Programming and Data Science for Biology (PDSB)

Session 10
Spring 2018

Today’s topics

1. Review of last assignment (ML)
2. Introduction to Model-based statistical inference.
3. Hands-on optimization
4. Hands-on Bayesian inference

Upcoming topics relate to projects

1. **Today:** Bayesian and frequentist statistics (scipy & pymc3)
2. **Next week:** Plotting methods: matplotlib, toyplot, folium, bokeh
3. **Two weeks:** Genomic analysis tools.
4. Then we begin presentations...

Last week's assignment:
Applying ML to a *records* dataset

Assignment

We created a simple library to query the GBIF API to generate a Pandas DataFrame full of occurrence records. Each record has >100 columns of data, including the taxonomy, location, and collection information.

Your task was to try to mine this data in some interesting and informative way while also learning to apply a machine learning algorithm from scikit-learn.

Observations

1. Simple validation methods are often insufficient to know whether your algorithm is working well. For example, 50% prediction rate may not be that good if your dataset is composed mostly of NaN values and it is matching NaN records together.

2. Filtering data is important, as is selecting informative columns and ensuring they are converted to the proper datatype (e.g., int, float, str).

Example

See Notebooks/nb-10.1-ML-applied.ipynb

Here I apply a GaussianNB model to latitude, longitude, and elevation for the *Bombus* data set to try to predict species identifications. I also did the following:

1. Filtered to remove NaN records
2. Filtered to keep only records for species that were observed >500 times.
3. Examined the confusion matrix to examine which species were confused.
4. Examined maps to see if my method was making sense.

Model based statistical inference

What is model based statistical inference?

We make *assumptions* when testing for statistical patterns in data. These assumptions have to do with the probability of observing certain outcomes.

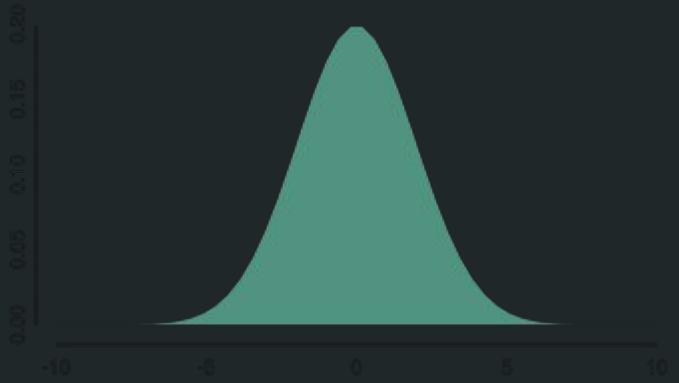
For example, to test if a coin is fair we would test whether heads vs tails deviates from a probability of 0.5. Performing this test thus requires us to first build a model of what we expect to observe.

What is model based statistical inference?

The central limit theorem underlies many of our statistical assumptions.

This states that when independent random variables are added, their properly normalized sum (e.g., mean) tends toward a normal distribution even if the original variables themselves are not normally distributed.

This key concept implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.



What is model based statistical inference?

This is readily apparent in a coin flip example. The mean of many **independent** coin flips will converge on the true probability of a given flip, and will be normally distributed.

We can easily test this with code:

The **binomial** distribution is used to draw binary values (Bernoulli trials) with a defined probability of success (1) or fail (0).

The binomial distribution ends up with the same mean and std as samples drawn directly from the normal distribution with those same parameters.

```
import numpy as np

# get 10000 random trials with p=0.5
arr = np.random.binomial(
     n=1, p=0.5, size=10000)

# calc mean and std
arr.mean(), arr.std())
(0.5, 0.5)


# get 10000 random normal values
arr = np.random.normal(
     loc=0.5, scale=0.5, size=10000)

# calc mean and std
arr.mean(), arr.std())
(0.5, 0.5)
```

What is model based statistical inference?

This was stated in your reading using the example of a coin flip.

In frequentist statistical methods the result (a probability statement) depends on the number of coin flips. We can only say how probable a flip is based on the data

In Bayesian statistical methods, the result (a probability statement) depends on our model (a Normal distribution) and our confidence is updated depending on how much data.

What is model based statistical inference?

We've modeled a **process** that we can described by a function (something we can code, like a coin flip) as a probability distribution.

To test how accurate our model is we simply need to ask how close it matches to a normal distribution.

How do we do that? With a likelihood function.

```
import numpy as np

# get 10000 random trials with p=0.5
arr = np.random.binomial(
     n=1, p=0.5, size=10000)

# calc mean and std
arr.mean(), arr.std())
(0.5, 0.5)


# get 10000 random normal values
arr = np.random.normal(
     loc=0.5, scale=0.5, size=10000)

# calc mean and std
arr.mean(), arr.std())
(0.5, 0.5)
```

Probability and Likelihood

Likelihood:
A likelihood function is a function of the parameters of a statistical model given data. In other words, it **returns** a value that depends on how well the data fit the model with the given parameters. It is typically used to infer the best set of parameters for a model.

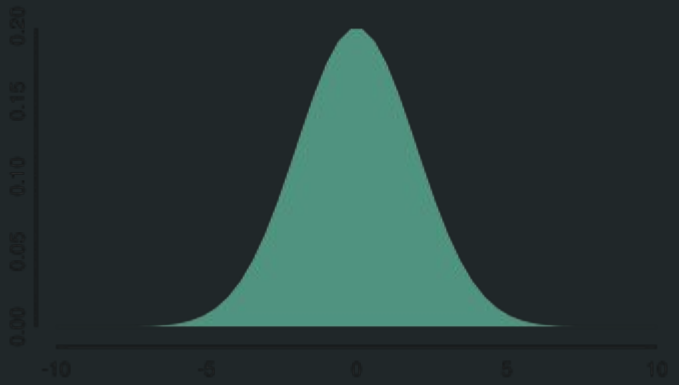
For example, to find the optimum parameter loc of the normal distribution that can best fit data drawn from a binomial distribution with p=0.75.

Probability and Likelihood

Some likelihood functions are very simple, such as the PDF of a distribution. It returns the likelihood for a value drawn from a distribution.

Other likelihood functions are more complex. For example, it could have parameters that fit a collection of statistically independent likelihood functions. In such a situation, the likelihood function factors into a product of individual likelihood functions.

See notebooks/nb-10.2-scipy-likelihood.ipynb



Probability and Likelihood

Likelihood:
A likelihood function is a function of the parameters of a statistical model given data. In other words, it **returns** a value that depends on how well the data fit the model with the given parameters. It is typically used to infer the best set of parameters for a model.

```
def coin_flip(p, nheads, ntails):
    ## calculate likelihood
    likelihood = p**(nheads) * (1-p)**(ntails)

    ## return negative likelihood
    return -likelihood


def coin_flip_log(p, nheads, ntails):
    ## calculate likelihood
    logp = nheads*np.log(p) + ntails*np.log(1.-p)

    ## return negative log-likelihood
    return -1*logp
```

Probability and Likelihood

Likelihood:

The product (i.e., multiplied product) of all independent probability events can be used to calculate a likelihood.

Or, the sum of the natural log of all independent probability events can be used to calculate a likelihood.

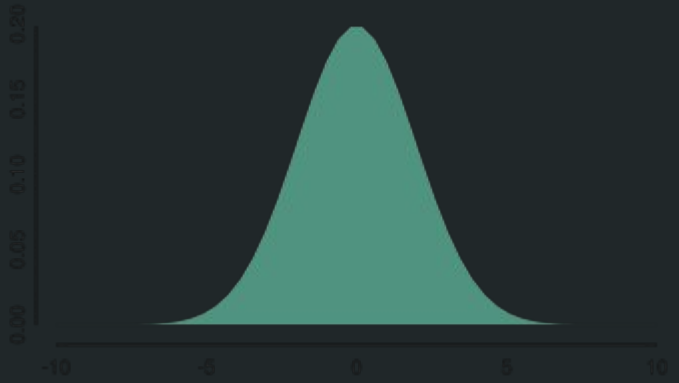
The latter is typically easier to work with.

```
def coin_flip(p, nheads, ntails):
    ## calculate likelihood
    likelihood = p**(nheads) * (1-p)**(ntails)

    ## return negative likelihood
    return -likelihood


def coin_flip_log(p, nheads, ntails):
    ## calculate likelihood
    logp = nheads*np.log(p) + ntails*np.log(1.-p)

    ## return negative log-likelihood
    return -1*logp
```

Probability and Likelihood

When fitting more generic models, such as how competition for light among plants affects their rate of growth, it is less clear what *distribution* should fit our data most closely.

Instead, we can compare our model fit to *observations,* wherein we expect that the **deviation** (residual error) between model estimates and observations are random variates.



Probability and Likelihood

This idea, that the distribution of the residual errors of a model should be distributed in a certain way is the basis for most statistical methods.

Least squares regression aims to minimize the sum of squared errors between a model fit and observed data.

Maximum likelihood is identical to ordinary least squares when errors are normally distributed.

See notebooks/nb-10.3-scipy-likelihood.ipynb

Probabilistic programming

What is model based statistical inference?

We make *assumptions* when testing for statistical patterns in data. These assumptions have to do with the probability of observing certain outcomes.

What if we could completely describe all aspects of our statistical inference problems in terms of probability distributions?

What is model based statistical inference?

Probabilistic programming is an emerging paradigm in statistical learning, of which Bayesian modeling is an important sub-discipline.

The signature characteristics of **probabilistic programming**–specifying variables as probability distributions and conditioning variables on other variables and on observations–makes it a powerful tool for building models in a variety of settings, and over a range of model complexity.

What is model based statistical inference?

Probabilistic programming is an emerging paradigm in statistical learning, of which Bayesian modeling is an important sub-discipline.

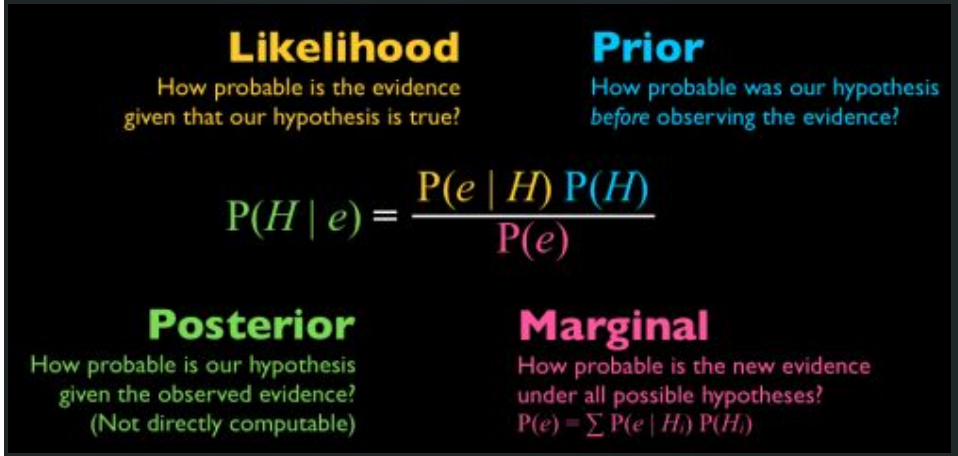
Probabilistic programming is particularly well suited for Bayesian inference since we can describe priors for random variates using probability distributions.

The philosophy of Bayesian Inference

We will not delve into Bayes' theorem too deeply here, since it was in your reading, and because we do not need to memorize the exact equation in order to understand how it works and implement it.

The main idea is important:

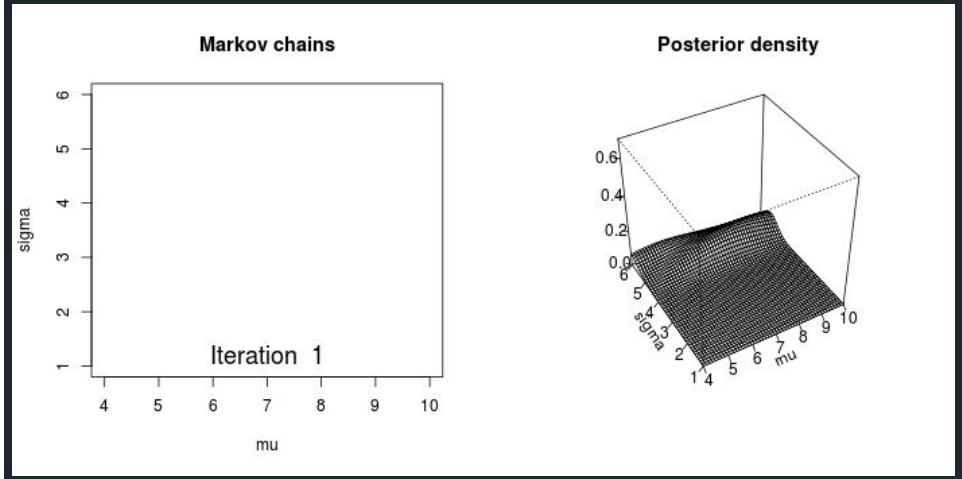
+ We define priors on our beliefs
+ We define a likelihood function that is maximized when data fits our model.
+ We sample data using MCMC (a simulation optimization method) to search over all possible hypotheses (*marginalize).*
+ We calculate the posterior (updated beliefs) as likelihood x priors.



The philosophy of Bayesian Inference

Markov chain Monte Carlo:

Marginalize over probabilities using *importance sampling*. The amount of time spent in any given part of parameter space is equal to its probability, under certain assumptions.



The philosophy of Bayesian Inference

In Bayesian inference we update our beliefs about an outcome; rarely can we be absolutely sure unless we rule out all other alternatives.

Although Bayesian inference can be used for model testing or model comparison, it is philosophically favored to describe results in terms of **posterior distributions** of parameter values, and **posterior predictive checks,** where we ask how well the model parameters can generate data that looks like our observed data.

In the latter sense it is similar to machine learning methods, and there is a lot of overlap between the two

Continued: Testing with simulations

Using simulated data

Learning statistics and statistical methods is hard, takes a long time, and the details are often over our head. The best way to **learn** and **apply** statistical methods is to:

1. Look for applications where others have analyzed similar types of data, or answered similar types of questions, and try to understand their implementations.

2. Simulate data to look like your results under a known generative model, to **validate** that the method is working how you think it is.

Simulating data

1. The numpy random library
2. Wrap as a pandas dataframe

You've seen this many times now, including in our examples today. Your assignments this week include using numpy to generate data that is fitting for a model in pymc3, and then fitting the model to infer the parameters you used to simulate the data.

Let’s work through pymc3 model fitting to simulated data

Implementations of examples from the pymc3 tutorial.

notebooks/nb-10.4-pymc3-regression.ipynb
notebooks/nb-10.5-pymc3-volatility.ipynb
notebooks/nb-10.6-pymc3-switchpoint.ipynb

Resources:

See the pymc3 github page. Links to several free books on Bayesian statistics that implement all exercises in pymc3.

https://github.com/pymc-devs/pymc3

https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

Assignments notebook is in assignments/
See assigned reading in syllabus:

Assignment: Link to Session 10 repo
Readings: See syllabus
Collaborate: Work together in this gitter chatroom