2018

Jupyter: Intro to Data Science - Lecture 4 Demonstrating Key Theorems

Grant Long
CUNY City College

NYC Tech-in-Residence Corps

Data Dive, Part I: Demonstrating Key Theorems

In [ ]:
```
%matplotlib notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import random as rd
```

Law of Large Numbers

The average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.

- In other words, the more data we are able to pull, the better we are able to estimate parameters and derive insights about the population.

[Wikipedia (https://en.wikipedia.org/wiki/Law_of_large_numbers)](https://en.wikipedia.org/wiki/Law_of_large_numbers)

In [ ]:
```
fig = plt.figure(2)
ax = fig.add_subplot(111)
fig.show()
fig.canvas.draw()

plt.ion()

total_sample = []
n = 100 # number of samples for each mean
k = 100 # number of iterations
mu = 0
sigma = 1

for i in range(k):

    sample = rd.normal(mu, sigma, size=n)

    total_sample += list(sample)

    plt.cla()
    ax.set_title('iteration %i, no. means=%i, mu=%0.7f' % (i+1, len(tota
l_sample), np.mean(total_sample)))
    ax.hist(total_sample, density=True, alpha=0.9, color='blue')
    plt.axvline(x=np.mean(total_sample), alpha=0.9, color='red')
    fig.canvas.draw()

plt.ioff()
```

In [ ]:

Central Limit Theorem

When independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a "bell curve") even if the original variables themselves are not normally distributed.

- The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.

[Wikipedia (https://en.wikipedia.org/wiki/Central_limit_theorem)](https://en.wikipedia.org/wiki/Central_limit_theorem)

In [ ]:

```
fig = plt.figure(1)
ax = fig.add_subplot(111)
fig.show()
fig.canvas.draw()

plt.ion()

sample_means = []
n = 100 # number of samples for each mean
m = 100 # number of means for each iteration
k = 100 # number of iterations
mu = 0
sigma = 1

for i in range(k):

    sample = rd.normal(mu, sigma, size=[n, m])

    sample_means += list(np.mean(sample, axis=1))

    plt.cla()
    ax.set_title('iteration %i, no. means=%i, mu=%0.7f' % (i+1, len(samp
le_means), np.mean(sample_means)))
    ax.hist(sample_means, density=True, alpha=0.9, color='blue')
    plt.axvline(x=np.mean(sample_means), alpha=0.9, color='red')

    fig.canvas.draw()

plt.ioff()
```

In [ ]:

In [ ]:

In [ ]: