


- در ابتدا، دسته‌های داده را به دو دسته **Numerical** و **Categorical** تقسیم می‌کنیم. در دسته **Numerical**، داده‌ها به دو دسته **Normal** و **Not Normal** تقسیم می‌شوند. **Perceptron** مدل است.

الگوریتم **Perceptron** که در کلاس درس مودل‌ها تکرار می‌شود.



THE PERCEPTRON LEARNING ALGORITHM

- Initialisation**
 - set all of the weights w_{ij} to small (positive and negative) random numbers
- Training**
 - for T iterations or until all the outputs are correct:
 - for each input vector:
 - compute the activation of each neuron j using activation function g :

$$y_j = g\left(\sum_{i=0}^m w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^m w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^m w_{ij}x_i \leq 0 \end{cases}$$
 - update each of the weights individually using:

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$$
- Recall**
 - compute the activation of each neuron j using:

$$y_j = g\left(\sum_{i=0}^m w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases}$$

از کتابخانه **sklearn** برای مدل **perceptron** استفاده می‌کنیم:

```
perceptron = Perceptron(eta0=0.4, tol=None, max_iter=1500)
# eta0 -> Learning rate
# tol -> The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol).
# max_iter -> The maximum number of passes over the training data (aka epochs)
perceptron.fit(X, y)
```

این مدل برای دیتای **test** امتحان $\text{accuracy} \approx 0.83$ را به دست می‌آورد.

بخش دوم

Non-linear Perceptron:

برای قسمت دوم سر درآوردن، استفاده از کرنل‌ها، از **feature map** مربوط به کرنل‌ها استفاده می‌کنیم.

به این صورت که بجای اینکه برای n داده ورودی، ماتریس $n \times n$ مربوط به **Kernel matrix** را حساب کرده و سپس با آلتوریت **Kernel perceptron** (HyperLink) مدل را آموزش دهیم، از **feature map** مربوط به کرنل **RBF** استفاده می‌کنیم.

این **feature map** به این صورت است که قبلاً از استفاده از مدل خطی، یک **feature map** از داده‌ها را حساب می‌کنیم و آن **feature map** را به عنوان ورودی مدل **perceptron** می‌دهیم.

HyperLink →

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

Gram matrix برای آلتوریت **kernel perceptron** استفاده می‌شود.

با استفاده از **feature mapping** انگار مدل خطی روی داده‌ها به بالا آموزش می‌دهیم.

- در کد از **RBF sampler** برای **feature map** مربوط به **kernel RBF** استفاده می‌کنیم. (Link)

- با استفاده از کرنل **RBF** (در واقع استفاده از **feature map** مربوط به **RBF**) مدل امتحان $\text{accuracy} = 0.91$ می‌کند.

