LOGISTIC REGRESSION

COLLIN PRATHER

Contents

Abstract. This paper comprises of a mathematical representation of the logistic regression algorithm, applied to the Iris dataset[4]

1. Logistic Regression in the Context of Machine Learning

Logistic regression has traditionally been strictly labeled as a statistical technique used for predicting classification of datapoints, however, since the rise of big data and the emergence of the field of machine learning, many would now also describe it as a machine learning algorithm. As early as 1959, Arthur Samuel famously defined machine learning as the "Field of study that gives computers the ability to learn without being explicitly programmed"[1]. More recently, in 1998, the revered Machine Learning researcher, Tom Mitchell posed Machine Learning problems as so: "A computer program is said to *learn* from experience E with respect to some task T, and some performance measure P, if its performance on T, as measured by P, improves with experience E"[1].

Logistic regression is the go-to algorithm when dealing with categorical data, and it's applications are widespread. To name a few, logistic regression algorithms can help us to identify fradulent versus valid credit card transactions, spam vs non-spam emails, or even whether or not an online ad will cause the user to buy a product.

Generally speaking, logistic regression allows us to identify structure in our data. This understanding of structure enables us to categorize observations previously unseen, given a set of characteristics.

Date: April 25, 2018.

2. Logistic Regression Introduction

Logistic Regression is a classification algorithm used to predict which class or category an observation falls into[3]. For the purposes of this paper, we will be talking about what's referred to as simple Logistic Regression; that is, an algorithm that assigns a datapoint to one of two categories (you could think of this as binary), given a number of characteristics about the datapoint. It is important to note, however, this process can be expanded to multi-class problems as well.
Specifically, we will be attempting to classify an iris flower as either an *iris setosa*, or an *iris versicolor*[1], given information about its sepal and petals.[2]

3. Linear Algebra Orientation

Logistic regression is a *linear classification algorithm*; which means that it classifies data points based on the value of a linear combination of the characteristics. Recall that a linear combination z can be defined as,

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n,$$

with θ_0 as the intercept term. In this case, z is a vector in $\mathbb{R}^m$, and $\theta_0 \dots \theta_n$ are weights (coefficients) to the vectors $x_1 \dots x_n$.[3], which can be represented as,

[1]Note that the data set contains three classes of iris flowers, but it has been manipulated to be a binary classification problem.

[2]The Iris Dataset is well-renowned in the machine learning community. Originating from a paper written by Ronald Fisher in 1936[2], and now residing in the UCI Machine learning repository, this dataset has been referenced in thousands of academic papers and is generally one of the first datasets that a blooming data scientist will encounter.

[3]This notation is borrowed from Andrew Ng's machine learning course taught at Stanford and to thousands on the online education platform, Coursera. Ng's course has quickly emerged as the go-to introduction to Machine Learning course to many, both in academia and industry, and thus, his notation is often regarded as the standard.

$$\begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \dots & x_n^m \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

Intuitively, as we multiply our $m \times n$ matrix and our $n \times 1$ vector, we get an $m \times 1$ vector. As aforementioned, each of the m rows can be seen as a datapoint, or observation (these terms will be used interchangeably from here on out). With this orientation, we can see how we may apply this linear combination to a spreadsheet of data. Our Iris data looks like

	Sepal Length (cm)	Sepal Width (cm)	Target
0	5.1	3.5	0
1	4.9	3.0	0
2	4.7	3.2	0
3	4.6	3.1	0
4	5.0	3.6	0
5	5.4	3.9	0
6	4.6	3.4	0
7	5.0	3.4	0
8	4.4	2.9	0
9	4.9	3.1	0

with each row representing the characteristics of one flower[4]. Note that the "Target" column, which we will refer to as y, contains the actual classes of each of the flowers. Our algorithm will disregard the target vector in making predictions, focusing solely on the numerical characteristics of each flower, then will compare our predictions z with y to see how well our logistic regression algorithm performs.

4. Hypothesis Function

4.1. **Representation.** Given our vector z, which is a linear combination of the feature matrix X and weights vector θ, we use a hypothesis function $h_\theta(x)$, that outputs the probability that the target value y_i is either 1 (Versicolor), or 0 (Setosa) in order to predict which type of iris the given flower is.
Since $h_\theta(x)$ outputs a probability, the output must be a between 0 and 1. In other words, $0 < h_\theta(x) < 1$. In machine learning, the most common function used that always outputs a value between 0 and 1 is called the sigmoid function, $g(z)$. In order to think about our

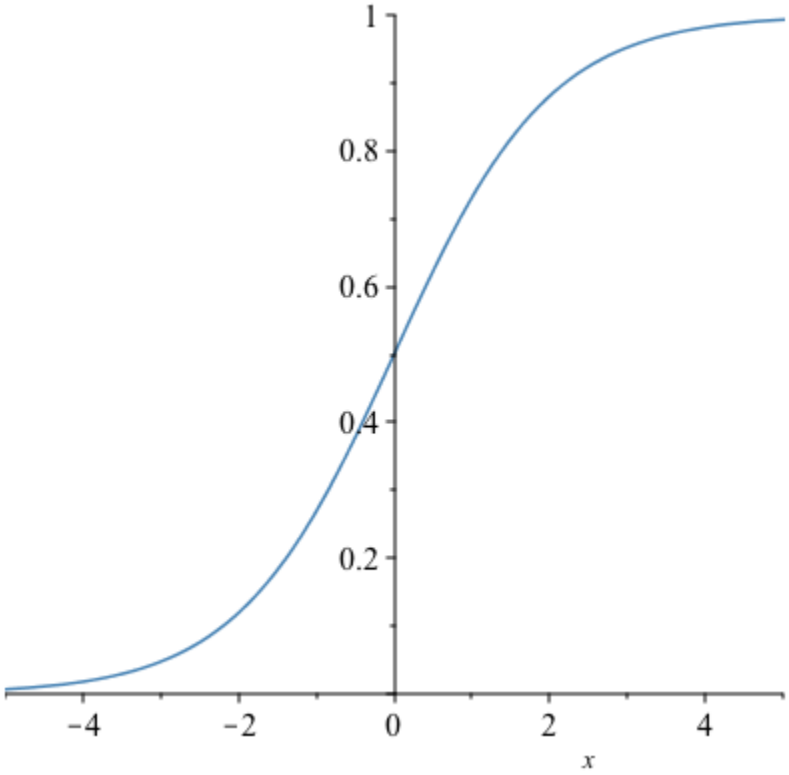
[4]This is just a sneakpeak of the data, and it is a bit misleading. The full dataset contains 150 observations, more of which are classified as 1 than 0.

hypothesis function, we'll first define our Sigmoid function as,

$$\text{Sigmoid/Logistic Function: } g(z) = \frac{1}{1+e^{-z}}$$

As depicted by the graph below, the output of our sigmoid function will always be between 0 and 1, even as $z \to \infty$ and $z \to -\infty$.

FIGURE 1. Sigmoid Function



4.2. **Interpretation.** Our hypothesis function, given weights $\theta_1, \theta_2 \dots \theta_n$ and vectors $x_1, x_2, \dots, x_n$ is denoted by $h_\theta(x)$, and can be expressed as,

$$h_\theta(x) = g(z) = \frac{1}{1+e^{-z}},$$

and outputs a vector of values between 0 and 1. This vector $h_\theta(x)$ contains the probability that each instance in our dataset is an Iris Versicolor. That is; if a prediction z_i is 0.7, then there is a 70% chance that the corresponding flower in our dataset is a Versicolor. To be clear,

$$h_\theta(x) = p(y = 1|x; \theta).$$

It may be obvious to state that an iris cannot be 70% Versicolor and 30% Setosa, it must either be one or the other. Our hypothesis function, however, only gives us a probability, not an explicit classification of the datapoint. Generally, the value of 0.5 is chosen as the classification threshold; that is, if $h_\theta(x) > 0.5$, then the iris is classified as a Versicolor, if $h_\theta(x) \leq 0.5$, then the iris is classified as a Setosa.

Our task is to determine the optimal weights $\theta_1, \theta_2 \dots \theta_n$ such that our hypothesis function perfectly[5] classifies our dataset into the correct class[6].

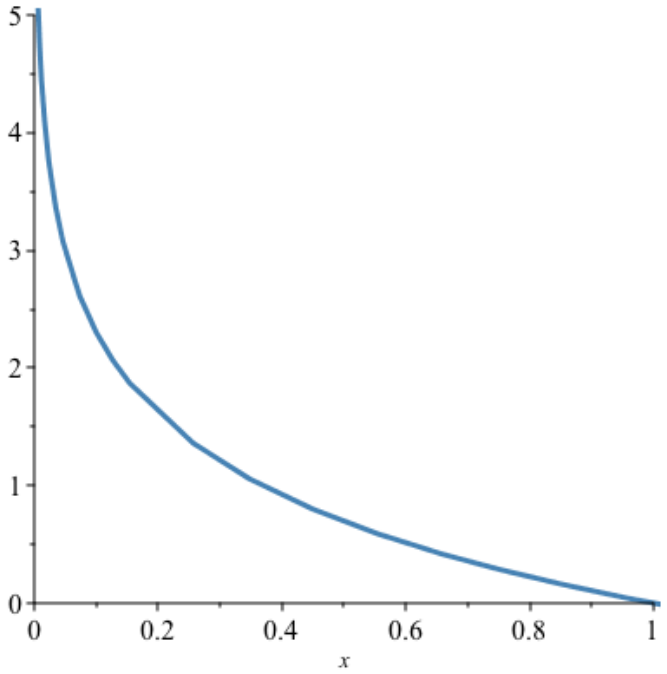
5. Cost Function

As aforementioned, we now define a cost function $\text{Cost}(h_\theta(x), y)$, that we will use to compare how correct (or incorrect) our predictions z, given θ, are compared to the actual classifications of each flower y.

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Let us draw our attention to the domain of the cost function. The inputs are $(h_\theta(x), y)$, both of which are between or equal to 0 and 1, therefore, the domain of $\text{Cost}(h_\theta(x), y)$ is $[0, 1]$.

5.1. **When y=1.** Now, the purpose of the cost function is to "penalize" our logistic regression algorithm by telling it just how wrong it is. To build intuition, let us take a look at the graph of our cost function when $y = 1$. As depicted below, if the actual classification of an individual datapoint is 1 (Versicolor), and our prediction is 0.9 (meaning that we've predicted there's a 90% chance that the flower is a Versicolor), the cost function outputs a small value, in this case, very close to zero. If, however, for this same datapoint we predict 0.1, the cost function will ouput a value closer 3, translating to a larger penalty for our algorithm.



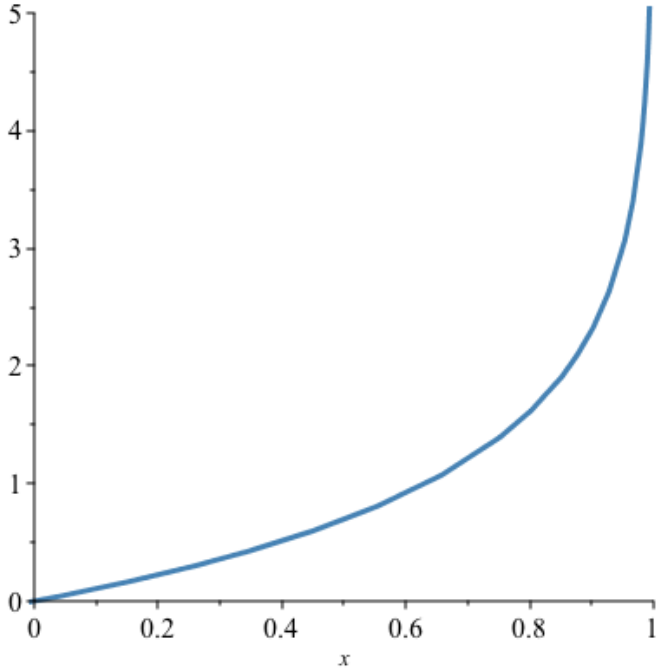
[5]To attain weights that "perfectly" models our dataset is often not optimal in reality, as we would most likely run into the problem of overfitting. This issue, however, lies outside the scope of this paper and will not be addressed.

[6]Again, there are two possible classes. If $z_i = 0$, then the iris is a Setosa. If $z_i = 1$, then the iris is a Versicolor.

Specifically,

$$\begin{aligned}\text{When } y = 1, \text{ as } \ & h_\theta(x) \to 1, \ \text{Cost}(h_\theta(x), y) \to 0 \\ \text{as } & h_\theta(x) \to 0, \ \text{Cost}(h_\theta(x), y) \to \infty\end{aligned}$$

5.2. **When $y = 0$.** As graphically displayed by our graph of $\text{Cost}(h_\theta(x), y)$ when $y = 0$,



$$\begin{aligned}\text{When } y = 0, \text{ as } \ & h_\theta(x) \to 1, \ \text{Cost}(h_\theta(x), y) \to \infty \\ \text{as } & h_\theta(x) \to 0, \ \text{Cost}(h_\theta(x), y) \to 0\end{aligned}$$

In short, the closer our algorithm is to predicting the correct class, the less harshly it is penalized.

5.2.1. *Simplified Cost Function.* Since y is always equal to either 0 or 1, there is actually a simplified way that we may write our cost function. Previously, we had defined $\text{Cost}(h_\theta(x), y)$ by cases like so:

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

However, if we compress our case-wise function by defining the cost function like so:

$$\text{Cost}(h_\theta(x), y) = \underbrace{-y \log(h_\theta(x))}_{=0 \text{ when } y=0} \underbrace{-(1 - y) \log(1 - h_\theta(x))}_{=0 \text{ when } y=1}$$

you'll notice that this function is precisely the same. When $y = 0$, the "$-\log(h_\theta(x))$" term cancels out, and when $y = 1$, the "$-\log(1 - h_\theta(x))$" term cancels out, giving us essentially the same case-wise function, only simplified.

Now, in creating a predictive model, we are interested in how accurate our predictions, $h_\theta(x)$, are, given $\theta_1, \theta_2, \ldots, \theta_n$. We aim to minimize the average cost of our predictions across the entire dataset. To do so, we introduce a function $J(\theta_1, \theta_2, \ldots, \theta_n)$. Given weights

$\theta_1, \theta_2, \ldots, \theta_n$, our function $J(\theta)$ will output the average cost of our entire dataset. Expressed mathematically,

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)}) \log\left(1-h_\theta(x^{(i)})\right)\right] \end{aligned}$$

6. Gradient Descent

Using our average cost function $J(\theta)$, we now are able to measure exactly how accurate our predictions are. All that is left to do is find the optimal weights $\theta_1, \theta_2, \ldots, \theta_n$ such that $J(\theta) \to 0$. The question remaining is, "How might we choose optimal weights $\theta_1, \theta_2, \ldots, \theta_n$?" We now introduce the Batch Gradient Descent algorithm, which we will use to iteratively update our weights until we reach that global minimum of $J(\theta)$, which represents weights that give us the most accurate predictions on our training set[7].
The Gradient Descent algorithm updates our weights θ as follows:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Let's break down this algorithm piece by piece.

6.0.1. *The Partial Derivative Term.* When we calculate the partial derivative term $\frac{\partial}{\partial \theta_j} J(\theta)$, we find that

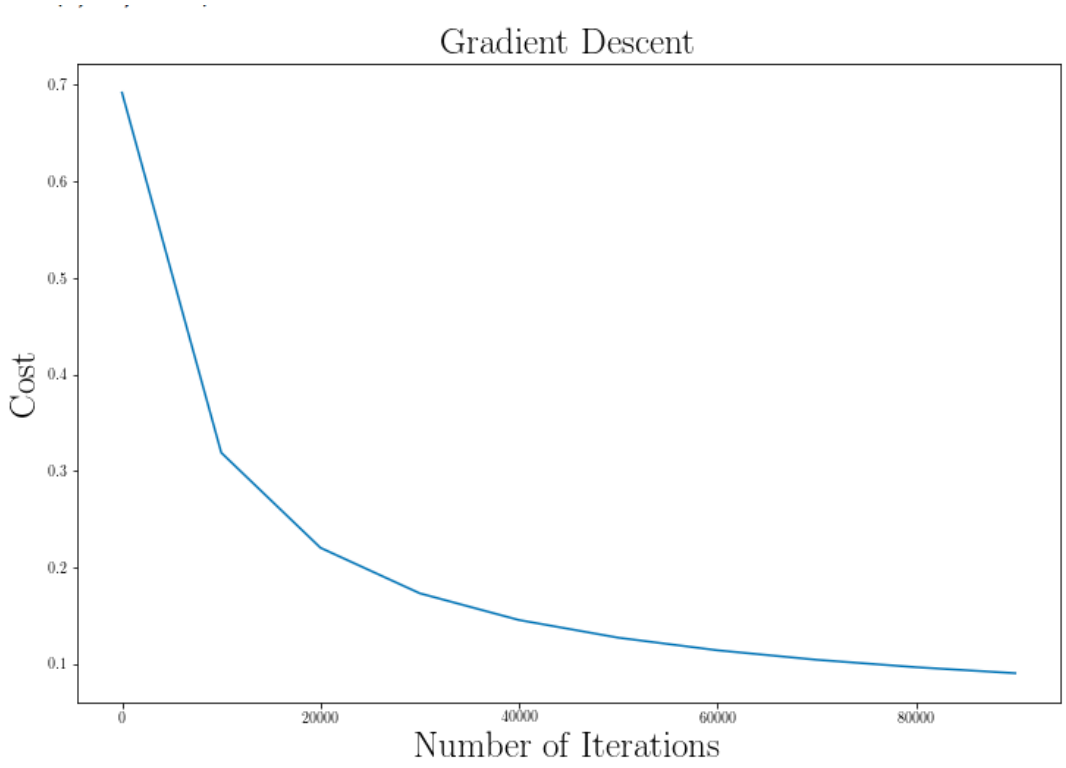
$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j}\left(-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)}) \log\left(1-h_\theta(x^{(i)})\right)\right]\right) \\ &= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)} \end{aligned}$$

Thus, we can re-write our Gradient Descent update rule with this new substitution like so:

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

6.0.2. *The Learning Rate α.* In order to explain the purpose of α, we first graph the convergence of our gradient descent algorithm by plotting the number of iterations of our of our gradient descent algorithm compared to the value of $J(\theta)$

[7]When applying Logistic Regression, we use our training set to find the optimal weights so that, given a new data point (one can think of this as a new row of data), we can plug this new datapoint x into $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ and classify that datapoint, though our models had never encoutered it before.



When the algorithm is working correctly, the graph should look like so. It is customary for the gradient descent algorithm to take anywhere from 100 to 1000 (or more) iterations to converge. Here's the rule of thumb: one can declare convergence if $J(\theta)$ decreases by less that $\frac{1}{1000}$ in one iteration.
The learning rate α determines how much $J(\theta)$ changes on each iteration. If α is too large, $J(\theta)$ may "overshoot" the global minimum and may never converge. Generally, $0.001 < \alpha < 1$. Finding the optimal α may take some trial and error.

6.0.3. *Batch Gradient Descent.* Finally, when it comes to implementing this algorithm, it is important to note that the "batch" in batch gradient descent implies that each weight θ_j gets updated on each iteration. Thus, in practice, our algorithm will repeat the following process on each iteration:

$$\theta_1 := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)}$$

$$\theta_2 := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_2^{(i)}$$

$$\vdots$$

$$\theta_n := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_n^{(i)}$$

Upon the convergence of our gradient descent algorithm, we find the optimal weights θ to classify our data. The final step is to use those weights θ in calculating our linear combinations z, then feed z through our hypothesis functions $h_\theta(x)$ in order to obtain our final classifications for each flower.
Please refer to my attached code to observe the accuracy of our predictions.

Appendix A. Variables Defined

- θ = a vector of weights (coefficients). $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

- X = the feature matrix containing numeric characteristics about the flowers. Each row $x^1, x^2, \ldots, x^m$ represents an individual flower. $m = 100$. Each column $x_1, x_2, \ldots, x_n$ represents a vector containing numeric data about a single characteristic of the flower.

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_n^1 \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \ldots & x_n^m \end{bmatrix}$$

- z = a vector in $\mathbb{R}^m$, with each element being the product of the corresponding θ and X value.

$$z = \begin{bmatrix} \theta_1 x_1 \\ \theta_2 x_2 \\ \vdots \\ \theta_n x_n \end{bmatrix}$$

- y = a vector in $\mathbb{R}^m$, with each element being the actual classification of the individual flower. The classification is binary with 1 representing a Versicolor and 0 representing a Setosa. For your reference, here's what it could look like:

$$y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- $h_\theta(X)$ = a vector in $\mathbb{R}^m$, and is the output of sending each element in z through the sigmoid funcion. Each value in $h_\theta(X)$ will be between 0 and 1 and can be interpreted as a probability that $y = 1$.

$$h_\theta(X) = \begin{bmatrix} 0.35 \\ 0.88 \\ \vdots \\ 0.41 \end{bmatrix}$$

- $\text{Cost}(h_\theta(x), y)$ = a vector filled with values that represent how wrong our predictions $h_\theta(x)$ are, in comparison with the actual classifications of each flower y. It could look

like this:

$$\text{Cost}(h_\theta(x), y) = \begin{bmatrix} 2.2 \\ 0.58 \\ \vdots \\ 0.37 \end{bmatrix}$$

- $J(\theta)$ = a real number that represents the average cost of our entire dataset. In theory, $J(\theta)$ decreases on each iteration of our algorithm. Below are our values for $J(\theta)$ = every 10,000 iterations.

```
cost: 0.6914086418407188
cost: 0.31914025762495013
cost: 0.22076282591769877
cost: 0.17370626624345922
cost: 0.14607836499121452
cost: 0.1278095149727622
cost: 0.11477019334946933
cost: 0.10495731501075146
cost: 0.09728061175028055
cost: 0.0910948324402139
```

References

[1] A. Ng, Machine Learning, Stanford University, Stanford, California, 2012.
[2] S. Fialoke. Classification of Iris Varieties, http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/. (3/21/18)
[3] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction To Statistical Learning, Springer, New York, 2013.
[4] D. Dheeru, E. Taniskidou, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences,http://archive.ics.uci.edu/ml. (3/21/18)