2020-1-PL01-KA203-082197

Moduł 12
Metody statystyczne_L2

iBigWorld:
Innovations for Big Data in a Real World

Zespół UBB

Co-funded by the Erasmus+ Programme of the European Union

Elementy statystyki. Rachunek prawdopodobieństwa i dedukcja statystyczna

Prawdopodobieństwo i teoria informacji

Teoria prawdopodobieństwa - matematyczne podejście do przedstawiania zdań niepewnych. Ale prawdopodobieństwo to nie tylko abstrakcyjne pojęcie w świecie matematyki, prawdopodobieństwo jest wszędzie wokół nas i może być zabawne obliczanie prawdopodobieństwa wydarzeń w naszym życiu. W zastosowaniach sztucznej inteligencji wykorzystujemy teorię prawdopodobieństwa na dwa główne sposoby.

- Po pierwsze, prawa prawdopodobieństwa mówią nam, w jaki sposób systemy sztucznej inteligencji powinny wnioskować, więc projektujemy nasze algorytmy, aby obliczać lub aproksymować różne wyrażenia wyprowadzone za pomocą teorii prawdopodobieństwa.
- Po drugie, możemy wykorzystać prawdopodobieństwo i statystyki do teoretycznej analizy zachowania proponowanych systemów sztucznej inteligencji.

Podczas gdy teoria prawdopodobieństwa pozwala nam na dokonywanie niepewnych twierdzeń i wnioskowania w warunkach niepewności, teoria informacji pozwala nam określić ilościowo wielkość niepewności za pomocą prawdopodobieństwa.

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Narzędzia do analizy Big Data

R (język pisany przez i dla statystyków, z R to język pisany przez i dla statystyków, który posiada już praktycznie wszystkie niezbędne komponenty: pakiety do wizualizacji danych, pakiety do łączenia R z innymi językami, np. Java i hurtowniami danych; od lat rozwija się nawet jako osobna infrastruktura do budowy aplikacji webowych - tzw. R Spark, praca z R na Apache Hadoop staje się coraz łatwiejsza)

Python (znany i lubiany język używany do tworzenia aplikacji do gier i w zasadzie wszystkiego, co można zrobić z programowaniem; Jeśli chodzi o analizę. Interesują nas głównie moduły NumPy,moduły SciPy, Sci-kit Learn, Pandas lub NLTK; Python wydaje się być łatwiejszy i bardziej intuicyjny niż R i lepszy do zadań eksploracji tekstu).

Matlab (znany i nadal powszechnie używany język do analizy danych, analizy danych, statystyki i wizualizacji danych; często porównywany z R, chociaż prawdopodobnie ustąpi w nadchodzących latach jego wolny kolega),

Apache Mahout (Biblioteka Java do uczenia maszynowego – słabo znane i stosunkowo nowe narzędzie; podobno jeden z najlepszych jeśli chodzi o zadania związane z przetwarzaniem języka naturalnego; Przede wszystkim fakt, że Mahout jest częścią. Rodzina Apache, sprawia, że jest to idealne rozwiązanie do łączenia z Hadoop).

iBigWorld:2020-1-PL01-KA203-082197

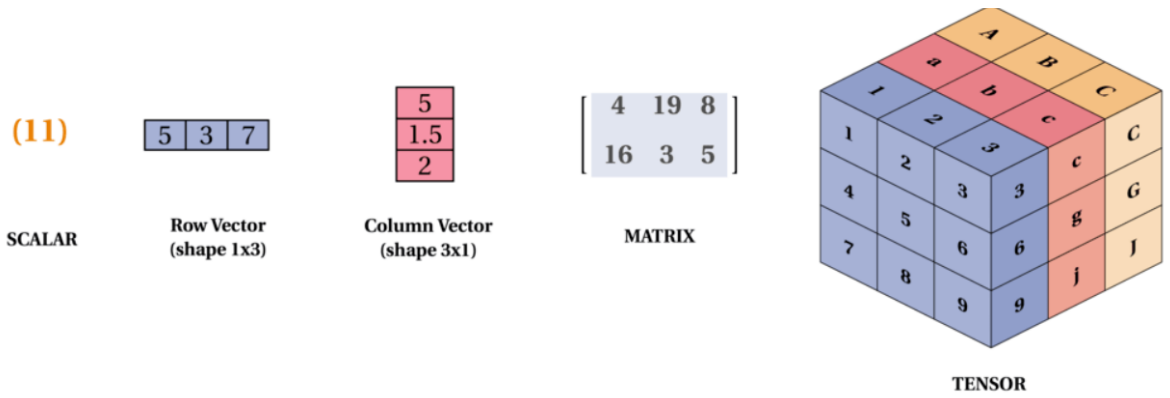
Co-funded by the Erasmus+ Programme of the European Union

Skalary, wektory, macierze i tensory

Skalary: to tylko jedna liczba. Na przykład temperatura, która jest oznaczona tylko jedną liczbą.

Wektory: to tablica liczb. Liczby są ułożone w kolejności i możemy zidentyfikować każdy indywidualny numer według jego indeksu w tej kolejności. Możemy myśleć o wektorach jako o punktach w przestrzeni, gdzie każdy element podaje współrzędną wzdłuż innej osi. Mówiąc prościej, wektor jest strzałką reprezentującą wielkość, która ma zarówno wielkość, jak i kierunek, przy czym długość strzałki reprezentuje wielkość, a orientacja wskazuje kierunek. Na przykład wiatr, który ma kierunek i wielkość.

Macierz: macierz jest dwuwymiarową tablicą liczb, więc każdy element jest identyfikowany przez dwa indeksy, a nie tylko jeden. Jeśli rzeczywista macierz wartości A ma wysokość m i szerokość n, wtedy mówimy, że . Identyfikujemy elementy macierzy jako , gdzie m reprezentuje wiersz, a n reprezentuje kolumnę.

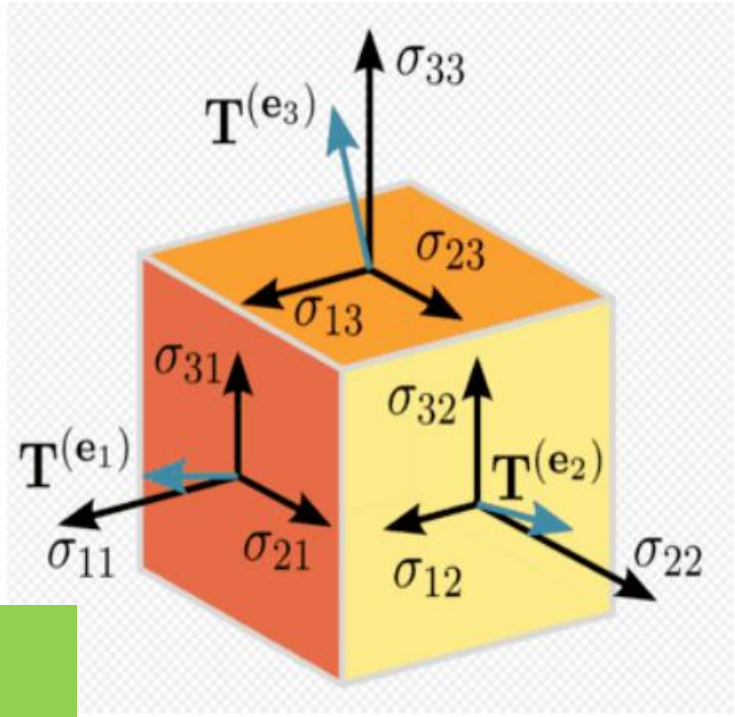


iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union

Tensor

W ogólnym przypadku tablica liczb ułożonych na regularnej siatce ze zmienną liczbą osi jest znana jako tensor. Ale aby naprawdę zrozumieć tensory, musimy rozwinąć sposób, w jaki myślimy o wektorach jako strzałach o odpowiedniej długości i kierunek. Pamiętaj, że wektor może być reprezentowany przez trzy składowe, a mianowicie składowe x, y i z (wektory bazowe). Jeśli masz długopis i papier, zróbmy mały eksperyment, umieść długopis pionowo na papierze i przechyl go pod pewnym kątem, a teraz skieruj światło z góry tak, aby cień długopisu padł na papier, ten cień, reprezentuje składnik x wektora „długopisu", a wysokość od papieru do końcówki długopisu jest składnikiem y. Teraz weźmy te składniki do opisania tensorów, wyobraźmy sobie, że jesteśmy uwięzieni w sześcianie, a trzy strzałki lecą w naszym kierunku z trzech ścian (reprezentujących oś x, y, z). Możemy myśleć o tych trzech strzałkach jako wektorach skierowanych do ciebie z trzech ścian sześcianu i możemy przedstawić te wektory (strzałki) x , y i z. Teraz jest to tensor (matryca) rzędu 2 z 9 składnikami. Pamiętaj, że jest to bardzo proste wyjaśnienie tensorów.



W tensorflow:

Tensor rangi 0 to skalar
Tensor rangi 1 to wektor
Tensor rangi 2 to macierz
Tensor rangi 3 to 3-wym Tensor
Tensor rangi n to n-wym Tensor

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Prawdopodobieństwo tensorflow (zestaw narzędzi do programowania prawdopodobieństwa)

- Tensorflow Probability – zestaw narzędzi prezentowany na Tensorflow Developer Summit 2018. Jest to zestaw narzędzi do programowania probabilistycznego dla badaczy i praktyków uczenia maszynowego do budowania modeli.
- Podamy praktyczne przykłady, abyś mógł sam korzystać z niektórych innych niesamowitych funkcji. Więc nawet jeśli znasz pojęcia prawdopodobieństwa, przynajmniej przejrzyj kod, aby zobaczyć, jak działa prawdopodobieństwo tensorflow

In [6]:
```
"""
At the moment of writing (05.20.2019) the only build that supports tensorflow probability is the tensorflow
nightly build so we will use that to install tensorflow 2.0 and tensorflow probability.
"""
# Install tensorflow 2.0 and tensorflow probability from the nightly build
#!pip install tf-nightly tfp-nightly seaborn
```

Out[6]:
```
'\nAt the moment of writing (05.20.2019) the only build that supports tensorflow probability is the tensorflow \nnightly build so we will use that to install tensorflow 2.0 and tensorflow probab
ility.\n'
```

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Prawdopodobieństwo tensorflow (zestaw narzędzi do programowania prawdopodobieństwa)

In [7]:

```
# Imports
import os
import random
import sys

import tensorflow as tf
import tensorflow_probability as tfp

# By convention, we generally refer to the tf probability distributions library as tfd.
tfd = tfp.distributions

import seaborn as sns

from matplotlib import pyplot as plt
from collections import defaultdict

# Import helpers file
"""
For some plots we need to convert tensors into numpy ndarrays. For that we use the evaluate function in
the helpers.py. If you are running this in Google Colab, make sure you upload the helpers.py found in the
notebooks folder to Google Colab but if you are running this in binder, you should be fine.
"""
from helpers import evaluate


# turning of tensorflow INFO, WARNING, and ERROR messages
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# plt axis colors setup
plt.rc_context({'axes.edgecolor':'orange', 'xtick.color':'red', 'ytick.color':'red', 'text.color':'orange'})

color_b = 'dodgerblue'
color_o = '#FF9A13'
color_sb = '#0504aa'
color_do = 'darkorange'
```

In [8]:

```
# Check the versions of tensorflow 2.0 and tensorflow probability

print("Tensorflow version: {}".format(tf.__version__))
print("Tensorflow probability version: {}".format(tfp.__version__))
```

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Biblioteka rozkładu prawdopodobieństwa Tensorflow

- Będziemy często używać modułu dystrybucji TFP i będziemy go nazywać tfd (=tfp.distributions).TF Probability wykorzystuje podklasy rozkładów do reprezentowania stochastycznych zmiennych losowych.
- Przypomnij sobie pierwszą przyczynę niepewności, wrodzoną stochastyczność. Oznacza to, że nawet gdybyśmy znali wszystkie wartości parametrów zmiennych, nadal byłoby to losowe. Zaczniemy od stworzenia rozkładu, a następnie, gdy z niego pobieramy próbki, stają się one tensorami tensorflow, którymi można deterministycznie manipulować.

Methods in tfd

- **sample(sample_shape=(), seed=None): Generates the specified sample size**
- **mean(): Calculates the average**
- **mode(): Calculates the mode**
- **variance(): Calculates variance**
- **stddev(): Calculates the standard deviation**
- **prob(value): Calculates the probability density/mass function**
- **log_prob(value): Calculates the logarithmic probability density/mass function.**
- **entropy(): Shannon entropy**

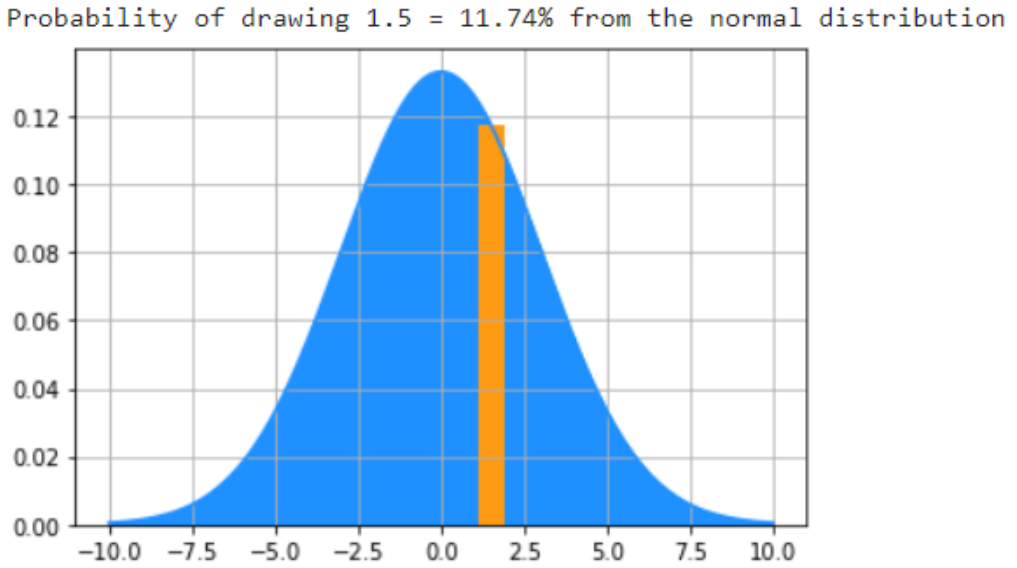
iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

prob(value): Oblicza funkcję gęstości prawdopodobieństwa/masy

In [16]:

```
"""
Let's say we want to find the probability of 1.55 (p(1.5)) from a continuous distribution. We can ofcourse
do the integral and find it but in tensorflow probability you have "prob()" which allows you to calculate
both Probability Mass Function and Probability Density Function.
For tfp.distributions.Normal "loc" is the mean and "scale" is the std deviation.

Also, there's nothing special about these numbers, play around with the scale, p(x) values and the k limits to
get a better understanding.
"""


# creating an x axis
samples = tf.range(-10, 10, 0.001)

# Create a Normal distribution with mean 0 and std deviation 3
normal_distribution = tfd.Normal(loc=0., scale=3)

# Then we calculate the PDFs of drawing 1.5
pdf_x = normal_distribution.prob(1.5)

# We can't plot tensors so evaluate is a helper function to convert to ndarrays
[pdf_x_] = evaluate([pdf_x])


# Finally, we plot both the PDF of the samples and p(1.5)
plt.plot(samples, normal_distribution.prob(samples), color=color_b)
plt.fill_between(samples, normal_distribution.prob(samples), color=color_b)
plt.bar(1.5, pdf_x_, color=color_o)
plt.grid()

print("Probability of drawing 1.5 = {:.4}% from the normal distribution".format(pdf_x*100))
```



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Wspólne rozkłady prawdopodobieństwa

Rozkład prawdopodobieństwa to funkcja opisująca prawdopodobieństwo uzyskania różnych możliwych wartości zmiennej losowej.

Bernoulli Distribution

Rozkład Bernoulliego to rozkład na pojedynczą binarną zmienną losową. Jest kontrolowany przez pojedynczy parametr φ∈[0,1]φ∈[0,1], który daje prawdopodobieństwo, że zmienna losowa będzie równa 1. Ma następujące właściwości:

$$P(\mathrm{x}=1)=\phi$$
$$P(\mathrm{x}=0)=1-\phi$$
$$P(\mathrm{x}=x)=\phi^{x}(1-\phi)^{1-x}$$
$$\mathbb{E}_{\mathrm{x}}[\mathrm{x}]=\phi$$
$$Var_{\mathrm{x}}(\mathrm{x})=\phi(1-\phi)$$

iBigWorld:2020-1-PL01-KA203-082197

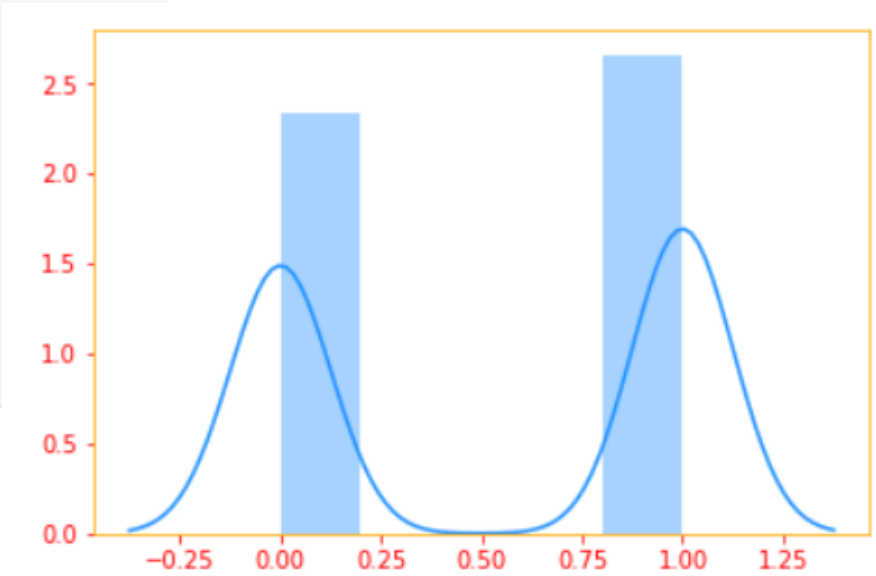
Co-funded by the Erasmus+ Programme of the European Union

Utwórz rozkład Bernoullego z prawdopodobieństwem 0,5 i wielkością próbki 1000

In [ ]:

```
# Create a Bernoulli distribution with a probability .5 and sample size of 1000
bernoulli_distribution = tfd.Bernoulli(probs=.5)
bernoulli_trials = bernoulli_distribution.sample(1000)

# Plot Bernoulli Distribution
sns.distplot(bernoulli_trials, color=color_b)

# Properties of Bernoulli distribution
property_1 = bernoulli_distribution.prob(1)
print("P(x = 1) = {}".format(property_1))

property_2 = bernoulli_distribution.prob(0)
print("P(x = 0) = 1 - {} = {}".format(property_1, property_2))

print("Property three is a generalization of property 1 and 2")

print("For Bernoulli distribution The expected value of a Bernoulli random variable  X is p (E[X] = p)")

# Variance is calculated as Var = E[(X - E[X])**2]
property_5 = bernoulli_distribution.variance()
print("Var(x) = {0} (1 - {0})".format(property_5))
```

```
P(x = 1) = 0.5
P(x = 0) = 1 - 0.5 = 0.5
Property three is a generalization of property 1 and two
For Bernoulli distribution The expected value of a Bernoulli random variable  X is p (E[X] = p)
Var(x) = 0.25 (1 - 0.25)
```



iBigWorld:2020-1-PL01-KA203-082197

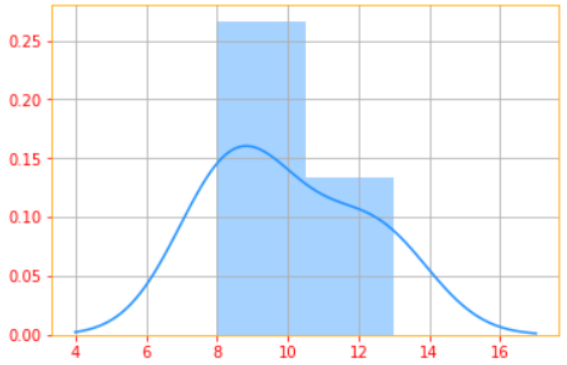
Co-funded by the Erasmus+ Programme of the European Union

Dystrybucja Multinoulli

Multinoulli lub rozkład kategoryczny to rozkład na pojedynczej zmiennej dyskretnej z k różnych stanów, gdzie k jest skończone. Rozkład wielomianowy jest szczególnym przypadkiem rozkładu wielomianowego, który jest uogólnieniem rozkładu dwumianowego. Rozkład wielomianowy to rozkład względem wektorów w 0,⋯,nk0,⋯,nk reprezentujący, ile razy każda z k kategorii odwiedzonych, gdy n próbek jest pobieranych z rozkładu wielomianowego.

In [ ]:

```
# For a fair dice
p = [1/6.]*6

# Multinoulli distribution with 60 trials and sampled once
multinoulli_distribution = tfd.Multinomial(total_count=60., probs=p)
multinoulli_pdf = multinoulli_distribution.sample(1)

print("""Dice throw values: {}
In sixty trials, index 0 represents the times the dice landed on 1 (= {} times) and
index 1 represents the times the dice landed on 2 (= {} times)\n""".format(multinoulli_pdf,
                                                                           multinoulli_pdf[0][0],
                                                                           multinoulli_pdf[0][1]))


g = sns.distplot(multinoulli_pdf, color=color_b)
plt.grid()
```




```
Dice throw values: [[ 8. 10. 13. 12.  9.  8.]]
In sixty trials, index 0 represents the times the dice landed on 1 (= 8.0 times) and
index 1 represents the times the dice landed on 2 (= 10.0 times)
```

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Gaussian Distribution

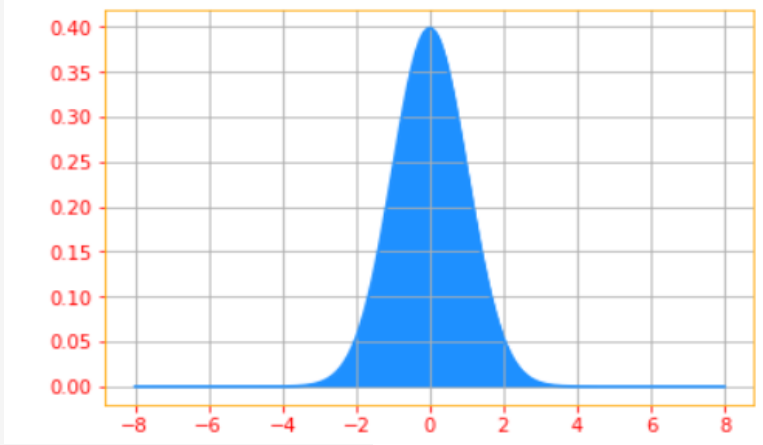
Najczęściej stosowanym rozkładem na liczbach rzeczywistych jest rozkład normalny, znany również jako rozkład Gaussa:

$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} exp\Big(-\frac{1}{2\sigma^2}(x-\mu)^2\Big)$$

Dwa parametry μ∈Rμ∈R i σ∈(0,∞)σ∈(0,∞) kontrolują rozkład normalny. Parametr μμ podaje współrzędną centralnego piku. Jest to również średnia z rozkładu: E[x]=μE[x]=μ. Odchylenie standardowe rozkładu jest podane przez σσ, a wariancję przez σ2σ2

In [ ]:

```
# We use linespace to create a range of values starting from -8 to 8 with incremants (= stop - start / num - 1)
rand_x= tf.linspace(start=-8., stop=8., num=150)

# Gaussian distribution with a standard deviation of 1 and mean 0
sigma = float(1.)
mu = float(0.)
gaussian_pdf = tfd.Normal(loc=mu, scale=sigma).prob(rand_x)

# convert tensors into numpy ndarrays for plotting
[rand_x_, gaussian_pdf_] = evaluate([rand_x, gaussian_pdf])

# Plot of the Gaussian distribution
plt.plot(rand_x_, gaussian_pdf_, color=color_b)
plt.fill_between(rand_x_, gaussian_pdf_, color=color_b)
plt.grid()
```



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Exponential Distributions

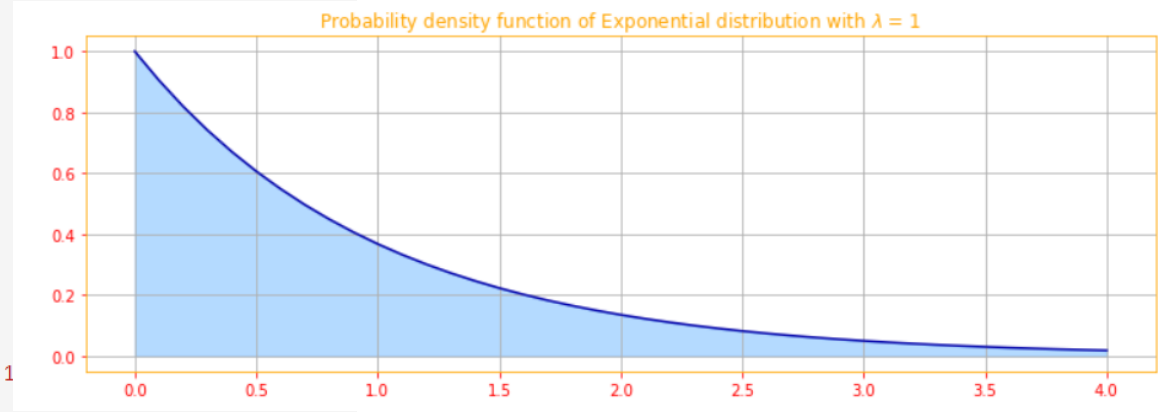
W kontekście uczenia głębokiego często chcemy mieć rozkład prawdopodobieństwa z ostrym punktem na x=0x=0. Aby to osiągnąć, możemy użyć rozkładu wykładniczego:

$$p(x;\lambda) = \lambda \mathbf{1}_{x \geq 0} exp(-\lambda x)$$

Rozkład wykładniczy wykorzystuje funkcję wskaźnika 1x≥01x≥0, aby przypisać prawdopodobieństwo zero do wszystkich ujemnych wartości xx.

In [ ]:

```
# We use linespace to create a range of values starting from 0 to 4 with incremants (= stop - start / num - 1)
a = tf.linspace(start=0., stop=4., num=41)

# the tf.newaxis expression is used to increase the dimension of the existing array by one more dimension
a = a[..., tf.newaxis]
lambdas = tf.constant([1.])

# We create a Exponential distribution and calculate the PDF for a
expo_pdf = tfd.Exponential(rate=1.).prob(a)

# convert tensors into numpy ndarrays for plotting
[a_, expo_pdf_] = evaluate([a,expo_pdf])

# Plot of Exponential distribution
plt.figure(figsize=(12.5, 4))
plt.plot(a_.T[0], expo_pdf_.T[[0]][0], color=color_sb)
plt.fill_between(a_.T[0], expo_pdf_.T[[0]][0],alpha=.33, color=color_b)
plt.title(r"Probability density function of Exponential distribution with $\lambda$ = 1
plt.grid()
```



Co-funded by the Erasmus+ Programme of the European Union

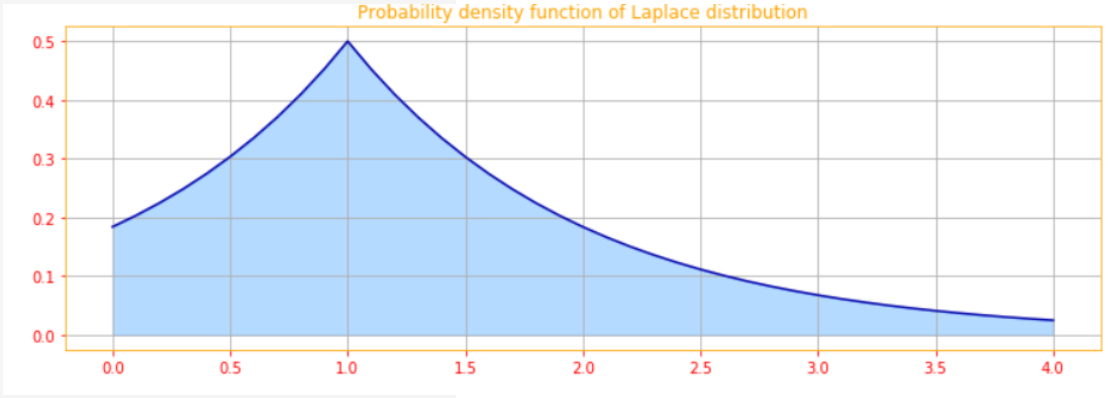
Dystrybucja Laplace'a

Ściśle powiązany rozkład prawdopodobieństwa, który pozwala nam umieścić ostry szczyt masy prawdopodobieństwa w dowolnym punkcie μμ, to rozkład Laplace'a:

$$\text{Laplace}(x;\mu,\gamma) = \frac{1}{2\gamma} exp\Big(-\frac{|x-\mu|}{\gamma}\Big)$$

In [ ]:

```
# We use linespace to create a range of values starting from 0 to 4 with incremants (= stop - start / num - 1)
a = tf.linspace(start=0., stop=4., num=41)

# the tf.newaxis expression is used to increase the dimension of the existing array by one more dimension
a = a[..., tf.newaxis]
lambdas = tf.constant([1.])

# We create a Laplace distribution and calculate the PDF for a
laplace_pdf = tfd.Laplace(loc=1, scale=1).prob(a)

# convert tensors into numpy ndarrays for plotting
[a_, laplace_pdf_] = evaluate([a, laplace_pdf])

# Plot of laplace distribution
plt.figure(figsize=(12.5, 4))
plt.plot(a_.T[0], laplace_pdf_.T[[0]][0], color=color_sb)
plt.fill_between(a_.T[0], laplace_pdf_.T[[0]][0],alpha=.33, color=color_b)
plt.title(r"Probability density function of Laplace distribution")
plt.grid()
```



iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Dystrybucja Diraca

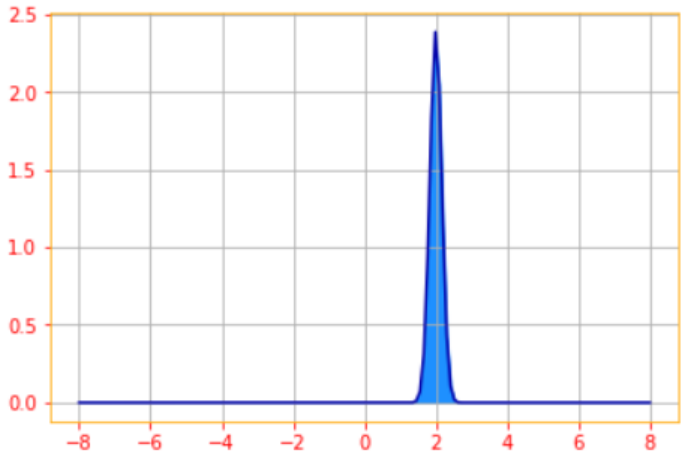
W niektórych przypadkach chcemy określić, że cała masa w rozkładzie prawdopodobieństwa skupia się wokół jednego punktu. Można to osiągnąć, definiując plik PDF za pomocą funkcji delta Diraca, δ(x)δ(x):

$$p(x) = \delta(x - \mu)$$

Funkcja delta Diraca jest zdefiniowana w taki sposób, że ma wartość zero wszędzie z wyjątkiem 0, ale całkuje do 1. Możemy myśleć o delcie Diraca jako o punkcie granicznym szeregu funkcji, które przypisują coraz mniejszą masę wszystkim punktom innym niż zero. Definiując, że p(x)p(x) przesunięte o δδ o −μ−μ otrzymujemy nieskończenie wąski, nieskończenie wysoki szczyt masy prawdopodobieństwa, gdzie x=μ

In [ ]:

```
"""
There is no dirac distribution in tensorflow, you will be able to plot using the fast fourier transform in
the tf.signal but that would take us outside the scope of the book so we use the normal distribution
to plot a dirac distribution. Play around with the delta and mu values to see how the distribution moves.
"""

# We use linespace to create a range of values starting from -8 to 8 with incremants (= stop - start / num - 1)
rand_x= tf.linspace(start=-8., stop=8., num=150)

# Gaussian distribution with a standard deviation of 1/6 and mean 2
delta = float(1./6.)
mu = float(2.)
dirac_pdf = tfd.Normal(loc=mu, scale=delta).prob(rand_x)

# convert tensors into numpy ndarrays for plotting
[rand_x_, dirac_pdf_] = evaluate([rand_x, dirac_pdf])

# Plot of the dirac distribution
plt.plot(rand_x_, dirac_pdf_, color=color_sb)
plt.fill_between(rand_x_, dirac_pdf_, color=color_b)
plt.grid()
```



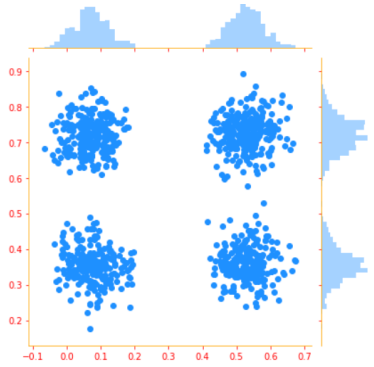
iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Mixtures of Distributions

Jednym z popularnych sposobów łączenia prostszych rozkładów w celu zdefiniowania rozkładu prawdopodobieństwa jest skonstruowanie rozkładu mieszanki. Rozkład mieszaniny składa się kilku rozkładów składników. W każdej próbie wybór, który rozkład składników powinien generować próbkę, jest określany przez pobranie tożsamości składnika z rozkładu multinoulli:

$$P(\mathbf{x}) = \sum_i P(c = i)\, P(\mathbf{x}|c = i)$$



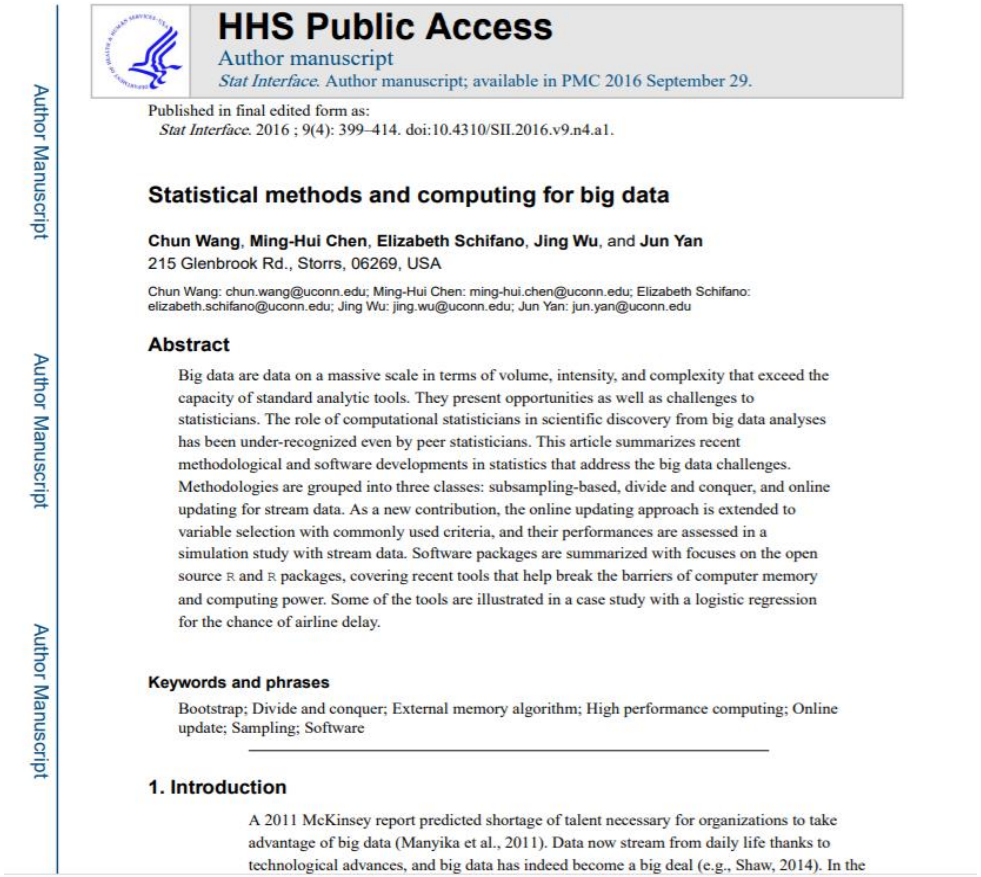
In [ ]:

```
"""
We will be creating two variable with two components to plot the mixture of distributions.

The tfd.MixtureSameFamily distribution implements a batch of mixture distribution where all components are from
different parameterizations of the same distribution type. In our example, we will be using tfd.Categorical to
manage the probability of selecting components. Followed by tfd.MultivariateNormalDiag as components.
The MultivariateNormalDiag constructs Multivariate Normal distribution on R^k
"""

num_vars = 2        # Number of variables (`n` in formula).
var_dim = 1         # Dimensionality of each variable `x[i]`.
num_components = 2  # Number of components for each mixture (`K` in formula).
sigma = 5e-2        # Fixed standard deviation of each component.

# Set seed.
tf.random.set_seed(77)

# categorical distribution
categorical = tfd.Categorical(logits=tf.zeros([num_vars, num_components]))

# Choose some random (component) modes.
component_mean = tfd.Uniform().sample([num_vars, num_components, var_dim])

# component distribution for the mixture family
components = tfd.MultivariateNormalDiag(loc=component_mean, scale_diag=[sigma])

# create the mixture same family distribution
distribution_family = tfd.MixtureSameFamily(mixture_distribution=categorical, components_distribution=components)

# Combine the distributions
mixture_distribution = tfd.Independent(distribution_family, reinterpreted_batch_ndims=1)

# Extract a sample from the distribution
samples = mixture_distribution.sample(1000).numpy()

# Plot the distributions
g = sns.jointplot(x=samples[:, 0, 0], y=samples[:, 1, 0], kind="scatter", color=color_b, marginal_kws=dict(bins=50))
plt.show()
```

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the Erasmus+ Programme of the European Union

Metody statystyczne i obliczenia dla dużych zbiorów danych



HHS Public Access
Author manuscript
Stat Interface. Author manuscript; available in PMC 2016 September 29.

Published in final edited form as:
Stat Interface. 2016 ; 9(4): 399–414. doi:10.4310/SII.2016.v9.n4.a1.

Statistical methods and computing for big data

Chun Wang, **Ming-Hui Chen**, **Elizabeth Schifano**, **Jing Wu**, and **Jun Yan**
215 Glenbrook Rd., Storrs, 06269, USA

Chun Wang: chun.wang@uconn.edu; Ming-Hui Chen: ming-hui.chen@uconn.edu; Elizabeth Schifano: elizabeth.schifano@uconn.edu; Jing Wu: jing.wu@uconn.edu; Jun Yan: jun.yan@uconn.edu

Abstract

Big data are data on a massive scale in terms of volume, intensity, and complexity that exceed the capacity of standard analytic tools. They present opportunities as well as challenges to statisticians. The role of computational statisticians in scientific discovery from big data analyses has been under-recognized even by peer statisticians. This article summarizes recent methodological and software developments in statistics that address the big data challenges. Methodologies are grouped into three classes: subsampling-based, divide and conquer, and online updating for stream data. As a new contribution, the online updating approach is extended to variable selection with commonly used criteria, and their performances are assessed in a simulation study with stream data. Software packages are summarized with focuses on the open source `R` and `R` packages, covering recent tools that help break the barriers of computer memory and computing power. Some of the tools are illustrated in a case study with a logistic regression for the chance of airline delay.

Keywords and phrases

Bootstrap; Divide and conquer; External memory algorithm; High performance computing; Online update; Sampling; Software

1. Introduction

A 2011 McKinsey report predicted shortage of talent necessary for organizations to take advantage of big data (Manyika et al., 2011). Data now stream from daily life thanks to technological advances, and big data has indeed become a big deal (e.g., Shaw, 2014). In the

Chun Wang, Ming-Hui Chen, Elizabeth Schifano, Jing Wu, and Jun Yan.
„Statistical methods and computing for big data”
Stat Interface. 2016 ; 9(4): 399–414.
doi:10.4310/SII.2016.v9.n4.a1.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5041595/pdf/nihms817697.pdf

Co-funded by the Erasmus+ Programme of the European Union

Dziękuję bardzo

iBigWorld:2020-1-PL01-KA203-082197

Co-funded by the
Erasmus+ Programme
of the European Union