Deep Learning Fundamentals
Assignment 1 - Predicting Diabetes: Perceptron Method

Ziyang Ye
The University of Adelaide
a1707805@adelaide.edu.au

Abstract

Diabetes is a chronic disease that can cause serious complications. Machine Learning can identify patterns in large amounts of data and make predictions based on those patterns. Perceptron, as a binary linear classifier, can predict whether a patient has diabetes based on patient-related information. Therefore, in the diagnosis of diabetes, the algorithm should be able to predict whether a patient has diabetes based on the patient's relevant information, such as age, gender, weight, diet, exercise, and blood sugar levels. This short paper describes its principles, implementation, and evaluation.

1. Introduction

Due to the development and growth of Artificial Intelligence (AI) technology and medical data, Machine Learning (ML) algorithms are widely applied in the medical field. The most common scenario is disease diagnosis.

Perceptron, a classic binary classification machine learning algorithm, draws inspiration from the functioning of biological neurons, and in the field of artificial neural networks, it is also referred to as a dense (linear) layer neural network. Perceptron can predict whether a patient has diabetes based on patient-related information such as age, gender, weight, diet, exercise, and blood sugar levels.

In this short paper, I will explore the application of the Perceptron algorithm in diabetes diagnosis. This paper includes the principles, implementation, and evaluation of the perceptron on the diabetes dataset. In addition, in order to better understand the Perceptron, it is necessary to add some other methods in the experiment section. By understanding the limitations of the Perceptron, we can better understand a series of machine learning algorithms.

2. Related Work

Logistic Regression is a probabilistic classification model that uses the logistic function (Sigmoid function) to map the linear combination of input features to probability values between $[0, 1]$. It outputs a probability, indicating the likelihood of belonging to the positive class [7].

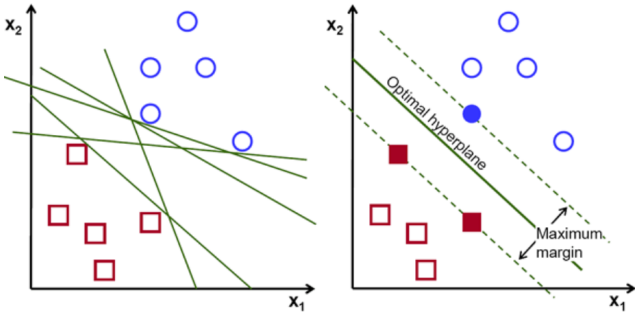


Figure 1. Right is a classic linear classifier, left is SVM with hard-margin.

In other words, logistic regression uses the Sigmoid function as its activation function and binary cross-entropy (BCE) as its loss function.

$$z = \theta^T x + b$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$H(x) = g(\theta^T x) = \frac{1}{1 + e^{-(\theta^T x + b)}}$$

$$L(\theta, b) = \sum_{i=1}^{m} y_i \log(H(x_i)) + (1 - y_i) \log(1 - H(x_i))$$

$$J(\theta, b) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(H(x_i)) + (1 - y_i) \log(1 - H(x_i))$$

where θ is weights, x is input features, $g(z)$ is the Sigmoid function, $H(x)$ represents the logistic regression itself, $L(\theta, b)$ is the log-likelihood function and $J(\theta, b)$ is the loss function.

Support Vector Machine (SVM) and perceptron are both used for classification, and they both use the sign function as the decision function. SVM optimizes its parameters by maximizing the minimum distance of support vectors from

the hyperplane [5].

SVM can address both linearly separable and non-linearly separable problems using hard-margin, soft-margin, and kernel methods. It can also be transformed into a dual form using Lagrange multipliers to enhance efficiency and subsequently introduce kernel methods [5].

Multi-Layer Perceptron (MLP) w.r.t. Feedforward Neural Network, is an extension of the single-layer Perceptron and is a typical deep learning model. Its main feature is having multiple layers of neurons. Typically, the first layer of the MLP is referred to as the input layer, the intermediate layers are called hidden layers, and the final layer is the output layer. The key to using the MLP lies in training the connection weights between the layers using the backpropagation [6] (BP) algorithm, and it employs nonlinear activation functions *e.g.* Rectified Linear Unit [2] (ReLU) (since Sigmoid has a high computational cost).

MLP does not specify the number of hidden layers, so you can choose an appropriate number of hidden layers based on your specific processing requirements. There are also no restrictions on the number of neurons in each layer of the hidden and output layers. Through hidden neurons, complex interactions between inputs are captured, and these neurons depend on the values of each input. We can easily design hidden nodes to perform arbitrary computations. Even with just one hidden layer, given enough neurons and correct weights, we can model any function.

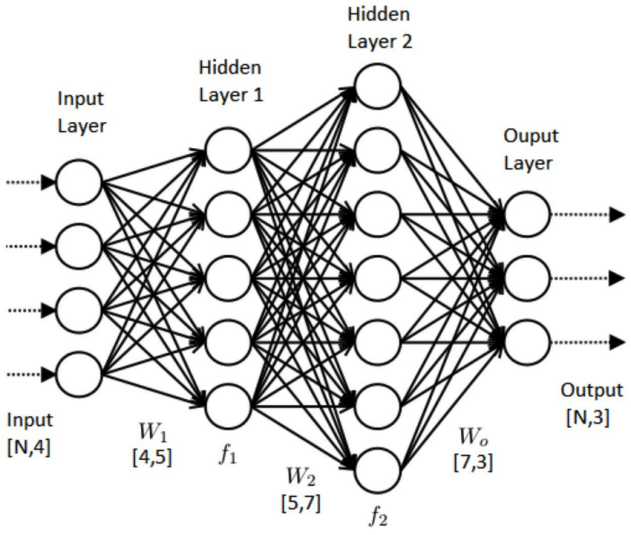


Figure 2. Graph shows a MLP has 2 hidden layers.

3. Dataset and Processing

Pima Indians Diabetes Database, this dataset originates from the National Institute of Diabetes and Digestive and Kidney Diseases and aims to predict, through diagnostic measurements provided in the dataset, whether or not a patient has diabetes. To create this dataset, specific criteria were applied when selecting instances from a larger database. Notably, all individuals in this dataset are female, at least 21 years of age, and of Pima Indian heritage [4].

Eight variables were selected as the basis for predicting the occurrence of diabetes within a five-year period in Pima Indian women. These specific variables were chosen because they have been identified as significant risk factors for diabetes in the Pima population or other similar groups, included:

- Number of times pregnant
- Plasma Glucose Concentration at 2 Hours in an Oral Glucose Tolerance Test (GTT)
- Diastolic Blood Pressure (mm Hg)
- Triceps Skin Fold Thickness (mm)
- 2-Hour Serum Insulin (μU/ml)
- Body Mass Index (Weight in kg / (Height in m)^2)
- Diabetes Pedigree Function
- Age (years)

Min-Max and **Z-score** normalization are applied to features of dataset, and replace 0 to -1 in labels.

$$Min\_Max: \ X_i' = \frac{X_i - \min(X_i)}{\max(X_i) - \min(X_i)}$$

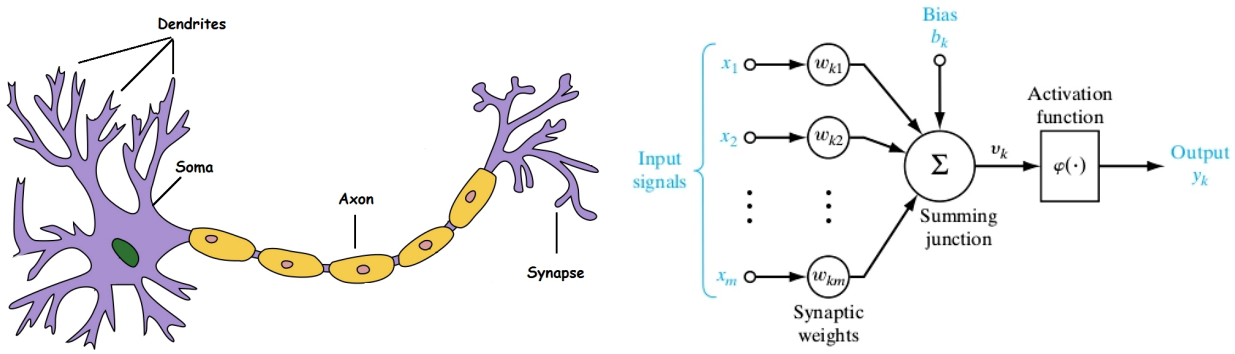
$$Z\_score: \ X_i' = \frac{X_i - \overline{X_i}}{\sigma_{X_i}}$$

where X_i' is the nomalized value, $\overline{X}_i$ is the mean value of i_{th} feaure and σ_{X_i} is the standard deviation of i_{th} feature.

After column-wisely applying Min-Max and Z-score normalizations, we can get the same data values to LibSVM "scale" version.

4. Perceptron

Perceptron, proposed by Rosenblatt in 1957, is a simple abstraction of biological neurons and serves as the foundation for neural networks and support vector machines. It is a linear classification model for binary classification, where the input consists of feature vectors for instances, and the output assigns a class label: $+1$ for the positive class and -1 for the negative class [3]. Perceptron is a discriminative model with the goal of separating positive and negative instances using a separating hyperplane. There is not a unique solution for the perceptron's hyperplane; the perceptron algorithm aims to find one feasible separation. When extended to high-dimensional spaces, it seeks a hyperplane.



(a) (b)

Figure 3. Left is a biological neuron, right is a McCulloch-Pitts neuron

4.1. Linear Separability

Given a dataset, if there exists a hyperplane, denoted as $S : (wx + b = 0)$, that can completely and correctly separate the positive and negative instances of the dataset, such that for $y = +1$ instances, $wx + b > 0$, and for $y = -1$ instances, $wx + b < 0$, then the dataset is considered linearly separable; otherwise, it is linearly inseparable.

4.2. Algorithm

Perceptron can be represented as following:

$$H(x) = sign(w \cdot x + b) \tag{1}$$

$$sign(x) = \begin{cases} +1, x \geq 0 \\ -1, x < 0 \end{cases} \tag{2}$$

where $w \in \mathbb{R}^n$ is weight and $b \in \mathbb{R}$ is bias, $w \cdot x$ is the inner product. $sign$ is the signature (activation) function.

First, write down the distance from any point x_0 in the input space $\mathbb{R}^n$ to the hyperplane S:

$$\frac{1}{\| w \|} \mid w \cdot x_0 + b \mid \tag{3}$$

where $\| w \|$ is $L2\ norm$ for w.

For misclassified data (x_i, y_i), the distance from x_i to hyperplane S,

$$-\frac{1}{\| w \|} y_i (w \cdot x_i + b) \tag{4}$$

Therefor, make the set of all misclassified point as M, the total distance to hyperplane S,

$$-\frac{1}{\| w \|} \sum_{x_i \in M} y_i (w \cdot x_i + b) \tag{5}$$

Ingore $\frac{1}{\|w\|}$ is the loss function of Perceptron,

$$L(w, b) = -\sum_{x_i \in M} y_i (w \cdot x_i + b) \tag{6}$$

Find parameters w, b to minimize the following loss function problem,

$$\min_{w,b} L(w, b) = -\sum_{x_i \in M} y_i (w \cdot x_i + b) \tag{7}$$

Perceptron learning algorithm is error-driven and specifically uses stochastic gradient descent [1] (SGD). Initially, it selects an arbitrary hyperplane represented by w_0, b_0 and then continuously minimizes the objective function (7) using gradient descent. During the minimization process, it does not perform gradient descent for all misclassified points in M at once, but instead randomly selects one misclassified point at a time for gradient descent. The gradient of the loss function $L(w, b)$:

$$\nabla_w L(w, b) = -\sum_{x_i \in M} y_i x_i \tag{8}$$

$$\nabla_b L(w, b) = -\sum_{x_i \in M} y_i \tag{9}$$

Randomly pick a misclassified point $(x_i, y_i) \in M$, update w, b:

$$w \leftarrow w + \eta y_i x_i \tag{10}$$

$$b \leftarrow b + \eta y_i \tag{11}$$

where η is the learning rate.

Through iterations, it is expected that the loss function $L(w, b)$ will continuously decrease, eventually reaching 0.

4.3. Implementation

By creating a class, we can abstract and break Perceptron into different method.

Firstly, we have a method named **sign**:

```
def sign(self,y):
  return -1 if y < 0 else 1
```

And Main function is **fit** method:

```
def fit(self,x_train,y_train):
  if not self.w:
    self.w = np.zeros(x_train.shape[1])
  for _ in range(self.epoch):
    err=0
    for i in range(len(x_train)):
      xi = x_train[i,:]
      yi = y_train[i]
      yi_hat = self._predict(xi)
      if yi * yi_hat != 1:
        err+=1
        self.w += self.l_rate * yi * xi
        self.b += self.l_rate * yi
    if err == 0:
      break
```

Lastly, **predict** method predicts the whole input dataset:

```
def predict(self,x):
  y_hat = x@self.w.T + self.b
  y_hat = np.where(y_hat<0,-1,1)
  return y_hat
```

5. Experiment

Accuracy refers to the percentage of correct predictions out of the total samples. While accuracy can assess the overall correctness, it may not be a good indicator when dealing with imbalanced datasets.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

Precision represents the probability that a sample predicted as positive is actually positive. Precision measures the accuracy of positive predictions, while accuracy considers both positive and negative predictions.

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

Recall refers to the probability that an actual positive sample is predicted as positive. Higher recall indicates a higher probability of correctly identifying actual positive cases.

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

The F1 score is designed to balance precision and recall, finding a trade-off point between the two.

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{15}$$

I used the above-mentioned metrics for experiment. In terms of algorithms and models, apart from the perceptron itself, I also utilized logistic regression, linear SVM, rbf SVM, and MLP for comparison and expansion. Specifically, I also employed K-fold cross-validation, setting K to vary between 3 and 5 to test the performance of different algorithms. I primarily used scikit-learn for this, after extensive hyperparameter tuning. I've omitted most of the process and only kept the results. The final results are as follows:

method	accuracy	precision	recall	f1
perceptron	0.747373	0.750533	0.747620	0.726802
logistic	0.769570	0.754407	0.722578	0.731989
linear-svm	0.772176	0.759074	0.722909	0.732924
rbf-svm	0.770894	0.758015	0.720205	0.730358
mlp	0.916667	0.928571	0.824627	0.873518

From the table, it can be observed that the perceptron has the poorest performance across all metrics, with logistic regression performing slightly better. Linear and kernel-based SVMs exhibit significantly better performance than the perceptron, while MLP outperforms all other methods.

6. Conclusion

One of the main drawbacks of the perceptron is that it can only address linearly separable problems, meaning it cannot handle situations where there are multiple nonlinear decision boundaries within the dataset. Additionally, perceptrons are sensitive to noise and may perform poorly when noise is present or when data is not entirely linearly separable.

References

[1] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436. Springer, 2012. 3

[2] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011. 2

[3] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 2

[4] Jack W Smith, James E Everhart, WC Dickson, William C Knowler, and Robert Scott Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the annual symposium on computer application in medical care*, page 261. American Medical Informatics Association, 1988. 2

[5] Shan Suthaharan and Shan Suthaharan. Support vector machine. *Machine learning models and algorithms for big data classification: thinking with examples for effective learning*, pages 207–235, 2016. 2

[6] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 2

[7] Raymond E Wright. Logistic regression. 1995. 1