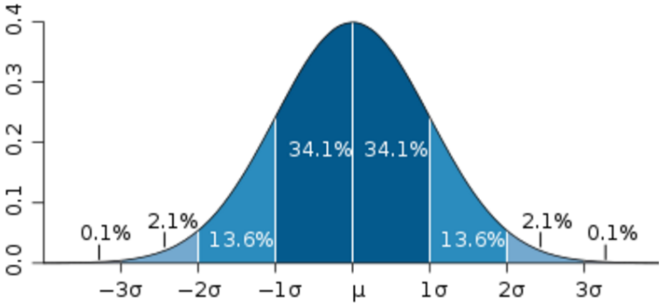
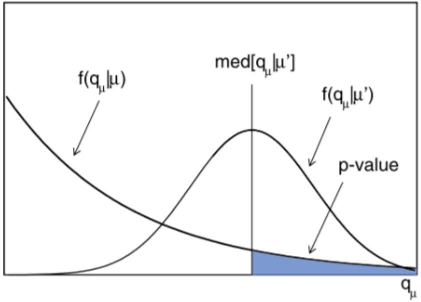




Confidence Intervals and a taste of ML

Applied Multi-Messenger Astronomy

Hans Niederhausen

TUM - winter term 2020/21

Large Sample Theory: Likelihood Ratio Testing

Reminder

given two hypotheses $H0 : \boldsymbol{\theta} = \boldsymbol{\theta}_0$ and $H1 : \boldsymbol{\theta} \neq \boldsymbol{\theta}_0$
the likelihood ratio test-statistic $\lambda\,(\mathrm{x})$ is defined as

$$\lambda\,(\mathrm{x}) = -2\log\Lambda\,(\mathrm{x}) = -2\log\left\{\frac{sup_{\nu}\ L\,(\boldsymbol{\theta}_0,\, \boldsymbol{\nu}\,|\,\mathrm{x})}{sup_{\nu,\,\theta}\ L\,(\boldsymbol{\theta},\, \boldsymbol{\nu}\,|\,\mathrm{x})}\right\} \tag{1}$$

to perform the hypothesis test, we also need to know the sampling distribution of this test-statistic:

$$\lambda \sim f_{\lambda}\,(\lambda;\, \boldsymbol{\theta},\, \boldsymbol{\nu}) \tag{2}$$

Often, this is non-trivial and one needs extensive Monte-Carlo computations (see example 3)
Luckily, as the sample size increases, the distribution is known to **converge**!
(beware of conditions!)

Large Sample Theory: Likelihood Ratio Testing

Wilk's Theorem

As the sample size increases, the distribution of the likelihood ratio test-statistic (eq. 10) converges to a χ^2 distribution with number of degrees of freedom k equal to the difference in number of free parameters specified by each hypothesis. In our notation $k = dim\,\boldsymbol{\theta}$.

$$f_\lambda\left(\lambda;\, \boldsymbol{\theta}_0\right) \underset{n\to\infty}{\longrightarrow} \chi^2\left(k\right) \tag{3}$$

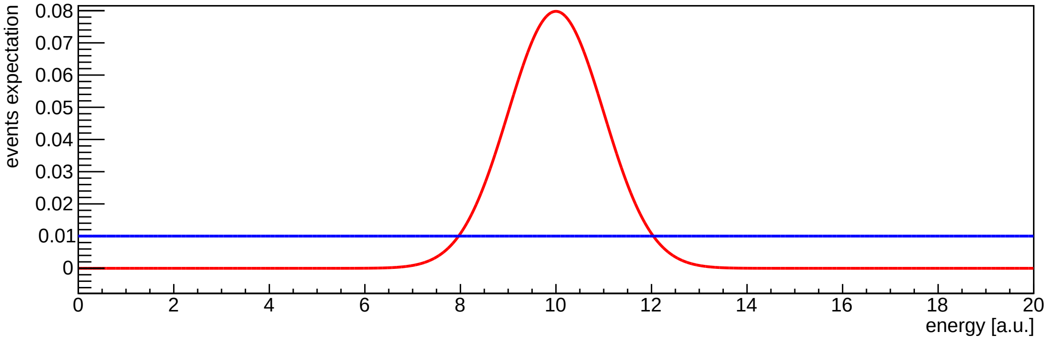
Wilk's Theorem (cont'd)
Unfortunately there are strict regularity conditions. Here are the two most important ones

- $\boldsymbol{\theta}_0$ needs to be an interior point of Θ
- nuisance parameters $\boldsymbol{\nu}$ that are only present under H1 are another issue
- ... several minor ones (typically not important)

Some extensions exists that might be useful (see Chernoff 1954, Gross, Vitells 2010) in such situations.

Large Sample Theory: The Toy Problem

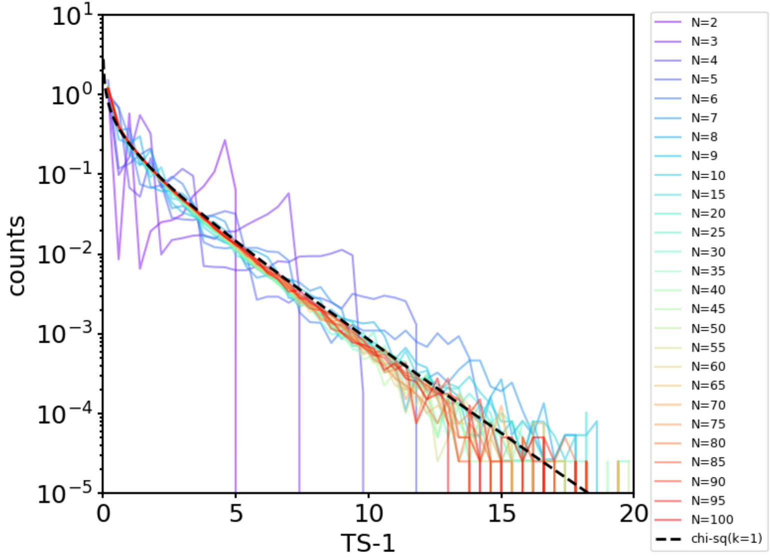
Application to our standard toy problem (with 2 parameters: p_s, μ_s)



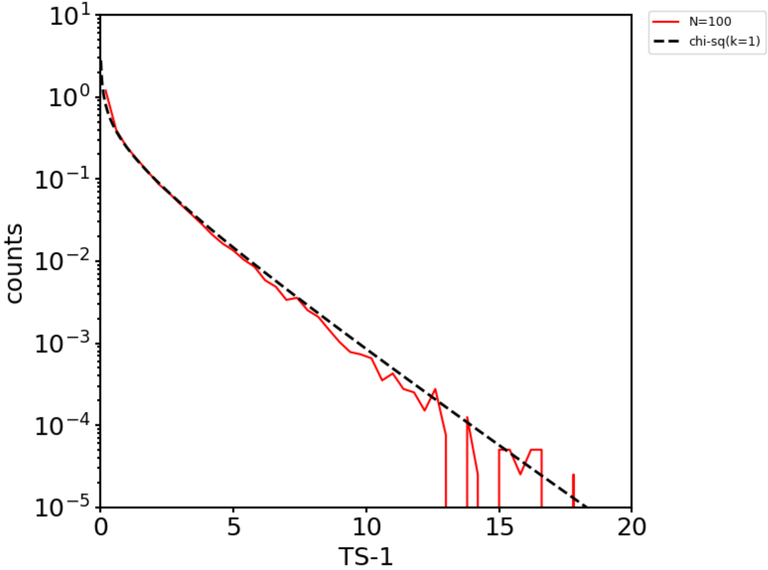
Two different hypothesis tests satisfying Wilk's theorem

Case 1: $H0 : p_s = 0.2$ and $H1 : p_s \neq 0.2$ (k=1) (μ_s is nuisance!)

Large Sample Theory: The Toy Problem

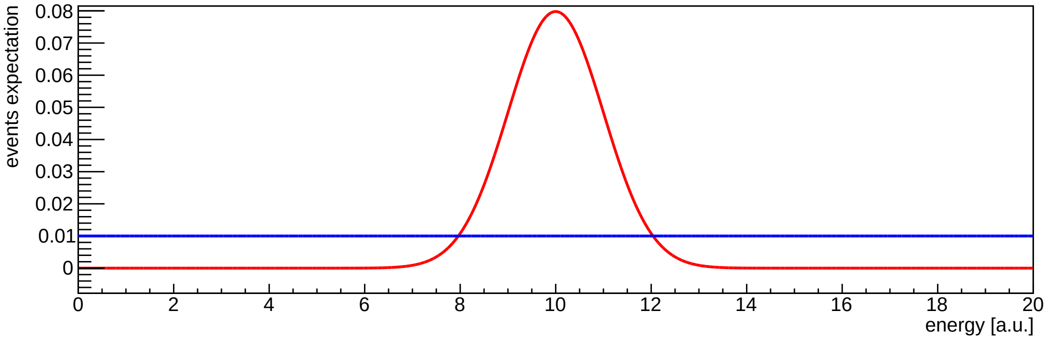


Large Sample Theory: The Toy Problem



Large Sample Theory: The Toy Problem

Application to our standard toy problem (with 2 parameters: p_s, μ_s)

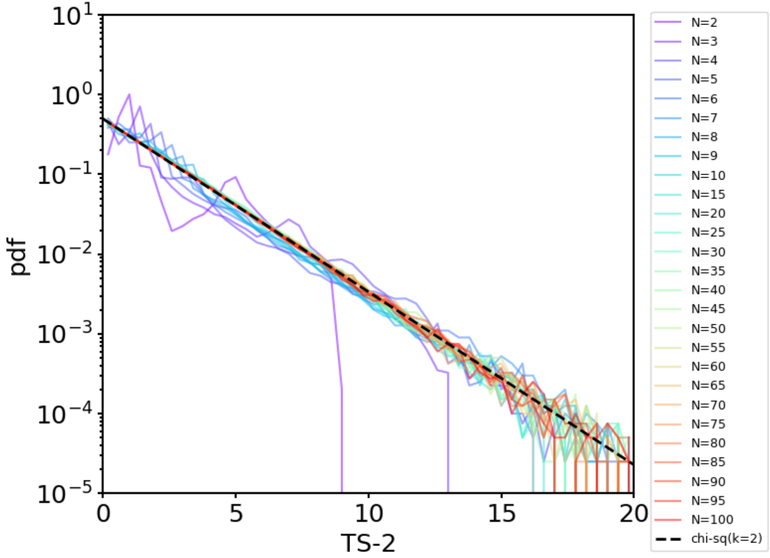


Two different hypothesis tests satisfying Wilk's theorem

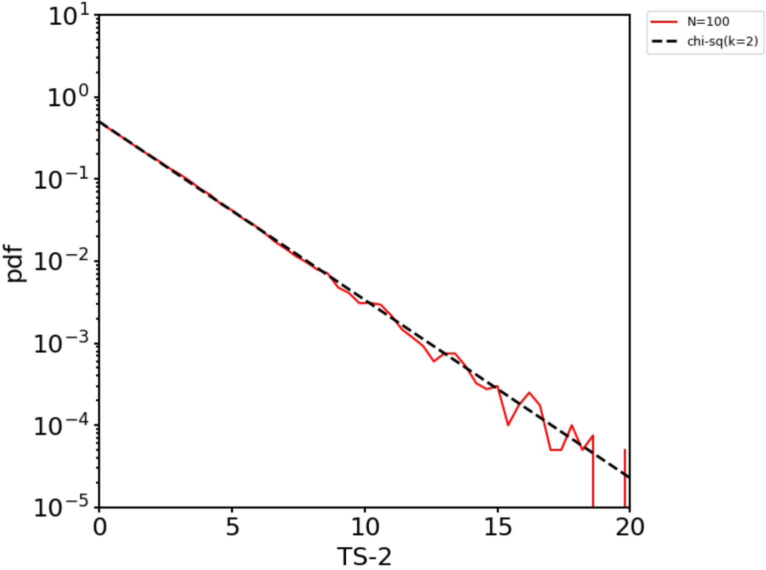
Case 1: $H0 : p_s = 0.2$ and $H1 : p_s \neq 0.2$ (k=1)

Case 2: $H0 : p_s = 0.2,\ \mu_s = 10.0$ and $H1 : p_s \neq 0.2,\ \mu_s \neq 10.0$ (k=2)

Large Sample Theory: The Toy Problem



Large Sample Theory: The Toy Problem

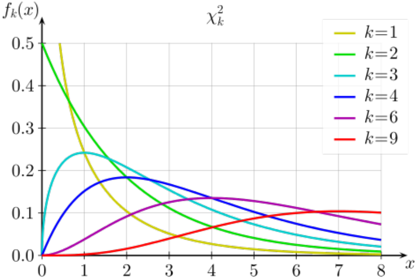


Critical Values from Chi-squared Distribution

for different number of degrees of freedom ($n = \Delta \mathrm{dim} \theta$)
and various choices of common levels α.

(here: critical value $c \equiv Q_\alpha$))

$1-\alpha$	Q_α				
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
0.683	1.00	2.30	3.53	4.72	5.89
0.90	2.71	4.61	6.25	7.78	9.24
0.95	3.84	5.99	7.82	9.49	11.1
0.99	6.63	9.21	11.3	13.3	15.1



Questions?

Confidence Intervals

Goal: calculate some range/region that has some probability to contain the true (unknown) parameter/s.

- Probability does not refer to the parameter (the true parameter is a fixed constant, not a random variable.) but to the region/interval that we obtain from the data.
- Generally speaking: different data results in a different region/interval (albeit construction is the same).

Confidence Intervals

Goal: calculate some range/region that has some probability to contain the true (unknown) parameter/s.

- Probability does not refer to the parameter (the true parameter is a fixed constant, not a random variable.) but to the region/interval that we obtain from the data.
- Generally speaking: different data results in a different region/interval (albeit construction is the same).

Mathematically, from data X we calculate function values $L(\mathrm{X})$ and $U(\mathrm{X})$ which are random variables.

$$[L(\mathrm{X}),\ U(\mathrm{X})] \quad (two - sided) \tag{4}$$

$$(-\infty,\ U(\mathrm{X})] \quad or \quad [L(\mathrm{X}),\ \infty) \quad (one - sided) \tag{5}$$

in physics: two-sided intervals often called "uncertainties", one-sided intervals often called "limits". (sometimes gets mixed ... e.g. hard to tell difference on bounded parameter spaces. always check how the construction was done.)

Confidence Intervals: Coverage

coverage := probability that the random interval $[L(X), U(X)]$ (or limit) happens to overlap with the unknown, true parameter value.

$$P_\theta\left(\theta \in [L(X),\ U(X)]\right) \tag{6}$$

confidence coefficient of an interval (denoted by $1-\alpha$) defined by

$$inf_\theta P_\theta\left(\theta \in [L(X),\ U(X)]\right) = 1-\alpha \tag{7}$$

Can not always guarantee exact coverage (hello nuisance parameters!) - strive to guarantee confidence coefficient (i.e. minimum coverage!). That is usually possible.

Confidence Intervals: Coverage in the normal mean problem

Consider the problem of constructing a confidence interval for the unknown mean μ of a normal distribution (variance σ^2 known) from n observations $(\mathrm{X} = \{X_1, ..., X_n\})$. This can be done using a **pivot** (a function of the parameter and observations, that has a distribution which is independent of the parameter).

$$Q(\mu, \mathrm{X}) = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \tag{8}$$

$$Q \sim N(0, 1) \tag{9}$$

i.e. here Q is a standard normal random variable. Thus can solve

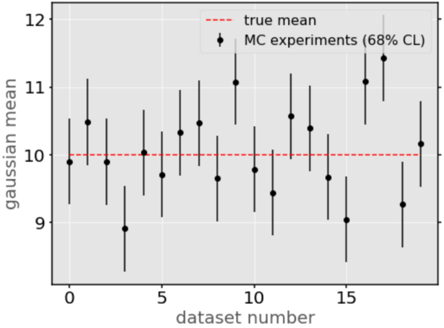
$$P_\mu\left(-a \leq Q \leq a\right) = 1 - \alpha \tag{10}$$

which corresponds to the following confidence set

$$\left\{\mu : \bar{X} - a\frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{x} + a\frac{\sigma}{\sqrt{n}}\right\} \tag{11}$$

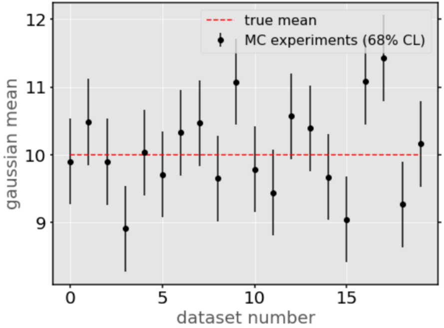
Confidence Intervals: Coverage in the normal mean problem

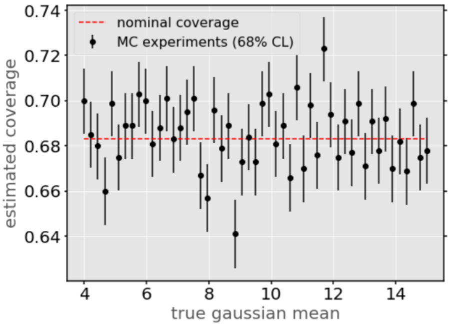
Check with Monte-Carlo (see ipython notebook)



Confidence Intervals: Coverage in the normal mean problem

Check with Monte-Carlo (see ipython notebook)





Confidence Intervals from inversion of hypothesis tests

If you can construct a level α hypothesis test for the unknown parameter/s specified by H_0 it is always possible to use this test to construct a confidence interval with guaranteed confidence coefficient $1 - \alpha$.

This is called **inverting a hypothesis test**. Whether you get two-sided or one-sided intervals depends on the alternative hypothesis

- $H_0 : \theta = \theta_0$ and $H_1 : \theta \neq \theta_0$ produces two-sided intervals
- $H_0 : \theta = \theta_0$ and $H_1 : \theta < \theta_0$ produces one-sided intervals (upper-limit)
- $H_0 : \theta = \theta_0$ and $H_1 : \theta > \theta_0$ produces one-sided intervals (lower-limit)

Confidence Intervals from inversion of hypothesis tests

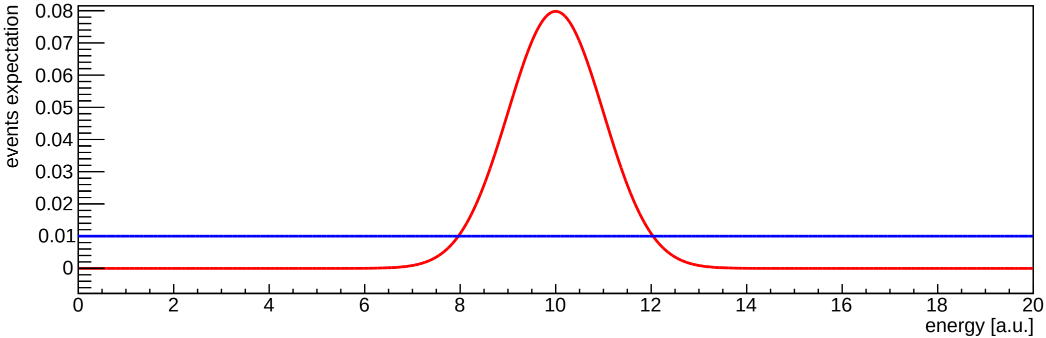
Why does it work?

- perform the test on every possible point in parameter space
- if the test rejects the point, simply discard it
- if the point is accepted, add the point to your confidence set
- Whats the coverage of this strategy? (probability that the random set contains true parameter)
- Probability to rejected a parameter if it is true is $\leq \alpha$ by definition (size of test)
- Thus, probability for true parameter to contribute to set is $\geq 1 - \alpha$.
- Hence, probability for set to cover true parameter is $\geq 1 - \alpha$ by construction

Questions?

Confidence Intervals from inversion of LRT

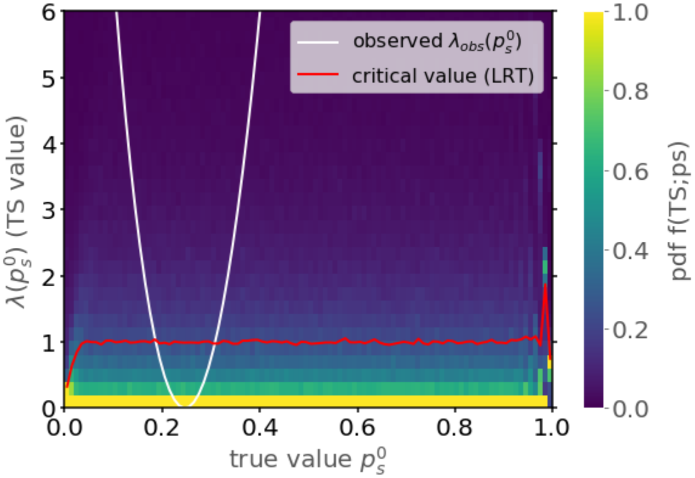
We have learned how to construct likelihood ratio tests. Let's invert a likelihood ratio test to obtain a confidence set on the signal fraction p_s in our toy model. (see ipython notebooks)



$$f_X(x;\mu,\sigma,p_s) = \left[p_s \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} + (1-p_s) \cdot \frac{1}{20}\right] \tag{12}$$

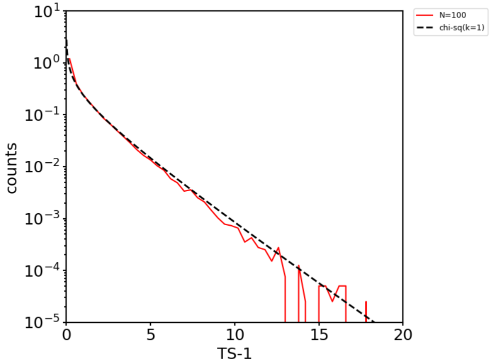
Confidence Intervals from inversion of LRT in toy problem.

$H_0 : p_s = p_s^0$ and $H_1 : p_s \neq p_s^0$, sample size n=100
endpoints of interval: intersection points of obs. TS value (white) with critical value (red)



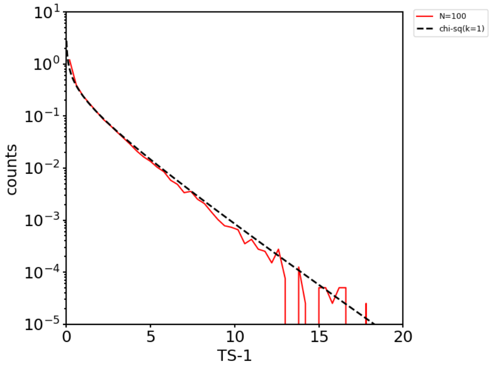
Reminder: Asymptotic Chi-Squared.

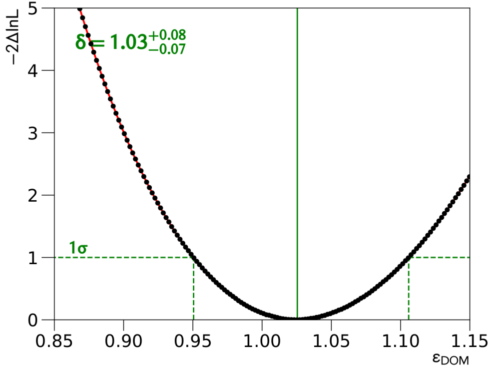
For **large n**:
The distribution of λ is independent from (true) p_s^0 with $\lambda \sim \chi^2(k=1)$
We just have to find the points, where λ changes by 1 (for $1-\alpha = 0.68$).
(because $p(\lambda \geq c) = \alpha$ yields $c = 1$ for k=1 and $1-\alpha = 0.68$.



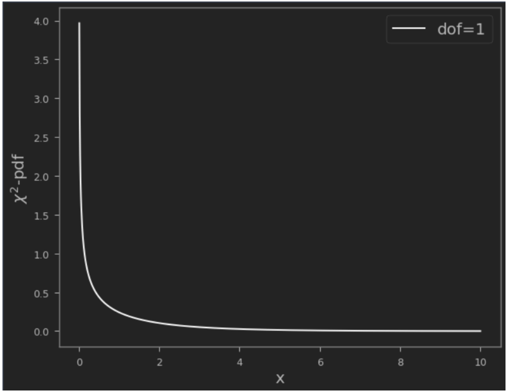
Reminder: Asymptotic Chi-Squared.

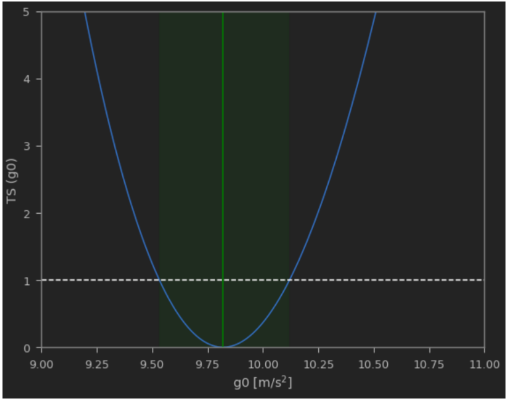
For **large n**:
The distribution of λ is independent from (true) p_s^0 with $\lambda \sim \chi^2(k=1)$
We just have to find the points, where λ changes by 1 (for $1-\alpha = 0.68$).
(because $p(\lambda \geq c) = \alpha$ yields $c = 1$ for k=1 and $1-\alpha = 0.68$.





Exercise 4, Problem 3



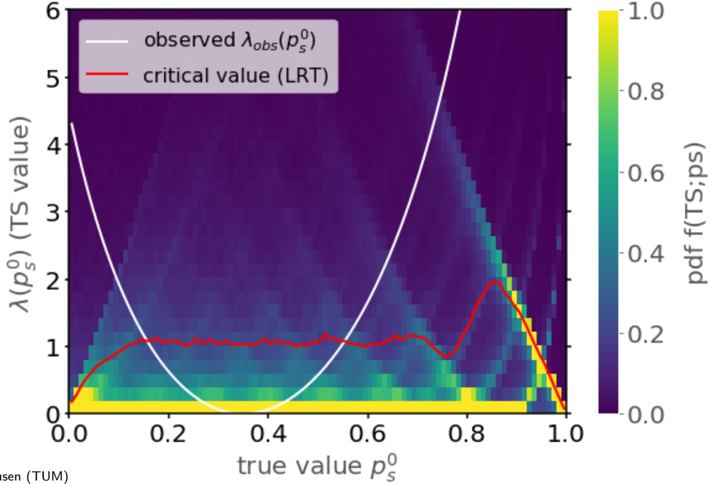


Questions?

Confidence Intervals from inversion of LRT in toy problem.

The problem is much harder, if Wilk's theorem does not apply.

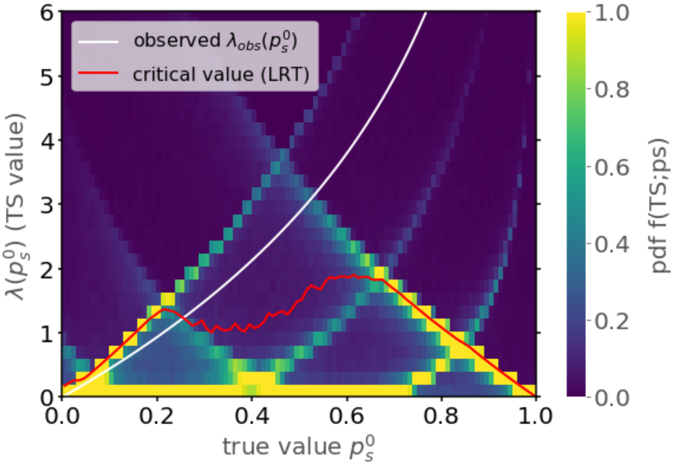
$H_0 : p_s = p_s^0$ and $H_1 : p_s \neq p_s^0$, sample size n=10
endpoints of interval: intersection points of obs. TS value (white) with critical value (red)



Confidence Intervals from inversion of LRT in toy problem.

The problem is much harder, if Wilk's theorem does not apply.

$H_0 : p_s = p_s^0$ and $H_1 : p_s \neq p_s^0$, sample size n=3
endpoints of interval: intersection points of obs. TS value (white) with critical value (red)

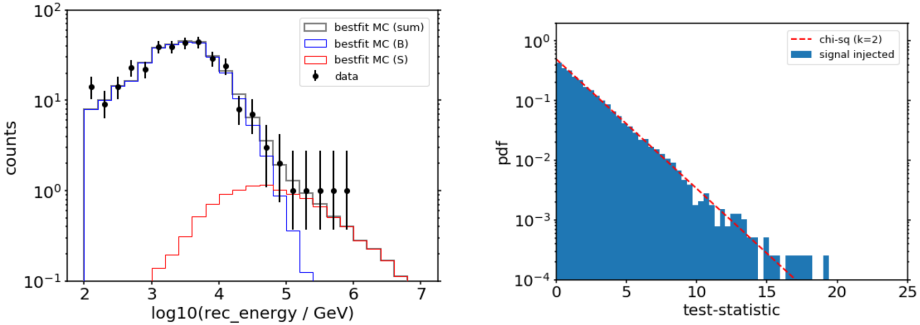


Summary: Confidence Intervals from inversion of LRT

- very simple if your measurement is in the asymptotic regime (lots of data!)
- obtain the critical value (red curve) from wilk's theorem (i.e. appropriate χ^2-pdf)
- generalizes well to high dimensions, if analysis remains asymptotic
- if asymptotics don't apply, you will run out of CPU quickly as the dimensionality increases (since you need to costruct the TS distributions for each point in parameter space)
- always check a few representative parameter combinations (and also a few extreme ones) first

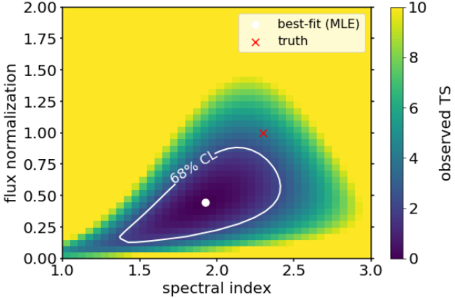
Example: Inversion of LRT in the IceCube diffuse flux measurement

To construct a joint confidence interval for the normalization and spectral index of the astrophysical neutrino flux, we need to invert a LRT:
$H_0 : (\Phi, \gamma) = (\Phi_0, \gamma_0)$ and $H_1 : (\Phi, \gamma) \neq (\Phi_0, \gamma_0)$
The asymptotic expectation for the TS distribution would be χ^2 with 2 dof.



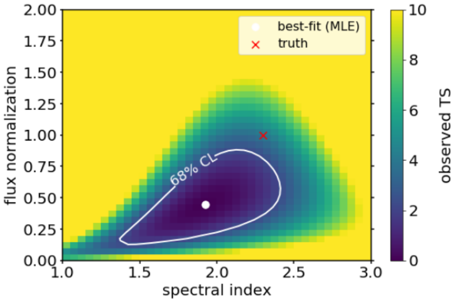
Example: Inversion of LRT in the IceCube diffuse flux measurement

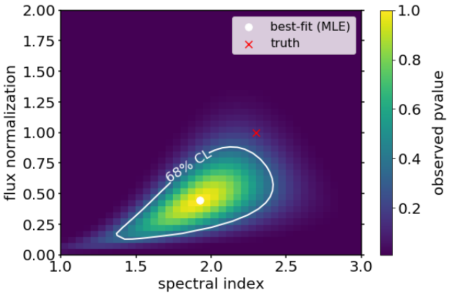
if we have sufficient data, we use the χ^2 pdf (left) otherwise we need to obtain (valid) p-values from MC simulations and use those to get the contours (right)



Example: Inversion of LRT in the IceCube diffuse flux measurement

if we have sufficient data, we use the χ^2 pdf (left) otherwise we need to obtain (valid) p-values from MC simulations and use those to get the contours (right)



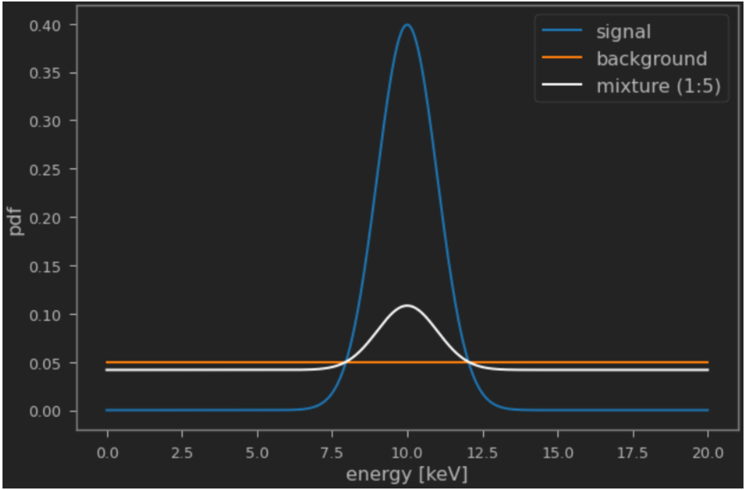


$$p(x_{\text{obs}}) = \sup_{\theta \in \Theta_0} P_\theta \left(TS(X) \geq TS(x_{\text{obs}}) \right) \tag{13}$$

Questions?

A taste of ML: Classification

Let's come back to our toy example.



$$f_X(x;\mu,\sigma,p_s) = \left[p_s \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} + (1-p_s)\cdot\frac{1}{20}\right]; \;\; p_s \frac{\lambda_s}{\lambda_s+\lambda_b} \qquad (14)$$

To classify events into signal and background based on the observed energy, we need to setup some notation.

- introduce binary random variable c for the class-membership of each event ($c = 1$ for signal, $c = 0$ for background).
- need to find posterior probability $p(c \mid E)$.
- one possible criterion: consider event as signal if $p(c = 1 \mid E) > 0.5$.

We need the following ingredients: $p(c)$, $f(E \mid c)$ and $f(E)$.

To classify events into signal and background based on the observed energy, we need to setup some notation.

- introduce binary random variable c for the class-membership of each event ($c = 1$ for signal, $c = 0$ for background).
- need to find posterior probability $p(c \mid E)$.
- one possible criterion: consider event as signal if $p(c = 1 \mid E) > 0.5$.

We need the following ingredients: $p(c)$, $f(E \mid c)$ and $f(E)$.

$$p(c \mid E) = \frac{f(E \mid c) \cdot p(c)}{f(E)} \tag{15}$$

where $f(E)$ is the usual energy pdf (mixture).

The marginal pmf $p(c)$ is easy.

- $p(c = 1) = \frac{\lambda_s}{\lambda_s + \lambda_b}$
- $p(c = 0) = 1 - p(c = 1) = \frac{\lambda_b}{\lambda_s + \lambda_b}$

```
def class_prob(c, pars):
    lambda_s, lambda_b = (pars['lambda_s'], pars['lambda_b'])
    lambda_tot = lambda_s + lambda_b

    if c == 0:
        return lambda_b / lambda_tot

    elif c == 1:
        return lambda_s / lambda_tot

    else:
        raise ValueError(f"ValueError: c can only be 0 or 1. but value c={c} gi

pars = {'lambda_s':200, 'lambda_b':1000}

print(class_prob(0, pars))
print(class_prob(1, pars))
```

```
0.8333333333333334
0.16666666666666666
```

And so is the conditional pdf $f(E \mid c)$.

For $c = 1$: this is just the normal $N(\mu, \sigma)$ distribution
For $c = 0$: we have the uniform $u(E_{min}, E_{max})$ distribution

```
def conditional_energy_pdf(energy, c, pars):

    if c == 0:
        return uniform.pdf(energy, pars['emin'], pars['emax'])

    elif c == 1:
        return norm.pdf(energy, pars['mu'], pars['sigma'])

    else:
        raise ValueError(f"ValueError: c can only be 0 or 1. but value c={c} given.")

pars.update({'emin':0., 'emax':20., 'mu':10., 'sigma':1.})

print(conditional_energy_pdf(10, 0, pars))
print(conditional_energy_pdf(10, 1, pars))
```

```
0.05
0.3989422804014327
```

Putting it all together ...

```
In [82]:
def joint_pdf(energy , c, pars):
    if c == 0 or c == 1:
        return conditional_energy_pdf(energy, c, pars) * class_prob(c, pars)
    else:
        raise ValueError(f"ValueError: c can only be 0 or 1. but value c={c} given.")

print(joint_pdf(10, 0, pars))
print(joint_pdf(10, 1, pars))
```

```
0.041666666666666667
0.06649038006690544
```

Putting it all together ...

```
In [82]: def joint_pdf(energy , c, pars):
             if c == 0 or c == 1:
                 return conditional_energy_pdf(energy, c, pars) * class_prob(c, pars)
             else:
                 raise ValueError(f"ValueError: c can only be 0 or 1. but value c={c} given.")

         print(joint_pdf(10, 0, pars))
         print(joint_pdf(10, 1, pars))

          0.04166666666666667
          0.06649038006690544
```

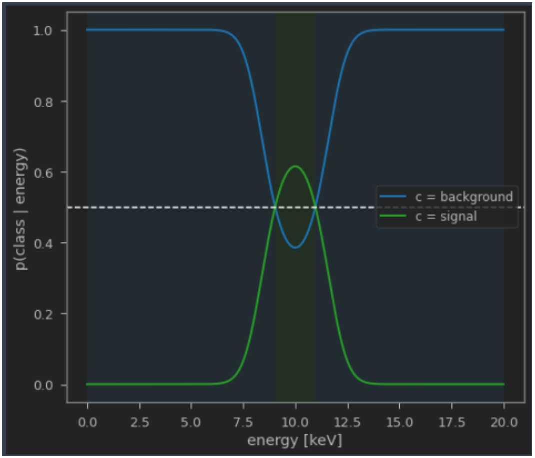
```
def class_prob_given_energy(c, energy, pars):
    if c == 0 or c == 1:
        p_joint = joint_pdf(energy, c, pars)


        p_marginal = 0
        for c in [0, 1]:
            p_marginal += joint_pdf(energy, c, pars)

        return p_joint / p_marginal

    else:
        raise ValueError(f"ValueError: c can only be 0 or 1. but value c={c} given.")

p0 = class_prob_given_energy(0, 10, pars)
p1 = class_prob_given_energy(1, 10, pars)
print(p0, p1, p0+p1)

 0.3852422743134431 0.614757725686557 1.0
```

Voila! The result.

```
energies = np.linspace(0.0, 20.0, 1000)
background_probs = class_prob_given_energy(0, energies, pars)
signal_probs = class_prob_given_energy(1, energies, pars)

# get a decision boundary
idx = background_probs < 0.5
boundary_idx = np.where(np.diff(np.array(idx, dtype=int))!=0)[0]

boundary_low = energies[boundary_idx[0]]
boundary_high = energies[boundary_idx[1]]

plt.plot(energies, background_probs, label='c = background', color='tab:blue', linewidth=2)
plt.plot(energies, signal_probs, label='c = signal', color='tab:green', linewidth=2)

plt.axvspan(pars['emin'], boundary_low, color='tab:blue', alpha=0.1)
plt.axvspan(boundary_high, pars['emax'], color='tab:blue', alpha=0.1)
plt.axvspan(boundary_low, boundary_high, color='tab:green', alpha=0.1)
plt.axhline(0.5, color='white', linestyle='dashed')

plt.xlabel("energy [keV]")
plt.ylabel("p(class | energy)")
plt.legend(fontsize=12)
```

Voila! The result.

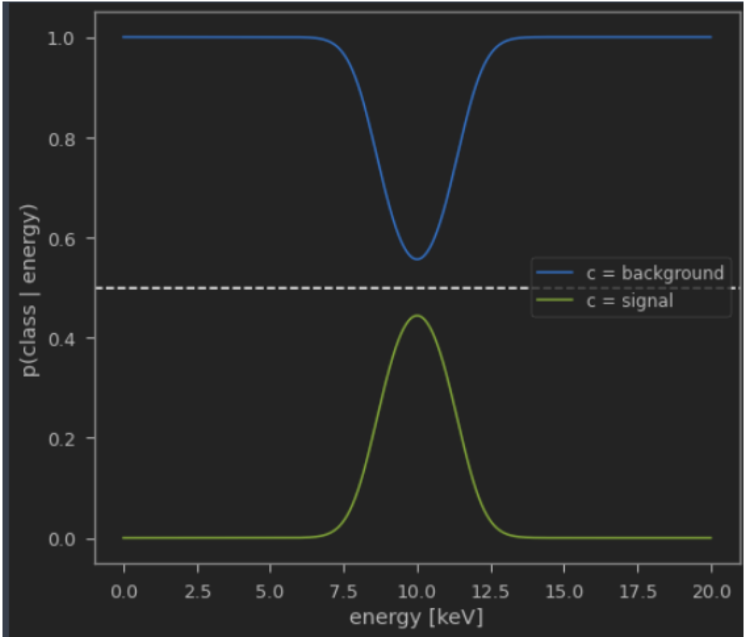
```
energies = np.linspace(0.0, 20.0, 1000)
background_probs = class_prob_given_energy(0, energies, pars)
signal_probs = class_prob_given_energy(1, energies, pars)

# get a decision boundary
idx = background_probs < 0.5
boundary_idx = np.where(np.diff(np.array(idx, dtype=int))!=0)[0]

boundary_low = energies[boundary_idx[0]]
boundary_high = energies[boundary_idx[1]]

plt.plot(energies, background_probs, label='c = background', color='tab:blue', linewidth=2)
plt.plot(energies, signal_probs, label='c = signal', color='tab:green', linewidth=2)

plt.axvspan(pars['emin'], boundary_low, color='tab:blue', alpha=0.1)
plt.axvspan(boundary_high, pars['emax'], color='tab:blue', alpha=0.1)
plt.axvspan(boundary_low, boundary_high, color='tab:green', alpha=0.1)
plt.axhline(0.5, color='white', linestyle='dashed')

plt.xlabel("energy [keV]")
plt.ylabel("p(class | energy)")
plt.legend(fontsize=12)
```



The green region shows where $f(c = 1 \mid E) > 0.5$, i.e. energies for which the conditional signal probability is larger than the background probability.

What happens if λ_s becomes small (e.g. $\lambda_s = 100$, $\lambda_b = 1000$)?

What happens if λ_s becomes small (e.g. $\lambda_s = 100$, $\lambda_b = 1000$)?



Questions?

What if we do not know the right model?

- true relationship $\mu(x) = g(x; \vec{\theta})$ e.g. in regression problems (like $s(t) = \frac{1}{2}gt^2$ in HW ex. 4, problem 2)
- correct sampling pdfs $f(x\,;\,\vec{\theta})$ e.g. for classification (like the mixture model in our toy-problem)

What if we do not know the right model?

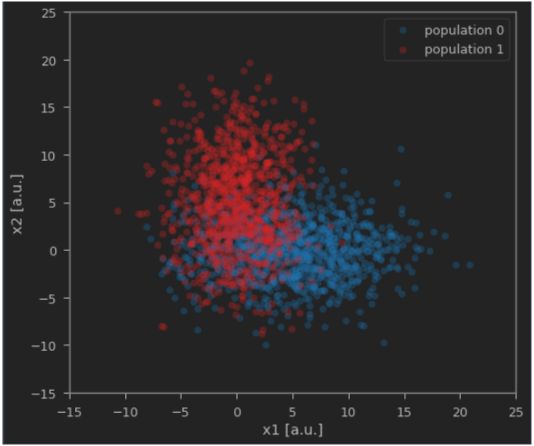
- true relationship $\mu(x) = g(x; \vec{\theta})$ e.g. in regression problems (like $s(t) = \frac{1}{2}gt^2$ in HW ex. 4, problem 2)
- correct sampling pdfs $f(x\,;\,\vec{\theta})$ e.g. for classification (like the mixture model in our toy-problem)

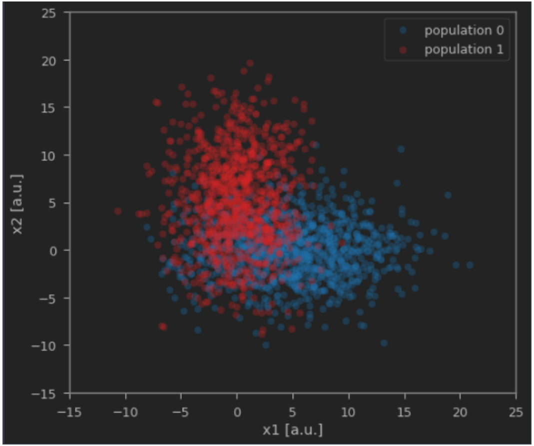
One can still perform inference and learn from the data!

- use flexible, *non-parametric* models and fit!
- re-formulate the problem in a clever way.

Example: consider data generated from a **bi-variate** gaussian mixture with 2 components.

Pretend we do not know the true data-generating process (i.e. the gaussians!)





Each data point comes with two observables: $\vec{x} = (x_1,\, x_2)$.

Goal: Classify data points depending on $\vec{c}$ into class 0 (blue) or class 1 (red) but without using the correct pdfs.

Goal: Calculate an estimate of $p_1(\vec{x}) \equiv p(c = 1;\, \vec{x})$.

Solution: Logistic regression - or more complex neural networks (or decision tree forests etc...).

Reminder: The multi-variate normal distribution

$$f\left(\vec{x}\right) = \left(\frac{1}{2\pi}\right)^{d/2} \left|\Sigma\right|^{-1/2} \exp\left(-\frac{1}{2}\left(\vec{x} - \vec{\mu}\right)\Sigma^{-1}\left(\vec{x} - \vec{\mu}\right)^{T}\right)$$

for $d = 2$ the covariance matrix Σ can be expressed as

$$\Sigma = \begin{vmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{vmatrix}$$

and $0 < \rho < 1$ measured the amount of correlation between the components x_1 and x_2.

(and now let's forget about it again ...)

Assume we have a **training dataset**, which consists of samples $\vec{x_i}$ with known/observed labels c_i ($c_i = 0$ or $c_i = 1$).

One reasonable statistical model would be the *tossing a biased coin*, which can be described with the **bernoulli pmf** for the random variable c.

$$c_i \sim \mathrm{bernoulli}(p_1(\vec{x_i})) \tag{16}$$

$$p(c) = p_1^c (1 - p_1)^{1-c}, \qquad c \in \{0, 1\} \tag{17}$$

where $0 \leq p_1(\vec{x}) \leq 1$ is the probability for class 1 (success) with some dependence on the values for $\vec{x} = (x_1, x_2)$.

Now we need a parameterization of $p_1(\vec{x})$.

A simple linear relationship $p_1(\vec{x}) \equiv g(\vec{x}) = \vec{\omega} \cdot \vec{x} + b$ won't work, since $g(\vec{x})$ ranges (in principle) from $-\infty$ to $+\infty$. But probabilities are bounded by 0 and 1.

Now we need a parameterization of $p_1(\vec{x})$.

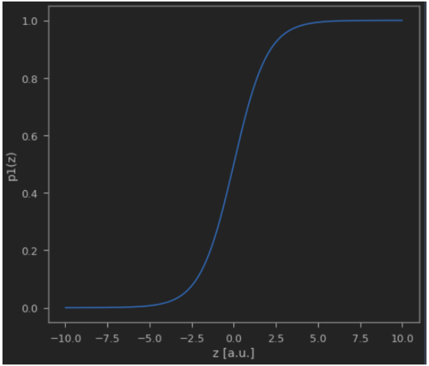
A simple linear relationship $p_1(\vec{x}) \equiv g(\vec{x}) = \vec{\omega} \cdot \vec{x} + b$ won't work, since $g(\vec{x})$ ranges (in principle) from $-\infty$ to $+\infty$. But probabilities are bounded by 0 and 1.

sigmoid function to the rescue! It maps from $(-\infty, +\infty)$ to $(0, 1)$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{18}$$

$$p_1(\vec{x}) = \sigma(g(\vec{x})) \tag{19}$$

$$= \sigma(\vec{\omega} \cdot \vec{x} + b) \tag{20}$$



What does this mean?

$$\vec{\omega} \cdot \vec{x} + b = \log\left(\frac{p_1}{1 - p_1}\right) \tag{21}$$

$$= \log\left(\frac{p_1(\vec{x})}{p_0(\vec{x})}\right) \tag{22}$$

Implicit Assumption: log-odds are linear in $\vec{x}$. This could easily be relaxed (this will yield more complex ML models).

What does this mean?

$$\vec{\omega} \cdot \vec{x} + b = \log \left(\frac{p_1}{1 - p_1} \right) \tag{21}$$

$$= \log \left(\frac{p_1(\vec{x})}{p_0(\vec{x})} \right) \tag{22}$$

Implicit Assumption: log-odds are linear in $\vec{x}$. This could easily be relaxed (this will yield more complex ML models).

Staying with the linear log-odds (logistic regression) gives 3 free parameters: slopes $\vec{\omega} = (\omega_1, \omega_2)$ and intercept b.

Can fit parameters using **maximum likelihood**!

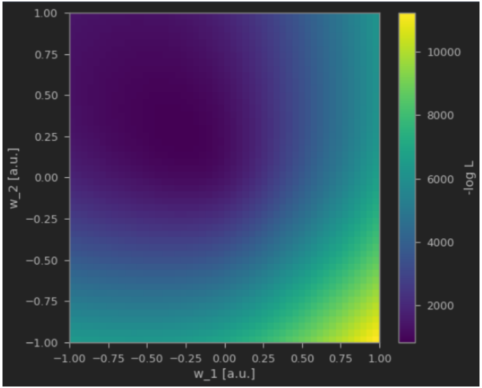
$$\log L(\vec{\omega}, b \,|\, \{\vec{x}_i,\ c_i\}) = \sum_{i=0}^{N} \{c_i \log[\sigma(g(\vec{x}, \vec{\omega}, b))] + (1 - c_i) \log[1 - \sigma(g(\vec{x}, \vec{\omega}, b))]\} \tag{23}$$

Can fit parameters using **maximum likelihood**!

$$\log L(\vec{\omega}, b \,|\, \{\vec{x}_i,\ c_i\}) = \sum_{i=0}^{N} \{c_i \log[\sigma(g(\vec{x}, \vec{\omega}, b))] + (1 - c_i) \log[1 - \sigma(g(\vec{x}, \vec{\omega}, b))]\} \tag{23}$$

```python
def neg_logl(pars, samples, labels):
    weights = pars[:2]
    b = pars[2]

    # compute z vector from all our samples
    zs = np.dot(weights, samples.T) + b

    # now run through sigmoid
    ps = 1./ (1 + np.exp(-zs))

    # and now calculate the neg_logl
    ll = np.sum(bernoulli.logpmf(labels, ps))
    return -ll
```

(for this plot: $b = 0$)

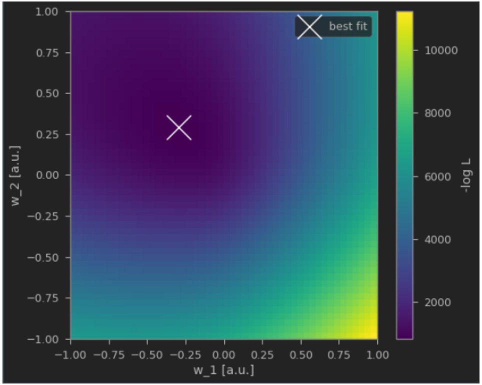


numerical minimization in (ω_1, ω_2, b) using scipy
(real problems: gradient descent)

```python
# now do the actual optimization!

from scipy.optimize import minimize

starting_slopes = np.array([-1., 1])
starting_intercepts = np.array([0.0])
pars_starting_point = np.concatenate([starting_slopes, starting_intercepts])

obj = lambda x: neg_logl(x, combined_samples, combined_labels)
r = minimize(obj, pars_starting_point, method='Powell')

print(r)
```

```
   direc: array([[ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00],
       [ 0.00000000e+00,  1.00000000e+00,  0.00000000e+00],
       [ 1.22671646e-02, -7.54470667e-04, -1.87984443e-02]])
     fun: 848.1590555566945
 message: 'Optimization terminated successfully.'
    nfev: 176
     nit: 5
  status: 0
 success: True
       x: array([-0.29170508,  0.2887275 , -0.02328438])
```

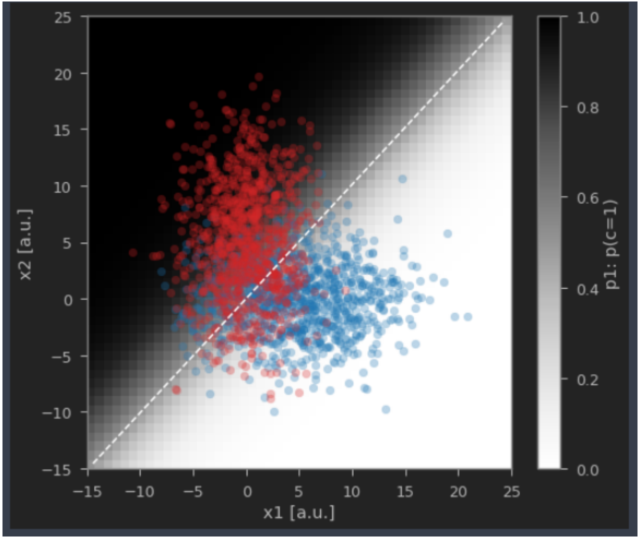
(for this plot: $b = 0$)



Let's **predict**! (assuming the best fit $\hat{\vec{\omega}}$, $\hat{b}$

```
def get_p(x, pars):
    weights = pars[:2]
    bs = pars[2:]
    z = np.dot(weights, x.T)
    p = 1./ (1 + np.exp(-z))
    return p
```

```
from matplotlib.colors import Normalize
edges = np.linspace(-15, 25, 51)
centers = 0.5*(edges[:-1] + edges[1:])

xv, yv = np.meshgrid(centers, centers, sparse=False, indexing='xy')
pos = np.stack([xv.flatten(), yv.flatten()], axis=1)
p1s = get_p(pos, best_pars)

fig, ax = plt.subplots()
h = ax.hist2d(xv.flatten(), yv.flatten(), bins=[edges]*2, weights = p1s.flatten(), norm=Normaliz
ax.set_xlabel("x1 [a.u.]")
ax.set_ylabel("x2 [a.u.]")

cbar = fig.colorbar(h[3], ax=ax)
cbar.set_label("p1: p(c=1)")

add_scatter_points(c0_samples, label="population 0", color="tab:blue")
add_scatter_points(c1_samples, label="population 1", color="tab:red")

ax.contour(centers, centers, p1s.reshape(len(centers), len(centers)), [0.5], colors=['white'], l

plt.show()
```



white dashed line: estimated that $p_0 = p_1 = 0.5$.

Summary

Achieved decent estimates of the **class-membersip probabilities** without knowledge of the data generating process (underlying process)!

For more complex problems: increase flexibility of $g(\vec{x})$ - introducing more parameters.

Neural networks (and other techniques) are natural extensions of this example. Sometimes involving several million parameters!

Eventually it turns into this:
:-)