Task

Supervised Learning – Logistic Regression

Visit our website

Introduction

We have worked with regression models and have been able to make predictions for the datasets for which the dependent variable is numerical. In this task, we are going to predict the outcome for datasets whose dependent variable is categorical.

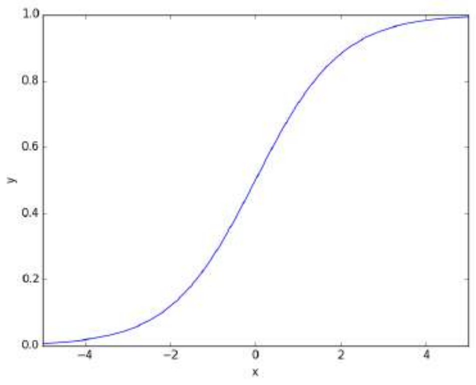
CLASSIFICATION PROBLEMS

The linear regression models discussed in the previous tasks assumed that the dependent variable **Y** is a **continuous numerical value**. However, it is very common in machine learning for problems to have **categorical variables** as the dependent variable. These variables take on distinct non-continuous values that correspond to a particular set of categories. For example, nurses in a town could either work in a “Hospital” or “Office” position, and these positions are represented as the values 0 and 1, respectively. This is an example of a categorical variable, where the category is encoded as a numeric value.

Predicting categorical variables is called **classification**. Classification problems are very common, perhaps even more so than problems suited for regression. Classification involves predicting the probability of each of the categories for a given observation and assigning the observation to the category with the highest probability. For example, a classifier might take an hourly wage and the number of years of practice of an employee and predict the probability that they are a hospital nurse and the probability that they are an office nurse. The model would then typically return either the probabilities or the category with the highest probability.

LOGISTIC REGRESSION

One approach to classification is logistic regression. Logistic regression is a common way of doing binary classification, which is classification into two categories. It works by using the logistic function. This function, also called the sigmoid function, is an S-shaped curve that maps input values **x** to output values **y**.



Logistic regression is similar to linear regression, however, the output is not continuous along a line, but a value between 0 and 1. That value can then be interpreted as the probability that the instance belongs to a certain category.

To give you an example, consider this sample of the Iris dataset:

SepalLength	SepalWidth	PetalLength	PetalWidth	Species
5.3	3.7	1.5	0.2	Iris-setosa
5	3.3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolour
6.4	3.2	4.5	1.5	Iris-versicolour

Our input values are the length and width of both the petal and sepal. The output will be the probability of the instance of being either a setosa or versicolour type.

The model will calculate the probability that a sample (consisting of a set of these measurements, or “features”) belongs to each category using a simple formula:

$$\mathbf{p(Species) = \beta_0 + \beta_1 SepalLength + \beta_2 SepalWidth + \beta_3 PetalLength + \beta_4 PetalWidth}$$

Let’s say we fit the model and it predicts the following output:

setosa	versicolour
0.4	0.6
0.7	0.3
0.2	0.8
0.9	0.1

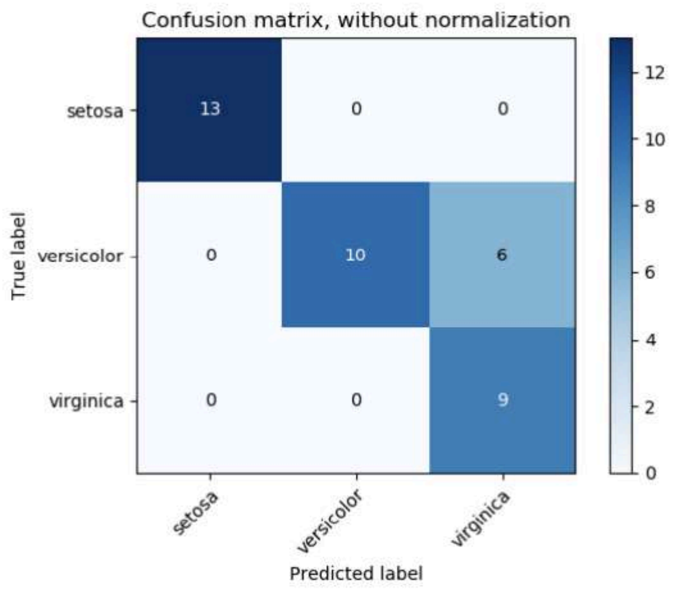
This table shows that the first observation is predicted with a 40% probability to be of the species “setosa”, and with a 60% probability to be the species “versicolour”. The prediction for the first instance is thus that it’s a member of the category “versicolour”.

EVALUATING CLASSIFICATION MODELS

In the tasks on regression, we evaluated our models by looking at the deviation of the predictions from the gold standard and computing the error. This works for continuous outcomes but not for categorical ones. Outcomes of classifiers are instead evaluated in different ways.

Confusion matrix

A confusion matrix is an NxN table that summarises a classification model's predictions. One axis of a confusion matrix is the label that the model predicted, and the other axis is the actual label. N represents the number of classes, and in a binary classification problem N = 2. In the complete Iris dataset with three classes, N = 3. Here is a sample confusion matrix for the iris classification problem:



The confusion matrix is a great way to see where the weaknesses of the classifier lie. In this example, we see that the model never classifies setosa instances as versicolour or vice versa, but on six occasions it classified versicolour instances as virginica instances, suggesting that those two types are, in the eyes of the system, harder to distinguish.

While the confusion matrix gives insights into the behaviour of the classifier, we need evaluation metrics to make claims about whether the model did well or not compared to other models. To illustrate the most common evaluation metrics, let's consider a hypothetical scenario.

A shop owner employs a security guard to keep a lookout for shoplifters. If the guard sees someone slip any merchandise into their pockets, he approaches the person he suspects of shoplifting, retrieves the stolen items, and escorts the shoplifter out of the shop. Unfortunately, looks can be deceiving, and the security guard sometimes makes mistakes and confronts people who may have only been putting their mobile phone back in their pocket. This makes innocent customers very upset and the shop owner gets angry.

Practised shoplifters also sometimes deceive the security guard and slip past him unnoticed with stolen goods, which also makes the shop owner angry as he is losing money. The security guard lets regular customers who do not shoplift pass peacefully out of the shop.

This story highlights two types of errors the security guard can make: accusing an innocent person of shoplifting and missing an actual shoplifting event.

If we state that “Shoplifter” is the positive class and “Not a shoplifter” is the negative class, then we can summarise our “shoplifting prediction” model using a 2x2 confusion matrix with the following four possible outcomes:

True Positive (TP):
- Reality: A person stole an item.
- Guard said: “Shoplifter.”
- Outcome: The owner is happy.

False Positive (FP):
- Reality: Regular customer, not stealing.
- Guard said: “Shoplifter.”
- Outcome: The owner is angry.

False Negative (FN):
- Reality: A person stole an item.
- Guard said: “Not a shoplifter.”
- Outcome: The owner is angry.

True Negative (TN):
- Reality: Regular customer, not stealing.
- Guard said: “Not a shoplifter.”
- Outcome: The owner is happy.

Correct predictions occur both for true positive predictions, where the model correctly predicts a positive (shoplifting) event, and true negative predictions, where the model correctly predicts a negative event (not shoplifting).

Incorrect predictions occur when the model predicts a false positive or a false negative. False positive predictions occur when the model predicts the positive class, but the reality is the negative class – i.e., a regular customer is accused of shoplifting. False negative predictions occur when the model predicts the negative class, but the reality is a positive class – i.e., the guard thinks a shoplifter is a regular customer and lets them pass.

For a multi-class classification problem that deals with more than two classes (e.g., setosa, versicolour, and virginica), a confusion matrix is made for each class (i.e., setosa vs not-setosa, versicolour vs not-versicolour, and virginica vs not-virginica) and evaluation metrics can be computed for each class and then averaged to get a sense of overall performance.

This brings us to common evaluation metrics, which are calculated using the number of TPs, FPs, TNs, and FNs in the confusion table.

Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

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

Thus, in the shoplifter example, accuracy is the number of correct predictions (both “Shoplifter” and “Not a shoplifter”) out of all the predictions made. For the iris example discussed earlier, the number of correct predictions is shown in the squares along the diagonal in the confusion matrix. The total number of predictions is the number of observations in the test set, i.e., the total of all the numbers in the confusion matrix.

Note that if a class is very rare, the value of TP will be low. This means that a model that does not predict a positive for any of the test items will have high accuracy. After all, the TN value will be high enough to offset the cost of the false negatives. For this reason, accuracy is often not the only measure you will want to use.

Precision

Precision is the proportion of predictions of the positive class that is correct. Precision is defined as follows:

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

In our shoplifter example, this metric tells us: “Of all the predictions that a person was a shoplifter (i.e., positive predictions), what proportion of these were correct?”

Recall

Recall is a measure of how many instances of a class the model was able to recognise. Recall is defined as follows:

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

Again in terms of our shoplifter example, this metric tells us: “Of all the actual shoplifters in the shop, what proportion of them were spotted by the security guard?”

F1 score

Precision and recall often form a trade-off: A model that predicts the category "Shoplifter" is often likely to identify all shoplifters (i.e., high recall), but is also likely to predict "Shoplifter" labels for many customers that are not shoplifters (i.e., low precision). In contrast, a model that is very careful about which customers it predicts to be shoplifters will have high precision but low recall, i.e., it is likely to let a lot of shoplifters exit the shop unconfronted, but won't misclassify many innocent customers as shoplifters. It is, therefore, useful to have a measure that is high for models that have both decent recall and precision. The most common is the F1 score.

The F1 score is the weighted average of precision and recall, and is defined as follows:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Instructions

- Read through this **blog** and the **Logistic_Regression.ipynb** file that comes with this task as guides to applying logistic regression.
- Read this **article** from DataCamp to learn about preprocessing categorical variables.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select "Request Review", the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you've done that, feel free to progress to the next task.

Auto-graded task

Follow these steps:

- Open a Jupyter notebook and create a new file named **iris_logistic_regression.ipynb**.
- Load the Iris dataset from the **Iris.csv** file into a DataFrame within the notebook. This dataset includes various measurements of iris flowers, such as petal length, petal width, sepal length, and sepal width, along with their corresponding species.
- The goal is to develop a classifier that can determine whether a given iris flower belongs to the `Iris-setosa` class. For this task, we will be working with two classes: `Iris-setosa` and not-`Iris-setosa`, which encompasses the other two species, `Iris-versicolour` and `Iris-virginica`.

    1. Identify your independent variable `x`.
    2. Encode your dependent variable `y` such that `Iris-setosa` is encoded as `0`, and `Iris-versicolour` and `Iris-virginica` are both encoded as `1`. Here, `0` corresponds to the `Iris-setosa` class, and `1` corresponds to the `not-Iris-setosa` class.
    3. Split the data into a training and test set.
    4. Use scikit-learn's `LogisticRegression` class to create a logistic regression model and fit it to the training data. Use the trained model to make predictions on the test data.
    5. Use scikit-learn to generate a confusion matrix that compares the predicted labels to the actual labels (gold labels).
    6. Review the confusion matrix to determine how well the model is performing. Based on your analysis, explain in a markdown cell whether you think the model will have higher precision, higher recall, or similar values for both.
    7. Instead of using sklearn's built-in function, write your **own code** to calculate accuracy, precision, and recall. Once you've calculated

these metrics manually, compare your results with those obtained from **scikit-learn** to see if they align.

Optional:

- Repeat this task but include all three categories, `Iris-setosa`, `Iris-versicolour`, and `Iris-virginica` corresponding to the numeric values 0, 1, and 2, respectively. This will now be a multiclass problem. Observe how this changes the confusion matrix.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.