Matematika Dasar

Pengantar Menuju Analisis Data

SANDY H.S. HERHO

Matematika Dasar

Pengantar Menuju Analisis Data

Sandy Hardian Susanto Herho[1]

24 Juni 2020

[1]sandyherho@protonmail.ch

Untuk publik sains terbuka Indonesia.

Kata Pengantar

Keilmuan analisis data menjadi primadona dalam perbincangan global di satu dekade terakhir. Hal ini ditandai dengan semakin terintegrasinya kehidupan umat manusia dengan teknologi yang berbuntut dengan lahirnya kumpulan data raksasa (*big data*) yang dapat kemudian dijadikan bahan bagi pemelajaran algoritma kecerdasan buatan untuk menghasilkan produk yang diharapkan dapat beradaptasi dengan kebutuhan pengguna.

Kecerdasan buatan di dalam beberapa waktu terakhir ini juga menelurkan perkembangan cukup pesat. Hal ini ditandai dengan terciptanya mobil swakemudi (*self-driving car*), diagnosa medis mutakhir, sistem kecerdasan yang dapat mengalahkan manusia dalam permainan catur dan GO, dll.

Progres di dalam penelitian dan pengembangan kecerdasan buatan tampak menjanjikan. Bukan tidak mungkin dalam beberapa dekade mendatang kita dapat membangun sistem kecerdasan buatan umum (*Artificial General Intelligence*) yang dapat menggantikan profesi manusia. Namun untuk mencapai hal tersebut diperlukan kerja keras dan kecerdasan dari para peneliti dan pengembang untuk menemukan dan mengeksekusi algoritma kecerdasan buatan hingga mencapai tahap siap untuk diproduksi secara massal. Mimpi ini tentu tidaklah mudah untuk dicapai karena untuk berkontribusi di dalam penelitian dan pengembangan algoritma kecerdasan buatan diperlukan dasar kemampuan matematis dan pemrograman yang cukup kuat.

Melalui catatan singkat ini penyusun hendak mengajak pembaca untuk berkenalan dengan konsep - konsep matematika yang diperlukan seba-

gai fondasi untuk memahami dan mengimplementasikan algoritma - algoritma yang digunakan di dalam analisis data melalui bahasa pemrograman Python dan R (penyusun mengharapkan familiaritas pembaca terhadap konsep - konsep kedua bahasa pemrograman tersebut sebelum mempelajari buku ini).

Catatan ini sendiri didesain sebagai jembatan kesenjangan pengetahuan di dunia industri dan dunia akademik terkait konsep - konsep dasar matematika yang dibutuhkan bagi para analis data junior, pengembang piranti lunak, mahasiswi/a dari jurusan non-eksakta, dan kaum awam guna berkontribusi di dalam pengembangan kecerdasan buatan melalui analisis data.

Buku ini akan membahas tiga konsep matematika utama yang menjadi fondasi algoritma kecerdasan buatan jaringan saraf tiruan (*neural networks*), yakni aljabar linier, kalkulus peubah banyak, dan teori probabilitas.

Selain konsep - konsep matematika, pembaca juga akan disuguhkan laboratorium pemrograman kecil pada setiap bab-nya dan latihan pemahaman konsep di bagian akhir buku. Pembaca diharapkan dapat memahami dan mengimplementasikan konsep - konsep matematika yang diterangkan di sini pada bidang keilmuannya masing - masing ketika selesai mempelajari catatan ini.

Catatan ini juga bersifat sumber terbuka, karena dituliskan dengan menggunakan LaTeX dan berlisensi milik publik (*copy-left*), sehingga para pembaca dapat menyalin dan mengubahnya, bahkan untuk kepentingan komersil sekalipun, secara cuma - cuma di `https://github.com/sandyherho/buku-mat-andat`.

Akhir kata, semoga catatan singkat dapat membantu dan saya menanti dengan tangan terbuka kolaborasi pembelajaran berbasis kode terbuka di laman GitHub penyusun.

Penyusun

Daftar Isi

1

Aljabar Linier

Skalar, vektor, matriks, dan tensor

Definisi skalar, vektor, dan matriks

Berikut ini merupakan definisi praktis dari sudut pandang keilmuan kecerdasan buatan untuk ketiga objek utama di dalam keilmuan aljabar linier ini:

- Skalar (x): nilai tunggal tanpa dimensi. Formulasi umumnya adalah:

$$X \tag{1.1}$$

  Contoh:

$$x = 12$$

- Vektor ($\mathbf{x}$): himpunan bilangan (*array*) berdimensi tunggal yang dapat berupa kolom ataupun baris dan dapat diidentifikasi melalui baris dan kolom. Formulasi umumnya adalah:

$$\begin{bmatrix} X_0 & X_1 & \cdots & X_n \end{bmatrix} \text{ atau } \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_n \end{bmatrix} \tag{1.2}$$

Contoh:

$$\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \text{ atau } \mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- Matriks ($\mathbf{X}$): *Array* berdimensi dua[1], di mana setiap elemennya dapat diakses melalui indeks baris dan kolom[2]. Formulasi umumnya adalah:

$$\mathbf{X} = \begin{bmatrix} X_{0,0} & X_{0,1} & \cdots & X_{0,n} \\ X_{1,0} & X_{1,1} & \cdots & X_{1,n} \\ \cdots & \cdots & \cdots & \cdots \\ X_{m,0} & X_{m,1} & \cdots & X_{m,n} \end{bmatrix} \tag{1.3}$$

Contoh:

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Operasi antar matriks

Operasi matriks seperti penjumlahan, pengurangan, dan perkalian sangat lazim dijumpai di dalam algoritma pemelajaran mesin.

Penjumlahan dan Pengurangan Matriks

Penjumlahan atau pengurangan matriks merupakan operasi penjumlahan atau pengurangan antar elemen dengan indeks yang sama pada kedua matriks, oleh karena itu prasyarat utamanya adalah kedua matriks yang hendak dijumlahkan harus mempunyai dimensi yang sama. Penjumlahan atau pengurangan antar matriks $\mathbf{A}$ dan $\mathbf{B}$ akan menghasilkan matriks $\mathbf{C}$ dengan

[1]Dimensi umumnya didefinisikan dalam format baris, kolom. Matriks merupakan *array* berdimensi dua karena mempunyai baris dan kolom, sementara vektor berdimensi tunggal karena hanya mempunyai baris atau kolom.

[2]Pengindeksan yang penyusun gunakan pada catatan ini berbeda dengan kebanyakan pengindeksan yang dijabarkan pada buku teks aljabar linier karena dalam kontek kecerdasan buatan yang lebih dekat pada dunia ilmu komputer, umumnya digunakan pengindeksan yang dimulai dari nol.

dimensi yang sama. Berikut adalah formulasi umum operasi penjumlahan atau pengurangan antar matriks tersebut:

$$\begin{aligned}\mathbf{A} \pm \mathbf{B} &= \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,0} & a_{m,1} & \cdots & a_{m,n} \end{bmatrix} + \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,0} & b_{m,1} & \cdots & b_{m,n} \end{bmatrix} \\ &= \begin{bmatrix} a_{0,0} \pm b_{0,0} & a_{0,1} \pm b_{0,1} & \cdots & a_{0,n} \pm b_{0,n} \\ a_{1,0} \pm b_{1,0} & a_{1,1} \pm b_{1,1} & \cdots & a_{1,n} \pm b_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,0} \pm b_{m,0} & a_{m,1} \pm b_{m,1} & \cdots & a_{m,n} \pm b_{m,n} \end{bmatrix}\end{aligned} \tag{1.4}$$

Contoh:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 7 \\ 5 & 6 & 8 \end{bmatrix}$$

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 3 & 4 & 4 \end{bmatrix}$$

Perkalian matriks

Perkalian antar matriks **A** dan **B** akan menghasilkan matriks baru, yakni **C**. Untuk melakukan operasi perkalian ini, matriks **A** harus mempunyai jumlah kolom yang sama dengan jumlah baris pada matriks **B**. Berikut adalah formulasi umum perkalian antar kedua matriks:

$$\begin{aligned}\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} * \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} &= \begin{bmatrix} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} & a_{0,0}b_{0,1} + a_{0,1}b_{1,1} \\ a_{0,1}b_{0,0} + a_{1,1}b_{1,0} & a_{1,0}b_{0,1} + a_{1,1}b_{1,1} \end{bmatrix} \\ &= \begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix}\end{aligned}$$

Atau secara umum dapat diformulasikan seperti pada persamaan 1.5 berikut ini:

$$\mathbf{C}_{i,j} = \Sigma_k \mathbf{A}_{i,k} \mathbf{B}_{j,k} \tag{1.5}$$

Perkalian matriks juga dapat terjadi pada matriks yang mempunyai dimensi yang berbeda, asalkan matriks pertama mempunyai kolom yang sama dengan jumlah baris pada matriks kedua,seperti pada contoh berikut ini:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} * \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 1(1)+2(2) & 1(3)+2(4) & 1(5)+2(6) \\ 3(1)+4(2) & 3(3)+4(4) & 3(5)+4(6) \\ 5(1)+6(2) & 5(3)+6(4) & 5(5)+6(6) \end{bmatrix}$$
$$= \begin{bmatrix} 5 & 11 & 17 \\ 11 & 25 & 39 \\ 17 & 39 & 61 \end{bmatrix}$$

Terdapat dua buah karakteristik utama dari perkalian matriks, yakni:

1. Distributif:
$$\mathbf{A}\left(\mathbf{B}+\mathbf{C}\right) = \mathbf{AB} + \mathbf{AC} \tag{1.6}$$

2. Asosiatif:
$$\mathbf{A}\left(\mathbf{BC}\right) = \left(\mathbf{AB}\right)\mathbf{C} \tag{1.7}$$

Transpos

Transpos merupakan operasi menukar indeks matriks pada diagonal utamanya. Dengan kata lain, transpos menukar indeks baris menjadi indeks kolom, dan sebaliknya.

Berikut ini adalah contohnya:

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}^T = \begin{bmatrix} a_{0,0} & a_{1,0} & a_{2,0} \\ a_{0,1} & a_{1,1} & a_{2,1} \\ a_{0,2} & a_{1,2} & a_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Berikut ini merupakan contoh operasi transpos pada matriks dengan dimensi baris dan kolom yang berbeda:

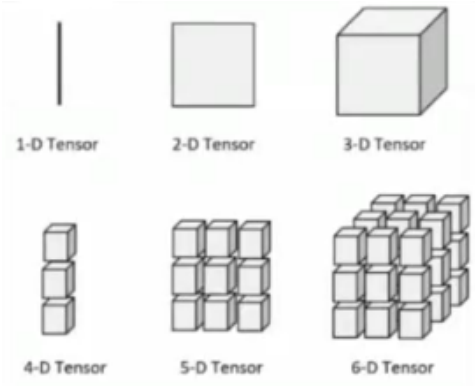
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Kita juga dapat melakukan operasi transpos pada vektor. Operasi transpos vektor ini lah yang sering digunakan dalam pemrosesan dan pembersihan data sebelum dijadikan input algoritma pemelajaran mesin:

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Tensor

Terkadang di dalam era data raksasa seperti saat ini, kita harus berurusan dengan data yang mempunyai sumbu dimensi lebih dari dua. Data dengan sumbu dimensi banyak inilah yang dinamakan tensor (vektor dan matriks juga termasuk ke dalam tensor dimensi satu dan dua, lihat Gambar 1.1) . Operasi dan fitur yang ditawarkan oleh tensor sama seperti yang telah kita ketahui pada vektor dan matriks, hanya saja berbeda pada dimensi saja, dan hal ini dapat dengan mudah diatasi oleh kemajuan fasilitas komputasi numerik saat ini.



Gambar 1.1: Ilustrasi tensor.

Laboratorium 1: skalar, vektor, matriks, dan tensor

Pada bagian ini, kita akan mencoba untuk mengimplementasikan konsep - konsep yang telah kita pelajari di lingkungan komputasi Python, khususnya dengan menggunakan pustaka numerik NumPy. Untuk itu, kita harus memastikan bahwa lingkungan yang kita gunakan telah sesuai, jalankan perintah berikut ini di Jupyter Notebook kalian masing - masing:

Memulai sesi NumPy

In [1]:
```
import sys
import numpy as np

print("Python: {}".format(sys.version))
print("NumPy: {}".format(np.__version__))
```

In [2]:
```
Python: 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0]
NumPy: 1.18.1
```

Kita dapat mendefinisikan skalar (yang mana merupakan nilai bilangan tunggal) dengan menggunakan operator `=` untuk penugasan ke suatu variabel tertentu:

In [3]:
```
## penugasan skalar
x = 128
x
```

Out[3]:
```
128
```

Dengan NumPy, kita dapat mendefinisikan vektor dengan menggunakan fungsi `array()`:

In [4]:
```
## penugasan vektor
x = np.array((1,2,8))
x
```

Out[4]:
```
array([1, 2, 8])
```

Array mempunyai beberapa atribut yang dapat memberitahu kita tentang informasi terkait dimensi dan ukuran suatu vektor:

In [5]:
```
print("Dimensi vektor: {}".format(x.shape))
print("Ukuran vektor: {}".format(x.size))
```

Out[5]:
```
Dimensi vektor: (3,)
Ukuran vektor: 3
```

Untuk mendefinisikan matriks, kita dapat menggunakan fungsi `matrix()` di pustaka NumPy. Guna memisahkan baris antar matriksnya, kita dapat menggunakan tanda `[]` dan koma:

In [6]:
```
X = np.matrix([[1,2,8],[2,2,0],[3,2,0]])
X
```

Out[6]:
```
matrix([[1, 2, 8],
        [2, 2, 0],
        [3, 2, 0]])
```

Untuk mengetahui dimensi dan ukuran matriks X, kita dapat menggunakan perintah yang sama:

In [7]:
```
print('Dimensi matriks: {}'.format(X.shape))
print('Ukuran matriks: {}'.format(X.size))
```

Out[7]:
```
Dimensi matriks: (3, 3)
Ukuran matriks: 9
```

NumPy memudahkan kita untuk mendefinisikan matriks secara cepat, misalkan pada kasus ini kita hendak mendefinisikan matriks identitas 3×3, kita tinggal menggunakan fungsi `ones()`:

In [8]:
```
X = np.ones((3,3))
X
```

Out[8]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [9]:
```
print('Dimensi matriks: {}'.format(X.shape))
print('Ukuran matriks: {}'.format(X.size))
```

Out[9]:
```
Dimensi matriks: (3, 3)
Ukuran matriks: 9
```

Sebagai manusia yang hidup di ruang tiga dimensi, tentu kita akan sangat kesulitan ketika memvisualisasikan tensor berdimensi banyak, namun NumPy mempermudah kita untuk melakukan pendefinisian ini dengan menggunakan satu baris perintah:

In [10]:
```
X = np.ones((3,3,3))
X
```

Out[10]:
```
array([[[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]])
```

In [11]:
```
print("Dimensi tensor: {}".format(x.shape))
print("Ukuran tensor: {}".format(x.size))
```

Out[11]:
```
Dimensi tensor: (3,)
Ukuran tensor: 3
```

Pengindeksan

Bagian ini mencoba untuk memfamiliarkan konvensi pengindeksan NumPy.

In [12]:
```
A = np.ones((5,5), dtype = np.int)
print(A)
```

Out[12]:
```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

In [13]:
```
## pengindeksan dimulai dari 0
A[0,1] = 2
print(A)
```

Out[13]:
```
[[1 2 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

In [14]:
```
## Penting untuk dicatat jika NumPy melakukan pengindeksan dengan konvensi baris, kolom
# Kita dapat melakukan penugasan seluruh baris atau kolom dengan menggunakan operator :
A[:,0] = 3
print(A)
```

Out[14]:
```
[[3 2 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]]
```

In [15]:
```
## Hal ini berlaku juga untuk Tensor berdimensi banyak
A = np.ones((5,5,5), dtype = np.int)

# Berikut ini contohnya
A[:,0,0] = 128
print(A)
```

Out[15]:

```
[[[128   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]]

 [[128   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]]

 [[128   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]]

 [[128   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]]

 [[128   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]
  [  1   1   1   1   1]]]
```

Operasi antar matriks

Bagian ini merupakan demo operasi penjumlahan, pengurangan, dan perkalian matriks.

In [16]:

```
A = np.matrix([[1,2],[3,4]])
B = np.ones((2,2), dtype = np.int)
```

In [17]:

```
print(A)
```

Out[17]:
```
[[1 2]
 [3 4]]
```

In [18]:
```
print(B)
```

Out[18]:
```
[[1 1]
 [1 1]]
```

In [19]:
```
## Penjumlahan antar elemen
C = A + B
print(C)
```

Out[19]:
```
[[2 3]
 [4 5]]
```

In [20]:
```
## Pengurangan antar elemen
C = A - B
print(C)
```

Out[20]:
```
[[0 1]
 [2 3]]
```

In [21]:
```
## Perkalian matriks
C = A*B
print(C)
```

Out[21]:
```
[[3 3]
 [7 7]]
```

Transpos matriks

Seperti yang telah kita pelajari, transpos digunakan untuk menukar elemen antar baris dan kolom pada suatu array. NumPy memudahkan kita untuk melakukan operasi transpos matriks secara efisien.

In [22]:
```
A = np.array(range(9))
A = A.reshape(3,3)
print(A)
```

Out[22]:
```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [23]:
```
## operasi transpos
B = A.T
print(B)
```

Out[23]:
```
[[0 3 6]
 [1 4 7]
 [2 5 8]]
```

In [24]:
```
## dengan melakukan double-transpose, maka kita
    akan mendapatkan matriks awal
C = B.T
print(C)
```

Out[24]:
```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Tensor berdimensi banyak

Seperti yang telah dijabarkan sebelumnya, NumPy mempermudah kita untuk melakukan pengolahan data pada tensor berdimensi raksasa. Berikut ini salah satu contoh pendefinisian array berdimensi 10:

In [25]:
```
A = np.ones((3,3,3,3,3,3,3,3,3,3))
```

In [26]:
```
print(A.shape)
print(len(A.shape))
print(A.size)
```

Out[26]:
```
(3, 3, 3, 3, 3, 3, 3, 3, 3, 3)
10
59049
```

Norma vektor dan matriks

Besaran suatu vektor dan matriks dapat diukur menggunakan fungsi yang dikenal sebagai norma (*norm*). Fungsi norma digunakan untuk memetakan suatu vektor ke nilai non-negatif yang menyatakan besaran dari vektor tersebut. Norma vektor $\mathbf{x}$ mengukur jarak antara titik pangkal vektor ke titik x. Di dalam dunia kecerdasan buatan norma ini banyak digunakan untuk menghitung fungsi kerugian (*loss function*) pada algoritma jaringan saraf tiruan (*neural networks*). Selain itu, algoritma pemelajaran mesin seperti jaringan pendukung vektor (*Support Vector Machine*: SVM) juga menggunakan norma L^2 untuk mengukur jarak antara diskriminan dengan masing - masing vektor pendukung (*support vector*). Formulasi umum dari norma L^P ditunjukkan pada persamaan 1.8 berikut ini:

$$\|\mathbf{x}\| = \left(\sum_i |x_i|^P\right)^{\frac{1}{P}} \tag{1.8}$$

Berikut ini adalah beberapa jenis norma yang umum digunakan di dalam penyelesaian algoritma kecerdasan buatan dan pemelajaran mesin:

Norma Euklidesan (L^2)

Ketika nilai $P = 2$, maka kita mendapatkan nilai norma Euklidesan atau dikenal juga sebagai norma L^2. Formulasi umumnya ditunjukkan pada persamaan 1.9 berikut ini:

$$\|\mathbf{x}\|_2 = \left(\sum_i |x_i|^2\right)^{\frac{1}{2}} \tag{1.9}$$

Karena begitu umum untuk digunakan, terkadang kita hanya cukup menuliskannya sebagai $\|\mathbf{x}\|$. Berikut adalah contoh penggunaan penggunaan norma Euklidesan:

$$\mathbf{x} = [3, 4]$$

$$\begin{aligned}\|\mathbf{x}\|_2 &= \sqrt{(3)^2 + (4)^2} \\ &= 5\end{aligned}$$

Nilai lima merupakan jarak vektor $\mathbf{x}$ dari koordinat awalnya.

Norma L^1

Norma ini digunakan pada kasus pembedaan antara nilai non-nol dan nilai nol pada jarak yang sangat kecil diperlukan. Di dalam penerapan algoritma jaringan saraf tiruan di mana dilakukan pembobotan dengan nilai - nilai yang sangat kecil, maka terkadang norma ini diperlukan. Formulasi umumnya ditunjukan pada persamaan 1.10 berikut ini:

$$\|\mathbf{x}\|_1 = \sum_i |x_i| \tag{1.10}$$

Norma maksimum (L^∞)

Norma L^∞ atau yang seringkali dikenal sebagai norma maksimum (*max norm*) merupakan norma yang paling banyak dijumpai di dalam kajian kecerdasan buatan dan pemelajaran mesin. Formulasi umumnya ditunjukkan pada persamaan 1.11 berikut ini:

$$\|\mathbf{x}\|_\infty = \max_i |x_i| \tag{1.11}$$

Meskipun tampak membingungkan, pada dasarnya persamaan 1.11 hanya bermakna untuk mencari nilai absolut terbesar dari elemen - elemen vektor $\mathbf{x}$.

Norma Frobenius

Besaran suatu matriks dapat diukur dengan menggunakan norma Frobenius, yang pada dasarnya merupakan norma L^2 untuk matriks. Norma Frobenius umum digunakan untuk menganalisis data raksasa seperti pada cabang keilmuan kecerdasan buatan, namun jarang digunakan di luar cabang keilmuan ini. Formulasi umumnya ditunjukkan pada persamaan 1.12 berikut ini:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} \tag{1.12}$$

Vektor dan matriks spesial

Pada algoritma kecerdasan buatan dan pemelajaran mesin, kita akan dengan mudah menjumpai beberapa jenis vektor dan matriks tertentu yang umumnya ditujukan untuk mempercepat pemrosesan data pada fase pelatihan (*training*) suatu algoritma. Beberapa vektor dan matriks yang sering kita jumpai di dalam literatur kecerdasan buatan dan pemelajaran mesin tersebut di antaranya adalah sebagai berikut:

Matriks diagonal

Suatu matriks dapat dikategorikan sebagai matriks diagonal, jika memenuhi persyaratan sebagai berikut:

$$a_{i,j} = 0 \text{ untuk seluruh } i \neq j$$

Contoh matriks diagonal adalah sebagai berikut:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Matriks diagonal sangat berguna di dalam tahap pemrosesan data, karena secara bersifat sangat efisien secara komputasi. Misalkan pada operasi perkalian diag$\mathbf{v} * \mathbf{x}$, kita hanya perlu untuk mempertimbangkan operasi antara x_i dengan v_i:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1(1) & 1(1) & 1(1) \\ 2(1) & 2(1) & 2(1) \\ 3(1) & 3(1) & 3(1) \end{bmatrix}$$

Matriks simetris

Matriks simetris merupakan matriks yang bernilai sama dengan transposnya sendiri:

$$\mathbf{A} = \mathbf{A}^T \tag{1.13}$$

Vektor satuan

Vektor satuan (*unit vector*) merupakan vektor dengan norma Euklidesan bernilai satu ($\|\mathbf{x}\|_2 = 1$). Kita dapat memperoleh vektor satuan dengan melakukan normalisasi terhadap vektor tersebut. Normalisasi tidak lain adalah proses pembagian suatu vektor dengan besarannya seperti yang ditunjukan pada persamaan 1.14 berikut ini:

$$\frac{\mathbf{x}}{\|\mathbf{x}\|_2} \tag{1.14}$$

Umumnya di dalam tahap pemrosesan data pada suatu algoritma pemelajaran mesin, kita memerlukan normalisasi vektor guna meningkatkan performa komputasi dari algoritma yang kita gunakan tersebut. Berikut ini adalah contoh penerapan normalisasi vektor:

$$\mathbf{x} = [1, 1, 1, 1]$$

$$\begin{aligned} \text{normalisasi} \rightarrow & \frac{\mathbf{x}}{\sqrt{(1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2}} \\ &= \frac{\mathbf{x}}{\sqrt{4}} \\ &= \left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right] \end{aligned}$$

Ortogonalitas

Vektor $\mathbf{x}$ dan $\mathbf{y}$ dinyatakan ortogonal satu dengan yang lain, jika memenuhi syarat:

$$\mathbf{x}^T\mathbf{y} = 0 \tag{1.15}$$

Secara intuitif, jika kedua buah vektor dinyatakan ortogonal dan keduanya mempunyai besaran bukan nol, maka sudut yang terbentuk antara kedua buah vektor tersebut adalah 90°. Jika kedua vektor tersebut merupakan vektor satuan, maka disebut sebagai ortonormal.

Dekomposisi eigen

Pada dasarnya ide dasar dari dekomposisi eigen adalah melakukan pemisahan dari suatu matriks ke dalam masing - masing komponen matematis penyusunnya. Maksud dari dekomposisi adalah untuk menguak informasi tentang sifat - sifat fungsional dari suatu matriks yang tidak langsung terlihat jika kita menampilkan matriks tersebut hanya sebagai susunan elemen - elemen bilangan.

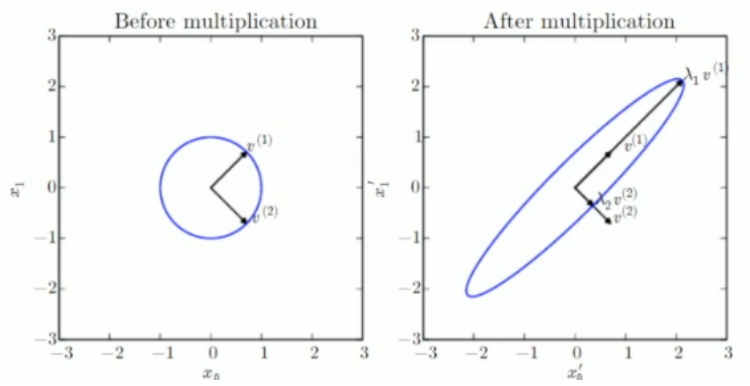
Nilai dan vektor eigen

Vektor - vektor eigen dari matriks persegi **A** merupakan vektor yang tidak bernilai nol, yang mana perkaliannya dengan matriks **A** hanya akan mengubah skala dari vektor **v** (persamaan 1.16):

$$\mathbf{Av} = \lambda\mathbf{v} \tag{1.16}$$

, dimana **v** adalah vektor - vektor eigen, dan λ merupakan nilai - nilai eigen yang berkorespondensi untuk setiap vektor eigen.

Untuk memahami pengaruh nilai dan vektor eigen pada suatu matriks perhatikanlah Gambar 1.2 berikut ini:



Gambar 1.2: Perkalian antar dua buah vektor ortonormal eigen: $\mathbf{v^1}$ dan $\mathbf{v^2}$ dengan matriks **A** mengubah lingkaran satuan (*unit circle*).

Dekomposisi eigen dimungkinkan jika matriks $\mathbf{A}$ mempunyai n vektor - vektor eigen yang bersifat independen secara linier, sehingga kita dapat membentuk sebuah matriks $\mathbf{V}$ yang dengan sebuah vektor eigen per kolomnya dan sebuah vektor λ yang memuat seluruh nilai - nilai eigen. Formulasi umum dari dekomposisi eigen untuk matriks $\mathbf{A}$ ditunjukkan pada persamaan 1.17 sebagai berikut:

$$\mathbf{A} = \mathbf{V}\text{diag}(\lambda)\mathbf{V}^{-}1 \tag{1.17}$$

Invers matriks

Invers suatu matriks ($\mathbf{A}^{-1}$) digambarkan pada persamaan 1.18 berikut ini:

$$\mathbf{A}.\mathbf{A}^{-1} = \mathbf{A}^{-1}.\mathbf{A} = \mathbf{I} \tag{1.18}$$

dimana $\mathbf{I}$ merupakan matriks identitas. Secara umum invers matriks ekuivalen dengan hubungan timbal-balik di dalam operasi perkalian di bilangan bulat.

Sifat - sifat dekomposisi eigen

Berikut adalah sifat - sifat dekomposisi eigen yang wajib kita pahami:

- Dekomposisi eigen tidak dapat diterapkan pada seluruh matriks.
- Suatu matriks dikatakan matriks singular (tidak mempunyai invers) jika terdapat salah satu nilai eigen-nya yang bernilai nol.
- Suatu matriks yang mempunyai seluruh nilai eigen positif disebut sebagai definit positif.
- Suatu matriks yang mempunyai seluruh nilai eigen negatif disebut sebagai definit negatif.

Motivasi penerapan dekomposisi eigen

Dekomposisi eigen digunakan di dalam algoritma analisis komponen utama (*Principle Component Analysis*: PCA). PCA merupakan prosedur statistik yang digunakan untuk mengubah suatu himpunan observasi dari variabel - variabel yang saling terkait menjadi suatu himpunan variabel yang secara linear tidak terkorelasi yang dinamakan komponen - komponen utama (*principle components*). Secara umum dapat dikatakan bahwa PCA merupakan suatu metode yang digunakan untuk merangkum dan mengompresi data.

Laboratorium 2: Norma dan dekomposisi eigen

Pada catatan sebelumnya, kita telah membahas tentang konsep norma pada vektor dan matriks, normalisasi vektor, dan dekomposisi eigen. Pada bagian ini , kita akan menggunakan NumPy untuk mempermudah proses komputasinya dan memperkuat pemahaman kita akan konsep - konsep tersebut.

Seperti biasa, sebelum memulai kita harus memastikan bahwa kita menggunakan lingkungan komputasi yang sama:

In [27]:
```
import sys
import numpy as np
from numpy import linalg

print('Python: {}'.format(sys.version))
print('NumPy: {}'.format(np.__version__))
```

Out[27]:
```
Python: 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0]
NumPy: 1.18.1
```

Norma vektor dan matriks

Untuk menghitung norma vektor atau matriks, kita hanya cukup menggunakan satu baris perintah dengan fungsi `linalg.norm()`:

In [28]:
```
## mendefinisikan array
A = np.arange(9) - 3
A
```

Out[28]:
```
array([-3, -2, -1,  0,  1,  2,  3,  4,  5])
```

In [29]:
```
## melakukan reshape membentuk matriks berukuran 3
    x 3
B = A.reshape((3,3))
B
```

Out[29]:
```
array([[-3, -2, -1],
       [ 0,  1,  2],
       [ 3,  4,  5]])
```

In [30]:
```
## Perhitungan Norma Euklidesan (L2)
print(np.linalg.norm(A))
print(np.linalg.norm(B))
```

Out[30]:
```
8.306623862918075
8.306623862918075
```

In [31]:
```
## Perhitungan Norma Frobenius (L2 Norm untuk
    matriks)
print(np.linalg.norm(B, 'fro'))
```

Out[31]:
```
8.306623862918075
```

In [32]:
```
## Perhitungan Norma L1
print(np.linalg.norm(A, 1))
print(np.linalg.norm(B, 1))
```

Out[32]:
```
21.0
8.0
```

In [33]:
```
## Perhitungan norma maks (P = tak hingga)
print(np.linalg.norm(A, np.inf))
print(np.linalg.norm(B, np.inf))
```

Out[33]:
```
5.0
12.0
```

Normalisasi vektor

In [34]:
```
## normalisasi untuk mendapatkan vektor satuan
norm = np.linalg.norm(A, 2)
sat_A = A / norm

print(sat_A)
```

Out[34]:
```
[-0.36115756 -0.24077171 -0.12038585  0.
   0.12038585  0.24077171
  0.36115756  0.48154341  0.60192927]
```

In [35]:
```
## norma dari vektor satuan adalah 1
np.linalg.norm(sat_A)
```

Out[35]:
```
1.0
```

Dekomposisi eigen

Kita dapat menghitung nilai dan vektor eigen dengan sangat mudah dengan menggunakan NumPy. Ingat bahwa vektor eigen dari matriks persegi $\mathbf{A}$ merupakan vektor bukan-nol $\mathbf{v}$, di mana perkalian dengan $\mathbf{A}$ hanya akan mengubah skalanya saja:

$$\mathbf{Av} = \lambda\mathbf{v}$$

Nilai skalar λ dikenal sebagai nilai eigen.

In [36]:
```
## mencari nilai dan vektor eigen untuk matriks
    persegi sederhana
A = np.diag(np.arange(1,4))
A
```

Out[36]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [37]:
```
nilai_eigen, vektor_eigen = np.linalg.eig(A)
print("Nilai - nilai eigen: {}".format(nilai_eigen)
    )
print("Vektor - vektor eigen: {}".format(vektor_
    eigen))
```

Out[37]:
```
Nilai - nilai eigen: [1. 2. 3.]
Vektor - vektor eigen: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [38]:
```
## nilai eigen w[i] berkorespondensi pada vektor
    eigen v[:, i]
print('Nilai eigen: {}'.format(nilai_eigen[1]))
print('Vektor eigen: {}'.format(vektor_eigen[:,1]))
```

Out[38]:
```
Nilai eigen: 2.0
Vektor eigen: [0. 1. 0.]
```

Kita dapat dengan mudah melakukan pengecekan kembali pada nilai dan vektor eigen ini dengan melakukan perhitungan sebagai berikut:

$$\mathbf{A} = \mathbf{V}diag(\lambda)\mathbf{V}^{-1}$$

In [39]:
```
## verifikasi - dekomposisi eigen untuk
    menghasilkan nilai A

matriks = np.matmul(np.diag(nilai_eigen), np.linalg
    .inv(vektor_eigen))
A = np.matmul(vektor_eigen, matriks).astype(np.int)
print(A)
```

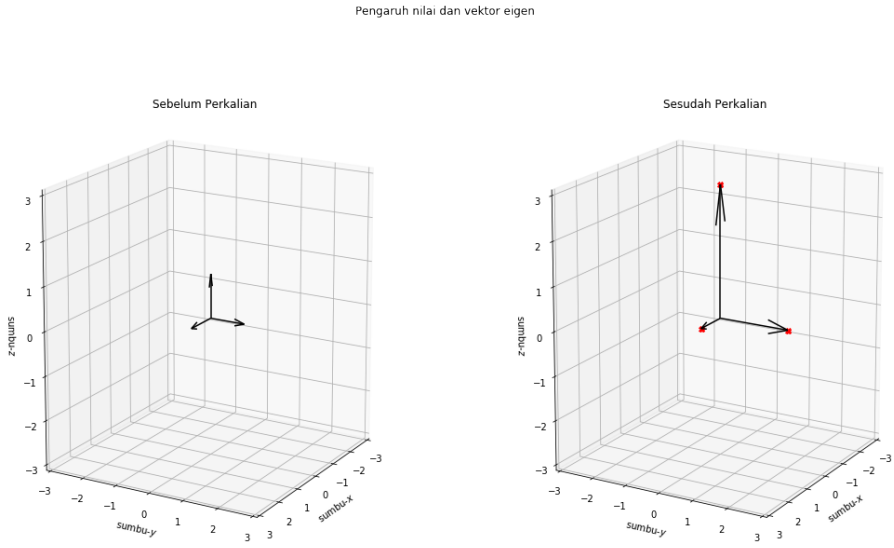
Out[39]:
```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Nilai dan vektor eigen umumnya sulit untuk dipahami secara konseptual, untuk itu pada contoh berikut ini, kita mencoba memvisualisasikan perkalian antara vektor - vektor eigen dengan matriks A dengan menggunakan pustaka matplotlib.

In [40]:
```
## mengimpor pustaka - pustaka yang diperlukan
    untuk visualisasi data
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm
\%matplotlib inline
```

In [41]:

```
## plot vektor - vektor eigen
titik_awal = [0,0,0]

fig = plt.figure(figsize=(18,10))
ax1 = fig.add_subplot(121, projection='3d')

ax1.quiver(titik_awal, titik_awal, titik_awal,
    vektor_eigen[0, :], vektor_eigen[1, :], vektor_
    eigen[2, :], color = 'k')
ax1.set_xlim([-3, 3])
ax1.set_ylim([-3, 3])
ax1.set_zlim([-3, 3])
ax1.set_xlabel('sumbu-$x$')
ax1.set_ylabel('sumbu-$y$')
ax1.set_zlabel('sumbu-$z$')
ax1.view_init(15, 30)
ax1.set_title("Sebelum Perkalian")

# perkalian matriks awal dengan vektor - vektor
    eigen
eig_baru = np.matmul(A, vektor_eigen)
ax2 = plt.subplot(122, projection='3d')

# plot vektor - vektor baru
ax2.quiver(titik_awal, titik_awal, titik_awal, eig_
    baru[0, :], eig_baru[1, :], eig_baru[2, :],
    color = 'k')

# plot nilai - nilai eigen untuk setiap vektor
ax2.plot((nilai_eigen[0]*vektor_eigen[0]), (nilai_
    eigen[1]*vektor_eigen[1]), (nilai_eigen[2]*
    vektor_eigen[2]), 'rX')
ax2.set_title("Sesudah Perkalian")
ax2.set_xlim([-3, 3])
ax2.set_ylim([-3, 3])
ax2.set_zlim([-3, 3])
ax2.set_xlabel('sumbu-$x$')
ax2.set_ylabel('sumbu-$y$')
ax2.set_zlabel('sumbu-$z$')
ax2.view_init(15, 30)

# tampilkan plot!
plt.show()
```

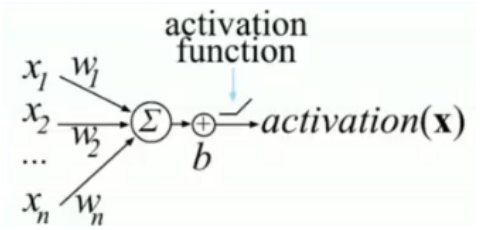


Gambar 1.3: Pengaruh nilai dan vektor eigen.

2

Kalkulus Peubah Banyak

Turunan peubah banyak lazim digunakan untuk mengoptimisasi jaringan saraf tiruan dengan menggunakan berbagai algoritma yang masuk dalam kategori penurunan gradien (*gradient descent*), utamanya dengan menggunakan algoritma penurunan gradien stokastik (*stochastic gradient descent*) (Gambar 2.1). Sementara integral digunakan untuk memahami distribusi probabilitas kontinyu dalam teori probabilitas. Bagian ini hanya akan membahas kulit luar konsep - konsep di dalam kalkulus karena sifatnya sebagai pengantar semata.



Gambar 2.1: Optimasi algoritma jaringan saraf tiruan.

Turunan

Secara geometris turunan merupakan ukuran kemiringan dari suatu kurva atau yang sering disebut sebagai garis tangen. Kemiringan ini menyatakan ukuran perubahan kurva pada suatu titik tertentu. Sedangkan secara fisis, turunan merupakan laju perubahan suatu fungsi. Turunan satu dimensi dari fungsi $y = f(x)$ dinyatakan melalui notasi sebagai berikut(persamaan 2.1):

$$\begin{aligned} f'(x) &= \frac{dy}{dx} \\ &= \frac{df(x)}{dx} \\ &= \frac{d}{dx} f(x) \end{aligned} \tag{2.1}$$

Terdapat beberapa aturan di dalam penggunaan turunan di antaranya adalah sebagai berikut:

Aturan turunan skalar

Konstanta

Suatu konstanta karena sifatnya yang tetap, tidak akan pernah mengalami perubahan, maka berlaku:

$$\frac{d}{dx} C = 0 \tag{2.2}$$

Contoh:

$$\frac{d}{dx}(128) = 0$$

Aturan pangkat

$$\frac{d}{dx} x^n = n x^{n-1} \tag{2.3}$$

Contoh:

$$\frac{d}{dx} x^3 = 3x^2$$

Perkalian

$$\frac{d}{dx}Cx^n = C\frac{d}{dx}x^n = Cx^{n-1} \tag{2.4}$$

Contoh:

$$\frac{d}{dx}(4x^3) = 4\frac{d}{dx}x^3 = 4(3)x^2 = 12x^2$$

Aturan utama turunan

Aturan penjumlahan

$$\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x) \tag{2.5}$$

Contoh:

$$\begin{aligned}\frac{d}{dx}(4x + 2x^2) &= 4\frac{d}{dx}x^2 + 2\frac{d}{dx}x^2 \\ &= 4 + 4x\end{aligned}$$

Aturan perkalian

$$\frac{d}{dx}(f(x).g(x)) = g(x)\frac{d}{dx}f(x) + f(x)\frac{d}{dx}g(x) \tag{2.6}$$

Contoh:

$$\begin{aligned}\frac{d}{dx}(x^2x) &= x\frac{d}{dx}x^2 + x^2\frac{d}{dx}x \\ &= x(2x) + x^2 \\ &= 3x^2\end{aligned}$$

Aturan rantai Aturan ini penting untuk dipahami karena banyak digunakan di dalam analisis jaringan saraf tiruan.

$$\frac{d}{dx}\left[f\left(u\right)\right] = \frac{d}{du}\left[f\left(u\right)\right]\frac{du}{dx} \tag{2.7}$$

Contoh:

$$\begin{aligned}\frac{d}{dx}\sin\left(x^2\right) &= \cos\left(x^2\right).2x \\ &= 2x\cos\left(x^2\right)\end{aligned}$$

Mengenal turunan parsial Turunan parsial digunakan untuk menyelesaikan permasalahan turunan yang melibatkan lebih dari satu peubah. Turunan parsial dua dimensi dinyatakan sebagai berikut(persamaan 2.8):

$$f'(x,y) = \nabla f(x,y) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right] \tag{2.8}$$

Contoh:

$$f(x,y) = 3x^2y$$

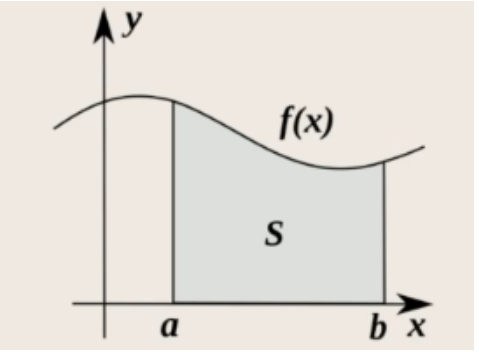
$$\nabla f(x,y) = \left[\frac{\partial(3x^2y)}{\partial x}, \frac{\partial(3x^2y)}{\partial y}\right] = \left[3y(2x), 3x^2(1)\right] = \left[6xy, 3x^2\right]$$

Integral

Konsep integral banyak digunakan untuk pemodelan distribusi probabilitas kontinyu yang digunakan dalam berbagai algoritma pemelajaran mesin. Metode integrasi umumnya digunakan untuk mencari luasan wilayah di bawah kurva (Gambar 2.2). Proses integrasi merupakan kebalikan dari diferensiasi (turunan), maka integral sering dikenal sebagai anti-turunan. Formulasi umum integral ditunjukkan pada persamaan 2.9 berikut ini:

$$\int f'(x)dx = f(x) + C \tag{2.9}$$

, dimana C merupakan konstanta integrasi.



Gambar 2.2: Ilustrasi integral untuk mencari luasan di bawah kurva.

Jenis - jenis integral

Terdapat dua jenis integral, yakni:

Integral tak-tentu

Integral tanpa batas yang diformulasikan sebagai berikut:

$$\int f(x)dx \tag{2.10}$$

Integral tentu Integral yang mempunyai batasan tertentu yang diformulasikan sebagai berikut:

$$\int_a^b f(x)dx \tag{2.11}$$

Aturan integral

Aturan pangkat

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \tag{2.12}$$

Contoh:

$$\int x^2 = \frac{x^3}{3} + C$$

Aturan konstanta

$$\int k dx = kx + C \tag{2.13}$$

Contoh:

$$\int 4 dx = 4x + C$$

Penyelesaian integral tentu

Berikut ini merupakan algoritma yang umum digunakan untuk menyelesaikan integral tentu:

$$\begin{aligned} \int_a^b f'(x)dx &= f(x)|_a^b \\ &= f(b) - f(a) \end{aligned} \tag{2.14}$$

Contoh:

$$\int_0^2 3x^2 dx = 3\int_0^2 x^2 dx = 3\left(\frac{x^3}{3}\right)\Big|_0^2 = x^3\Big|_0^2 = (2)^3 - (0)^3 = 8$$

Gradien

Gradien umum digunakan untuk mengoptimasi algoritma jaringan saraf tiruan. Konsep gradien pada dasarnya berangkat dari turunan parsial yang seperti telah bersama kita ketahui dapat diorganisasikan ke dalam bentuk vektor(persamaan 2.15 merupakan contoh implementasi gradien pada dimensi tiga).

$$\nabla f(x,y) = \left[\frac{\partial f(x,y,z)}{\partial x}, \frac{\partial f(x,y,z)}{\partial y}, \frac{\partial f(x,y,z)}{\partial z}\right] \tag{2.15}$$

Seperti juga turunan, gradien merepresentasikan derajat kemiringan dari suatu fungsi. Gradien mengarah pada arah laju perubahan terbesar pada fungsi, dan besarannya menyatakan kemiringan pada arah tersebut.

Dengan demikian gradien sangatlah cocok untuk digunakan untuk meminimalkan fungsi kerugian pada banyak algoritma pemelajaran mesin. Gradien tidak hanya terbatas pada dimensi tiga, kita dapat melakukan ekspansi pada dimensi - dimensi yang lebih besar seperti yang umum dilakukan ketika menerapkan algoritma pemelajaran mesin pada data raksasa.

Vektor gradien digunakan untuk menyusun turunan - turunan parsial dari suatu fungsi skalar. Ketika gradien diterapkan pada banyak fungsi, maka didefinisikan ke dalam bentuk matriks Jacobian. Formulasi matriks Jacobian untuk dua fungsi dengan dua buah peubah ditampilkan pada persamaan 2.16 sebagai berikut:

$$J = \begin{bmatrix} \nabla f(x,y) \\ \nabla g(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{bmatrix} \tag{2.16}$$

Formulasi umum untuk kasus dengan:

$$f(\mathbf{x}) = f(x_1, x_2, \cdots, x_n)$$

, adalah sebagai berikut (persamaan 2.17)

$$J = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \frac{\partial f_m(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{2.17}$$

Laboratorium 3: Visualisasi gradien menggunakan matplotlib

Pada bagian ini kita hendak melakukan visualisasi gradien untuk persamaan 2D.

Disini kita hendak berfokus pada pemahaman algoritma gradien, namun kita juga akan mempelajari beberapa konsep *scripting* dengan menggunakan Python. Konsep - konsep tersebut antara lain adalah penggunaan

meshgrid yang sangat berguna untuk menampilkan informasi yang berhubungan dengan titik - titik berbeda di dalam suatu *array*. Selain itu, kita juga akan mempelajari tentang pustaka matplotlib dengan menggunakan plot *quiver* dan *pcolor*.

In [42]:
```
import sys
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

print('Python: {}'.format(sys.version))
print('NumPy: {}'.format(np.__version__))
print('Matplotlib: {}'.format(matplotlib.__version__))
```

Out[42]:
```
Python: 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0]
NumPy: 1.18.1
Matplotlib: 3.2.1
```

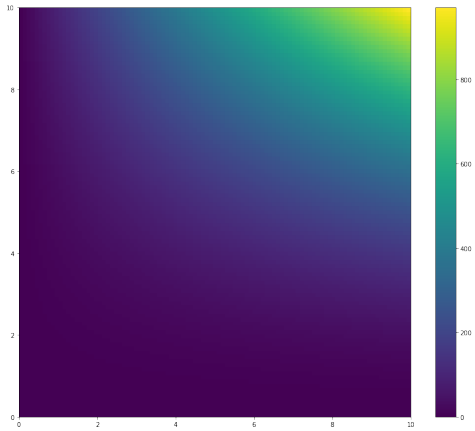
Sel di atas digunakan untuk mengimpor beberapa pustaka yang kita gunakan dalam sesi komputasi ini.

Dengan menggunakan NumPy, kita membuat sebuah meshgrid untuk setiap titik x dan y. *Meshgrid* ini merupakan *array* dua dimensi yang akan kita gunakan untuk visualisasi. Nilai z pada setiap titik x dan y dihitung dengan menggunakan fungsi sebagai berikut:

In [43]:
```
## menghasilkan meshgrid 2D
nx, ny = (100, 100)

x = np.linspace(0, 10, nx)
y = np.linspace(0, 10, ny)

xv, yv = np.meshgrid(x,y)

# mendefinisikan fungsi untuk plotting
def f(x,y):
    return x * (y**2)

# menghitung nilai z untuk setiap titik x,y
z = f(xv, yv)
```

Sekarang kita telah mempunyai *meshgrid* dan telah menghitung $f(x, y)$ untuk seluruh titik di *meshgrid*. Saatnya untuk memvisualisasikan hasilnya melalui *colormap*.

In [44]:
```
## Membuat colorplot untuk menampilkan data
plt.figure(figsize=(14,12))
plt.pcolor(xv, yv, z)
plt.colorbar()
plt.show()
```



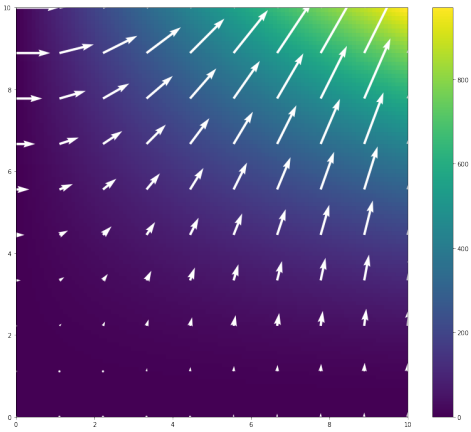
Gambar 2.3: *Colorplot* 2D dari fungsi $f(x, y) = xy^2$.

Sekarang saatnya kita menambahkan gradien ke dalam plot tersebut. Kita tidak akan menghitung gradien untuk setiap titik di dalam grafik ini, kita akan mendefinisikan *meshgrid* baru dengan titik - titik yang lebih sedikit. Kita dapat memanfaatkan fungsi `gradient()` di dalam pustaka NumPy untuk menghitung gradien pada setiap titik. Di sini kita harus berhati - hati karena prinsip kerja NumPy berbasis *array*, maka luaran dari fungsi ini berupa baris, kolom bukan dalam format x, y.

In [45]:
```
## membuat meshgrid 2D untuk gradien
nx, ny = (10, 10)
x = np.linspace(0, 10, nx)
y = np.linspace(0, 10, ny)
xg, yg = np.meshgrid(x,y)

# menghitung gradien untuk fungsi f(x,y)
# Catatan: NumPy menghasilkan luaran dalam format
    baris (y), kolom(x)
Gy, Gx = np.gradient(f(xg, yg))
```

Kemudian kita memvisualisasikan gradien menggunakan *quiverplot* (Gambar 2.4). Arah gradien direpresentasikan melalui anak panah, sedangkan besarannya direpresentasikan oleh panjang panah.

In [46]:
```
## Memvisualisasikan gradien dengan colorplot
plt.figure(figsize=(14,12))
plt.pcolor(xv, yv, z)
plt.colorbar()
plt.quiver(xg, yg, Gx, Gy, scale = 1000, color = 'w')
plt.show()
```



Gambar 2.4: Gradien dari fungsi $f(x, y) = xy^2$.

In [47]:
```
## Memvisualisasikan gradien dengan colorplot
plt.figure(figsize=(14,12))
plt.pcolor(xv, yv, z)
plt.colorbar()
plt.quiver(xg, yg, Gx, Gy, scale = 1000, color = 'w')
#plt.title('Gradient of f(x,y) = xy^2')
plt.show()
```

Plot di atas nampak sempurna. Anak - anak panah-nya nampak mengarah ke titik maksimum dan besarannya sesuai dengan kemiringan di setiap lokasi. Namun bagaimana kita bisa tahu kalau NumPy telah melakukan kalkulasi dengan benar? Untuk itu kita harus mencari turunan parsial

melalui persamaan:

$$\nabla f(x,y) = \begin{bmatrix} \frac{d}{dx} f(x,y) & \frac{d}{dy} f(x,y) \end{bmatrix}$$

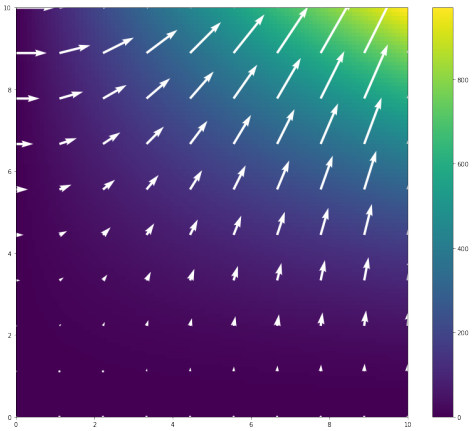
$$\nabla f(x,y) = \begin{bmatrix} y^2 & 2xy \end{bmatrix}$$

Dengan mendefinisikan fungsi sebagai berikut:

In [48]:
```
## menghitung gradien fungsi: f(x,y) = xy^2
def ddx(x,y):
    return y ** 2

def ddy(x,y):
    return (2 * y * x)

Gx = ddx(xg,yg)
Gy = ddy(xg,yg)
```

Kemudian kita memvisualisasikannya (Gambar 2.5):

In [49]:
```
## Plot
plt.figure(figsize=(14,12))
plt.pcolor(xv, yv, z)
plt.colorbar()
plt.quiver(xg, yg, Gx, Gy, scale = 1000, color = 'w
    ')
plt.show()
```



Gambar 2.5: Visualisasi $[y^2, 2xy]$.

Optimasi

Pada bagian ini kita akan menggabungkan beberapa konsep yang telah kita pelajari pada bagian - bagian sebelumnya untuk memahami permasalahan optimasi pada fungsi konveks satu dimensi. Di dunia nyata tentunya kita akan disuguhkan oleh perkara optimasi yang melibatkan banyak dimensi, hanya karena buku ini hanya berupa pengantar matematis singkat, maka yang dibahas hanya pada perkara optimasi fungsi konveks berdimensi tunggal.

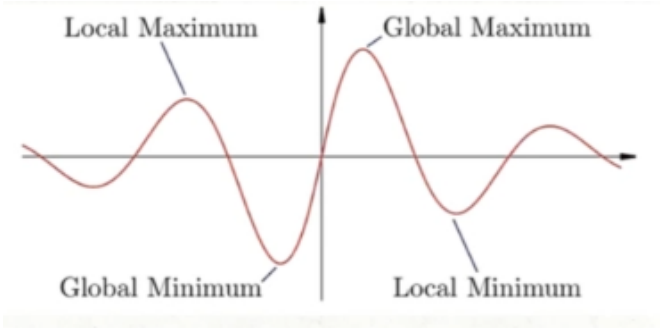
Persamaan 2.18 menunjukkan formulasi umum dari optimasi fungsi konveks:

$$\begin{aligned} \min_{x} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, i = \{i, \cdots, k\}, \\ & h_j(x) = 0, j = \{1, \cdots, k\} \end{aligned} \tag{2.18}$$

Berikut adalah beberapa terminologi yang wajib kita pahami:

- $f_0(x)$: merupakan fungsi objektif yang hendak kita minimalkan (dalam kasus pemelajaran mesin merupakan fungsi kerugian).
- x: merupakan peubah objektif, umumnya di dalam algoritma jaringan saraf tiruan merupakan vektor masukan yang hendak kita *update*.
- $f_i(x)$ dan $h_j(x)$ merupakan fungsi - fungsi kendala (*constraint functions*) yang bentuknya dapat beragam.

Optimasi digunakan untuk mencari titik minimum global (Gambar 2.6) dari suatu fungsi objektif yang mana harus memenuhi beberapa fungsi kendala.



Gambar 2.6: Visualisasi titik - titik kritis dari suatu fungsi objektif.

Jika fungsi objektif berupa fungsi konveks (cembung atau cekung), maka permasalahan optimasinya juga dikenal sebagai optimasi konveks. Pada fungsi konveks titik minimum yang hendak kita optimasi sudah pasti berupa titik minimum global.

Optimasi merupakan topik yang sangat penting di dalam analisis data karena hampir digunakan pada seluruh algoritma pemelajaran mesin, beberapa di antaranya adalah:

- **Klasifikasi**:

$$\min_{w} \sum_{i=1} \log(1 + \exp(-y_i x_i^T w) \tag{2.19}$$

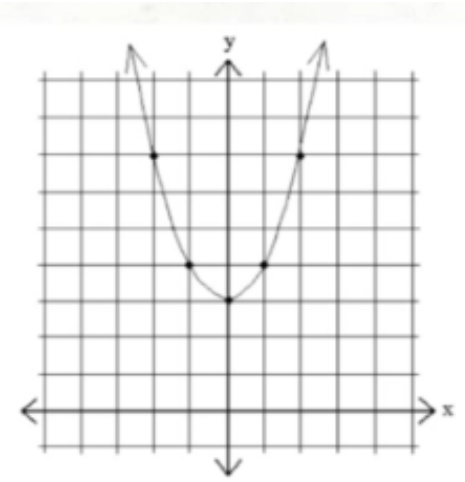
- **K-means**:

$$\min_{\mu_1,\cdots,\mu_k} J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2 \tag{2.20}$$

- **Regresi logistik**:

$$\min_{w} \|X_w - y\|^2 \tag{2.21}$$

Untuk lebih memahami persoalan optimasi ini kita akan mencoba menggunakan konsep turunan untuk mencari titik - titik kritis dari fungsi objektif $y = x^2 + 3$ (Gambar 2.7) dengan menggunakan turunan pertama dan kedua.



Gambar 2.7: Fungsi $y = x^2 + 3$.

Yang pertama kali harus kita lakukan adalah mencari titik kritis dengan mengatur turunan pertama sama dengan nol:

$$y = x^2 + 3$$

$$y' = 2x$$

$$0 = 2x$$

$$x = 0$$

Maka kita telah mengetahui titik kritis globalnya (karena fungsi ini merupakan fungsi konveks, maka seluruh titik kritisnya berlaku global). Untuk menguji apakah titik ini titik maksimum atau minimum, kita perlu melakukan penurunan kedua, yakni:

$$y'' = 2$$

Jika penurunan kedua bersifat positif, maka titik tersebut merupakan titik minimum, sedangkan jika bersifat negatif maka titik tersebut merupakan titik maksimum. Dalam kasus ini, maka titik minimum global dari fungsi $y = x^3 + 3$ adalah $x = 0$.

Pencarian titik minimum global secara matematis tersebut umumnya tidak berlaku dalam konteks pemelajaran mesin. Karena volume data yang berjumlah sangat besar, maka diperlukan proses iteratif dengan menggunakan gradien. Jika gradien bernilai positif, maka kita menggeser ke arah lain hingga mendapatkan gradien yang bernilai negatif, dan hal ini dilakukan secara berulang hingga menemukan titik minimum yang dicari. Berikut ini contoh kasus yang mungkin dilakukan oleh algoritma pemelajaran mesin untuk mencari titik minimum:

1. Tebakan pertama: $x = 3$ merupakan titik minimum.

2. Evaluasi $f(x)$:
$$y = (3)^2 + 3 = 12$$

3. Evaluasi nilai dari turunan:
$$y' = 2(3) = 6$$
Ternyata gradien bernilai positif, maka kita harus mengubah arah tebakan.

4. Tebakan kedua: $x = -1$ merupakan nilai minimum.

5. Evaluasi $f(x)$:
$$y = (-1)^2 + 3 = 4$$

6. Evaluasi nilai dari turunan:
$$y' = 2(-1) = -2$$
Ternyata tebakannya *overshot*, maka kita ubah lagi arah tebakan.

7. Begitu seterusnya hingga kita menemukan nilai $x = 0$ sebagai titik minimum.

Proses ini dinamakan sebagai optimasi gradien stokastik. Pada algoritma jaringan saraf tiruan, jarak yang digunakan untuk mengubah arah tebakan umumnya tidak sebesar contoh di atas (dari 3 ke -2), melainkan sangatlah kecil (dari 3 ke 2,99 misalnya). Oleh karena volume data yang umumnya sangat besar, proses iterasi ini dapat berlangsung hingga ribuan, bahkan jutaan kali.

Di dalam dunia nyata tentu tidak seluruh fungsi objektif yang hendak kita optimasi merupakan fungsi konveks, sehingga kita harus memperhitungkan juga titik - titik kritis yang bersifat lokal.

3

Teori Probabilitas

Pendahuluan

Teori probabilitas digunakan untuk mengkuantifikasikan ketidakpastian. Teori probabilitas menyediakan aksioma - aksioma yang dapat digunakan untuk menurunkan pernyataaan - pernyataan matematis tentang ketidakpastian.

Di bidang analisis data sendiri, hukum - hukum probabilitas digunakan sebagai fondasi tentang bagaimana suatu sistem algoritma menimbang permasalahan dan memberikan keputusan untuk menyelesaikan permasalahan tersebut. Dapat dikatakan bahwa teori probabilitas merupakan batu penjuru dari kemampuan dari suatu sistem kecerdasan buatan untuk dapat berpikir dan menimbang permasalahan selayaknya manusia biasa yang kesehariannya tidak luput dari ketidakpastian.

Terdapat tiga macam sumber ketidakpastian ketika kita hendak menganalisis data, yakni:

- Ketidakpastian stokastik yang bersifat inheren, yakni secara natural sistem yang hendak kita modelkan diatur oleh hukum - hukum alam yang bersifat tidak pasti. Contoh yang paling terkenal adalah pemodelan mekanika kuantum.

- Ketidaklengkapan hasil observasi. Dalam kasus ini kita tidak mempunyai seluruh data yang hendak dimodelkan. Jenis ketidakpastian inilah yang utamanya sering dijumpai ketika kita melakukan penganalisisan data. Misalnya, ketika kita hendak menerapkan algoritma pengenalan objek - objek lalulintas pada mobil swakemudi, tentu kita hanya melatih sistem tersebut dengan ratusan (atau bahkan ribuan) objek - objek fotografi rambu - rambu lalulintas, sedangkan pada kenyataannya mobil swakemudi ini harus berhadapan dengan seluruh rambu - rambu lalulintas di seluruh jalan raya di Indonesia.

- Ketidaklengkapan pemodelan. Seringkali karena keterbatasan sumberdaya komputasi kita hanya memodelkan beberapa fitur penting saja, sehingga beberapa fitur yang mendetail sengaja kita abaikan karena keterbatasan sumberdaya komputasi. Salah satu contohnya adalah pemodelan iklim global yang mungkin saja gagal menangkap fenomena regional seperti sistem monsun.

Oleh karena tiga ketidakpastian tersebut, kita membutuhkan teori probabilitas untuk memodelkan data yang telah kita peroleh.

Berdasarkan kesepakatan para ilmuwan, teori probabilitas terbagi ke dalam dua mazhab utama, yakni:

- Probabilitas frekuensi, yang mana lebih menekankan pada frekuensi terjadinya suatu kejadian tertentu. Contohnya adalah seberapa sering kita memperoleh dadu dengan jumlah enam ketika dua kali melemparkan dadu.

- Probabilitas Bayesian, yang mana lebih menekankan pada derajat kepercayaan. Contohnya adalah ketika dokter menyatakan seorang pasien mempunyai 40% peluang terjangkit COVID-19.

Distribusi - distribusi probabilitas

Peubah acak

Peubah acak (*random variable*) merupakan suatu peubah yang dapat memperoleh nilai secara acak dari suatu himpunan nilai. Peubah acak X dapat memperoleh nilai dari himpunan bilangan $\{x_1, x_2, x_3, \cdots, x_n\}$. Distribusi probabilitas sendiri bertugas untuk menentukan seberapa besar peluang suatu nilai di dalam himpunan bilangan yang dimaksud untuk terpilih sebagai peubah acak. Terdapat dua jenis peubah acak, yakni:

- **Diskrit**, yang mana mempunyai nilai - nilai tertentu (bersifat finit), dan
- **Kontinyu**, yang mana himpunan kandidat peubah acak tersebut merupakan bilangan riil yang bersifat kontinyu.

Fungsi massa peluang

Distribusi probabilitas untuk peubah acak diskrit dikenal sebagai fungsi massa peluang (*Probability Mass Function*: PMF). Terdapat tiga kriteria yang menyatakan bahwa suatu fungsi dapat dikategorikan sebagai PMF:

- Domain dari P harus memenuhi setiap nilai dari x.
- $0 \leq P(X) \leq 1$
- $\Sigma P(X) = 1$, oleh karena itu distribusi probabilitas pada PMF selalu ternormalisasi.

Terdapat dua jenis PMF, yakni:

- Distribusi probabilitas gabungan, yang mana merupakan PMF yang berlaku pada banyak peubah. $P(X = x, Y = y)$ berlaku untuk setiap $X = x$ dan $Y = y$ secara bersamaan.

- Distribusi seragam, pada distribusi ini seluruh elemen di dalam himpunan peubah acak mempunyai peluang yang sama untuk terpilih. Berikut adalah formulasi umumnya (persamaan 3.1):

$$P(X = x_i) = \frac{1}{k} \tag{3.1}$$

Untuk kasus PMF sendiri akan lebih mudah dihitung karena mempunyai nilai - nilai yang bersifat diskrit:

$$\sum_i P(X = x_i) = \sum_i \frac{1}{k} = \frac{k}{k} = 1$$

Fungsi kepadatan peluang

Fungsi kepadatan peluang (*Probability Density Function*: PDF) banyak dijumpai di dalam riset - riset tentang kecerdasan buatan. PDF berlaku pada peubah acak yang bersifat kontinyu. Terdapat tiga kriteria yang menyatakan bahwa suatu fungsi dapat dikategorikan sebagai PMF:

- Domain dari p harus memenuhi setiap nilai dari x.
- $p(x) \geq 0$
- $\int p(x)dx = 1$

Probabilitas marjinal

Merupakan distribusi probabilitas bagian dari seluruh peubah. Jenis probabilitas ini banyak dijumpai di dalam kegiatan analisis data karena umumnya kita tidak mempunyai seluruh peubah yang dibutuhkan untuk implementasi pemelajaran mesin. Berikut ini formulasi umum probabilitas marjinal:

- Peubah acak diskrit:

$$P(X = x) = \sum_y P(X = x, Y = y) \tag{3.2}$$

- Peubah acak kontinyu:

$$p(x) = \int p(x, y)dy \tag{3.3}$$

Probabilitas bersyarat

Probabilitas bersyarat adalah probabilitas dari suatu kejadian yang bergantung pada kejadian lainnya. Formulasi umumnya dapat dilihat pada persamaan 3.4:

$$P(Y = y|X = x) = \frac{P(Y = y, X = x)}{P(X = x)} \tag{3.4}$$

Ekspektasi, varian, dan kovarian

Ekspektasi

Ekspektasi dari suatu fungsi $f(x)$ dengan mempertimbangkan distribusi probabilitas $P(x)$ merupakan nilai rata - rata ketika kita melakukan eksperimen terhadap x pada distribusi P. Berikut adalah formulasi umum ekspektasi pada:

- Peubah acak diskrit:

$$E_{x\sim\ p}\left[f(x)\right] = \sum_{x} P(x)f(x) \tag{3.5}$$

- Peubah acak kontinyu:

$$E_{x\sim\ p}\left[f(x)\right] = \int p(x)f(x)dx \tag{3.6}$$

Varian dan standar deviasi

Secara formal varian (σ^2) berarti ekspektasi dari kuadrat deviasi peubah acak dari rata - ratanya. Secara informal varian dapat dikatakan sebagai

ukuran seberapa jauh bilangan acak yang diambil dari distribusi probabilitas $P(x)$ menyebar dari nilai rata - ratanya. Berikut merupakan formulasi umum varian (persamaan 3.7):

$$\text{Var}(f(x)) = E\left[(f(x) - E[f(x)])^2\right] \tag{3.7}$$

Sementara itu, standar deviasi (σ) tidak lain merupakan akar kuadrat dari varian.

Kovarian

Kovarian merupakan ukuran keterkaitan linier antara dua peubah acak (persamaan 3.8):

$$\text{Cov}(f(x), g(y)) = E\left[f(x) - E[f(x)]\right](g(y) - E[g(y)]) \tag{3.8}$$

Kovarian digunakan untuk mengukur dependensi linier antar peubah.

Matriks kovarian

Matriks kovarian dari vektor acak $\mathbf{x}$ merupakan matriks $n \times n$, di mana berlaku:

$$\text{Cov}(\mathbf{x})_{i,j} = \text{Cov}(x_i, x_j) \tag{3.9}$$

Elemen - elemen pada diagonal matriks kovarian merupakan varian:

$$\text{Cov}(x_i, x_i) = \text{Var}(x_i) \tag{3.10}$$

Ketika berbicara mengenai varian di dalam ranah pemelajaran mesin, maka umumnya kita akan mengacu matriks kovarian.

Laboratorium 4: Visualisasi distribusi probabilitas

Pastikan kalian memulai laboratorium ini dengan memilih kernel R pada Jupyter Notebook kalian masing - masing.

Contoh yang akan kita gunakan di dalam kegiatan kali ini adalah tentang IQ siswa dengan rata - rata: 100 dan standar deviasi: 15. Yang menjadi pertanyaan adalah berapa persen siswa dengan IQ > 115?

Mendefinisikan parameter untuk distribusi normal:

```
rata2 <- 100
std <- 15
```

Mendefinisikan batas atas dan batas bawah *Region of Interest* (ROI):

```
bawah <- 115
atas <- Inf
```

Membuat sikuen bilangan dan distribusi normal:

```
x <- seq(-4,4, length = 100) * std + rata2
prob <- dnorm(x, rata2, std)
```

Memvisualisasikan distribusi probabilitas dan menambahkan label sumbu:

```
plot(x,prob,
     type="n",
     xlab="Skor IQ",
     ylab="p(x)",
     main="Distribusi normal skor IQ", axes=FALSE)
lines(x,prob)
axis(1, at=seq(40, 160, 20), pos=0)
```

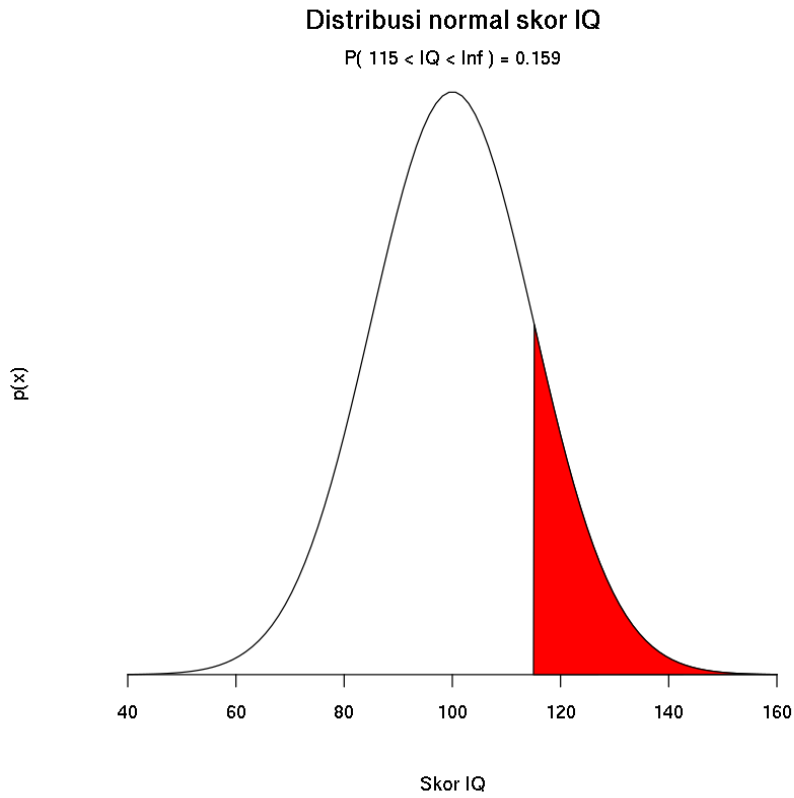
Mendefinisikan poligon untuk ROI:

```
plot(x,prob, type="n", xlab="Skor IQ",
     ylab="p(x)",
     main="Distribusi normal skor IQ",
     axes=FALSE)
```

```
lines(x,prob)
axis(1, at=seq(40, 160, 20), pos=0)
i <- x >= bawah & x <= atas
polygon(c(bawah,x[i],atas), c(0,prob[i],0), col="red")
```

Menghitung luasan wilayah di bawah kurva ROI dan menampilkan hasilnya (Gambar 3.1):

```
plot(x,prob, type="n", xlab="Skor IQ", ylab="p(x)",
     main="Distribusi normal skor IQ", axes=FALSE)
lines(x,prob)
axis(1, at=seq(40, 160, 20), pos=0)
i <- x >= bawah & x <= atas
polygon(c(bawah,x[i],atas), c(0,prob[i],0), col="red")

area <- 1 - pnorm(bawah, rata2, std)
hasil <- paste("P(",bawah,"< IQ <",atas,") =",
               signif(area, digits=3))
mtext(hasil,3)
```



Gambar 3.1: Distribusi normal IQ siswa.

Laboratorium 5: Implementasi matriks kovarian di lingkungan R

Kita akan membuat matriks kovarian di R dengan dua cara, yakni dengan membuat dari awal dan menggunakan fungsi bawaan (*built-in*) dari R.

Mendefinisikan vektor kolom:

```
a <- 1:6
b <- seq(1, 11,by=2)
c <- seq(10, 60, by=10)
d <- c(2, 5, 5, 2, 1, 0)
e <- c(4, 5, 6, 7, 8, 9)
```

Membuat matriks dari vektor - vektor di atas:

```
M <- cbind(a,b,c,d,e)
k <- ncol(M)
n <- nrow(M)
print(M)
```

```
     a  b  c d e
[1,] 1  1 10 2 4
[2,] 2  3 20 5 5
[3,] 3  5 30 5 6
[4,] 4  7 40 2 7
[5,] 5  9 50 1 8
[6,] 6 11 60 0 9
```

Mencari rata - rata (ekspektasi) untuk setiap kolom:

```
k_rata2 <-  matrix(data = 1, nrow = n) %*%
cbind(mean(a), mean(b), mean(c), mean(d), mean(e))
print(k_rata2)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  3.5    6   35  2.5  6.5
[2,]  3.5    6   35  2.5  6.5
[3,]  3.5    6   35  2.5  6.5
[4,]  3.5    6   35  2.5  6.5
[5,]  3.5    6   35  2.5  6.5
[6,]  3.5    6   35  2.5  6.5
```

Mendefinisikan perbedaan antar matriks:

```
diffM <- M - k_rata2
print(diffM)
```

```
        a  b   c    d    e
[1,] -2.5 -5 -25 -0.5 -2.5
[2,] -1.5 -3 -15  2.5 -1.5
[3,] -0.5 -1  -5  2.5 -0.5
[4,]  0.5  1   5 -0.5  0.5
[5,]  1.5  3  15 -1.5  1.5
[6,]  2.5  5  25 -2.5  2.5
```

Mendefinisikan matriks kovarian:

```
Mkovar <- (n-1)^-1 * t(diffM) %*%
diffM # kovarian sampel
print(Mkovar)
```

```
     a  b   c     d    e
a  3.5  7  35  -2.5  3.5
b  7.0 14  70  -5.0  7.0
c 35.0 70 350 -25.0 35.0
d -2.5 -5 -25   4.3 -2.5
e  3.5  7  35  -2.5  3.5
```

Mencari variansi dari matriks kovarian:

```
var <- diag(Mkovar)
print(var)
```

```
    a     b     c     d     e
  3.5  14.0 350.0   4.3   3.5
```

Kita dapat menggunakan fungsi *built-in* `cov()` untuk mendefinisikan matriks kovarian di lingkungan R:

```
print(cov(M))

     a  b   c     d    e
a  3.5  7  35  -2.5  3.5
b  7.0 14  70  -5.0  7.0
c 35.0 70 350 -25.0 35.0
d -2.5 -5 -25   4.3 -2.5
e  3.5  7  35  -2.5  3.5
```

Distribusi - distribusi spesial

Terdapat beberapa distribusi probabilitas yang umum dijumpai di hampir seluruh literatur tentang pemelajaran mesin, di antaranya adalah:

Distribusi Bernoulli

Merupakan suatu distribusi untuk peubah acak diskrit tunggal, dengan formulasi umum sebagai berikut:

$$\begin{aligned} P(X = 1) &= \phi \\ P(X = 0) &= 1 - \phi \end{aligned} \tag{3.11}$$

Distribusi ini dapat dikembangkan dengan melibatkan banyak peubah acak menjadi distribusi multinomial.

Distribusi multinomial

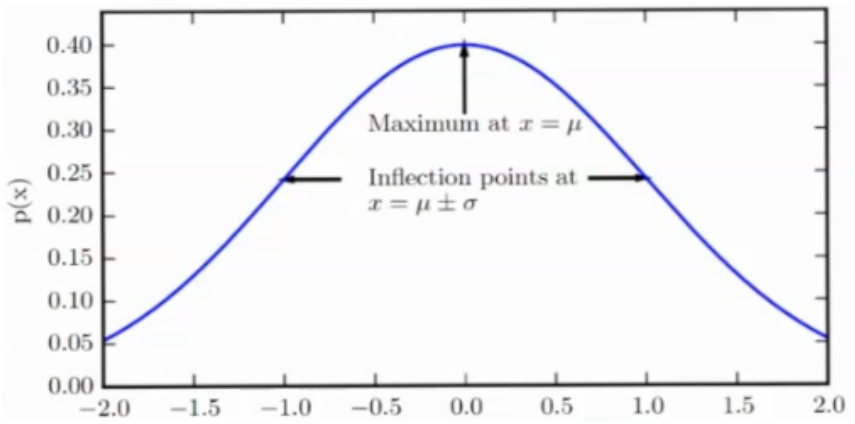
Distribusi yang melibatkan banyak peubah acak diskrit yang mana merupakan pengembangan dari distribusi Bernoulli. Kedua distribusi ini dapat dikatakan bisa memodelkan hampir seluruh distribusi probabilitas diskrit.

Distribusi Gaussian

Distribusi Gaussian (Gambar 3.2) juga dikenal sebagai distribusi normal. Distribusi ini merupakan distribusi untuk peubah acak kontinyu yang dapat

memodelkan hampir 90% permasalahan statistik yang melibatkan bilangan riil. Oleh karenanya distribusi ini merupakan bagian integral dari hampir seluruh algoritma pemelajaran mesin yang melibatkan peubah acak kontinyu. Formulasi umum dari distribusi ini ditampilkan pada persamaan 3.12:

$$\mathcal{N}(x;\, \mu,\, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{3.12}$$



Gambar 3.2: Distribusi normal.

Terdapat tiga buah aturan penting terkait sebaran data di dalam distribusi Gaussian, di antaranya adalah:

- 68% data tersebar di dalam jangkauan $\mu \pm \sigma$.
- 95% data tersebar di dalam jangkauan $\mu \pm 2\sigma$.
- 99,7% data tersebar di dalam jangkauan $\mu \pm 3\sigma$.

Distribusi eksponensial

Merupakan distribusi probabilitas kontinyu dengan titik lancip pada $x = 0$. Formulasi umumnya ditunjukkan pada persamaan 3.13 berikut ini:

$$p(x;\, \lambda) = \lambda 1_{x \geq 0} \exp\left(-\lambda x\right) \tag{3.13}$$

Distribusi Laplace

Distribusi probabilitas dengan titik tajam pada $x = \mu$. μ di sini tidak harus berupa nilai rata - rata, melainkan titik sebarang tempat tempat titik lancip berada. Formulasi umumnya ditunjukkan pada persamaan 3.14 berikut ini:

$$\text{Laplace}(x;\, \mu,\, \gamma) = \frac{1}{2\gamma} \exp\left(\frac{-|x - \mu|}{\gamma}\right) \tag{3.14}$$

4

Latihan Pemantapan Konsep

Soal

1. Dengan menggunakan NumPy hitunglah jumlah baris dan kolom pada matriks - matriks berikut ini:

   (A) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

   (B) $\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 8 \end{bmatrix}$

   (C) $\begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$

   (D) $\begin{bmatrix} 1 & 2 \\ 8 & 1 \end{bmatrix}$

   (E) $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 2 & 1 \end{bmatrix}$

   (F) $\begin{bmatrix} 2 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 \end{bmatrix}$

2. Dengan menggunakan NumPy, hitunglah transpos pada matriks - matriks berikut ini:

   (A) $\begin{bmatrix} 6 & 5 & 4 & 2 \\ 1 & 5 & 2 & 1 \\ 7 & 0 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$

   (B) $\begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix}$

3. Dengan menggunakan NumPy, hitunglah norma L^2 dari vektor $\mathbf{x}$ sebagai berikut: $\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$

4. Carilah turunan pertama dari:

   (A) $f(x, y) = x^4 y + 2x$

   (B) $f(x, y) = 2y + 3x^2$

5. **Pilihlah jawaban yang paling tepat!**

   Arah penurunan dari fungsi objektif $f(x)$ dapat diketahui melalui:

   (A) Gradien (∇)

   (B) Gradien negatif ($-\nabla$)

   (C) Matriks Jacobian ($\mathbf{J}$)

   (D) Laplacian (∇^2)

6. **Lengkapi bagian kosong pada soal sebagai berikut:**

   Integral _____ dari $f(x)$ merupakan bilangan representasi dari wilayah di bawah kurva dari $x = a$ hingga $x = b$. Integral _____ dari $f(x)$ tidak mempunyai batas dan hasil akhirnya berupa suatu fungsi.

7. **Lengkapi bagian kosong pada soal sebagai berikut:**

   Titik kritis dari suatu fungsi konveks sudah pasti merupakan titik minimum _____.

8. Carilah titik minimum global dari fungsi objektif $f(x) = 2x^2 - 3$!

9. **Lengkapi bagian kosong pada soal sebagai berikut:**

   _____ merupakan sumber dari sifat stokastik yang melekat pada sistem yang dimodelkan, pemodelan yang tidak tepat, dan tidak nya data pengamatan.

10. **Lengkapi bagian kosong pada soal sebagai berikut:**

    Distribusi probabilitas untuk peubah acak kontinyu dinamakan sebagai _____.

11. **Lengkapi bagian kosong pada soal sebagai berikut:**

    Simbol matematis untuk rata - rata, varian, dan standar deviasi adalah: _____, _____, _____.

12. Tuliskanlah persamaan yang digunakan untuk memodelkan ekspektasi pada peubah acak diskrit!

13. **Lengkapi bagian kosong pada soal sebagai berikut:**

    Distribusi probabilitas yang umum dijumpai di alam adalah distribusi _____.

Jawaban

1.
```python
# Mengimpor NumPy
import numpy as np

# Mendeklarasikan matriks
A = np.eye(3,3)
B = np.array([[2,1,2],[1,2,8]])
C = np.array([1, 1,1])
D = np.array([[1,2], [8,1]])
E = np.array([[1,0], [0,1], [0,0], [2,1]])
F = np.array([[2,1,1,2],[1,2,2,1]])

# Jawaban
```

```
print('Jumlah baris A: {}, Jumlah kolom A: {}.'.format(np.
    shape(A)[0], np.shape(A)[1]))
print('Jumlah baris B: {}, Jumlah kolom B: {}.'.format(np.
    shape(B)[0], np.shape(B)[1]))
print('Jumlah baris C: {}, Jumlah kolom C: {}.'.format(np.
    shape(C)[0], np.shape(C)[1]))
print('Jumlah baris D: {}, Jumlah kolom D: {}.'.format(np.
    shape(D)[0], np.shape(D)[1]))
print('Jumlah baris E: {}, Jumlah kolom E: {}.'.format(np.
    shape(E)[0], np.shape(E)[1]))
print('Jumlah baris F: {}, Jumlah kolom F: {}.'.format(np.
    shape(F)[0], np.shape(F)[1]))
```

Hasilnya:

```
Jumlah baris A: 3, Jumlah kolom A: 3.
Jumlah baris B: 2, Jumlah kolom B: 3.
Jumlah baris C: 3, Jumlah kolom C: 3.
Jumlah baris D: 2, Jumlah kolom D: 2.
Jumlah baris E: 4, Jumlah kolom E: 2.
Jumlah baris F: 2, Jumlah kolom F: 4.
```

2.

```
# Mengimpor NumPy
import numpy as np

# Mendeklarasikan matriks
A = np.array([[6,5,4,2],[1,5,2,1],[7,0,2,1],[1,2,3,4]])
B = np.array([[2,1,1],[1,3,2]])

# Menghitung transpose
## A
print(A.T)
print('##################')
## B
print(B.T)
```

Hasilnya:

```
[[6 1 7 1]
```

```
 [5 5 0 2]
 [4 2 2 3]
 [2 1 1 4]]
################
[[2 1]
 [1 3]
 [1 2]]
```

3.
```python
# Mengimpor NumPy
import numpy as np

# Deklarasi vektor
x = np.arange(1,5)

# Menghitung norm L2
euNorm = np.linalg.norm(x)

# Menampilkan hasil
print('Norma Euklidesan dari vektor x adalah: {}'.format(
    euNorm))
```

Hasilnya:

```
Norma Euklidesan dari vektor x adalah: 5.477225575051661
```

4. (A) $\nabla f(x, y) = [4x^3 + 2,\ x^4]$
   (B) $\nabla f(x, y) = [6x,\ 2]$

5. **B**. Gradien negatif $(-\nabla)$

6. Integral **tentu** dari $f(x)$ merupakan bilangan representasi dari wilayah di bawah kurva dari $x = a$ hingga $x = b$. Integral **tak-tentu** dari $f(x)$ tidak mempunyai batas dan hasil akhirnya berupa suatu fungsi.

7. Titik kritis dari suatu fungsi konveks sudah pasti merupakan titik minimum **global**.

8. $f(x) = 2x^3 - 3$

   $\frac{df(x)}{dx} = 2(2)x - 4x$

   - Turunan pertama dinyatakan dalam bentuk nol:
     $0 = 4x$
     $x = 0$ (merupakan titik kritis)
   - Uji coba turunan kedua:

     $\frac{d^2 f(x)}{dx} = 4$

     Karena turunan kedua bernilai positif dan tidak ada titik - titik kritis lainnya, maka $x = 0$ dinyatakan sebagai titik minimum global.

9. **Ketidakpastian** merupakan sumber dari sifat stokastik yang melekat pada sistem yang dimodelkan, pemodelan yang tidak tepat, dan tidak nya data pengamatan.

10. Distribusi probabilitas untuk peubah acak kontinyu dinamakan sebagai **fungsi kepadatan peluang** (**probability density function**).

11. Simbol matematis untuk rata - rata, varian, dan standar deviasi adalah: μ, σ^2, dan σ.

12. $E_{x\sim\ p}\,[f(x)] = \sum_x P(x)f(x)$

13. Distribusi probabilitas yang umum dijumpai di alam adalah distribusi **Gaussian (normal)**.

Bibliografi

[1] Andriy Burkov. *The Hundred-Page Machine Learning Book*. 2019.

[2] Gilbert Strang. *Linear Algebra and Learning from Data*. Wesley-Cambridge Press, 2019.

[3] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

BERLAJAR

Everything Worth Learning