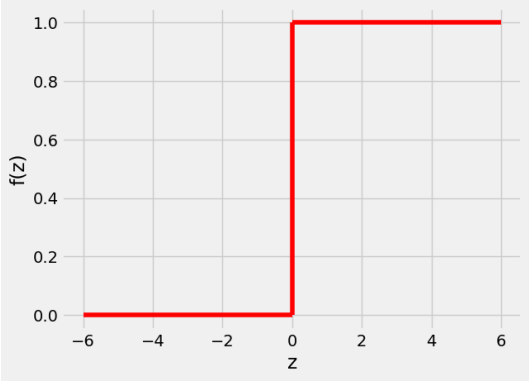
Artificial Intelligence
Kuliah 3: Regresi Logistik

Ali Akbar Septiandri
Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Al Azhar Indonesia
aliakbars@live.com

1 Klasifikasi

Bisakah hasil dari model regresi linear kita ubah untuk kasus yang lebih "sederhana"? Sederhana yang dimaksudkan di sini adalah nilai yang diprediksi tidak lagi dalam $\mathbb{R}$, tetapi hanya $\{0, 1\}$. Jadi, kita perlu fungsi yang dapat memetakan $f : \mathbb{R} \rightarrow \{0, 1\}$. Untuk melakukan hal itu, ide yang paling mudah adalah dengan menggunakan *step function* seperti pada Gambar 1.

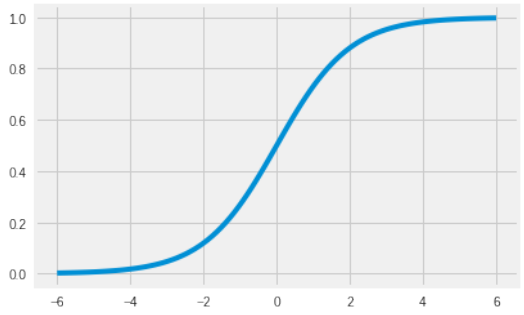


Gambar 1: Step function

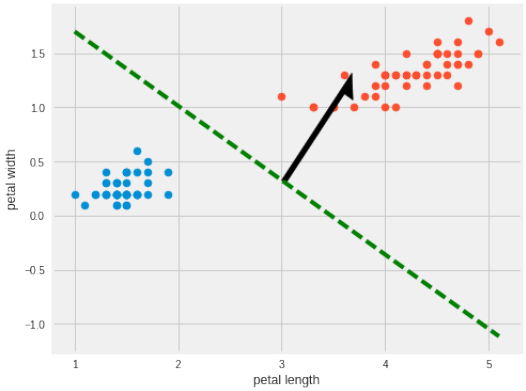
Namun, di kebanyakan kasus, kita tidak terlalu yakin dengan hasil klasifikasi tersebut. Oleh karena itu, pendekatan yang lebih baik adalah dengan memberikan nilai berupa probabilitas terhadap keluaran dari model yang dibentuk. Untuk kebutuhan ini, kita dapat menggunakan **fungsi logistik** atau **sigmoid** seperti pada Gambar 2. Dengan kata lain, fungsi regresi linear yang telah dikenakan fungsi logistik dapat dituliskan sebagai:

$$p(y = 1|\mathbf{x}) = \sigma(f(\mathbf{x})) = \sigma(\mathbf{w}^T\mathbf{x}) = \sigma(w_0x_0 + w_1x_1)$$

Nilai $\mathbf{w}$ yang dihasilkan nanti akan menghasilkan batas keputusan (*decision boundary*) yang tegak lurus terhadap vektor $\mathbf{w}$ tersebut. Contoh batas keputusan dapat dilihat pada Gambar 3. Garis putus-putus berwarna hijau merupakan batas keputusan yang tegak lurus terhadap vektor yang digambarkan sebagai garis hitam dengan panah pada gambar tersebut. Untuk dimensi yang lebih tinggi, batas keputusan yang dihasilkan akan berupa ***hyperplane***. Yang menjadi masalah sekarang adalah bagaimana cara kita dapat menemukan nilai $\mathbf{w}$?



Gambar 2: Fungsi sigmoid



Gambar 3: Batas keputusan dan vektor bobot untuk klasifikasi dua kelas

2 Optimasi

2.1 Maximum Likelihood

Seperti halnya pada kasus regresi linear, yang perlu kita lakukan untuk mendapatkan nilai $\mathbf{w}$ adalah dengan melakukan optimasi. Jadi, yang perlu didefinisikan di awal adalah nilai yang ingin kita optimasi. Dalam kasus regresi logistik, kita dapat mengoptimasi nilai *likelihood*, i.e. probabilitas untuk melihat data seperti yang kita punya jika diketahui nilai parameter $\mathbf{w}$, atau meminimalkan nilai galatnya. Nilai *likelihood* didefinisikan sebagai

$$\begin{aligned} p(\mathcal{D}|\mathbf{w}) &= \prod_{i=1}^{N} p(y = y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i=1}^{N} p(y = 1|\mathbf{x}_i, \mathbf{w})^{y_i}(1 - p(y = 1|\mathbf{x}_i, \mathbf{w}))^{1-y_i} \end{aligned}$$

dengan $\mathbf{x}$ adalah vektor input dan y adalah kelas yang ingin kita prediksi.

Untuk memudahkan optimasinya, kita dapat mengubah nilainya dengan menerapkan fungsi logaritma yang tidak mengubah sifat monotonik dari fungsi aslinya, tetapi membuat fungsinya lebih mudah untuk dicari turunannya. Dengan demikian, fungsi di atas berubah menjadi

$$\begin{aligned} L(\mathbf{w}) &= log\ p(\mathcal{D}|\mathbf{w}) \\ &= \sum_{i=1}^{N} y_i log\ \sigma(\mathbf{w}^T\mathbf{x}_i) + (1 - y_i) log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i)) \end{aligned}$$

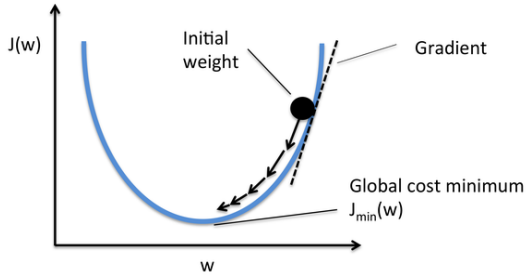
sehingga derivatif untuk setiap dimensi w menjadi

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{N}(y_i - \sigma(\mathbf{w}^T\mathbf{x}_i))x_{ij}$$

Sayangnya, tidak ada solusi tertutup untuk nilai turunan ini sehingga kita harus menggunakan **metode optimasi numerik**, e.g. *gradient descent*. Meski demikian, nilai optimum untuk kasus ini unik, i.e. *convex*.

2.2 Gradient Descent

Secara terminologi, *gradient descent* berarti menuruni (*descent*) ke arah dengan kecuraman tertinggi (*gradient*). Jika fungsi galatnya bersifat kuadratik, maka prosesnya akan seperti pada Gambar 4. Dalam kasus regresi logistik ini, nilai $J(w)$ atau $E(\mathbf{w})$ dapat diganti dengan memaksimalkan $L(\mathbf{w})$ atau meminimalkan $-L(\mathbf{w})$. Maka, *learning* sejatinya adalah menuruni permukaan fungsi galat pada kasus ini.



Gambar 4: Menuruni lembah fungsi error J(w) [1]

Dalam bentuk *pseudo-code*, metode *gradient descent* dapat ditulis sebagai

begin
  Inisialisasi $\mathbf{w}$
  while $E(\mathbf{w})$ *masih terlalu besar* **do**
    Hitung $\mathbf{g} \leftarrow \nabla_{\mathbf{w}} E(\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{g}$
  end
  return $\mathbf{w}$
end

Algorithm 1: Gradient descent

Nilai η dalam kasus tersebut dikenal sebagai laju pembelajaran atau *learning rate*. Nilai dari laju pembelajaran ini biasanya diatur agar tidak terlalu besar sehingga nilai optimal dari fungsi galat tidak terlompati, tetapi juga tidak terlalu kecil sehingga butuh waktu yang lama untuk mencapai nilai optimal.

Dalam implementasinya, metode *gradient descent* bisa menjadi sangat mahal jika kita harus mengubah nilai $\mathbf{w}$ pada setiap iterasi dengan menjumlahkan semua error (*batch*) terlebih dahulu. Oleh karena itu, kita dapat menggunakan pendekatan yang berbeda dengan menggunakan metode *online*, i.e. mengubah nilai $\mathbf{w}$ dengan mengambil nilai error dari salah satu data saja. Metode yang dikenal dengan nama *stochastic gradient descent* ini mewajibkan agar data diacak terlebih dahulu sehingga tidak terjadi *overfitting*.

References

[1] Raschka, S. 2015. Single-Layer Neural Networks and Gradient Descent. Available at: `http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html`.