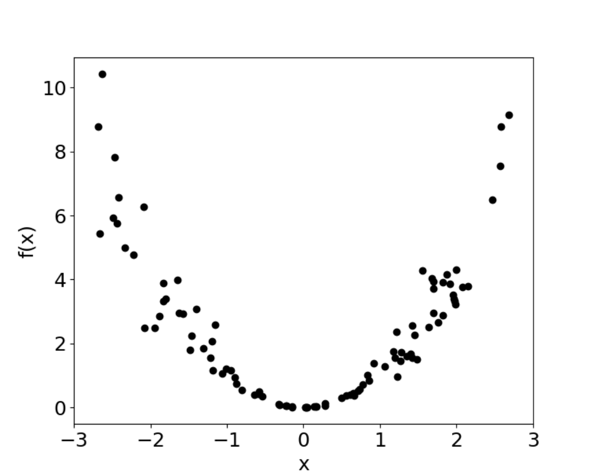


Gaussian Processors: Application on $f(x) = x^3$



- Found Parameters: $\sigma_f = 3.15 \pm 0.07$, $\Delta = 0.69 \pm 0.05$ and $\sigma_n = 0.53 \pm 0.08$
- Δ and σ_n roughly reflect the parameters that have been used to generate the above data: $\Delta x = 0.8$ and $\Delta y_{stat} \approx 0.3$
- Prediction/Extrapolation of the data outside the data limits fails, but:
  - One can easily modify the kernel-function
  - Leave points out during minimization $\Rightarrow$ Access to systematic uncertainties

Another noisy x^2-Function



- Given are a few noisy data points which seem to follow a x^2-distribution
- Would like to find the underlying distribution + prediction uncertainty
  $\Rightarrow$ Gaussian Processor

Setting up the Gaussian Processor

$$k(x_i, x_j) = \underbrace{\left[\exp\left\{-\frac{1}{2}\left(\frac{x_i - x_j}{\text{length\_scale}}\right)^2\right\}\right]}_{\text{RBF}} + \alpha(x_i, x_j)\delta(x_i, x_j) + \underbrace{\text{noise\_level}\cdot\delta(x_i, x_j)}_{\text{WhiteKernel}}$$

```
#1.) Define the kernel:
#RBF: Radial Basis Function = exponential squared
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
#White kernel: Corresponds to sigma_n --> constant noise
   + WhiteKernel(noise_level=1.0, noise_level_bounds=(1e-10, 1e+1))
```

Setting up the Gaussian Processor

$$k(x_i, x_j) = \underbrace{\left[\exp\left\{-\frac{1}{2}\left(\frac{x_i - x_j}{\text{length\_scale}}\right)^2\right\}\right]}_{\text{RBF}} + \alpha(x_i, x_j)\delta(x_i, x_j) + \underbrace{\text{noise\_level}\cdot\delta(x_i, x_j)}_{\text{WhiteKernel}}$$

```
#1.) Define the kernel:
#RBF: Radial Basis Function = exponential squared
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
#White kernel: Corresponds to sigma_n --> constant noise
   + WhiteKernel(noise_level=1.0, noise_level_bounds=(1e-10, 1e+1))

#2.) Setup the processor:
my_gp = GaussianProcessRegressor(
       kernel=kernel,
       n_restarts_optimizer=10, #--> How many times to run the minimization
       alpha=0.0 #--> similar to sigma_n if constant,
       #can be set for each data point individually--> individual error
 )
```

Setting up the Gaussian Processor

$$k(x_i, x_j) = \underbrace{\left[\exp\left\{-\frac{1}{2}\left(\frac{x_i - x_j}{\text{length\_scale}}\right)^2\right\}\right]}_{\text{RBF}} + \alpha(x_i, x_j)\delta(x_i, x_j) + \underbrace{\text{noise\_level} \cdot \delta(x_i, x_j)}_{\text{WhiteKernel}}$$

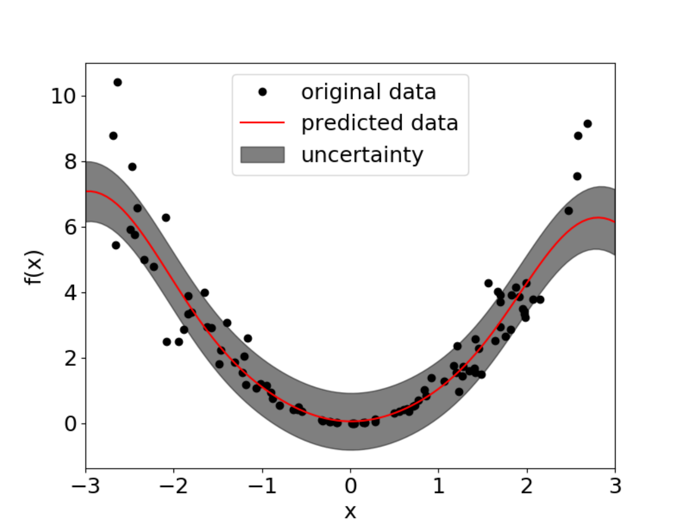
```
#1.) Define the kernel:
#RBF: Radial Basis Function = exponential squared
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
#White kernel: Corresponds to sigma_n --> constant noise
   + WhiteKernel(noise_level=1.0, noise_level_bounds=(1e-10, 1e+1))

#2.) Setup the processor:
my_gp = GaussianProcessRegressor(
       kernel=kernel,
       n_restarts_optimizer=10, #--> How many times to run the minimization
       alpha=0.0 #--> similar to sigma_n if constant,
       #can be set for each data point individually--> individual error
 )

 #3.) Set the parameters:
 my_gp.fit(x_values,y_values)
```

Setting up the Gaussian Processor

$$k(x_i, x_j) = \underbrace{\left[\exp\left\{-\frac{1}{2}\left(\frac{x_i - x_j}{\text{length\_scale}}\right)^2\right\}\right]}_{\text{RBF}} + \alpha(x_i, x_j)\delta(x_i, x_j) + \underbrace{\text{noise\_level} \cdot \delta(x_i, x_j)}_{\text{WhiteKernel}}$$

```
#1.) Define the kernel:
#RBF: Radial Basis Function = exponential squared
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
#White kernel: Corresponds to sigma_n --> constant noise
   + WhiteKernel(noise_level=1.0, noise_level_bounds=(1e-10, 1e+1))

#2.) Setup the processor:
my_gp = GaussianProcessRegressor(
       kernel=kernel,
       n_restarts_optimizer=10, #--> How many times to run the minimization
       alpha=0.0 #--> similar to sigma_n if constant,
       #can be set for each data point individually--> individual error
 )

 #3.) Set the parameters:
 my_gp.fit(x_values,y_values)

 #4.) Get the predictions:
 predictions, covariances = my_gp.predict(x_values,return_cov=True)
```

Applying the Gaussian Processor



- Points far outside not matched properly
- Could improve results by including a datapoint dependent α

Gaussian Processors: Summary

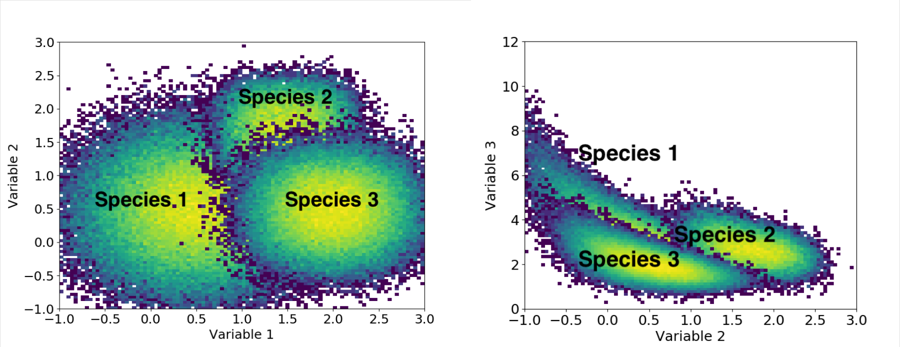
- Based on sampling from a multidimensional Gaussian distribution
- Use Kernel-Function to handle correlation between data points
- Determine covariance matrix from fit
- Also available in scikit: **GaussianProcessClassifier**
- Some hyper parameter optimizer make use of Gaussian processors

Classification with Machine Learning

- Most prominent application for machine learning algorithms

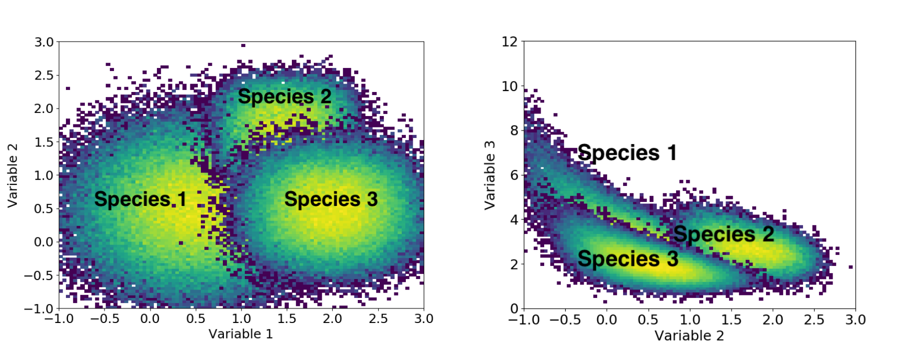
Classification with Machine Learning

- Most prominent application for machine learning algorithms
- Consider three species (e.g. particles, customer groups, car engine states...)



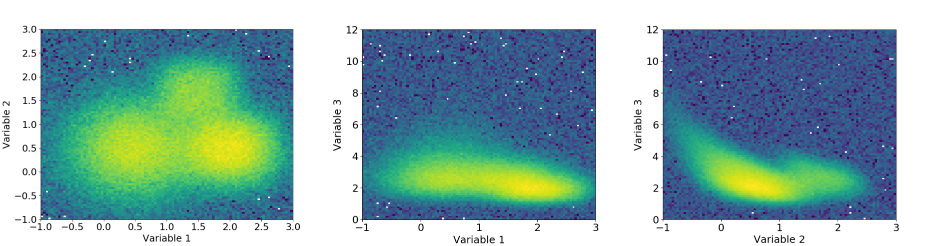
Classification with Machine Learning

- Most prominent application for machine learning algorithms
- Consider three species (e.g. particles, customer groups, car engine states...)
- Each species is defined by a set of three variables / features



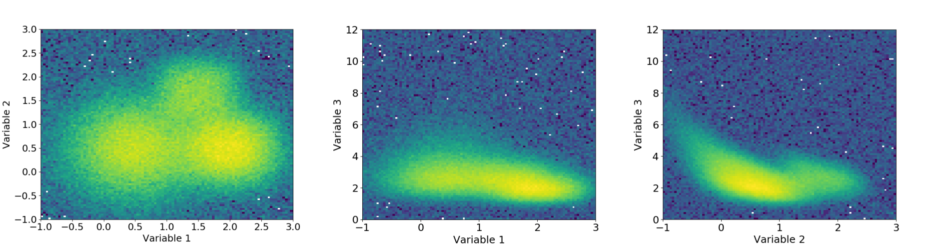
Classification with Machine Learning

- Most prominent application for machine learning algorithms
- Consider three species (e.g. particles, customer groups, car engine states...)
- Each species is defined by a set of three variables / features
- All species are measured / represented within one data set
  - Relative abundance between species is unequal (e.g. N(species 1) > N(species 2))
  - Noise contributions



Classification with Machine Learning

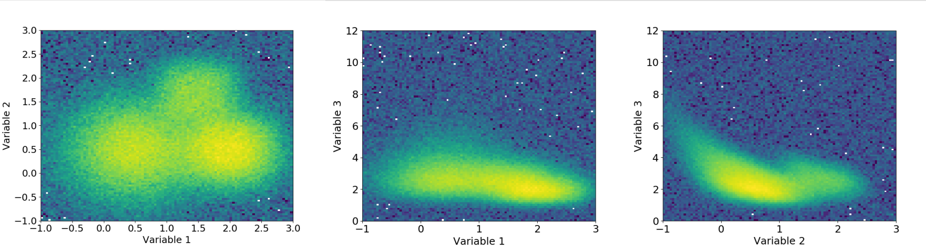
- Most prominent application for machine learning algorithms
- Consider three species (e.g. particles, customer groups, car engine states...)
- Each species is defined by a set of three variables / features
- All species are measured / represented within one data set
  - Relative abundance between species is unequal (e.g. N(species 1) > N(species 2))
  - Noise contributions
- **Goal:** Filter each species according to its features



Classification with Machine Learning

- Most prominent application for machine learning algorithms
- Consider three species (e.g. particles, customer groups, car engine states...)
- Each species is defined by a set of three variables / features
- All species are measured / represented within one data set
  - Relative abundance between species is unequal (e.g. N(species 1) > N(species 2))
  - Noise contributions
- **Goal:** Filter each species according to its features

⇒ Use a classification algorithm, aka classifier



The Training Data

- **Assumptions:**
  a) We possess a training data set
  b) The training set is a realistic representation of the measured/"real" data we want to analyze later
  c) Each species within the training set is labeled[6]: species 1 $\leftrightarrow$ 0, species 2 $\leftrightarrow$ 1 and species 3 $\leftrightarrow$ 2

[6] We want to train the classifier in such a way that it will be able to map: Features $\mapsto$ Label

The Training Data

- **Assumptions:**
  a) We possess a training data set
  b) The training set is a realistic representation of the measured/"real" data we want to analyze later
  c) Each species within the training set is labeled[6]: species 1 $\leftrightarrow$ 0, species 2 $\leftrightarrow$ 1 and species 3 $\leftrightarrow$ 2
- First, load the data and have a look at it

```
import pandas as pd
data = '/Volumes/BunchOfStuff/classifier_testData/fsu_ml_data3.csv'
data_df = pd.read_csv(data)

print(data_df.head(10)) #---> Look at the first 10 entries:
```

[6] We want to train the classifier in such a way that it will be able to map: Features $\mapsto$ Label

The Training Data

- **Assumptions:**
  a) We possess a training data set
  b) The training set is a realistic representation of the measured/"real" data we want to analyze later
  c) Each species within the training set is labeled[6]: species 1 $\leftrightarrow$ 0, species 2 $\leftrightarrow$ 1 and species 3 $\leftrightarrow$ 2
- First, load the data and have a look at it

	var1	var2	var3	label
0	2.140464	0.871710	1.634352	2.0
1	1.788192	1.992385	2.423125	1.0
2	0.602616	-0.480471	4.399315	0.0
3	1.354940	1.914728	2.849413	1.0
4	3.008098	0.649694	1.575176	2.0
5	1.241234	0.621577	2.915842	0.0
6	1.013267	-0.695762	4.493476	0.0
7	1.576466	2.001228	6.066541	0.0
8	0.920714	2.119301	2.783376	1.0
9	0.128073	1.348074	3.360470	0.0

[6] We want to train the classifier in such a way that it will be able to map: Features $\mapsto$ Label

The Training Data

- **Assumptions:**
  a) We possess a training data set
  b) The training set is a realistic representation of the measured/"real" data we want to analyze later
  c) Each species within the training set is labeled[6]: species 1 $\leftrightarrow$ 0, species 2 $\leftrightarrow$ 1 and species 3 $\leftrightarrow$ 2
- First, load the data and have a look at it
- Second prepare the data for the classifier

```
from sklearn.utils import shuffle

#Get the features / three variables for each species from the DataFrame
X = data_df[['var1','var2','var3']].values
#Get the labels / target values
Y = data_df['label'].values

#Shuffle the data
x_train, y_train = shuffle(X,Y,random_state=0)
```

[6] We want to train the classifier in such a way that it will be able to map: Features $\mapsto$ Label

Setup and train a MLP

- Prepare the MLP for training

```
from sklearn.neural_network import MLPClassifier

my_mlp = MLPClassifier(
        hidden_layer_sizes=(5),
        activation='tanh',
        solver='sgd',
        shuffle=True,
        validation_fraction=0.25,
        early_stopping=True,
        max_iter = 100,
        learning_rate_init=0.01,
        warm_start=True,
        tol=1e-6
)
```

Setup and train a MLP

- Prepare the MLP for training
- Train the MLP on the given data

```
#Do the mapping: Features |--> Label
my_mlp.fit(x_train,y_train)

#And get the learning / validation curve:
training_curve = my_mlp.loss_curve_
validation_curve = my_mlp.validation_scores_

plt.rcParams.update({'font.size': 18})
plt.plot(training_curve,label='training data')
plt.plot(validation_curve,label='validation data')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```

Setup and train a MLP

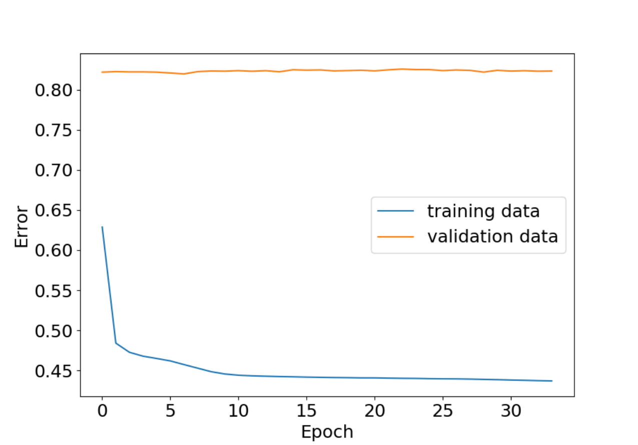
- Prepare the MLP for training
- Train the MLP on the given data

```
#Do the mapping: Features |--> Label
my_mlp.fit(x_train,y_train)

#And get the learning / validation curve:
training_curve = my_mlp.loss_curve_
validation_curve = my_mlp.validation_scores_

plt.rcParams.update({'font.size': 18})
plt.plot(training_curve,label='training data')
plt.plot(validation_curve,label='validation data')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```

- **Note:** This network has NOT been tuned for the upcoming analysis → Just "best-guess" settings

Setup and train a MLP

- Prepare the MLP for training
- Train the MLP on the given data
- **Note:** This network has NOT been tuned for the upcoming analysis $\rightarrow$ Just "best-guess" settings
- The error / loss-function for the mlp is given by the cross-entropy



Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

- with: $x_i = \sum_j w_{ji} o_j + b_i$, o_j = output from neuron j in the last hidden layer

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

- with: $x_i = \sum_j w_{ji} o_j + b_i$, o_j = output from neuron j in the last hidden layer
- By definition, all outputs of the mlp are normalized between 0 and 1

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

- with: $x_i = \sum_j w_{ji} o_j + b_i$, $o_j =$ output from neuron j in the last hidden layer
- By definition, all outputs of the mlp are normalized between 0 and 1
- How to transfer this to a label?

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label $\Rightarrow$ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

- with: $x_i = \sum_j w_{ji} o_j + b_i$, $o_j =$ output from neuron j in the last hidden layer
- By definition, all outputs of the mlp are normalized between 0 and 1
- How to transfer this to a label?
- In most cases[7]:
  label $i = \max\{\text{softmax(output 1), softmax(output 2), softmax (output3)}\}$

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Inspecting / Understanding the Network Output

- We have three species to identify / label ⇒ MLP has three outputs, one for each species
- The activation function for the output layer is the softmax-function

$$\text{softmax (output i)} = \exp(x_i)/(\sum_i \exp(x_i)) \tag{1}$$

- with: $x_i = \sum_j w_{ji} o_j + b_i$, o_j = output from neuron j in the last hidden layer
- By definition, all outputs of the mlp are normalized between 0 and 1
- How to transfer this to a label?
- In most cases[7]:
  label $i = \max\{\text{softmax(output 1), softmax(output 2), softmax (output3)}\}$
- **Example:** Suppose softmax(output 2) shows the largest response → This event would be labeled with 1

[7] In some frameworks you are able to set a threshold, i.e. suppress certain outputs

Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction

```
 #Get predicted labels:
predictions = my_mlp.predict(X) #---> X is the entire training data set
#Get the mlp outputs:
probabilities = my_mlp.predict_proba(X)

#Add them to the data frame:
data_df['prediction'] = predictions #--> 3D vector, because we have 3 species
data_df['probability1'] = probabilities[:,0]
data_df['probability2'] = probabilities[:,1]
data_df['probability3'] = probabilities[:,2]
#Have another look at the DataFrame:
print(data_df.head(10))
```

Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction

	var1	var2	var3	label	prediction	probability1	probability2	probability3
0	2.140464	0.871710	1.634352	2.0	2.0	0.016688	0.006019	0.977293
1	1.788192	1.992385	2.423125	1.0	1.0	0.033223	0.947372	0.019406
2	0.602616	-0.480471	4.399315	0.0	0.0	0.991312	0.002963	0.005725
3	1.354940	1.914728	2.849413	1.0	1.0	0.070873	0.891176	0.037950
4	3.008098	0.649694	1.575176	2.0	2.0	0.011981	0.005719	0.982301
5	1.241234	0.621577	2.915842	0.0	0.0	0.876290	0.058686	0.065025
6	1.013267	-0.695762	4.493476	0.0	0.0	0.985066	0.005237	0.009697
7	1.576466	2.001228	6.066541	0.0	1.0	0.307625	0.418755	0.273619
8	0.920714	2.119301	2.783376	1.0	1.0	0.108732	0.829198	0.062070
9	0.128073	1.348074	3.360470	0.0	0.0	0.632649	0.212343	0.155009

Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction

	var1	var2	var3	label	prediction	probability1	probability2	probability3
0	2.140464	0.871710	1.634352	2.0	2.0	0.016688	0.006019	0.977293
1	1.788192	1.992385	2.423125	1.0	1.0	0.033223	0.947372	0.019406
2	0.602616	-0.480471	4.399315	0.0	0.0	0.991312	0.002963	0.005725
3	1.354940	1.914728	2.849413	1.0	1.0	0.070873	0.891176	0.037950
4	3.008098	0.649694	1.575176	2.0	2.0	0.011981	0.005719	0.982301
5	1.241234	0.621577	2.915842	0.0	0.0	0.876290	0.058686	0.065025
6	1.013267	-0.695762	4.493476	0.0	0.0	0.985066	0.005237	0.009697
7	1.576466	2.001228	6.066541	0.0	1.0	0.307625	0.418755	0.273619
8	0.920714	2.119301	2.783376	1.0	1.0	0.108732	0.829198	0.062070
9	0.128073	1.348074	3.360470	0.0	0.0	0.632649	0.212343	0.155009

Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction

	var1	var2	var3	label	prediction	probability1	probability2	probability3
0	2.140464	0.871710	1.634352	2.0	2.0	0.016688	0.006019	0.977293
1	1.788192	1.992385	2.423125	1.0	1.0	0.033223	0.947372	0.019406
2	0.602616	-0.480471	4.399315	0.0	0.0	0.991312	0.002963	0.005725
3	1.354940	1.914728	2.849413	1.0	1.0	0.070873	0.891176	0.037950
4	3.008098	0.649694	1.575176	2.0	2.0	0.011981	0.005719	0.982301
5	1.241234	0.621577	2.915842	0.0	0.0	0.876290	0.058686	0.065025
6	1.013267	-0.695762	4.493476	0.0	0.0	0.985066	0.005237	0.009697
7	1.576466	2.001228	6.066541	0.0	1.0	0.307625	0.418755	0.273619
8	0.920714	2.119301	2.783376	1.0	1.0	0.108732	0.829198	0.062070
9	0.128073	1.348074	3.360470	0.0	0.0	0.632649	0.212343	0.155009

Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction

	var1	var2	var3	label	prediction	probability1	probability2	probability3
0	2.140464	0.871710	1.634352	2.0	2.0	0.016688	0.006019	0.977293
1	1.788192	1.992385	2.423125	1.0	1.0	0.033223	0.947372	0.019406
2	0.602616	-0.480471	4.399315	0.0	0.0	0.991312	0.002963	0.005725
3	1.354940	1.914728	2.849413	1.0	1.0	0.070873	0.891176	0.037950
4	3.008098	0.649694	1.575176	2.0	2.0	0.011981	0.005719	0.982301
5	1.241234	0.621577	2.915842	0.0	0.0	0.876290	0.058686	0.065025
6	1.013267	-0.695762	4.493476	0.0	0.0	0.985066	0.005237	0.009697
7	1.576466	2.001228	6.066541	0.0	1.0	0.307625	0.418755	0.273619
8	0.920714	2.119301	2.783376	1.0	1.0	0.108732	0.829198	0.062070
9	0.128073	1.348074	3.360470	0.0	0.0	0.632649	0.212343	0.155009

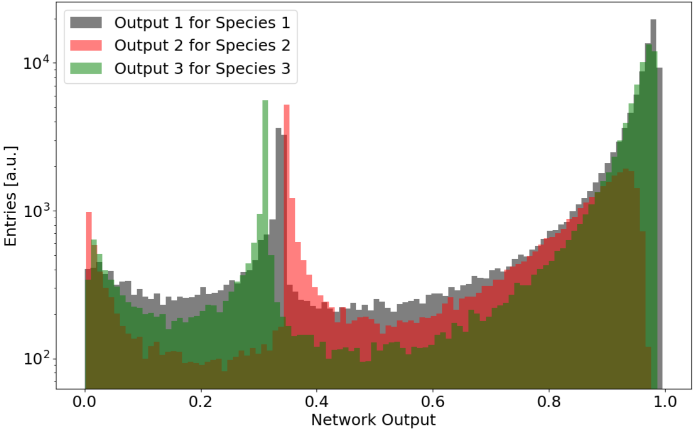
Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction
- **Always** look at the network output

```
#Plot the probabilities:
n_bins = 100
plt.hist(
      data_df[data_df['label']==0]['probability1'],
      bins=n_bins,
      facecolor='k',
      label='Output 1 for Species 1',
      alpha=0.5,
      log=True)

plt.hist(
     data_df[data_df['label']==1]['probability2'],
     ...)
plt.hist(
     data_df[data_df['label']==2]['probability3'],
     ...)
plt.xlabel('Network Output')
plt.ylabel('Entries [a.u.]')
plt.legend()
plt.show()
```

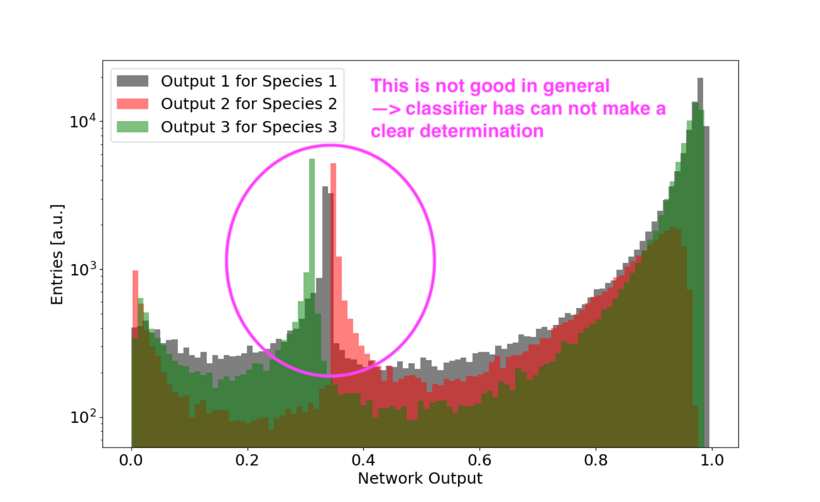
Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction
- **Always** look at the network output



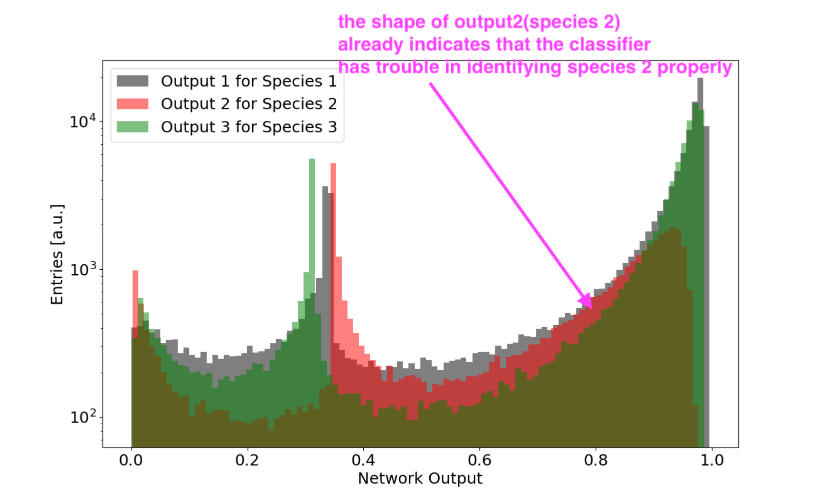
Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction
- **Always** look at the network output



Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction
- **Always** look at the network output



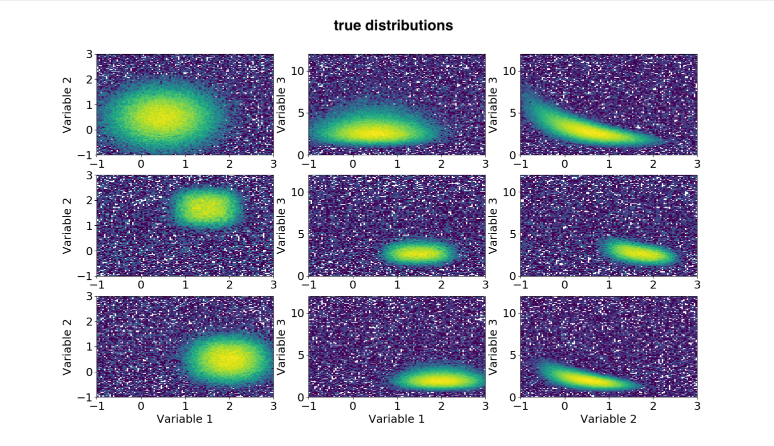
Accessing the MLP Output and Predictions

- Scikit allows to access both, the labeled and raw prediction
- **Always** look at the network output
- **Note:** The network / classifier output is often called "probabilty", which is technically not correct

⇒ Depending on how the algorithm has been trained, the output is not well defined between 0 and 1

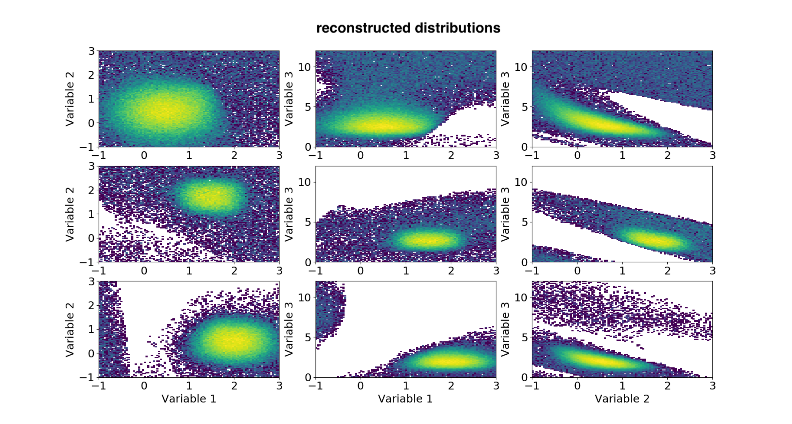
⇒ Need to calibrate the output: **On Calibration of Modern Neural Networks**

Monitoring the Classifier Performance



- **First thing to do:** Look at the features before / after classification
- Top row: true species 1 / Center row: true species2 / Bottom row: true species3

Monitoring the Classifier Performance



- **First thing to do:** Look at the features before / after classification
- Top row: identified species 1 / Center row: identified species2 / Bottom row: identified species3

The ROC-Curve

- ROC - Receiving Operator Characteristics

The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier

$$\text{False Positive Rate (species i)} = \frac{\#\text{Events identified as species i}}{\#\text{Events which do NOT contain species i}} \quad (2)$$

$$\text{True Positive Rate (species i)} = \frac{\#\text{Events with species i \& identified as species i}}{\#\text{Events only contain species i}} \quad (3)$$

The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier

$$\text{False Positive Rate (species i)} = \frac{\#\text{Events identified as species i}}{\#\text{Events which do NOT contain species i}} \quad (2)$$

$$\text{True Positive Rate (species i)} = \frac{\#\text{Events with species i \& identified as species i}}{\#\text{Events only contain species i}} \quad (3)$$

- Access ROC-curve in scikit

```
from sklearn.metrics import roc_curve

#Get the ROC-curve for each species:
fpr_s1, tpr_s1, th_s1 = roc_curve(
        data_df['label'].values,
        data_df['probability1'].values,
        pos_label=0)
fpr_s2, tpr_s2, th_s2 = roc_curve(
        data_df['label'].values,
        data_df['probability2'].values,
        pos_label=1)
...
```

The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier

$$\text{False Positive Rate (species i)} = \frac{\#\text{Events identified as species i}}{\#\text{Events which do NOT contain species i}} \quad (2)$$

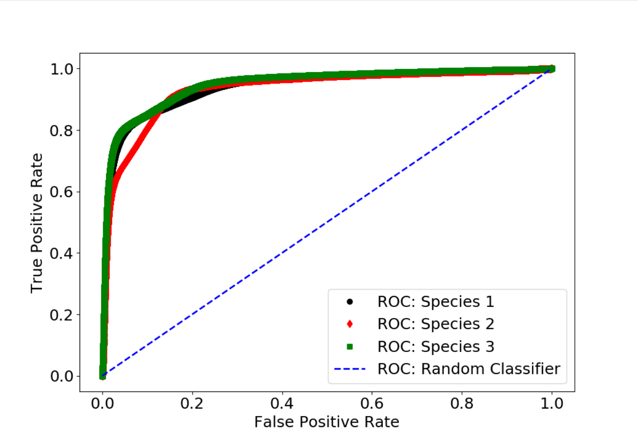
$$\text{True Positive Rate (species i)} = \frac{\#\text{Events with species i \& identified as species i}}{\#\text{Events only contain species i}} \quad (3)$$

- Access ROC-curve in scikit
- Plot the ROC-Curve

```
#Plot roc-curves:
plt.plot(fpr_s1,tpr_s1,'ko',label='ROC: Species 1')
plt.plot(fpr_s2,tpr_s2,'rd',label='ROC: Species 2')
...
plt.show()
```

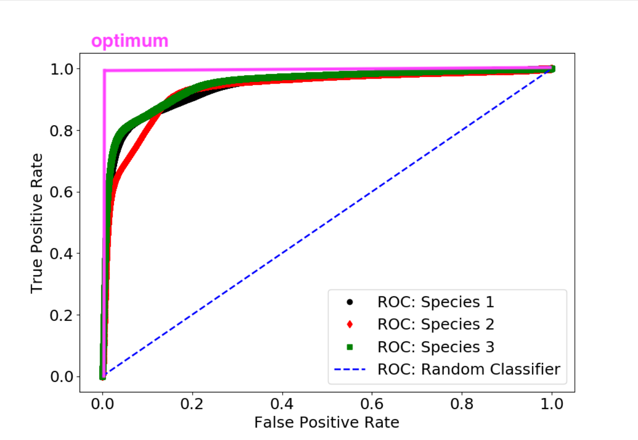
The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier
- Access ROC-curve in scikit
- Plot the ROC-Curve



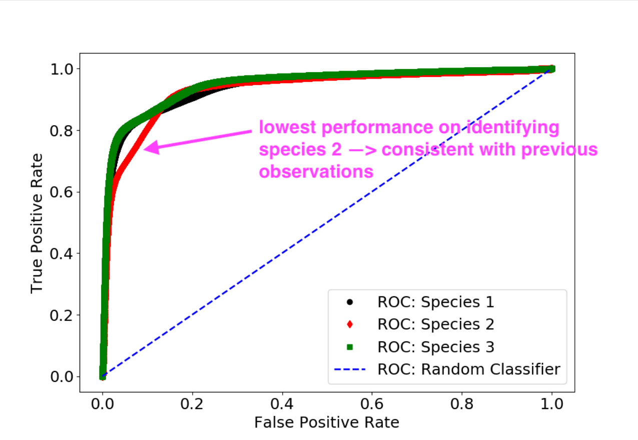
The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier
- Access ROC-curve in scikit
- Plot the ROC-Curve



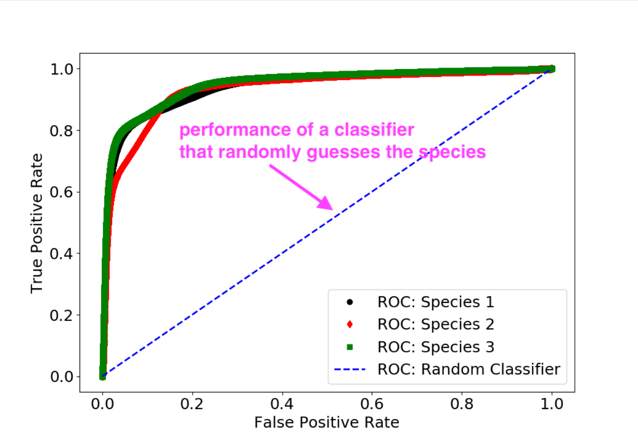
The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier
- Access ROC-curve in scikit
- Plot the ROC-Curve



The ROC-Curve

- ROC - Receiving Operator Characteristics
- This is one of the most important tools to monitor the performance of your classifier
- Access ROC-curve in scikit
- Plot the ROC-Curve



The Confusion Matrix

- Right after the ROC, the second most important monitoring tool

The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix

The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- The elements in the confusion matrix $\hat{C}$ are defined:

$$c_{ij} \equiv \sum_{k=0}^{N-1} \delta(L_{true,k} - \ell_i) \times \delta(L_{pred,k} - \ell_j) \quad (2)$$

$$\delta(x) = \begin{cases} 1, & \text{if } x = 0, \\ 0 & \text{else} \end{cases} \quad (3)$$

  With L_{true} / L_{pred} being the true / predicted label of event k and ℓ being the label you are interested in

- **NOTE:** The definition of the above equation depends on which axis holds the true / predicted label

The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- The elements in the confusion matrix $\hat{C}$ are defined:

$$c_{ij} \equiv \sum_{k=0}^{N-1} \delta(L_{true,k} - \ell_i) \times \delta(L_{pred,k} - \ell_j) \tag{2}$$

$$\delta(x) = \begin{cases} 1, & \text{if } x = 0, \\ 0 & \text{else} \end{cases} \tag{3}$$

  With L_{true} / L_{pred} being the true / predicted label of event k and ℓ being the label you are interested in

- **NOTE:** The definition of the above equation depends on which axis holds the true / predicted label

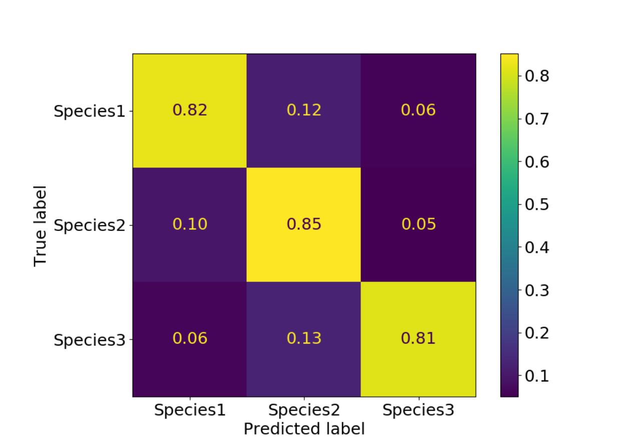
The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- Scikit handles the confusion matrix for you

```
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(my_mlp, #---> Your classifier
        X, #---> Your features
        Y, #---> Your labels
        display_labels=['Species1','Species2','Species3'],
        values_format='.2f',
        normalize='true') #---> Normalize with respect to true-axis
plt.show()
```

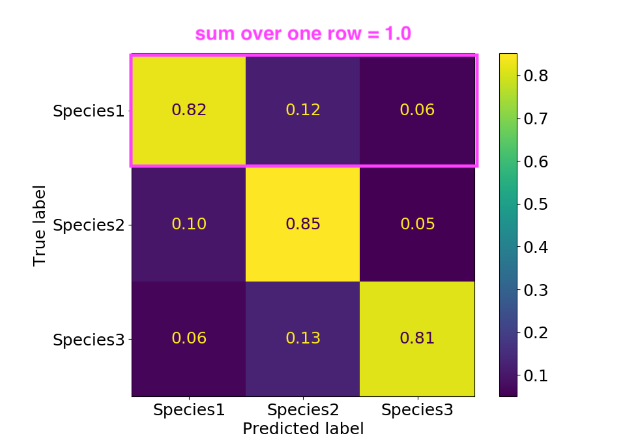
The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- Scikit handles the confusion matrix for you



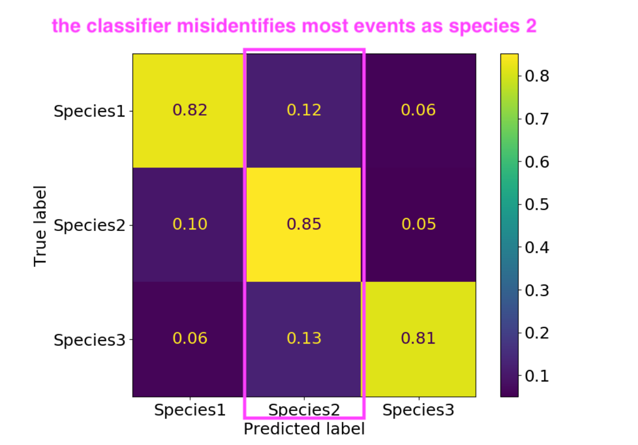
The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- Scikit handles the confusion matrix for you



The Confusion Matrix

- Right after the ROC, the second most important monitoring tool
- Nearly all performance measures (accuracy, F1 score, purity, mcc, efficiency,...) are directly derived from this matrix
- Scikit handles the confusion matrix for you



Summary Part II

- Include validation data into training of a neural network
  - Avoid overfitting
  - Enable generalization
- Gaussian processors for data regression
  - Sample from multidimensional Gaussian distribution via kernel function
  - Provide covariance matrix
- Classification with machine learning
  - Data set with three species, each defined by three features
  - Introduced important performance monitoring tools
    i) ROC-curve
    ii) Confusion matrix
  - Scikit nicely provides these tools $\Rightarrow$ Saves time in coding!
- Next part:
  - Introduce other machine learning algorithms
  - Performance metrics
  - Hyper parameter optimization