Machine Learning Applied to Diagnose Diabetes

Shreya Dhingra

Abstract

Machine learning is a field of computer science that allows data scientists to use statistical methods and functions from a coding language to analyze data and make insightful conclusions [1, 4, 3, 2]. The problem we consider in this paper is that of supervised learning. We will use logistic regression to create a computational model that can make predictions and later help us make decisions. We will discuss the process and the factors to consider while coming up with the model.

Introduction

As diabetes cases are rising and becoming a widespread problem, it is important to be able to diagnose diabetes. By using machine learning, we can compute a model that can predict what specific numbers for each variable (ex: age, blood pressure, insulin) are likely to cause diabetes. This way the person can make certain lifestyle changes to prevent getting diabetes. While the model won't always be accurate in predicting whether a person with certain statistics will get diabetes, it will be mostly accurate and better than not knowing at all whether a person will get diabetes later on in their life.

As mentioned earlier, the type of problem we are dealing with is supervised learning. This means the data consists of many examples that each have features and a label. The features can be considered the independent variables and the label can be considered the dependent variable. In a diabetes data set taken from Kaggle, the examples are female patients. The features are the number of pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, and age. The label is basically the conclusion: does the patient have diabetes or not. 0 indicates that the patient does not have diabetes while 1 indicates that the patient does have diabetes. Because there are two different outcomes for the label (0 and 1), this problem is known as a binary classification problem. The table below represents the data set. Five out of 392 patients' statistics are shown.

Pregnancies	Glucose	BP	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
3	78	50	32	88	31.0	0.248	26	1

2	197	70	45	543	30.5	0.158	53	1
1	189	60	23	846	30.1	0.398	59	1

Table 1

Initially the data was a set of 768 examples. However, there were too many 0s in the data for almost every feature. Since 0 is not a realistic number for each category, we removed it from the data. This cut down the data to 392 examples.

Afterward, we checked to make sure that the data set was well balanced with 0s and 1s. If the outcome is mostly 1 or 0, the data set needs to be amended so that the model can be as accurate as possible. There ended up being 262 0s and 130 1s which is pretty good.

Setting Up to Create the Model

Once the data set itself was ready (has realistic numbers and is well balanced), the next step was to store the outcomes into a 1D array called y using the numpy library. Since there are 392 examples, 392 values are stored in the y array where $y = [y_1, y_2, y_3, \dots, y_{392}]$. y_i is the ith label. We will store the features in X. However, since there are multiple features for each example, X must be of type 2D numpy array, which is why we have saved it as capital X. $X = [\ [x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{1,9}], [x_{2,1}, x_{2,2}, x_{2,3}, \dots, x_{2,9}], \dots, [x_{392,1}, x_{392,2}, x_{392,3}, \dots, x_{392,9}]$.

Next, the features and labels must be split into a training and validation set. The training set will be used to develop the model while the validation set will be used to test the accuracy of the model. Since the validation set was not used to develop the model, it is a good test to see whether the model is accurate for examples other than the one it was trained for. 25% of the examples will be randomly chosen for the validation set.

The features must also be scaled which means they become numbers not too small or large and not too spread out or clumped. This makes the data easier to work with.

Binary Cross Entropy Error

The model will make predictions based on the features given. $\widehat{y_i}$ will be the prediction from our model while y_i will be the actual label given in the dataset. y_i will be either 0 or 1 to indicate whether the person has diabetes or no while $\widehat{y_i}$ will be between 0 and 1 to detect the probability of the person having diabetes. If the decimal is closer to 0, the person does not have diabetes, and if the decimal is closer to 1, the person does have diabetes.

$-\ (y_i \log(\widehat{y_i}) + (1 - y_i) \log(1 - \widehat{y_i}))$ is the binary cross entropy error for a specific example

with features x_{ij} from the one dimensional numpy array x_i and the label y_i. If the features from x_i are plugged into the model, we will get $\widehat{y_i}$ and the error of the model's prediction is given by the expression above. The smaller the error is, the closer the estimation is to the actual label in the dataset. If they are exactly equal, the error is 0.

However, the expression above only gives the error for one example. In order to find the error overall, $-\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\widehat{y_i}) + (1 - y_i)\log(1 - \widehat{y_i}))$ is used. It gives the average of all the binary cross entropy errors of all the examples.

Logistic Regression

There are a few different methods that can be used for supervised learning problems. Some are logistic regression and neural networks. Neural networks is not used here because the data set doesn't have that many examples, so using neural networks won't make our model be more accurate. Logistic regression will give similar results and is exclusively for binary classification problems which is what we are dealing with here. The function for this is $\sigma(x) = 1/(1 + e^{-x})$. When using logistic regression our input is $w_1x_1 + w_2x_2 + \ldots + w_kx_k + b$ and the model $\sigma(x)$ gives $\widehat{y_i}$. w_1, w_2, …, w_k, and b are the parameters which are adjusted so that the model gives the smallest possible binary cross entropy error. After our model is created, we use the command "model.predict(X_train_scaled)" to get the predicted labels from the training scaled set of features. "model.predict(X_val_scaled)" is used to get the predicted labels from the validation scaled set (which was not used to create the model). Next the predicted labels from the training and validation sets are changed to 0s and 1s based on which number the decimal value is closest to. Then the classification report is printed for both the training and validation sets to see the precision and recall of the model. The precision measures how many of the "yes" predictions were actually true. The recall measures how many of the yeses in the actual data set were predicted correctly by the model. Did the model catch most of the patients with diabetes? The recall is the better measure of the model's reliability since even if precision is high, it just means that whatever yeses the model predicted, were right. It doesn't mean that the model was able to get most of the yeses in the actual dataset. The recall for the 0s was 86% while the recall for the 1s was 56%. That means the average recall was 71%, meaning the model is mediocre. It does a good job getting most of the people who do not have diabetes, but an okay job predicting the people who do have diabetes. However, the model we predict cannot really get any better than this. This is what we have, and it is not that bad in making true predictions.

References

[1] Andriy Burkov. The hundred-page machine learning book, volume 1. Andriy Burkov Canada, 2019.

[2] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. The elements of statistical learning, volume 1. Springer series in statistics New York, 2001.

[3] Tom M Mitchell et al. Machine learning. 1997.

[4] Toby Segaran. Programming collective intelligence: building smart web 2.0 applications. ” O’Reilly Media, Inc.”, 2007.