Pengantar Sain Data

Konsep dan Aplikasi untuk Bisnis

To blah, blah, and blah.

Daftar Isi

References **liii**

Selamat di Pengantar Sain Data

Second Edition.

Version date: October 30, 2023.

This textbook and its supplements, including slides, labs, and interactive tutorials, may be downloaded for free at
openintro.org/book/ims[1].

Source files for this book can be found on GitHub at
github.com/OpenIntroStat/ims[3].

[1] http://openintro.org/book/ims
[2] https://www.openintro.org/go/?id=creativecommons_org&referrer=ims1_pdf
[3] https://github.com/OpenIntroStat/ims

0

Charting in Colaboratory

A common use for notebooks is data visualization using charts. Colaboratory makes this easy with several charting tools available as Python imports.

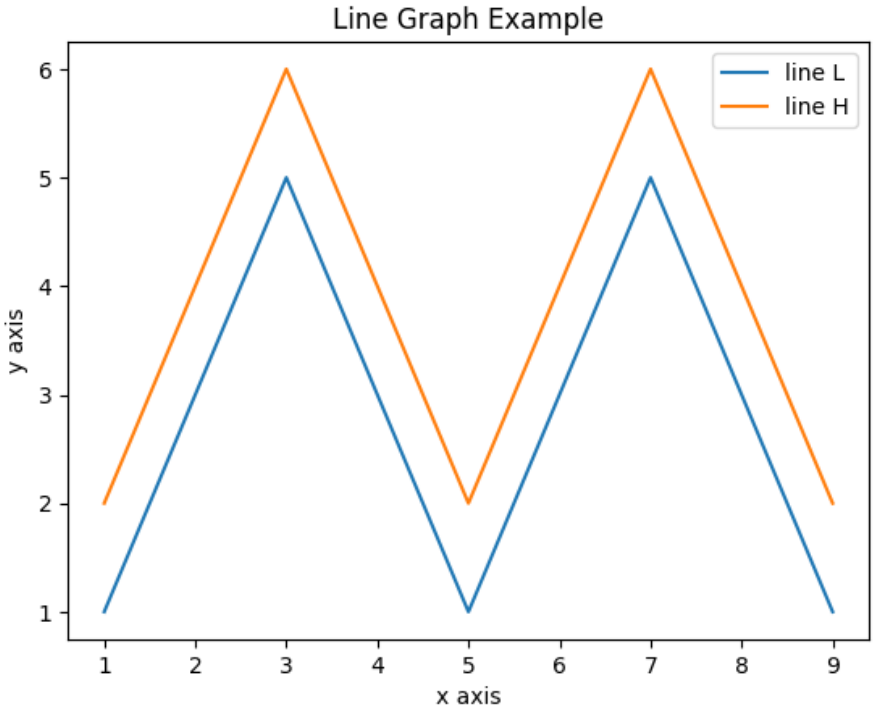
0.1 Matplotlib

Matplotlib[4] is the most common charting package, see its documentation[5] for details, and its examples[6] for inspiration.

0.1.1 Line Plots

```
import matplotlib.pyplot as plt

x  = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y1 = [1, 3, 5, 3, 1, 3, 5, 3, 1]
y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]
plt.plot(x, y1, label="line L")
plt.plot(x, y2, label="line H")
plt.plot()

plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("Line Graph Example")
plt.legend()
plt.show()
```

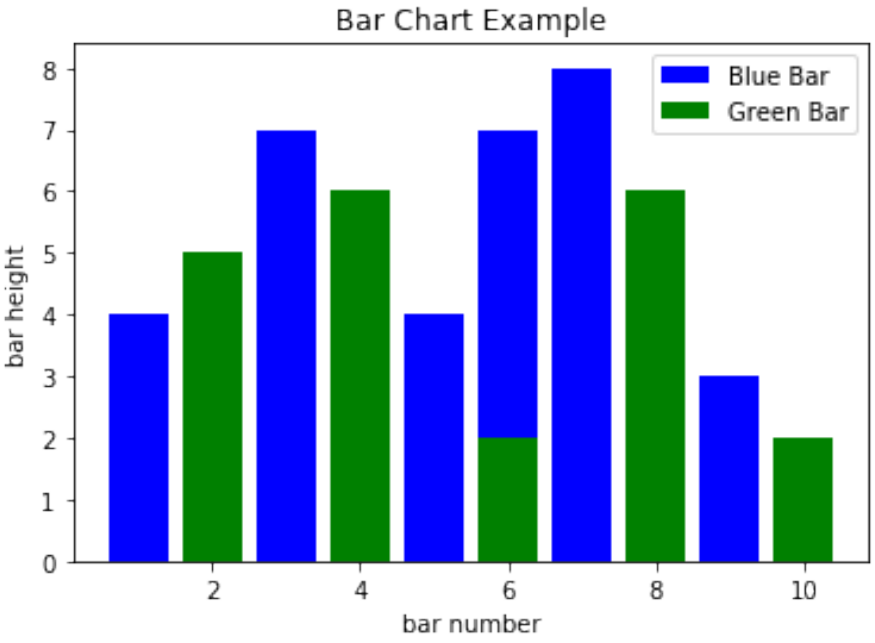
[4] http://matplotlib.org/
[5] http://matplotlib.org/api/pyplot_api.html
[6] http://matplotlib.org/gallery.html#statistics



0.1.2 Bar Plots

```
import matplotlib.pyplot as plt

# Look at index 4 and 6, which demonstrate overlapping cases.
x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

# Colors: https://matplotlib.org/api/colors_api.html

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
```

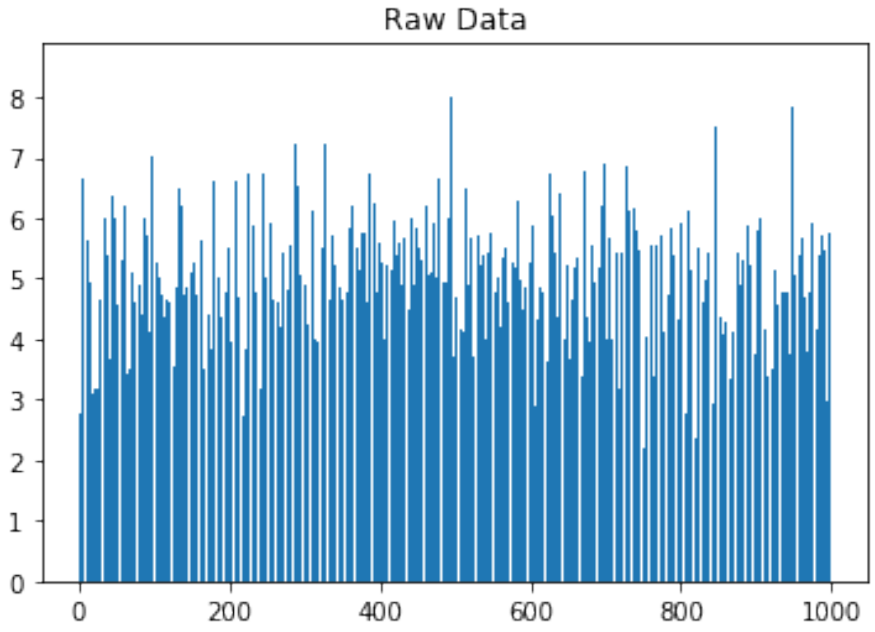
```
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```

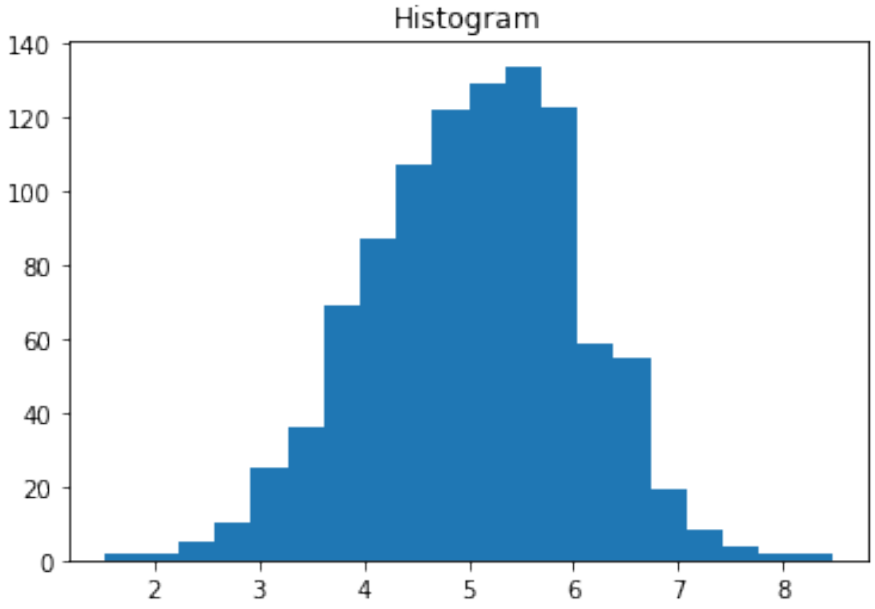


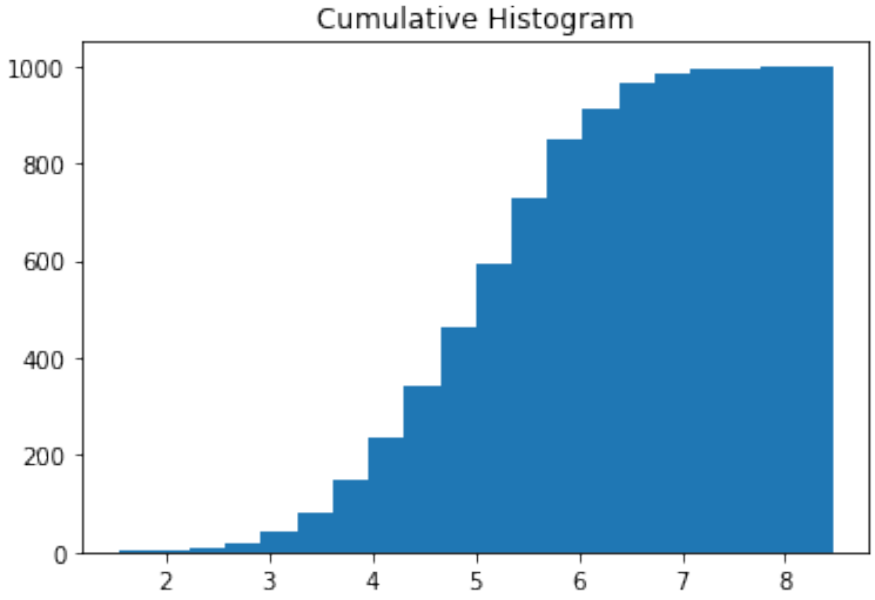
0.1.3 Histograms

```
import matplotlib.pyplot as plt
import numpy as np

# Use numpy to generate a bunch of random data in a bell curve
 ↪  around 5.
n = 5 + np.random.randn(1000)

m = [m for m in range(len(n))]
plt.bar(m, n)
plt.title("Raw Data")
plt.show()

plt.hist(n, bins=20)
plt.title("Histogram")
```

```
plt.show()

plt.hist(n, cumulative=True, bins=20)
plt.title("Cumulative Histogram")
plt.show()
```

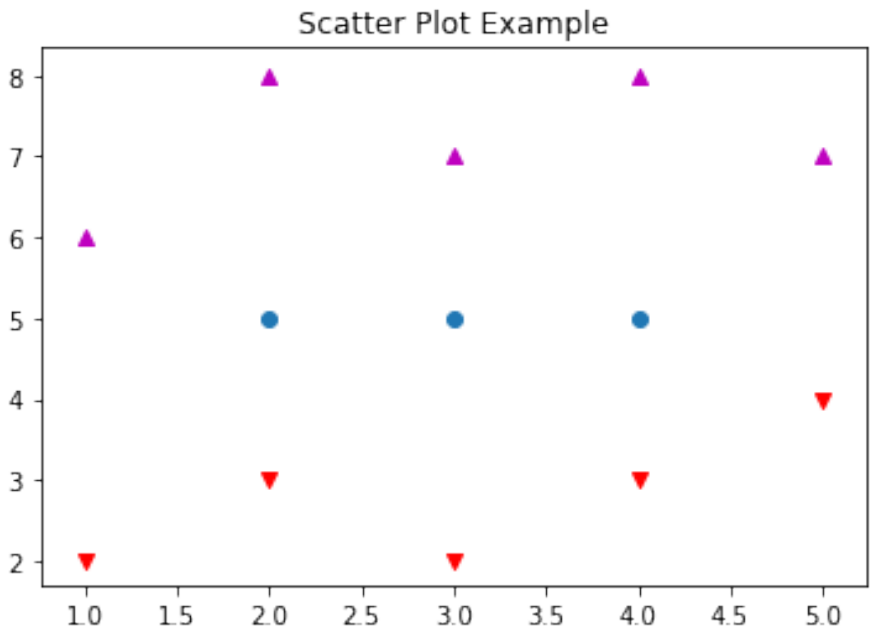



Histogram
140
120
100
80
60
40
20
0
2
3
4
5
6
7
8


Cumulative Histogram
1000
800
600
400
200
0
2
3
4
5
6
7
8

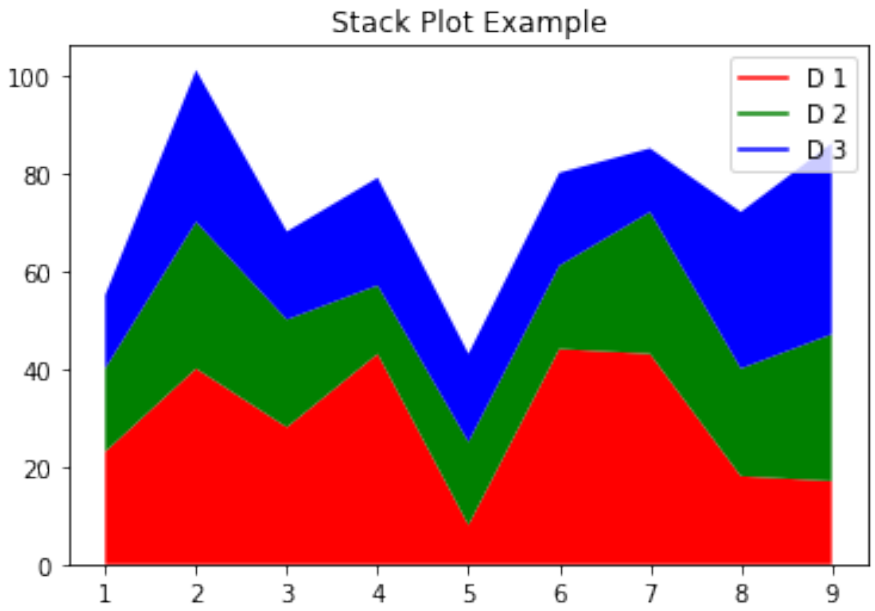
0.1.4 Scatter Plots

```
import matplotlib.pyplot as plt

x1 = [2, 3, 4]
y1 = [5, 5, 5]

x2 = [1, 2, 3, 4, 5]
y2 = [2, 3, 2, 3, 4]
y3 = [6, 8, 7, 8, 7]

# Markers: https://matplotlib.org/api/markers_api.html

plt.scatter(x1, y1)
plt.scatter(x2, y2, marker='v', color='r')
plt.scatter(x2, y3, marker='^', color='m')
plt.title('Scatter Plot Example')
plt.show()
```



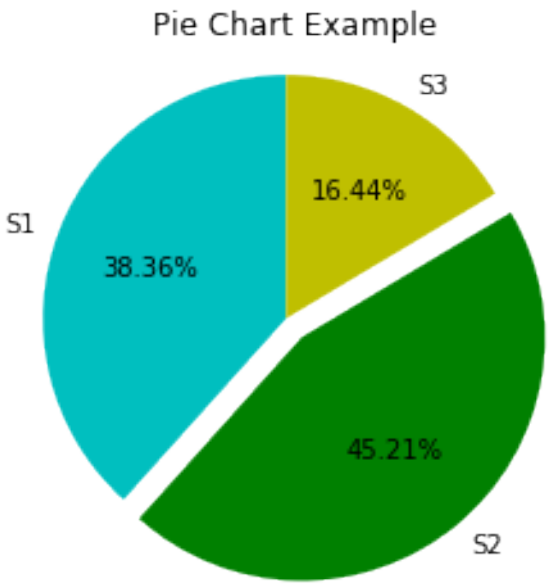
0.1.5 Stack Plots

```
import matplotlib.pyplot as plt

idxes = [ 1,  2,  3,  4,  5,  6,  7,  8,  9]
arr1  = [23, 40, 28, 43,  8, 44, 43, 18, 17]
arr2  = [17, 30, 22, 14, 17, 17, 29, 22, 30]
arr3  = [15, 31, 18, 22, 18, 19, 13, 32, 39]

# Adding legend for stack plots is tricky.
plt.plot([], [], color='r', label = 'D 1')
plt.plot([], [], color='g', label = 'D 2')
plt.plot([], [], color='b', label = 'D 3')

plt.stackplot(idxes, arr1, arr2, arr3, colors= ['r', 'g', 'b'])
plt.title('Stack Plot Example')
plt.legend()
plt.show()
```



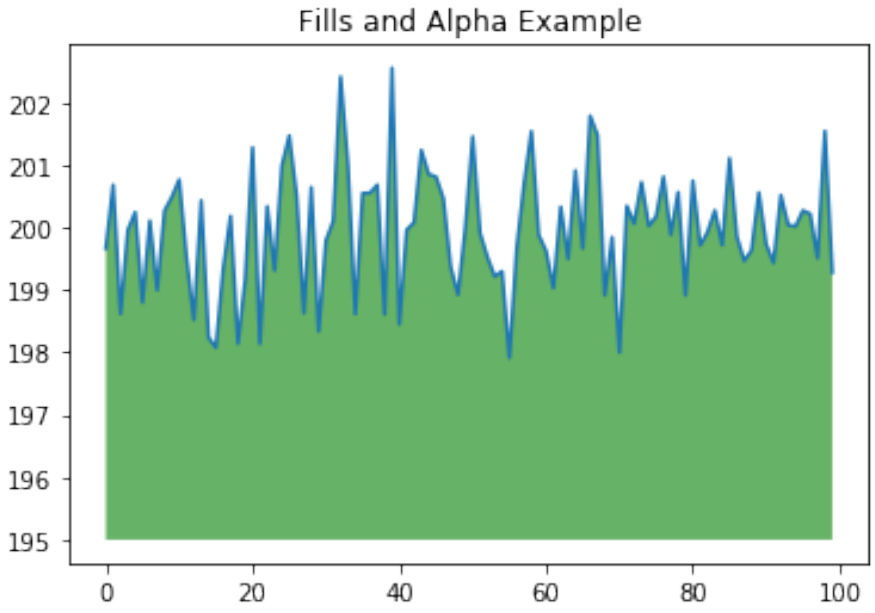
0.1.6 Pie Charts

```
import matplotlib.pyplot as plt

labels = 'S1', 'S2', 'S3'
sections = [56, 66, 24]
colors = ['c', 'g', 'y']

plt.pie(sections, labels=labels, colors=colors,
        startangle=90,
        explode = (0, 0.1, 0),
        autopct = '%1.2f%%')

plt.axis('equal') # Try commenting this out.
plt.title('Pie Chart Example')
plt.show()
```



0.1.7 fill_between and alpha

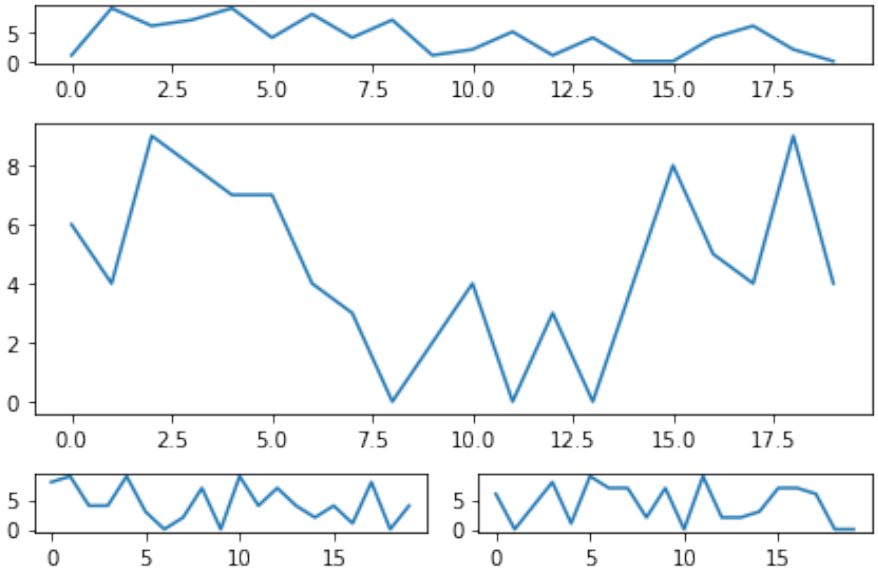
```
import matplotlib.pyplot as plt
import numpy as np

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g',
 ↪  alpha=0.6)

plt.title("Fills and Alpha Example")
plt.show()
```



0.1.8 Subplotting using Subplot2grid

```
import matplotlib.pyplot as plt
import numpy as np

def random_plots():
```

```
  xs = []
  ys = []

  for i in range(20):
    x = i
    y = np.random.randint(10)

    xs.append(x)
    ys.append(y)

  return xs, ys

fig = plt.figure()
ax1 = plt.subplot2grid((5, 2), (0, 0), rowspan=1, colspan=2)
ax2 = plt.subplot2grid((5, 2), (1, 0), rowspan=3, colspan=2)
ax3 = plt.subplot2grid((5, 2), (4, 0), rowspan=1, colspan=1)
ax4 = plt.subplot2grid((5, 2), (4, 1), rowspan=1, colspan=1)

x, y = random_plots()
ax1.plot(x, y)

x, y = random_plots()
ax2.plot(x, y)

x, y = random_plots()
ax3.plot(x, y)

x, y = random_plots()
ax4.plot(x, y)

plt.tight_layout()
plt.show()
```



0.2 Plot styles

Colaboratory charts use Seaborn's[7] custom styling by default. To customize styling further please see the matplotlib docs[8].

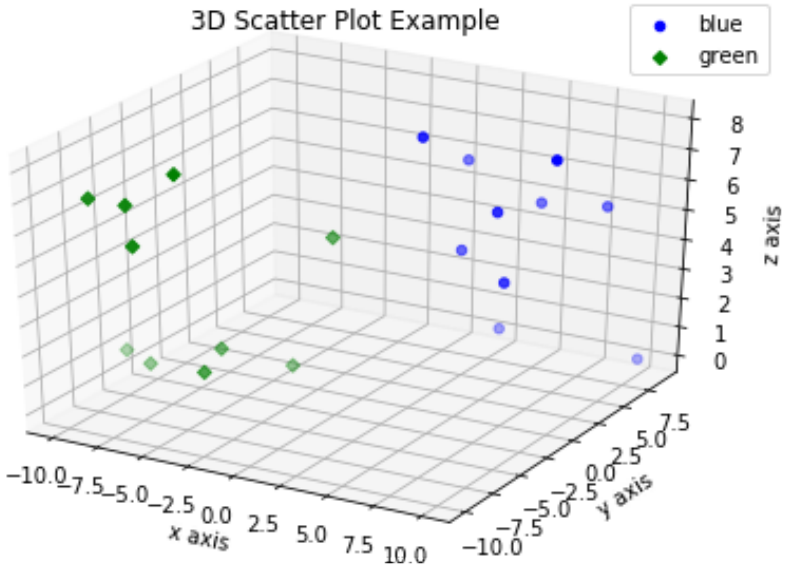
0.3 3D Graphs

0.3.1 3D Scatter Plots

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import axes3d

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
```

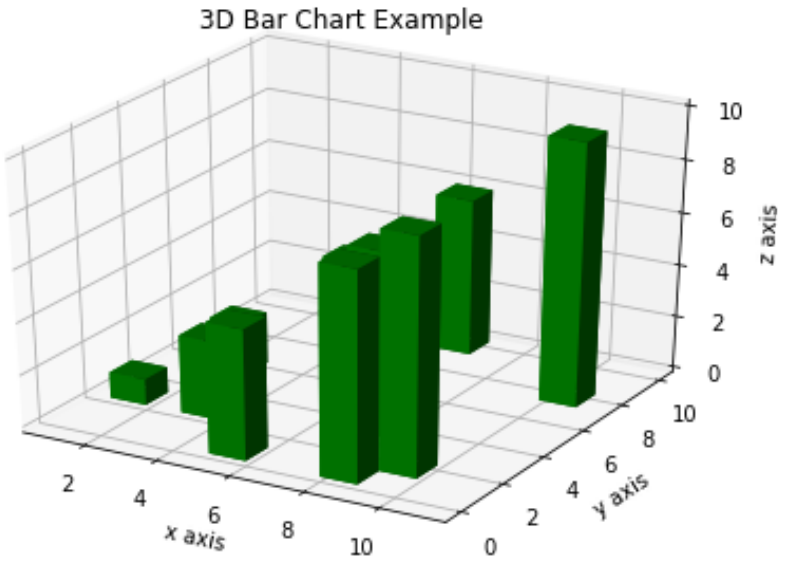
[7] https://seaborn.pydata.org
[8] https://matplotlib.org/users/style_sheets.html

```
x1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y1 = np.random.randint(10, size=10)
z1 = np.random.randint(10, size=10)

x2 = [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
y2 = np.random.randint(-10, 0, size=10)
z2 = np.random.randint(10, size=10)

ax.scatter(x1, y1, z1, c='b', marker='o', label='blue')
ax.scatter(x2, y2, z2, c='g', marker='D', label='green')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Scatter Plot Example")
plt.legend()
plt.tight_layout()
plt.show()
```



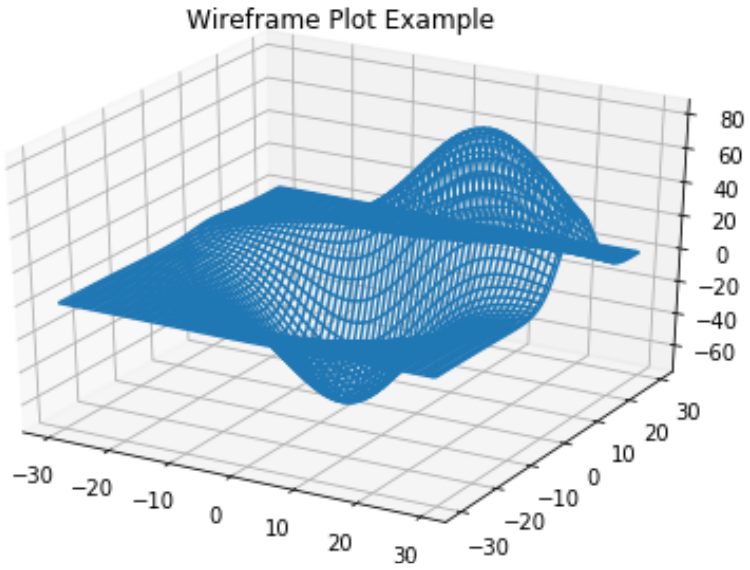
0.3.2 3D Bar Plots

```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.randint(10, size=10)
z = np.zeros(10)

dx = np.ones(10)
dy = np.ones(10)
dz = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

ax.bar3d(x, y, z, dx, dy, dz, color='g')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Bar Chart Example")
plt.tight_layout()
plt.show()
```



0.3.3 Wireframe Plots

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x, y, z = axes3d.get_test_data()

ax.plot_wireframe(x, y, z, rstride = 2, cstride = 2)

plt.title("Wireframe Plot Example")
plt.tight_layout()
plt.show()
```

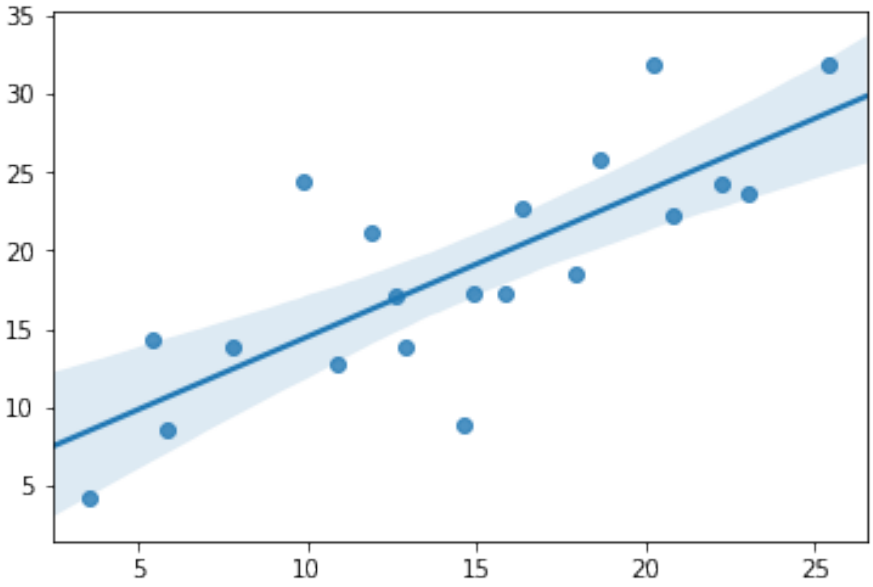


0.4 Seaborn

There are several libraries layered on top of Matplotlib that you can use in Colab. One that is worth highlighting is Seaborn[9]:

[9] http://seaborn.pydata.org

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Generate some random data
num_points = 20
# x will be 5, 6, 7... but also twiddled randomly
x = 5 + np.arange(num_points) + np.random.randn(num_points)
# y will be 10, 11, 12... but twiddled even more randomly
y = 10 + np.arange(num_points) + 5 * np.random.randn(num_points)
sns.regplot(x, y)
plt.show()
```



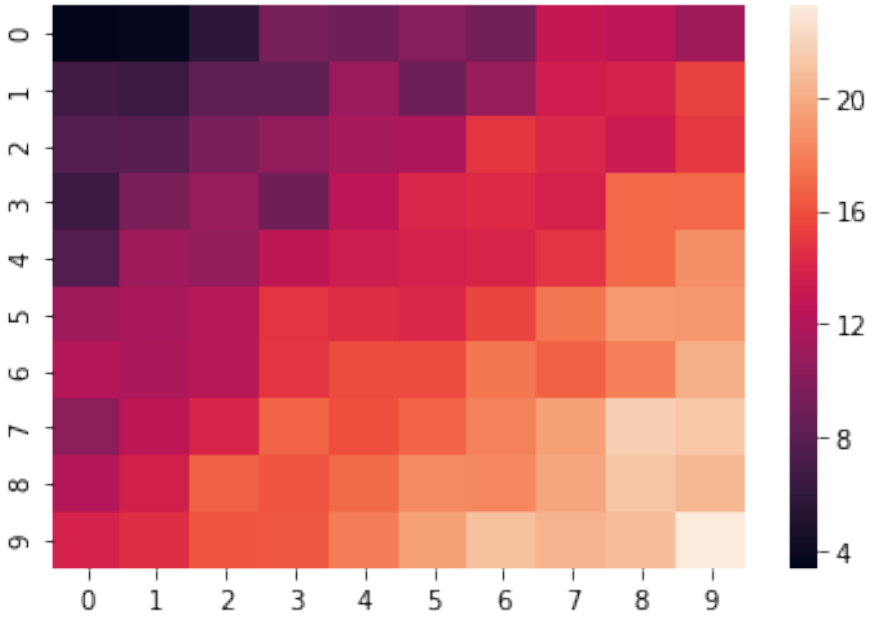
That's a simple scatterplot with a nice regression line fit to it, all with just one call to Seaborn's regplot[10].

Here's a Seaborn heatmap[11]:

```
import matplotlib.pyplot as plt
import numpy as np
```

[10] http://seaborn.pydata.org/generated/seaborn.regplot.html#seaborn.regplot
[11] https://seaborn.pydata.org/generated/seaborn.heatmap.html

```
# Make a 10 x 10 heatmap of some random data
side_length = 10
# Start with a 10 x 10 matrix with values randomized around 5
data = 5 + np.random.randn(side_length, side_length)
# The next two lines make the values larger as we get closer to
 ↪  (9, 9)
data += np.arange(side_length)
data += np.reshape(np.arange(side_length), (side_length, 1))
# Generate the heatmap
sns.heatmap(data)
plt.show()
```



0.5 Altair

Altair[12] is a declarative visualization library for creating interactive visualizations in Python, and is installed and enabled in Colab by default.

[12] http://altair-viz.github.io

For example, here is an interactive scatter plot:

```
import altair as alt
from vega_datasets import data
cars = data.cars()

alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
).interactive()
```

```
alt.Chart(...)
```

For more examples of Altair plots, see the Altair snippets notebook[13] or the external Altair Example Gallery[14].

0.6 Plotly

0.6.1 Sample

```
from plotly.offline import iplot
import plotly.graph_objs as go

data = [
    go.Contour(
        z=[[10, 10.625, 12.5, 15.625, 20],
           [5.625, 6.25, 8.125, 11.25, 15.625],
           [2.5, 3.125, 5., 8.125, 12.5],
           [0.625, 1.25, 3.125, 6.25, 10.625],
           [0, 0.625, 2.5, 5.625, 10]]
    )
]
iplot(data)
```

```
<IPython.core.display.HTML object>
```

```
Unable to display output for mime type(s): text/html
```

[13] /notebooks/snippets/altair.ipynb
[14] https://altair-viz.github.io/gallery/

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

0.7 Bokeh

0.7.1 Sample

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Call once to configure Bokeh to display plots inline in the
 ↪  notebook.
output_notebook()
```

```
N = 4000
x = np.random.random(size=N) * 100
y = np.random.random(size=N) * 100
radii = np.random.random(size=N) * 1.5
colors = ["#%02x%02x%02x" % (r, g, 150) for r, g in
 ↪  zip(np.floor(50+2*x).astype(int),
 ↪  np.floor(30+2*y).astype(int))]

p = figure()
p.circle(x, y, radius=radii, fill_color=colors, fill_alpha=0.6,
 ↪  line_color=None)
show(p)
```

```
Unable to display output for mime type(s): text/html
```

0

Kernels and Feature maps: Theory and intuition

Table of Contents

Following the series on SVM, we will now explore the theory and intuition behind Kernels and Feature maps, showing the link between the two as well as advantages and disadvantages.

The notebook is divided into two main sections: 1. Theory, derivations and pros and cons of the two concepts 2. An intuitive and visual interpretation in 3 dimensions

The section part of this notebook seved as a basis for the following answer on stats.stackexchange: - https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is/355046#355046

0

Theory and derivations

0.7.2 Notation and terminology

- **Attributes** of a problem are the original inputs: $x_1, x_2, ...x_n$
- **Feature mapping** is a function of the input attributes $\phi(x)$
- **Features** are the new set of quantities that result from applying the mapping

For example we could have

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

To obtain more complex, non linear, decision boundaries, we may want to apply the SVM algorithm to learn some features $\phi(x)$ rather than the input attributes x only. To do so we replace x everywhere in the previous formuals with $\phi(x)$ and repeat the optimization procedure.

The problem is that the features may live in very *high dimensional space*, possibly infinite, which makes the computation of the dot product $< \phi(x^{(i)}, \phi(x^{(j)}) >$ very difficult. This is where we introduce the notion of a **Kernel** which will greatly help us perform these computations.

0.7.3 Kernels: definition and example

Given a feature mapping ϕ we define the corresponding Kernel as

$$K(x, z) = \phi(x)^T \phi(z)$$

Hence we can replace the inner product $< \phi(x), \phi(z) >$ with $K(x, z)$ in the SVM algorithm. What is interesting is that the kernel may be very inexpensive to calculate, and may correspond to a mapping in very high dimensional space. So we can train an SVM in such space without having to explicitly calculate the inner product.

Consider the example where $x, z \in \mathbb{R}^n$ and $K(x, z) = (x^T z)^2$. We can also write this as

$$\begin{aligned} K(x,z) &= \left(\sum_i^n x_i z_i\right)\left(\sum_j^n x_j z_j\right) \\ &= \sum_i^n \sum_j^n x_i x_j z_i z_j \\ &= \sum_{i,j}^n (x_i x_j)(z_i z_j) \\ &= \phi(x)^T \phi(z) \end{aligned}$$

Where the feature mapping ϕ is given by (in this case $n = 2$)

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix}$$

Calculating the feature mapping is of complexity $O(n^2)$ due to the number of features, whereas calculating $K(x,z)$ is of complexity $O(n)$ as it is a simple inner product $x^T z$ which is then squared $K(x,z) = (x^T z)^2$.

Another example of Kernel is

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j}^n (x_i x_j)(z_i z_j) + \sum_i^n (\sqrt{2c} x_i)(\sqrt{2c} x_i) + c^2 \end{aligned}$$

which corresponds to the features mapping

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \\ \sqrt{2c} x_1 \\ \sqrt{2c} x_2 \end{bmatrix}$$

so the parameter c controls the relative weighting of the first and second order polynomials. More generally the kernel $K(x,z) = (x^T z + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space, corresponding to all monomials that are up to order d. Despite working in this $O(n^d)$ dimensional space, computing $K(x,z)$ is of order $O(n)$.

An intuitive view of Kernels would be that they correspond to functions that measure how closely related vectors x and z are. So when x and z are

similar the Kernel will output a large value, and when they are dissimilar K will be small. Knowing this justifies the use of the Gaussian Kernel as a measure of similarity

$$K(x,z) = \exp[\left(-\frac{||x-z||^2}{2\sigma^2}\right)$$

because the value is close to 1 when they are similar and close to 0 when they are not.

0.7.4 Necessary and sufficient conditions

The following are necessary and sufficient conditions for a function to be a valid kernel. Let G be the **Kernel matrix** or **Gram matrix** which is square of size $m \times m$ and where each i, j entry corresponds to $G_{i,j} = K(x^{(i)}, x^{(j)})$ of the data set $X = \{x^{(1)}, ..., x^{(m)}\}$

- The function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a valid kernel if and only if
- the kernel matrix G is symmetric, positive semi-definite

This is both a *necessary* and *sufficient* condition (i.e. goes both ways) and is called Mercer's theorem.

0.7.5 Properties

- Kernels are **symmetric**: $K(x,y) = K(y,x)$
- Kernels are **positive, semi-definite**: $\sum_{i=1}^{m}\sum_{j=1}^{m} c_i c_j K(x^{(i)}, x^{(j)}) \geq 0$
- Sum of two kernels is a kernel: $K(x,y) = K_1(x,y) + K_2(x,y)$
- Product of two kernels is a kernel: $K(x,y) = K_1(x,y)\ K_2(x,y)$
- Scaling by any function on both sides is a kernel: $K(x,y) = f(x)K_1(x,y)f(y)$
- Kernels are often scaled such that $K(x,y) \leq 1$ and $K(x,x) = 1$

0.7.6 Common Kernels

- Linear: is the inner product: $K(x,y) = x^T y$
- Gaussian / RBF / Radial : $K(x,y) = \exp(-\gamma(x-y)^2)$
- Polynomial: is the inner product: $K(x,y) = (1 + x^T y)^p$
- Laplace: is the inner product: $K(x,y) = \exp(-\beta|x-y|)$
- Cosine: is the inner product: $K(x,y) = \exp(-\beta|x-y|)$

0.7.7 Gaussian kernels

In general the Squared Exponential Kernel, or Gaussian kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Sigma (\mathbf{x} - \mathbf{x}')\right)$$

If Σ is diagnonal then this can be written as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}(x_j - x_j')^2\right)$$

Where the parameter σ_j^2 is the characteristic length scale of dimension j. if $\sigma_j^2 = \infty$ the dimension is ignored, hence this is known as the **ARD** kernel.

Finally if Σ is sperical, we get the isotropic kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

Which is a **radial basis function** or RBF kernel as it is only a function of $||\mathbf{x} - \mathbf{x}'||^2$. σ^2 is known as the bandwidth parameter.

0.7.8 Gram Matrix vs Feature Map

Consider a dataset of m data points which are n dimensional vectors $\in \mathbb{R}^n$, the **gram matrix** is the $m \times m$ matrix for which each entry is the kernel between the corresponding data points.

$$G_{i,j} = K(x^{(i)}, x^{(j)})$$

Since a Kernel function corresponds to an inner product in some (possibly infinite dimensional) feature space, we can also write the kernel as a **feature mapping**

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$$

When using a Kernel in a linear model, it is just like transforming the input data, then running the model in the transformed space.

For the linear kernel, the Gram matrix is simply the inner product $ G_{i,j} = x^{(i) T} x^{(j)}$. For other kernels, it is the inner product in a feature space with feature map ϕ: i.e. $ G_{i,j} = (x^{(i)})^T (x^{(j)})$

0.7.9 Advantages and disadvantages of Gram matrix and Feature mapping

Grams matrix: reduces computations by pre-computing the kernel for all pairs of training examples

- On the other hand, the Gram matrix may be impossible to hold in memory for large m
- The cost of taking the product of the Gram matrix with weight vector may be large

Feature maps: are computationally very efficient

- As long as we can transform and store the input data efficiently
- The drawback is that the dimension of transformed data may be much larger than the original data
- memory required to store the features and cost of taking the product to compute the gradient.
- finally, feature maps may require infinite dimensional space (e.g. Gaussian Kernel) which requires approximation

As a result there exists systems trade offs and rules of thumb

- When the number of examples is very large, **feature maps are better**
- When transformed features have high dimensionality, **Grams matrices** are better

---

0.7.10 Sources and further reading

- http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture4.pdf
- https://disi.unitn.it/~passerini/teaching/2014-2015/MachineLearning/slides/17_kernel_machines/hand
- Stanford CS229 - Lecture notes
- Introduction to Statistical Learning
- Elements of Statistical Learning

0

A visual example to help intuition

From the following stats.stackexchange post: - https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is/355046#355046

Consider the following dataset where the yellow and blue points are clearly not linearly separable in two dimensions.

If we could find a higher dimensional space in which these points were **linearly separable**, then we could do the following:

- Map the original features to the higher, transformer space (feature mapping)
- Perform linear SVM in this higher space
- Obtain a set of weights corresponding to the decision boundary hyperplane
- Map this hyperplane back into the original 2D space to obtain a non linear decision boundary

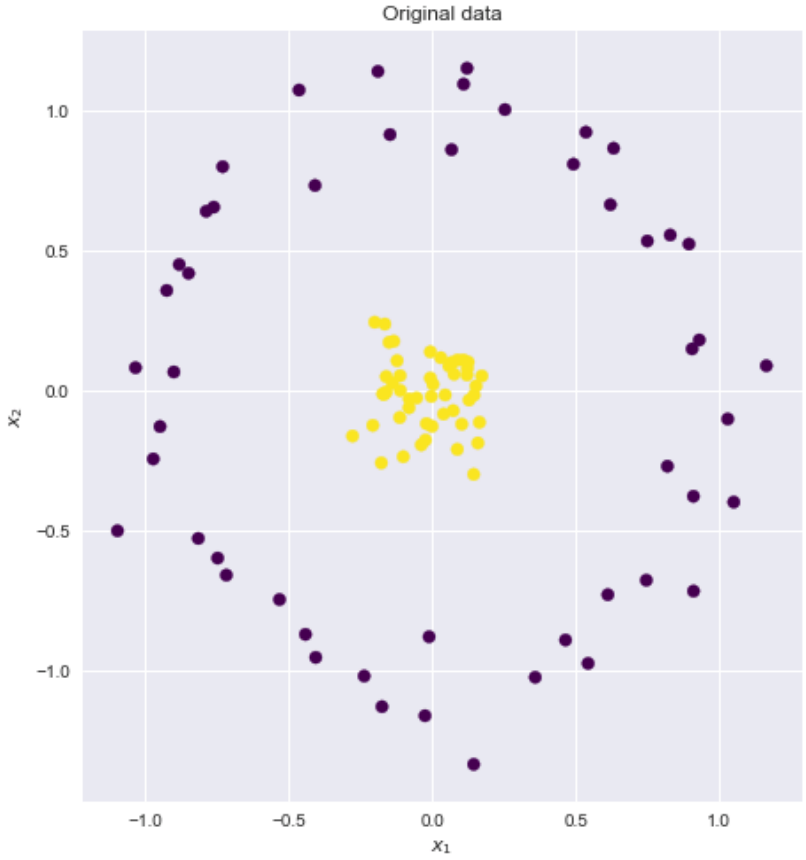
There are many higher dimensional spaces in which these points are linearly separable. Here is one example

$$x_1, x_2 :\rightarrow z_1, z_2, z_3$$
$$z_1 = \sqrt{2}x_1x_2 \quad z_2 = x_1^2 \quad z_3 = x_2^2$$

This is where the Kernel trick comes into play. Quoting the above great answers

---

> Suppose we have a mapping $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that brings our vectors in $\mathbb{R}^n$ to some feature space $\mathbb{R}^m$. Then the dot product of $\mathbf{x}$ and $\mathbf{y}$ in this space is $\varphi(\mathbf{x})^T\varphi(\mathbf{y})$. A kernel is a function k that corresponds to this dot product, i.e. $k(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^T\varphi(\mathbf{y})$



15

Figure 1: enter image description here

If we could find a kernel function that was equivalent to the above feature map, then we could plug the kernel function in the linear SVM and perform the calculations very efficiently.

0.7.11 Polynomial kernel

It turns out that the above feature map corresponds to the well known **polynomial kernel** : $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^d$. Let $d = 2$ and $\mathbf{x} = (x_1, x_2)^T$ we get

$$k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) = (x_1 x_2' + x_2 x_2')^2$$
$$= 2x_1 x_1' x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2$$
$$= (\sqrt{2}x_1 x_2 \; x_1^2 \; x_2^2) \begin{pmatrix} \sqrt{2}x_1' x_2' \\ x_1'^2 \\ x_2'^2 \end{pmatrix}$$

$$k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

$$\phi(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} \sqrt{2}x_1 x_2 \\ x_1^2 \\ x_2^2 \end{pmatrix}$$

0.7.12 Visualizing the feature map and the resulting boundary line

- Left hand side plot shows the points plotted in the transformed space together with the SVM linear boundary hyper plane
- Right hand side plot shows the result in the original 2-D space

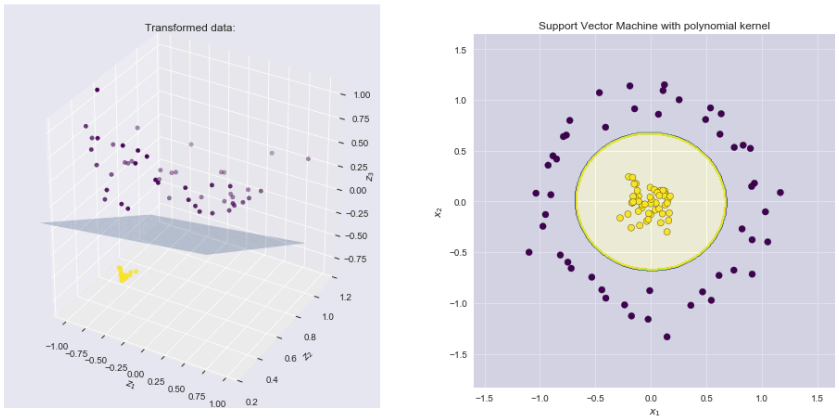


Figure 2: enter image description here

0

Python implementation of various feature maps and kernels

0.7.13 Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d
from IPython.display import HTML, Image

%matplotlib inline
sns.set()

from sklearn.datasets.samples_generator import make_circles
```

0.7.14 Feature map n. 1)

Sum of polynomials

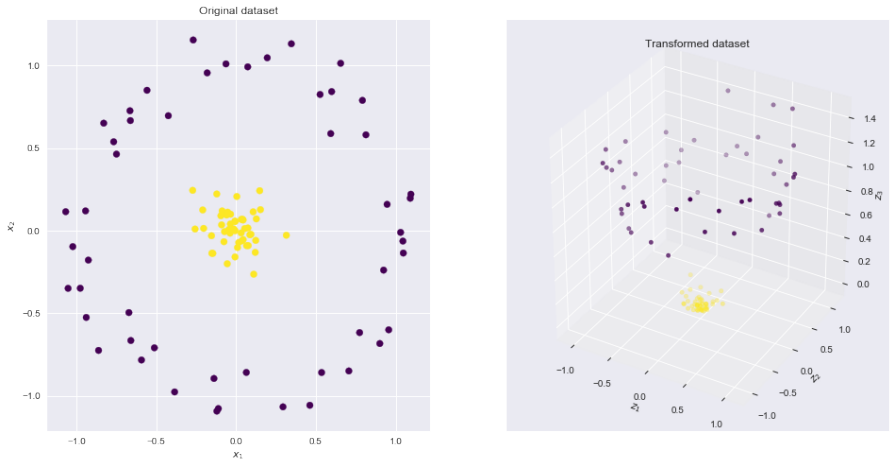
$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

```
def feature_map_1(X):
    return np.asarray((X[:,0], X[:,1], X[:,0]**2 + X[:,1]**2)).T
```

```
#Generate dataset and feature-map
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)
Z = feature_map_1(X)

#2D scatter plot
fig = plt.figure(figsize = (16,8))
```

```
ax = fig.add_subplot(1, 2, 1)
ax.scatter(X[:,0], X[:,1], c = y, cmap = 'viridis')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Original dataset')

#3D scatter plot
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.scatter3D(Z[:,0],Z[:,1], Z[:,2],c = y, cmap = 'viridis' )
 ↪  #,rstride = 5, cstride = 5, cmap = 'jet', alpha = .4,
 ↪  edgecolor = 'none' )
ax.set_xlabel('$z_1$')
ax.set_ylabel('$z_2$')
ax.set_zlabel('$z_3$')
ax.set_title('Transformed dataset')
```

```
Text(0.5,0.92,'Transformed dataset')
```



0.7.15 Feature map n. 2)

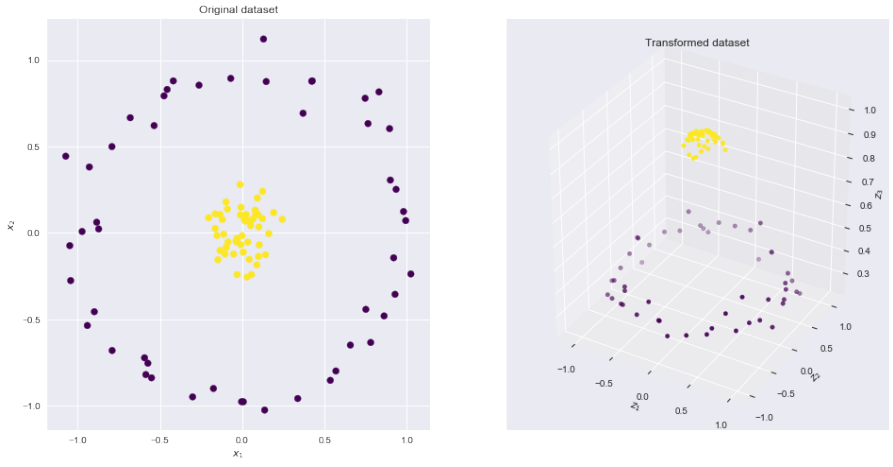
Gaussian Radial Basis Function (RBF) centered at $0, 0$

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, e^{-[x_1+x_2^2]})$$

```
def feature_map_2(X):
    return np.asarray((X[:,0], X[:,1], np.exp( -( X[:,0]**2 +
     ↪  X[:,1]**2)))).T
```

```
#Generate dataset and feature-map
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)
Z = feature_map_2(X)

#2D scatter plot
fig = plt.figure(figsize = (16,8))
ax = fig.add_subplot(1, 2, 1)
ax.scatter(X[:,0], X[:,1], c = y, cmap = 'viridis')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Original dataset')

#3D scatter plot
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.scatter3D(Z[:,0],Z[:,1], Z[:,2],c = y, cmap = 'viridis' )
 ↪  #,rstride = 5, cstride = 5, cmap = 'jet', alpha = .4,
 ↪  edgecolor = 'none' )
ax.set_xlabel('$z_1$')
ax.set_ylabel('$z_2$')
ax.set_zlabel('$z_3$')
ax.set_title('Transformed dataset')
```

```
Text(0.5,0.92,'Transformed dataset')
```



0.7.16 Feature map n. 3) - also a polynomial kernel

$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^d$

Let $d = 2$ and $\mathbf{x} = (x_1, x_2)^T$ we get

$$\begin{aligned} k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) &= (x_1 x_2' + x_2 x_2')^2 \\ &= 2x_1 x_1' x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 \\ &= (\sqrt{2}x_1 x_2 \; x_1^2 \; x_2^2) \begin{pmatrix} \sqrt{2}x_1' x_2' \\ x_1'^2 \\ x_2'^2 \end{pmatrix} \end{aligned}$$

$$k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

Where

$$\phi(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} \sqrt{2}x_1 x_2 \\ x_1^2 \\ x_2^2 \end{pmatrix}$$

In the plot of the transformed data we map

$$x_1, x_2 :\rightarrow z_1, z_2, z_3$$

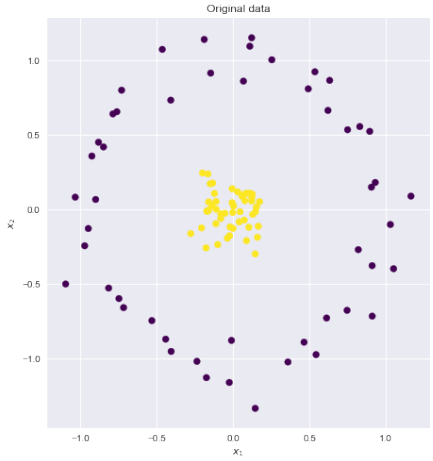
$$z_1 = \sqrt{2}x_1 x_2 \quad z_2 = x_1^2 \quad z_3 = x_2^2$$

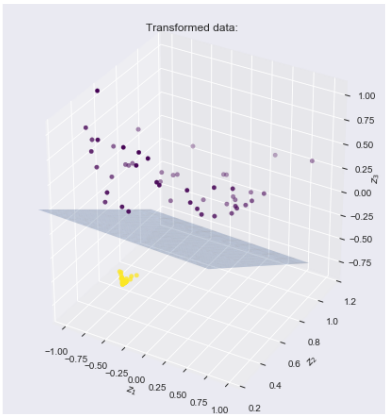
```
def feature_map_3(X):
    return np.asarray(( np.sqrt(2) *X[:,0] * X[:,1], X[:,0]**2,
     ↪  X[:,1]**2)).T
```

```
#Generate dataset and feature-map
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1, random_state = 0)
Z = feature_map_3(X)

#2D scatter plot
fig = plt.figure(figsize = (16,8))
ax = fig.add_subplot(1, 2, 1)
ax.scatter(X[:,0], X[:,1], c = y, cmap = 'viridis')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Original data')

#3D scatter plot
ax = fig.add_subplot(1, 2, 2, projection='3d')
```

```
ax.scatter3D(Z[:,0],Z[:,1], Z[:,2],c = y, cmap = 'viridis' )
 ↪  #,rstride = 5, cstride = 5, cmap = 'jet', alpha = .4,
 ↪  edgecolor = 'none' )
ax.set_xlabel('$z_1$')
ax.set_ylabel('$z_2$')
ax.set_zlabel('$z_3$')
ax.set_title('Transformed data: ')

#SVM using kernel 3 - feature map 3
clf = svm.SVC(C = 1, kernel = 'linear')
clf.fit(Z, y)

w = clf.coef_.flatten()
b = clf.intercept_.flatten()
print('w=',w,'b=',b)

# create x,y
xx, yy = np.meshgrid(np.linspace(-1,1), np.linspace(0,1))

# calculate corresponding z
boundary = (-w[0] * xx - w[1] * yy - b) * 1. /w[2]


# plot the surface

ax.plot_surface(xx, yy, boundary, alpha = .3)
ax.set_ylim(.2,1.2)
ax.set_zlim(-.9,1.1)
#ax.view_init(0, 260)

plt.show()
```

```
w= [-0.05481854 -2.53191791 -2.52028513] b= [1.14976292]
```


Original data
1.0
0.5
0.0
−0.5
−1.0
x_2
−1.0
−0.5
0.0
0.5
1.0
x_1


Transformed data:
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
z_2
1.2
1.0
0.8
0.6
0.4
0.2
z_3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
z_1

0

From Feature Maps to Kernels

0.7.17 Kernel as the inner product in the transformed space

$$K(\mathbf{x^{(i)}}, \mathbf{x^{(j)}}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

0.7.18 Grams matrix as an outer product in the transformed space

$$G_{i,j} = K(\mathbf{x^{(i)}}, \mathbf{x^{(j)}})$$

$$G = \tilde{X}\tilde{X}^T$$

0.8 Sklearn implementation of SVC with Gram matrix

```
from sklearn import svm
from sklearn.metrics import accuracy_score

clf = svm.SVC(kernel='precomputed')
# kernel computation
gram = np.dot(feature_map_2(X), feature_map_2(X).T)
clf.fit(gram, y)

# prediction errors on training examples
np.sum(y - clf.predict(gram))
```

```
0
```

0.9 Sklearn implementation using custom Kernel

```
def my_kernel_1(X,Y):
    return np.dot(feature_map_1(X),feature_map_1(Y).T )

def my_kernel_2(X,Y):
    return np.dot(feature_map_2(X),feature_map_2(Y).T )

def my_kernel_3(X,Y):
    return np.dot(feature_map_3(X),feature_map_3(Y).T )
```

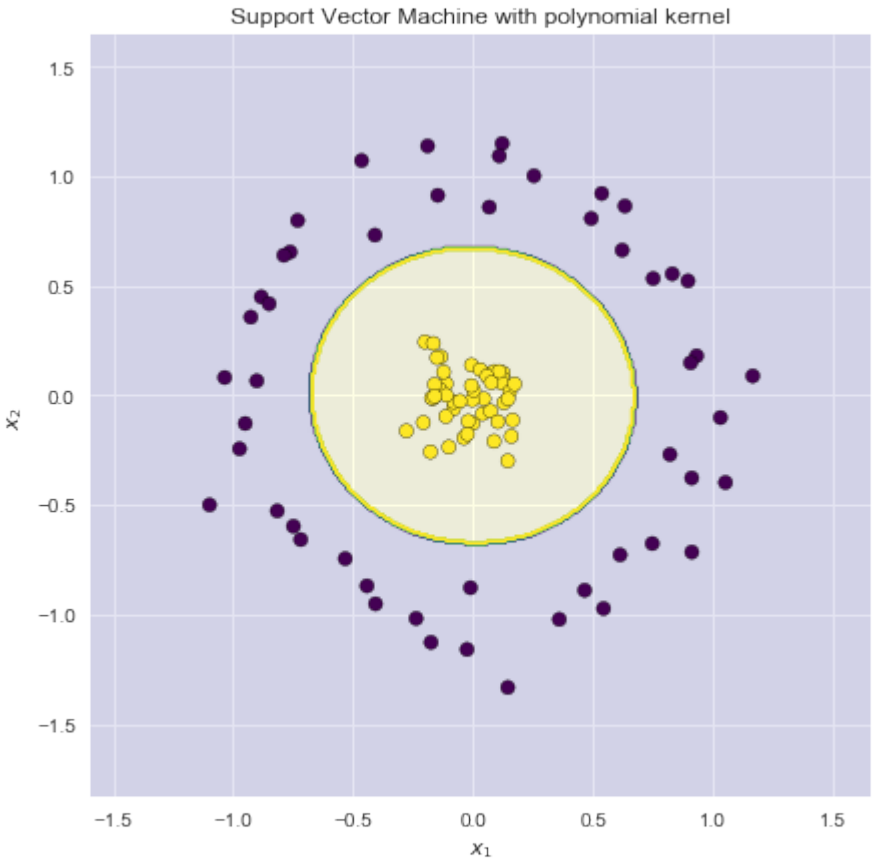
```
#SVM using kernel 1 - feature map 1
clf = svm.SVC(kernel=my_kernel_1)
clf.fit(X, y)

# predict on training examples - print accuracy score
print('Accuracy score using feature map n1',accuracy_score(y,
 ↪  clf.predict(X)))

#SVM using kernel 2 - feature map 2
clf = svm.SVC(kernel=my_kernel_2)
clf.fit(X, y)

# predict on training examples - print accuracy score
print('Accuracy score using feature map n2',accuracy_score(y,
 ↪  clf.predict(X)))

#SVM using kernel 3 - feature map 3
clf = svm.SVC(kernel=my_kernel_3)
clf.fit(X, y)

# predict on training examples - print accuracy score
print('Accuracy score using feature map n3',accuracy_score(y,
 ↪  clf.predict(X)))
```

```
Accuracy score using feature map n1 1.0
Accuracy score using feature map n2 1.0
Accuracy score using feature map n3 1.0
```

0.10 Plot decision boundary

```
clf = svm.SVC(kernel=my_kernel_3, C = 1)
# kernel computation
clf.fit(X, y)

#Initialize data
h = .01 #Stepsize in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 ↪  np.arange(y_min, y_max, h))

#Predict on meshgrid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(figsize = (7,7))
plt.contourf(xx, yy, Z,1, colors = ['darkblue','yellow'], alpha
 ↪  = .1)
plt.contour(xx, yy, Z, cmap = 'viridis')

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors =
 ↪  'k')
plt.title('Support Vector Machine with polynomial'
          ' kernel')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
```

```
Text(0,0.5,'$x_2$')
```


Support Vector Machine with polynomial kernel
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x_1

0

Contoh

$$\chi^2 = \sum_{i,j} \frac{\left(O_{ij} - E_{ij}\right)^2}{E_{ij}}$$

makanan itu dapat digunakan dengan $x + 2$

```
  pip install academics-scholar-scraper
```

```
Collecting academics-scholar-scraper
  Downloading academics_scholar_scraper-0.1.1-py3-none-any.whl (5.5 kB)
Collecting argparse (from academics-scholar-scraper)
  Downloading argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from academics-sc
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from academics-schola
Collecting openai (from academics-scholar-scraper)
  Downloading openai-0.27.8-py3-none-any.whl (73 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.6/73.6 kB 3.5 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from openai->acade
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from re
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from a
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohtt
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aio
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aioh
Installing collected packages: argparse, openai, academics-scholar-scraper
Successfully installed academics-scholar-scraper-0.1.1 argparse-1.4.0 openai-0.27.8
```

```
Unable to display output for mime type(s): application/vnd.colab-display-data+json
```

```
  pip install argparse requests tqdm concurrent.futures openai
```

```
Collecting argparse
```

```
  Using cached argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.65.0)
ERROR: Could not find a version that satisfies the requirement concurrent.futures (from versions: none)
ERROR: No matching distribution found for concurrent.futures
```

0

Summary

In summary, this book has no content whatsoever.

References