Basic Statistics with Python

Aris Budi Wibowo

PyCon APAC 2016

COEX Seoul, August 2016

Profile

- Aris Budi Wibowo
- Mathematics student of Bandung Institute of Technology
- Currently focusing on sport analytics
- http://github.com/arisbw

Introduction

NumPy

SciPy.org

Sponsored By ENTHOUGHT

scikit learn

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Getting started

Tools in this presentation

- Python 3.5.1
- Environment: Anaconda
- Libraries:
  - Numpy
  - Scipy
  - Pandas
  - Scikit-learn
- Spyder

Data Types

- Categorical:
  - Boolean (0/1)
  - Nominal (more than 2 categories)
  - Ordinal (ordered, have logical sequence)

- Numerical
  - Numerical Continuous
    - Ex: Height of students
  - Numerical Discrete
    - Ex: Number of Children

Descriptive Statistics

Let’s say I want to squeeze some important information of data

- What is the general information we really need?
- Is it possible to describe the data in some simplified ways?

Types of Descriptive Statistics

- Center
  - Mean
  - Median
  - Mode
- Spread
  - Variance
  - Std. Deviation
  - Range
  - Percentiles
- Form
  - Skewness
  - Kurtosis

Mean

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Variance and Standard Deviation

$$var = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}$$

$$s = \sqrt{var}$$

Variance and Standard Deviation of Sample

$$var = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1}$$

```
import numpy as np
import pandas as pd
import scipy as sp

np.random.seed(0)

s = sp.randn(10)

print(s)

>> array([ 1.8831507 , -1.34775906, -1.270485  , 0.96939671, -
1.17312341,
1.94362119, -0.41361898, -0.74745481, 1.92294203, 1.48051479])
```

```
print("Mean: {0:8.6f}".format(s.mean()))
print("Variance: {0:8.6f}".format(s.var()))
print("Standard Deviation: {0:8.6f}".format(s.std()))
print("Min: {0:8.6f}".format(s.min()))
print("Max: {0:8.6f}".format(s.max()))
print("Range: {0:8.6f}".format(np.ptp(s)))
```

```
>> Mean: 0.324718
>> Variance: 1.863543
>> Standard Deviation: 1.365117
>> Min: -1.347759
>> Max: 1.943621
>> Range: 3.291380
```

```
#other ways
print("Mean : {0:8.6f}".format(sp.mean(s)))
print("Variance : {0:8.6f}".format(sp.var(s)))
print("Standard deviation : {0:8.6f}".format(sp.std(s)))

sp.stats.describe(s)

>> DescribeResult(nobs=10, minmax=(-1.347759061142464,
1.9436211856492926), mean=0.3247184151835216,
variance=2.070603770052156, skewness=0.007437318627555,
kurtosis=-1.7848125676623792)
```

```
#using pandas
data = pd.DataFrame(data={'s': s})
data.describe()
```

```
                s
count  10.000000
mean    0.324718
std     1.438959
min    -1.347759
25%    -1.066706
50%     0.277889
75%     1.782492
max     1.943621
```

Probability Distributions

Now after we know the descriptive information, we want to learn more about our data

- What are our characteristics of our data?
- Is our data really that unique?

Probability Mass Function (PMF)

- A discrete function which defines the probability to obtain a given number of events in an experiment or observation.

Let X be a discrete random variable with range $R_X = \{x_1, x_2, x_3, \dots\}$ (finite or countably infinite). The function

$$P_X(x_k) = \mathrm{P}(X = x_k), \text{ for } k = 1, 2, 3, \dots,$$

is called the *probability mass function (PMF)* of X.

Probability Density Function (PDF)

- A continuous function which defines the likelihood to find a random variable with a certain value. The probability to find the value of the random variable in a given interval is the integral of the probability density function over that interval.
- The general function varies for each of continuous distributions.

Cumulative Distribution Function

- The probability to find a random variable with a value lower than the given one.

The cumulative distribution function (CDF) of random variable X is defined as

$$F_X(x) = P(X \leq x), \text{ for all } x \in \mathbb{R}.$$

Types of Distribution

- Discrete
  - Bernoulli
  - **Binomial**
  - Poisson
- Continuous
  - **Norm**
  - Student-T
  - Chi-Squared
  - F (Fischer)

Discrete Distributions

- Binomial Distribution:
  - In N-trials, how many will succeed?
  - Example:
    - Flipping coin
    - Price of a certain stock
    - Match on Tinder

- PMF

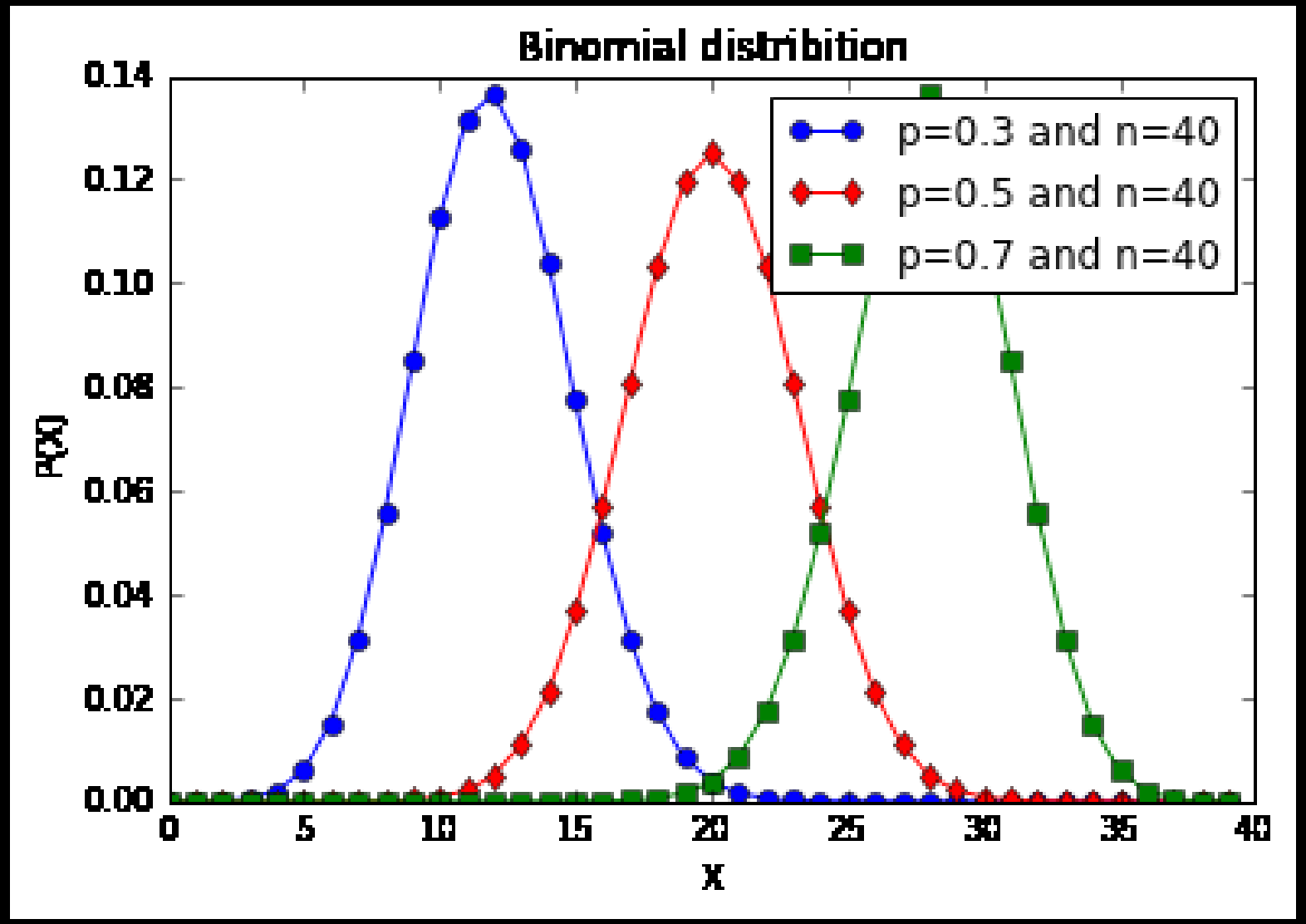
$$f(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- CDF

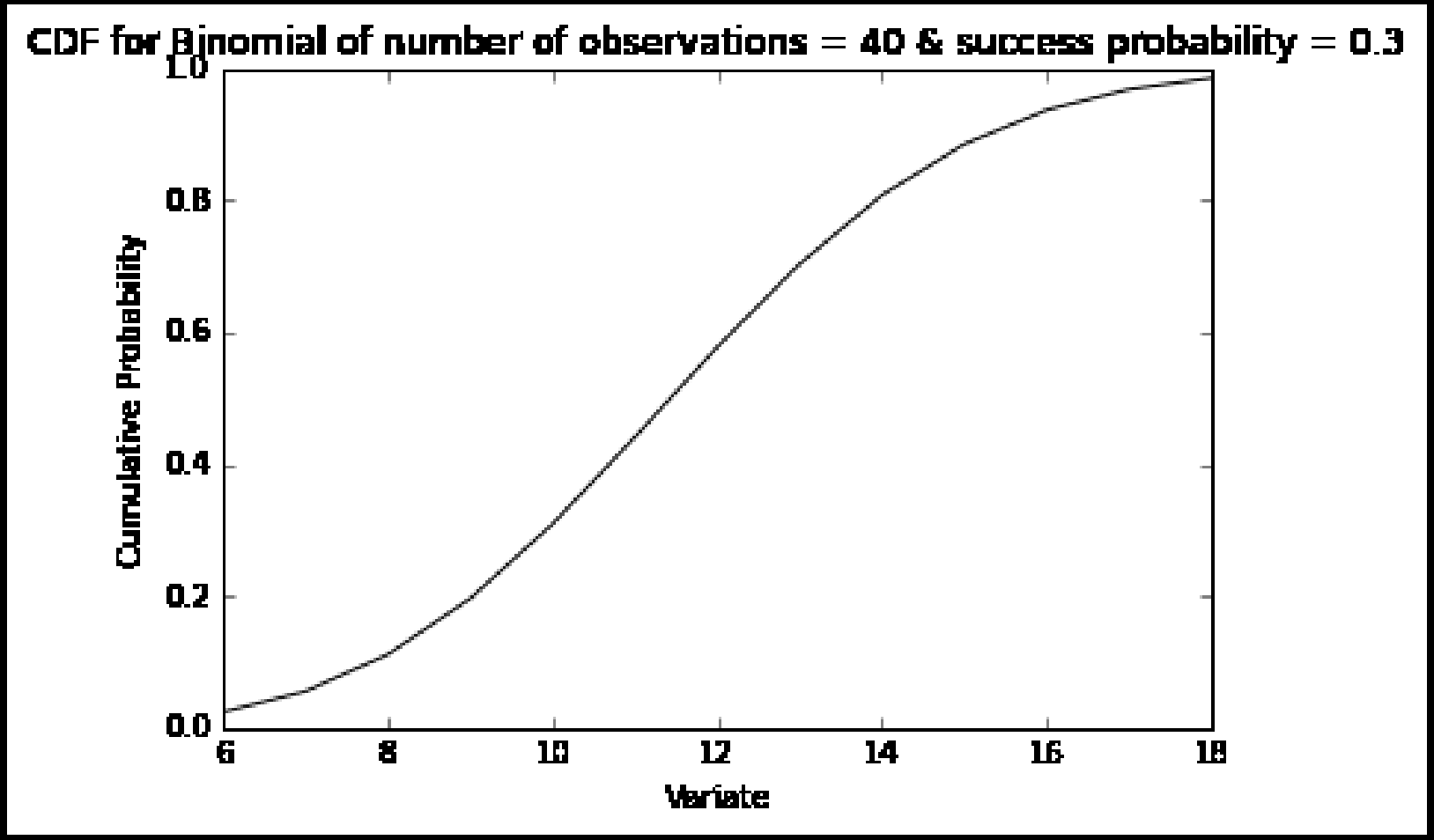
$$F(k;n,p) = \Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$$

```
bd1 = sp.stats.binom(40, 0.3)
bd2 = sp.stats.binom(40, 0.5)
bd3 = sp.stats.binom(40, 0.7)
k = np.arange(40)
plot(k, bd1.pmf(k), 'o-b')
hold(True)
plot(k, bd2.pmf(k), 'd-r')
plot(k, bd3.pmf(k), 's-g')
title('Binomial distribition')
legend(['p=0.3 and n=40', 'p=0.5 and n=40', 'p=0.7 and n=40'])
xlabel('X')
ylabel('P(X)')
```


Binomial distribition
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
P(X)
0
5
10
15
20
25
30
35
40
X
p=0.3 and n=40
p=0.5 and n=40
p=0.7 and n=40

```
def binom_cdf(n, p):
        x = np.arange(sp.stats.binom.ppf(0.01, n, p),
                      sp.stats.binom.ppf(0.99, n, p))
        # CDF at these values
        y = sp.stats.binom.cdf(x, n, p)
        plt.plot(x,y, color="black")
        plt.xlabel("Variate")
        plt.ylabel("Cumulative Probability")
        plt.title("CDF for Binomial of observation = {0} & success
                  probability = {1}".format( n, p))
        plt.draw()

binom_cdf(40,0.3)
```


CDF for Binomial of number of observations = 40 & success probability = 0.3
Cumulative Probability
1.0
0.8
0.6
0.4
0.2
0.0
6
8
10
12
14
16
18
Variate

Continuous Distributions

- Normal Distribution
  - Commonly used in real life
  - Most distributions can be transformed to normal distribution
  - Example:
    - Height of people
    - Components produced by machines
    - Test scores

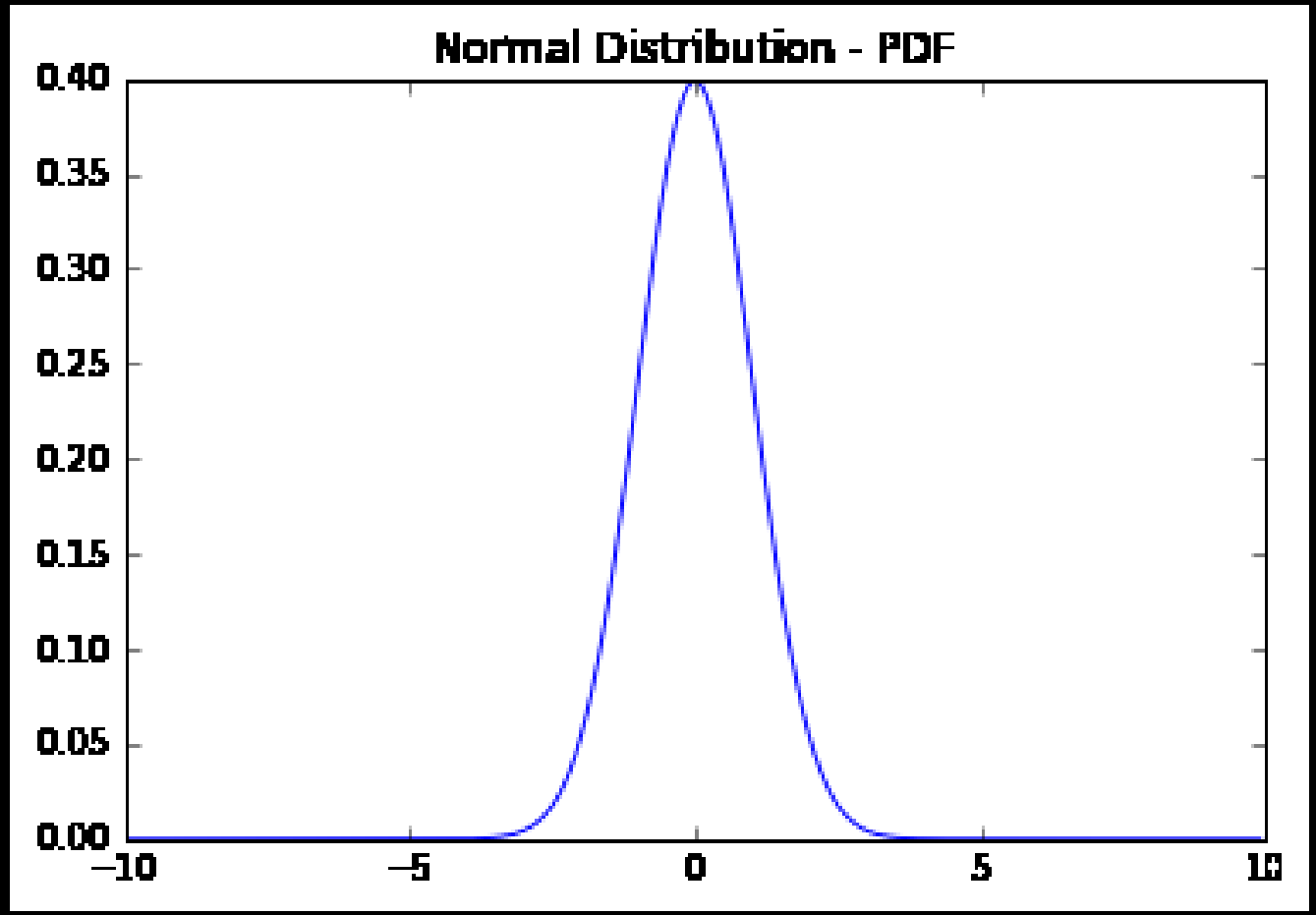
- PMF

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

- CDF

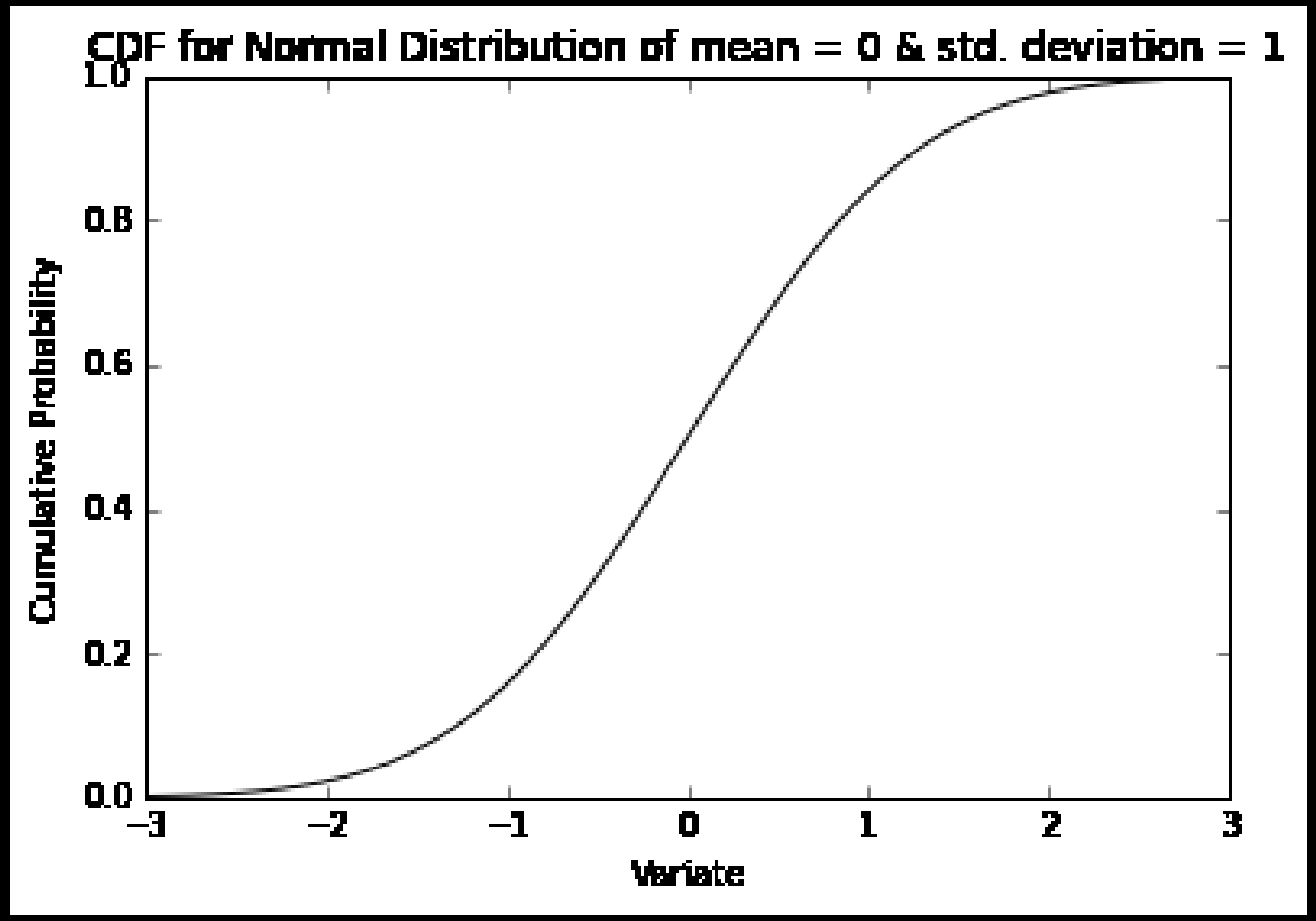
$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,dt$$

```
x = np.arange(-10,10,0.1)
n1 = sp.stats.norm(0,1)
plot(x,n1.pdf(x))
xlim([-10,10])
title('Normal Distribution - PDF')
```


Normal Distribution - PDF
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−10
−5
0
5
10

```
def norm_cdf(mean=0, std=1):
        x = sp.linspace(-3*std, 3*std, 50)
        y = sp.stats.norm.cdf(x, loc=mean, scale=std)
        plt.plot(x,y, color="black")
        plt.xlabel("Variate")
        plt.ylabel("Cumulative Probability")
        plt.title("CDF for Normal Distribution of mean = {0} & std.
                  deviation = {1}".format( mean, std))
        plt.draw()

norm_cdf()
```


CDF for Normal Distribution of mean = 0 & std. deviation = 1
Cumulative Probability
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3
Variate

Moment of Distributions

Definitions.

(1) $E(X^k)$ is the k^{th} **(theoretical) moment** of the distribution (**about the origin**), for $k = 1, 2, \ldots$

(2) $E\left[(X-\mu)^k\right]$ is the k^{th} **(theoretical) moment** of the distribution (**about the mean**), for $k = 1, 2, \ldots$

(3) $M_k = \frac{1}{n}\sum_{i=1}^{n} X_i^k$ is the k^{th} **sample moment**, for $k = 1, 2, \ldots$

(4) $M_k^* = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^k$ is the k^{th} **sample moment about the mean**, for $k = 1, 2, \ldots$

Moment of Distributions

- Moment 1: Mean
- Moment 2: Variance
- Moment 3: Skewness
- Moment 4: Kurtosis
- Higher degree of moment: ?

What's so 'moment' about 'moments' of a probability distribution?

44

According to the paper "First (?) Occurrence of Common Terms in Mathematical Statistics" by H.A. David, the first use of the word 'moment' in this situation was in a 1893 letter to *Nature* by Karl Pearson entitled "Asymmetrical Frequency Curves".

Neyman's 1938 *Biometrika* paper "A Historical Note on Karl Pearson's Deduction of the Moments of the Binomial" gives a good synopsis of the letter and Pearson's subsequent work on moments of the binomial distribution and the method of moments. It's a really good read. Hopefully you have access JSTOR for I don't have the time now to give a good summary of the paper (though I will this weekend). Though I will mention one piece that may give insight as to why the term 'moment' was used. From Neyman's paper:

> It [Pearson's memoir] deals primarily with methods of approximating continuous frequency curves by means of some processes involving the calculation of easy formulae. One of these formulae considered was the "point-binomial" or the "binomial with loaded ordinates". The formula
> differs from what to-day we call a binomial, viz. (4), only by a factor α, representing the area under the continuous curve which it is desired to fit.

This is what eventually led to the 'method of moments.' Neyman goes over the Pearson's derivation of the binomial moments in the above paper.

And from Pearson's letter:

> We shall now proceed to find the first four moments of the system of rectangles round GN. If the inertia of each rectangle might be considered as concentrated along its mid vertical, we should have for the s^{th} moment round NG, writing $d = c(1 + nq)$.

This hints at the fact that Pearson used the term 'moment' as an allusion to 'moment of inertia,' a term common in physics.

Getting to the Next Level

Data Imputation

- Replacing missing data with new values.
- Why?
  - In order to avoid dropping incomplete data that can lead to a worse model.

```
from sklearn.datasets import load_boston
from sklearn.preprocessing import Imputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import cross_val_score

dataset = load_boston()
rng = np.random.RandomState(0)
X_full, y_full = dataset.data, dataset.target
n_samples = X_full.shape[0]
n_features = X_full.shape[1]
```

```
# Estimate the score on the entire dataset, with no missing values
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_full, y_full).mean()
print("Score with the entire dataset = %.2f" % score)

# Add missing values in 75% of the lines
missing_rate = 0.75
n_missing_samples = np.floor(n_samples * missing_rate)
missing_samples = np.hstack((np.zeros(n_samples - n_missing_samples,
                                      dtype=np.bool),
                             np.ones(n_missing_samples, dtype=np.bool)))
rng.shuffle(missing_samples)
missing_features = rng.randint(0, n_features, n_missing_samples)
```

```
# Estimate the score without the lines containing missing values
X_filtered = X_full[~missing_samples, :]
y_filtered = y_full[~missing_samples]
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_filtered, y_filtered).mean()
print("Score without the samples containing missing values = %.2f" % score)

# Estimate the score after imputation of the missing values
X_missing = X_full.copy()
X_missing[np.where(missing_samples)[0], missing_features] = 0
y_missing = y_full.copy()
estimator = Pipeline([("imputer", Imputer(missing_values=0,
                                          strategy="mean", axis=0)),
                      ("forest", RandomForestRegressor(random_state=0,
                                                       n_estimators=100))])
score = cross_val_score(estimator, X_missing, y_missing).mean()
print("Score after imputation of the missing values = %.2f" % score)
```

```
Score with the entire dataset = 0.56
Score without the samples containing missing values = 0.48
Score after imputation of the missing values = 0.55
```

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam
dpkingma@uva.nl

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

References

- Haslwanter, T. (2016). An introduction to statistics with python. *Statistics and Computing*. doi:10.1007/978-3-319-28316-6
- Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization.*arXiv preprint arXiv:1412.6980*.
- tatistics) (2016). . In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Imputation_(statistics)
- Imputing missing values before building an estimator — scikit-learn 0.17.1 documentation. (2010). Retrieved August 12, 2016, from http://scikit-learn.org/stable/auto_examples/missing_values.html#example-missing-values-py
- Marsja, E. (2016, February 4). *Descriptive statistics using python - Erik Marsja*. Retrieved August 12, 2016, from Open Science, http://www.marsja.se/pandas-python-descriptive-statistics/
- Normal Distribution. Retrieved August 12, 2016, from Wikipedia, https://en.wikipedia.org/wiki/Normal_distribution
- Prasanth. (2011, February 28). *Simple statistics with SciPy*. Retrieved August 10, 2016, from https://oneau.wordpress.com/2011/02/28/simple-statistics-with-scipy/#probability-density-function-pdf-and-probability-mass-function-pmf
- State, T. P. (2016). *Method of moments*. Retrieved August 12, 2016, from https://onlinecourses.science.psu.edu/stat414/node/193
- What's so "moment" about "moments" of a probability distribution? (2016). Retrieved August 12, 2016, from http://stats.stackexchange.com/questions/17595/whats-so-moment-about-moments-of-a-probability-distribution

For full script:
https://github.com/arisbw/PyCon-APAC-2016/

Thank You!