Untitled

August 12, 2019

[1]:
```
# src: https://towardsdatascience.com/
↪a-gentle-introduction-to-maximum-likelihood-estimation-9fbff27ea12f
```

[2]:
```
# MLE in a nutshell helps us answer this question:
# Which are the best parameters/coefficients for my model?

# Bayes Theorem:
# P(y) = P(y) x P() / P(y)
# posterior = likelihood x prior / evidence

# The distinction between probability and likelihood is
# fundamentally important:
# Probability attaches to possible results;
# Likelihood attaches to hypotheses.

# MLE is Frequentist, but can be motivated from a
# Bayesian perspective:
# 1. Frequentists can claim MLE because its a point-wise
# estimate (not a distribution) and it assumes no prior
# distribution (technically, uninformed or uniform).
# 2. Also, MLEs do not give the 95% probability region for
# the true parameter value.
# 3. However, MLE is a special form of MAP, and uses the
# concept of likelihood, which is central to the
# Bayesian philosophy.
```

[3]:
```
# Four major steps in applying MLE:
# 1. Define the likelihood, ensuring youre using the correct
# distribution for your regression or classification problem.
# 2. Take the natural log and reduce the product function to
# a sum function.
# 3. Maximize  or minimize the negative of  the objective function.
# 4. Verify that uniform priors are a safe assumption!
# Otherwise, you could attribute the data to a generating function
# or model of the world that fails the Law of Parsimony.
```

[4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import seaborn as sns

import numpy as np
import pandas as pd

import scipy.stats as stats
from scipy.optimize import minimize

import statsmodels.api as sm
from statsmodels.base.model import GenericLikelihoodModel

import pymc3 as pm3
import numdifftools as ndt
```

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS
functions.
```

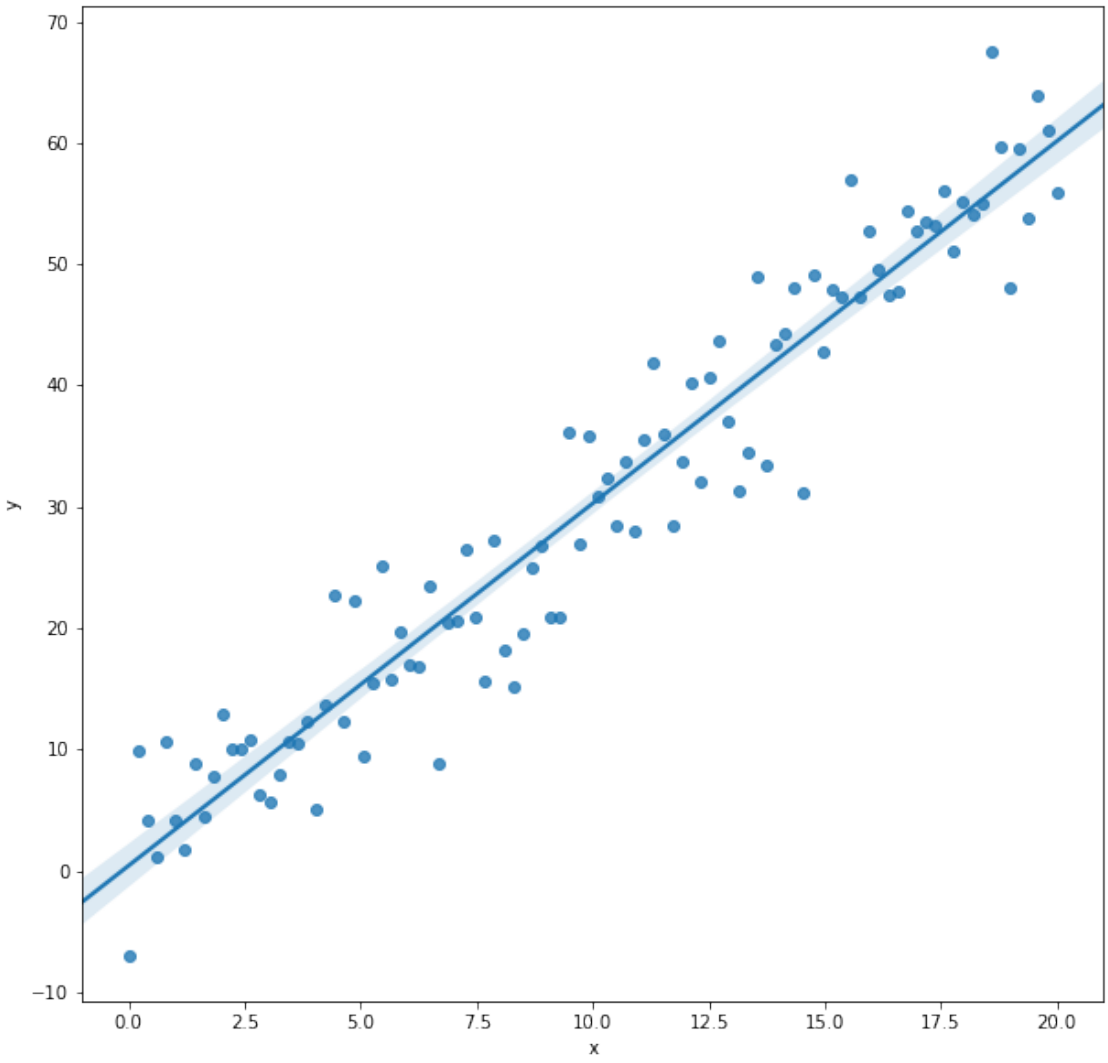
[5]:
```
# Generate data on linspace

N = 100

x = np.linspace(0, 20, N)

# Normally distributed errors
eps = np.random.normal(loc=0.0, scale=5.0, size=N)
```

[6]:
```
# Truth function

y = 3 * x + eps
```

[7]:
```
df = pd.DataFrame( {'x': x, 'y': y} )
df['const'] = 1

display(df.head(10))
```

	x	y	const
0	0.000000	-6.968115	1
1	0.202020	9.947630	1
2	0.404040	4.205770	1
3	0.606061	1.180839	1
4	0.808081	10.604320	1
5	1.010101	4.231410	1
6	1.212121	1.775587	1
7	1.414141	8.873452	1
8	1.616162	4.469828	1
9	1.818182	7.828227	1

[8]:
```
fig = plt.figure(figsize=(10, 10))
sns.regplot(df.x, df.y); plt.show()
```



[9]:
```
# Modeling OLS - calculate the best coefficients and
# log-likelihood as a benchmark

X = df[ ['const', 'x'] ]

sm.OLS(y, X).fit().summary()

# display(df)
```

[9]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.926
Model:                            OLS   Adj. R-squared:                  0.926
Method:                 Least Squares   F-statistic:                     1235.
Date:                Mon, 12 Aug 2019   Prob (F-statistic):           2.41e-57
Time:                        12:34:38   Log-Likelihood:                -300.87
No. Observations:                 100   AIC:                             605.7
Df Residuals:                      98   BIC:                             611.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4576      0.983      0.465      0.643      -1.493       2.409
x              2.9841      0.085     35.137      0.000       2.816       3.153
==============================================================================
Omnibus:                        0.424   Durbin-Watson:                   2.250
Prob(Omnibus):                  0.809   Jarque-Bera (JB):                0.454
Skew:                          -0.151   Prob(JB):                        0.797
Kurtosis:                       2.865   Cond. No.                         23.1
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[10]:
```
# Maximize LL to solve for Optimal Coefficienst

# Use custom functions to see if we can calculate the same
# OLS results using MLE methods

def MLERegression(params):
    intercept, beta, sd = params[0], params[1], params[2]
    yhat = intercept + beta * x  # predictions


    # flip the bayessian question
    # compute PDF of observed values normally distributed around mean (yhay)
    # with standard deviation of sd
    negLL = -np.sum( stats.norm.logpdf(y, loc=yhat, scale=sd) )

    return negLL
```

[11]:
```
# Now that we have a cost function, let's initialize and
# minimize it

# start with some random coefficient guesses and optimize
```

```
guess = np.array([5, 5, 2])

results = minimize(MLERegression, guess, method='Nelder-Mead', options={'disp':␣
 ↪True})
```

```
Optimization terminated successfully.
         Current function value: 300.874991
         Iterations: 119
         Function evaluations: 216
```

[12]:
```
results
```

[12]:
```
 final_simplex: (array([[0.45762622, 2.98408389, 4.90283996],
       [0.4576888 , 2.98408153, 4.90283767],
       [0.45769975, 2.98407696, 4.90280701],
       [0.45760993, 2.98408902, 4.90282974]]), array([300.87499129, 300.8749913
, 300.8749913 , 300.8749913 ]))
           fun: 300.87499129425197
       message: 'Optimization terminated successfully.'
          nfev: 216
           nit: 119
        status: 0
       success: True
             x: array([0.45762622, 2.98408389, 4.90283996])
```

[13]:
```
# drop results into df and round to match statsmodels

resultsdf = pd.DataFrame({'coef':results['x']})
resultsdf.index=['constant','x','sigma']
np.round(resultsdf.head(2), 4)

display(resultsdf)  # note: numbers almost match the OLS model!
```

```
              coef
constant  0.457626
x         2.984084
sigma     4.902840
```

[14]:
```
# Is using MLE to find our coefficients as robust?

# YES!

# 1. MLE is consistent with OLS.
# 2. With infinite data, it will estimate the optimal ,
# and approximate it well for small but robust datasets.
# 3. MLE is efficient; no consistent estimator has lower
```

```
# asymptotic mean squared error than MLE.
```

[15]:
```
# Why use MLE instead of OLS

# 1. MLE is generalizable for regression and classification!
# 2. MLE is efficient; no consistent estimator has lower
# asymptotic error than MLE if youre using the right distribution.

# We can think of MLE as a modular way of fitting models by
# optimizing a probabilistic cost function!
```