Laporan Stochastic Gradient Descent Menggunakan Dataset Iris

Gama Candra Tri K

15/378060/PA/16535

Latar Belakang

1. Dataset Iris

Kumpulan data bunga Iris atau kumpulan data **Fisher's Iris** adalah kumpulan data multivariat yang diperkenalkan oleh ahli statistik Inggris dan ahli biologi **Ronald Fisher** dalam makalahnya pada tahun 1936. Penggunaan beberapa pengukuran dalam masalah taksonomi sebagai contoh analisis diskriminan linier. Hal ini kadang-kadang disebut data **Iris Anderson** karena **Edgar Anderson** mengumpulkan data untuk mengukur variasi morfologis bunga Iris dari tiga spesies terkait. Dua dari tiga spesies dikumpulkan di Semenanjung Gaspé "semua dari padang rumput yang sama, dan dipetik pada hari yang sama dan diukur pada waktu yang sama oleh orang yang sama dengan peralatan yang sama".

Kumpulan data terdiri dari 50 sampel dari masing-masing tiga spesies Iris (Iris setosa, Iris virginica dan Iris versicolor). Empat fitur diukur dari masing-masing sampel: panjang dan lebar sepal dan kelopak, dalam sentimeter. Berdasarkan kombinasi keempat fitur ini, Fisher mengembangkan model diskriminan linier untuk membedakan spesies satu sama lain.

2. Stochastic Gradient Descent

Estimasi statistik dan pembelajaran mesin mempertimbangkan masalah meminimalkan fungsi tujuan yang memiliki bentuk penjumlahan:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

Dalam statistik klasik, masalah penjumlahan minimal muncul dalam kuadrat terkecil dan estimasi kemungkinan maksimum (untuk observasi independen). Kelas umum estimator yang muncul sebagai minimizer jumlah disebut estimator-M. Namun, dalam statistik, telah lama mengakui bahwa membutuhkan bahkan minimisasi lokal terlalu ketat untuk beberapa masalah estimasi kemungkinan maksimum. Oleh karena itu, teori statistik kontemporer sering menganggap titik stasioner dari fungsi kemungkinan (atau nol dari turunannya, fungsi skor, dan persamaan memperkirakan lainnya).

Bila digunakan untuk meminimalkan fungsi di atas, metode gradien standar (atau "batch") akan melakukan iterasi berikut:

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

dimana *eta* adalah step atau learning rate di machine learning

Dalam banyak kasus, fungsi peubah memiliki bentuk sederhana yang memungkinkan evaluasi murah dari jumlah-fungsi dan jumlah gradien. Misalnya, dalam statistik, satu keluarga eksponensial satu parameter memungkinkan evaluasi fungsi dan evaluasi gradien yang ekonomis.

Namun, dalam kasus lain, mengevaluasi jumlah-gradien mungkin memerlukan evaluasi mahal dari gradien dari semua fungsi peubah. Ketika set pelatihan sangat besar dan tidak ada rumus sederhana, mengevaluasi jumlah gradien menjadi sangat mahal, karena mengevaluasi gradien memerlukan evaluasi semua gradien fungsi summand. Untuk menghemat biaya komputasi pada setiap iterasi, stochastic gradien keturunan sampel subset dari fungsi peubah di setiap langkah. Hal ini sangat efektif dalam kasus skala besar masalah pembelajaran mesin.

Implementasi

Dataset yang digunakan dalam percobaan kali ini adalah data bunga Iris yang dikumpulkan oleh R. A. Fisher yang tersedia secara gratis di https://archive.ics.uci.edu/ml/datasets/iris

Data terdiri dari 5 atribut, yaitu:

1. panjang sepal (dalam cm)
2. lebar sepal (dalam cm)
3. panjang petal (dalam cm)
4. lebar petal (dalam cm)
5. kelas, yaitu : *Iris-setosa*, *Iris-versicolor*, dan *Iris-virginica*

Terdapat 150 data dalam dataset, 50 data untuk masing masing kelas, urut dari *Iris-setosa* sampai *Iris-virginica*. Untuk percobaan kali ini, digunakan 100 data pertama (*Iris-setosa* dan *Iris-versicolor*). Lalu, diulang sebanyak 60x dari 100 dataset tersebut untuk digunakan sebagai *epoch*. Lalu dari masing-masing kelas diambil 10 data sebagai data pengukian untuk validasi.

Model yang digunakan pada ujicoba kali ini adalah sebuah neural network sederhana dengan sebuah input layer dengan input 4 atribut pertama pada Iris dataset dan sebuah output layer dengan output sebuah nilai biner yang menunjukan kelas dari Iris dataset (0 untuk Iris-setosa dan 1 untuk Iris-versicolor).

Model menggunakan fungsi sebagai berikut:

$$f(x)=\sum_{i=1}^{4} x_i w_i + b$$

Dengan x_i adalah atribut ke i, w_i adalah bobot untuk atribut ke i, dan b adalah bias.

Fungsi aktivasi untuk fungsi di atas adalah:

$$a(f(x))=\frac{1}{1+e^{-f(x)}}$$

Sedangkan untuk penentuan hasil akhir dari model tersebut menggunakan *threshold* di nilai 0.1, hasil akhir adalah 1 jika nilai hasil fungsi aktivasi di atas 0.1 dan 0 jika sebaliknya.

Pembelajaran dalam percobaan kali ini menggunakan fungsi *error* sebagai berikut:

$$c=(y-a)^2$$

Dengan *y* adalah kelas dari data latih dan *a* adalah hasil dari fungsi aktivasi.

Untuk menghitung kontribusi nilai *error* masing-masing bobot dan bias, digunakan rumus sebagai berikut:

$$c_{w_i}=2x_i(y-a)(1-a)a$$

untuk bobot input

$$c_b=2(y-a)(1-a)a$$

dan untuk bias.

Sedangkan untuk mengubah nilai dari bobot dan bias, digunakan rumus:

$$w_i=w_i+\alpha c_{w_i}$$

untuk bobot input

$$b=b+\alpha c_b$$

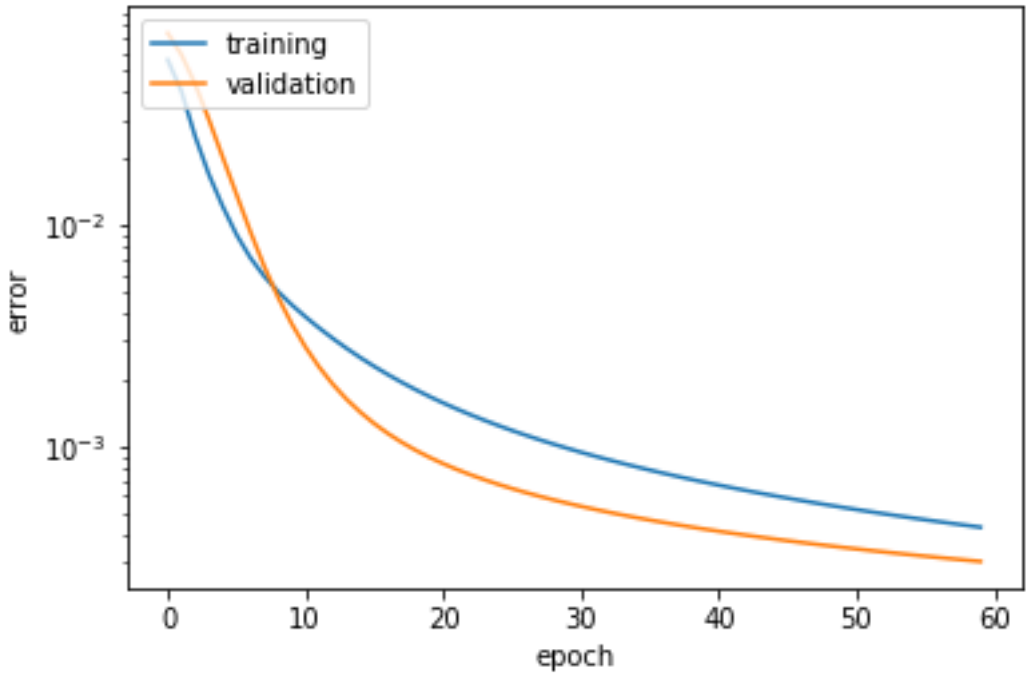
dan untuk bias.

Dengan α adalah *learning rate*.

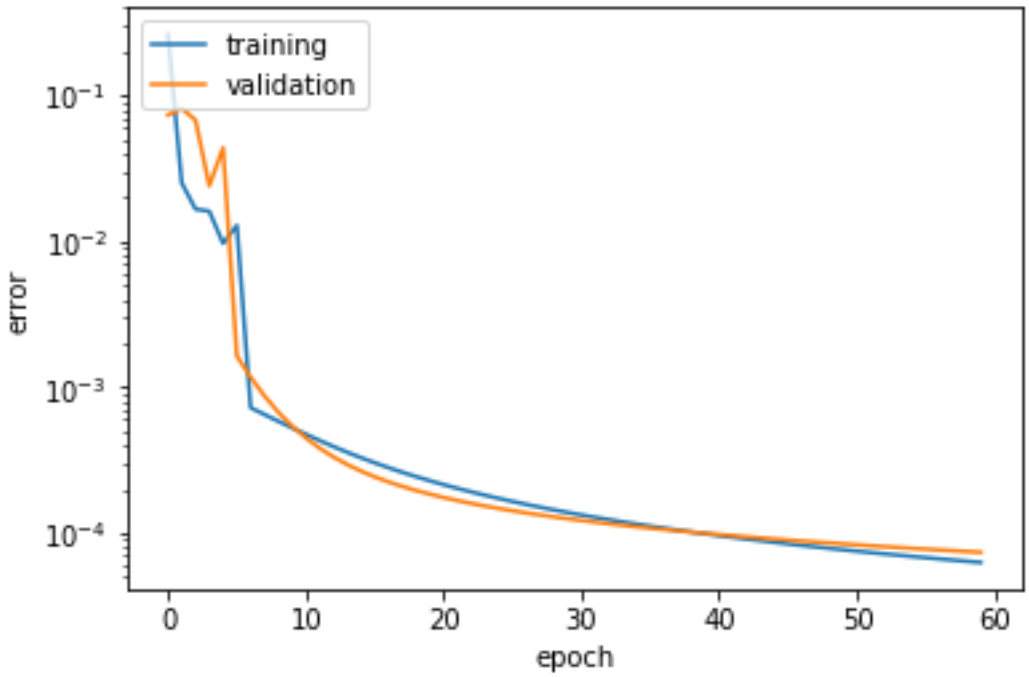
Pembelajaran dilakukan sebanyak 60 *epoch* dengan sistem *online learning*, dan dilakukan dengan nilai α = 0.1 dan α = 0.8.

Hasil

Untuk Learning rate 0.1



Untuk Learning rate 0.7



Source code

```
import math
import pandas as pd
import matplotlib.pyplot as plt

def read_data():
    data = pd.read_csv("/iris.csv", header=None)
    return data

def target_function(x, theta, bias):
    ans = 0.0
    for i in range (4):
        ans = ans + x[i] * theta[i]
    ans = ans + bias
    return ans

def error_function(prediction, result):
    return (prediction - result) ** 2

def activation_function(pre_prediction):
    return 1/(1+math.exp(-1.0 * pre_prediction))

def theta_gradient(x, theta, bias, result):
    ans = []
    for i in range(4):
        h = activation_function(target_function(x, theta,
bias))
        temp = 2 * x[i] * ( result -  h) * (1 - h) * h
        ans.append(temp)
    return ans

def bias_gradient(x, theta, bias, result):
    h = activation_function(target_function(x, theta, bias))
    temp = 2 * ( result -  h) * (1 - h) * h
    return temp

def theta_now(theta, theta_grad, learning_rate):
    ans = []
    for i in range(4):
        ans.append(theta[i] + theta_grad[i] * learning_rate)
    return ans
def bias_now(bias, bias_grad, learning_rate):
    return bias + learning_rate*bias_grad

data = read_data()
data = data.head(100)
```

```
data[4] = data[4].map({'Iris-setosa' : 0.0, 'Iris-
versicolor' : 1.0})
validation = pd.concat([data[40:50], data[90:100]])
data = pd.concat([data[0:40], data[50:90]])
theta = [0.1, 0.1, 0.1, 0.1]
bias = 0.1
error = []
v_error = []
for i in range(60):
    total_error = 0.0
    for index,row in data.iterrows():
        x = row[0:4]
        function_value = target_function(x, theta, bias)
        after_activated = activation_function(function_value)
        err = error_function(after_activated, row[4])
        total_error = total_error + err
        theta_grad = theta_gradient(x, theta, bias, row[4])
        bias_grad = bias_gradient(x, theta, bias, row[4])
        theta = theta_now(theta, theta_grad, 0.1)
        bias = bias_now(bias, bias_grad, 0.1)

    total_v_error = 0.0
    for index,row in validation.iterrows():
        x = row[0:4]
        function_value = target_function(x, theta, bias)
        after_activated = activation_function(function_value)
        v_err = error_function(after_activated, row[4])
        total_v_error = total_v_error + v_err

    print("average error epoch " + str(i+1) + " = " +
str(total_error / 100.0)
            + ". average validation error = " +
str(total_v_error / 100.0) )
    error.append(total_error / 100.0)
    v_error.append(total_v_error / 100.0)

plt.plot(error)
plt.plot(v_error)
plt.ylabel('error')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper left')
plt.yscale('log')
plt.show()
```

##Dengan nilai 0.1 sebagai learning rate

###Untuk 0.7 learning rate terdapat di github https://github.com/Saltfarmer/Pembelajaran-Mesin

Kesimpulan

Penggunaan nilai *learning rate* yang besar dapat mempercepat pembelajaran, namun dapat juga membuat pencarian bobot optimal melewati bobot yang ideal. Dalam percobaan kali ini, alpha 0.7 mendapatkan hasil yang lebih baik daripada alpha 0.1 meskipun 0.7 terkesan tidak stabil atau konsisten di awal.

Pada *epoch-epoch* awal, nilai *error* dari data validasi masih bisa di atas nilai *error* dari data latih. Kemudian, pada *epoch-epoch* setelahnya, nilai *error* dari data validasi tepat dibawah mengikuti nilai *error* data latih. Hal ini wajar, karena pada awal pembelajaran, nilai bobot dapat berfluktuasi sangat jauh sehingga memungkinkan *error* data validasi lebih besar dari *error* data latih.

Namun ketika *error* pada data validasi semakin membesar saat *epoch* sudah jauh, berarti model mulai mengalami *overfit*, sehingga bobot yang tepat sebagai bobot acuan adalah bobot ketika *error* data validasi mulai memotong *error* data latih, seperti yang ditunjukkan pada grafik aplha 0.7

Daftar Pustaka

https://github.com/Saltfarmer/Pembelajaran-Mesin
https://en.wikipedia.org/wiki/Stochastic_gradient_descent
https://en.wikipedia.org/wiki/Iris_flower_data_set
https://www.pyimagesearch.com/2016/10/17/stochastic-gradient-descent-sgd-with-python/
https://machinelearningmastery.com/linear-regression-tutorial-using-gradient-descent-for-machine-learning/
https://archive.ics.uci.edu/ml/datasets/iris