

[Open in app](#)

Published in Towards Data Science



Graham Harrison

[Follow](#)

May 2, 2021 · 10 min read · ✨ · 🎧 Listen



Save



Calculating and Setting Thresholds to Optimise Logistic Regression Performance

How to create a simple class to optimise thresholds for precision, recall, f1 score, accuracy, tpr — fpr or custom cost functions



Photo by [Sean Lim](#) on [Unsplash](#)

Background



[Open in app](#)

Also, I had seen various examples online about how to recalculate the true and false classifications given a chosen threshold but these examples fell short of the detail I needed to put thresholds to work in the real world.

I suspected there might be a way to wrap threshold optimisation into a simple, object-oriented class so I could use them easily in future hence this article documents my learning journey to achieve these objectives.

Preparation

Let's start by importing the libraries we will need ...




[Open in app](#)

[big-data/overview.](#)

I have performed some cleaning of that data including dealing with the null values, converting the categorical features and balancing the data to evenly represent the true and false classifications which is outside of the scope of this article. The data imported below includes all of those data cleaning steps ...





Open in app

0	482087.0	6	0.0	11.0	26.3	685960.0	1.0	1.0	99999999.0	47386.0	...	0	0
1	1025487.0	10	0.0	15.0	15.3	1181730.0	0.0	0.0	264968.0	394972.0	...	0	0
2	751412.0	8	0.0	11.0	35.0	1182434.0	0.0	0.0	99999999.0	308389.0	...	0	0
3	805068.0	6	0.0	8.0	22.5	147400.0	1.0	1.0	121396.0	95855.0	...	0	0
4	776264.0	8	0.0	13.0	13.6	385836.0	1.0	0.0	125840.0	93309.0	...	0	0
...
5938	2833185.0	6	0.0	18.0	21.3	280170.0	0.0	0.0	437404.0	108889.0	...	0	0
5939	1257610.0	8	0.0	14.0	16.5	821480.0	0.0	0.0	448052.0	167428.0	...	0	0
5940	402192.0	0	0.0	3.0	8.5	107866.0	0.0	0.0	129360.0	73492.0	...	0	0
5941	1533984.0	1	0.0	10.0	26.5	686312.0	0.0	0.0	444048.0	456399.0	...	0	0
5942	1878910.0	6	0.0	12.0	32.1	1778920.0	0.0	0.0	99999999.0	477812.0	...	0	0

5943 rows × 33 columns

Image by Author

Let’s complete the preparation by splitting the data into testing and training so we can optimise against the training data and come back to the test data later on ...

[Open in app](#)

`((4754, 32), (1189, 32), (4754,), (1189,))`

Benchmarking

Initial Analysis

OK, let's fit a basic `LogisticRegression` to the test data and then review the results ...



[Open in app](#)

A visualisation of the confusion matrix will help us to evaluate the performance ...

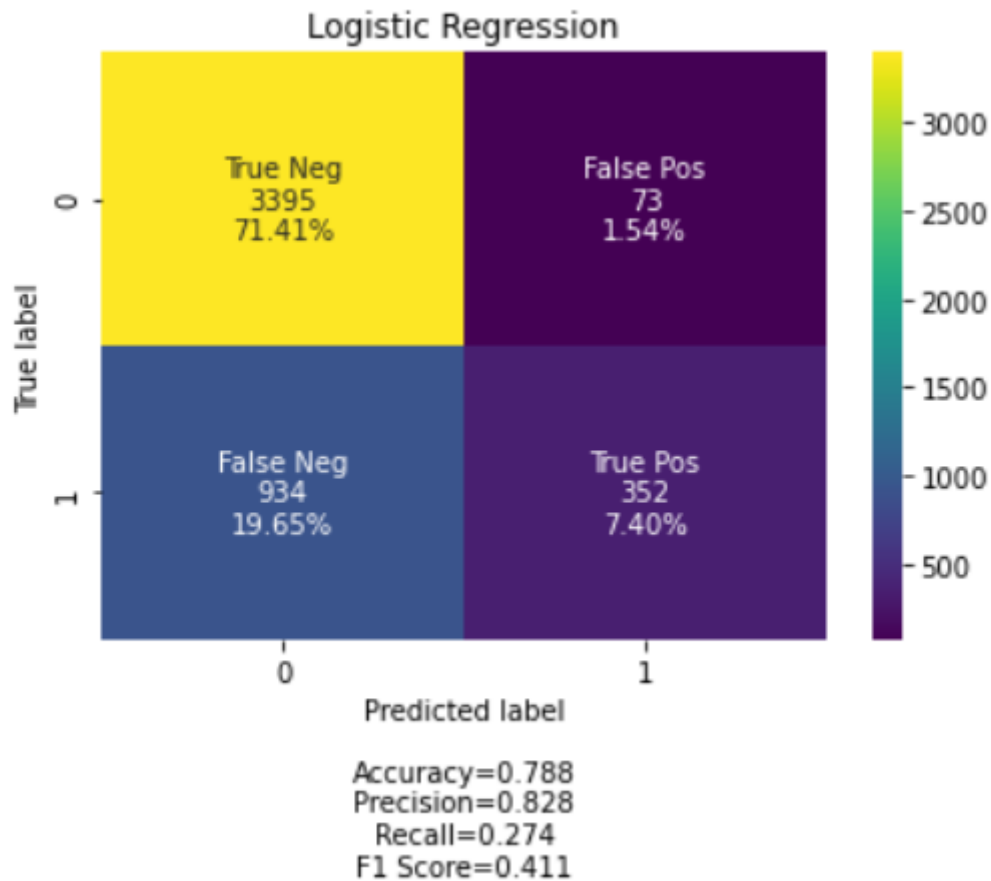


Image by Author

The banking data is being used to predict customers who will go on to default against their credit agreements and those who will not default.

We can see straight away that a basic logistic regression has 79% accuracy having predicted 352 customers that will default and 3395 customers who will not.

That sounds pretty good right? Well if I was one of the managers taking delivery of this algorithm my immediate question would be — “what about the 934 false negatives?”. There are nearly a 1000 customers that the algorithm did not predict as defaulting who went on to default.

If the average cost of a default account to the bank is £10,000, my data science team



[Open in app](#)

- The accuracy measure may not always be the best way to evaluate the performance of a classification algorithm.
- The way in which a classification algorithm is optimised is heavily dependent on what the business is trying to achieve.
- The algorithm must be appropriately optimised to achieve the desired business outcomes.

Further Analysis

A bit more digging will reveal all of the key metrics for the performance of our basic algorithm -

accuracy = 0.7881783761043332 precision = 0.8282352941176471

recall = 0.2737169517884914

f1 score = 0.4114552893045003

true positive rate (tpr) = 0.2737169517884914

false positive rate (fpr) = 0.02104959630911188

tpr-fpr = 0.2526673554793796

I would also usually take a look at the ROC (Receiver Operating Characteristic) curve -



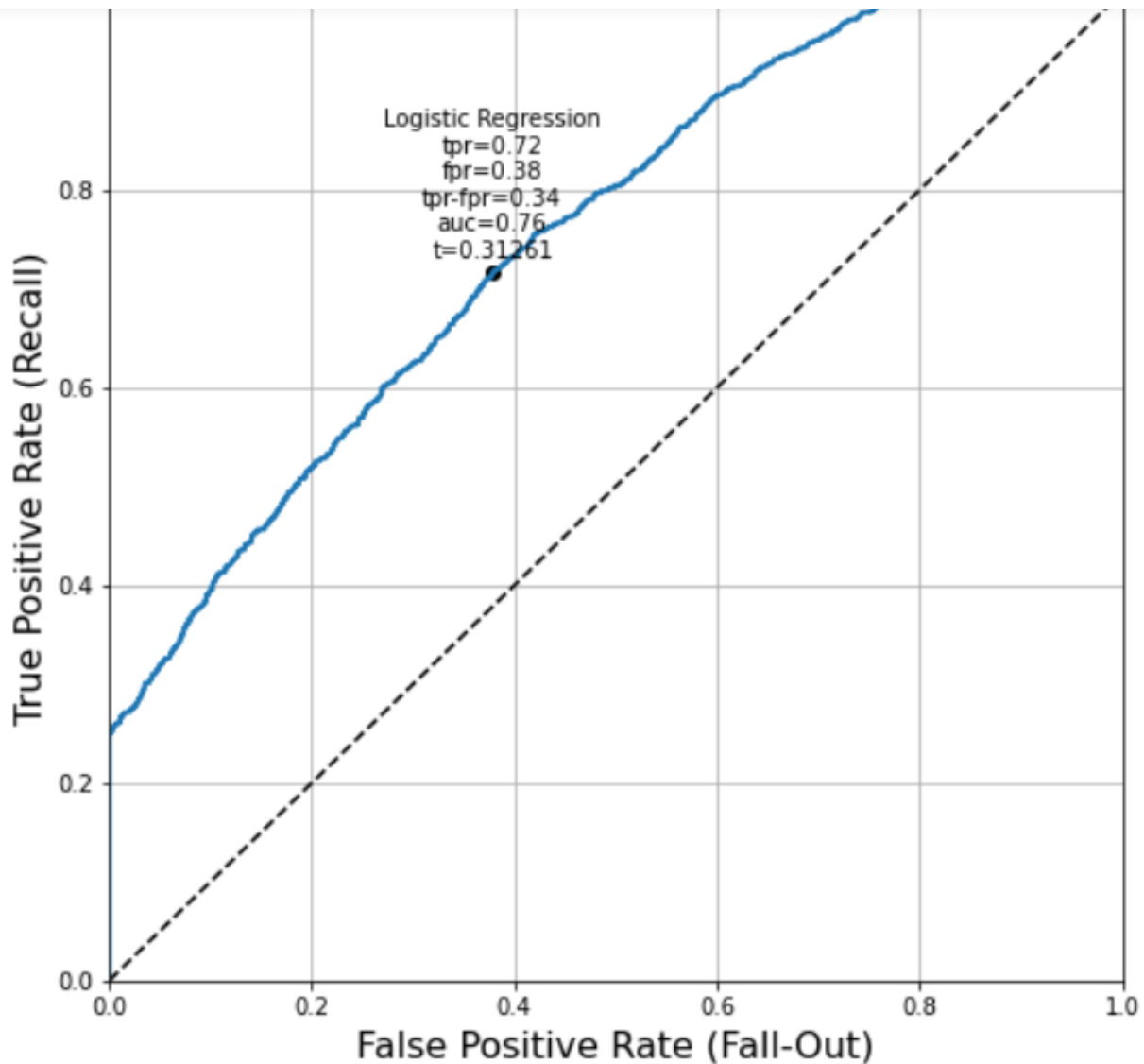
[Open in app](#)

Image by Author

The black dot represents one interpretation of the optimum point.

One measure that can be used is for calculating the optimum point on a ROC curve is $TPR-FPR$ where TPR = True Positive Rate and FPR = False Positive Rate. The point at which the $TPR-FPR$ is at its maximum value is the optimum point.

The graph is showing that if we set the threshold of the logistic regression to 0.31261 instead of the default which is 0.5 then we optimise the logistic regression algorithm for $TPR-FPR$.

A Quick Recap on Thresholds

If all this talk of thresholds is confusing there are many online articles that will



[Open in app](#)

for each row where that probability is ≥ 0.5 i.e. 0.5 is the default threshold.

Once we understand a bit more about how this works we can play around with that 0.5 default to improve and optimise the outcome of our predictive algorithm.

Analysis Conclusion

From this quick analysis, the evidence suggests we can do better than accepting the default threshold of 0.5 that is implemented within the `LogisticRegression` algorithm that is part of the `sklearn.linear_model` library. For example, we already know that if we change that default to 0.31261 we will optimise for $TPR-FPR$.

However, there is no easy way to change the threshold within the `LogisticRegression` class and typically data scientists will do this bit manually and duplicate effort across multiple projects.

What we need is a simple class that will do this for us every time we want to optimise the threshold.

Automating the Threshold Calculation

Here is my first attempt at writing a class that extends the `LogisticRegression` class to optimise the threshold based on the $TPR-FPR$ calculation. It consists of a class definition, a method called `threshold_from_optimal_tpr_minus_fpr` and a `predict` method which over-rides the base class method enabling the `threshold` parameter to be passed in ...



[Open in app](#)

Armed with this new utility class we can now easily fit the training data to it, ask the class to tell us the optimal threshold for $TPR-FPR$ and then use that threshold to return a new set of predictions ...



[Open in app](#)

(0.3126109044627986, 0.3383474055618039)

Logistic Regression (Threshold Optimized for TPR - FPR = 0.31261)

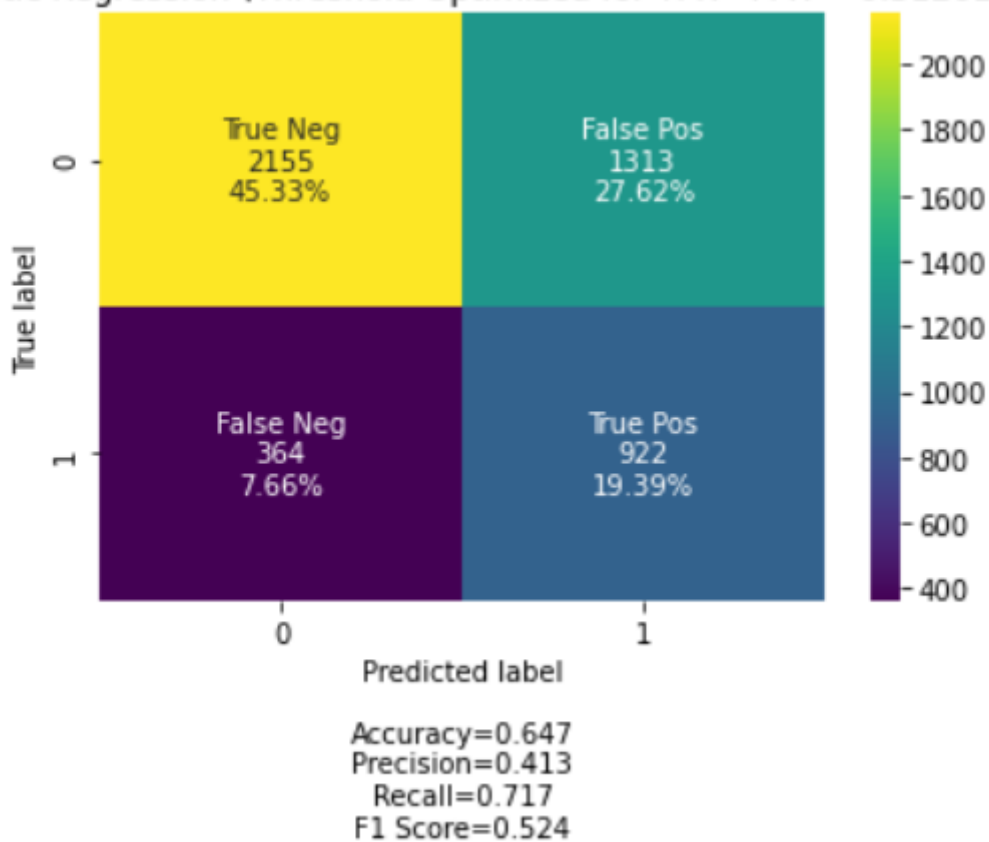


Image by Author

Well, this optimisation has certainly changed the shape of the outcomes as shown by the confusion matrix. The false negatives and true negatives have reduced whilst the false positives and true positives have increased.

This finding led me to ask the question — “what other approaches are there to optimisation?”



[Open in app](#)

precision and recall for the confusion matrix rather than the FPR and TPR.

As a quick reminder, precision and recall are calculated as follows -

$$\text{Precision} = \frac{\text{TruePositives}}{(\text{TruePositives} + \text{FalsePositives})}$$

$$\text{Recall} = \frac{\text{TruePositives}}{(\text{TruePositives} + \text{FalseNegatives})}$$

Image by Author

I.e. Precision is the right hand column of the confusion matrix and precision is the bottom row.

Whereas $TPR-FPR$ is commonly used to pick the optimum point on the ROC curve, the F1 score can be used to pick the optimum point on the precision-recall curve. The F1 Score is calculated as follows -

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{(\text{Precision} + \text{Recall})}$$

Image by Author

The F1 score is the harmonic mean of precision and recall. We use the harmonic mean instead of a simple average because it punishes extreme values.

The plot of precision vs. recall for our default algorithm looks like this -



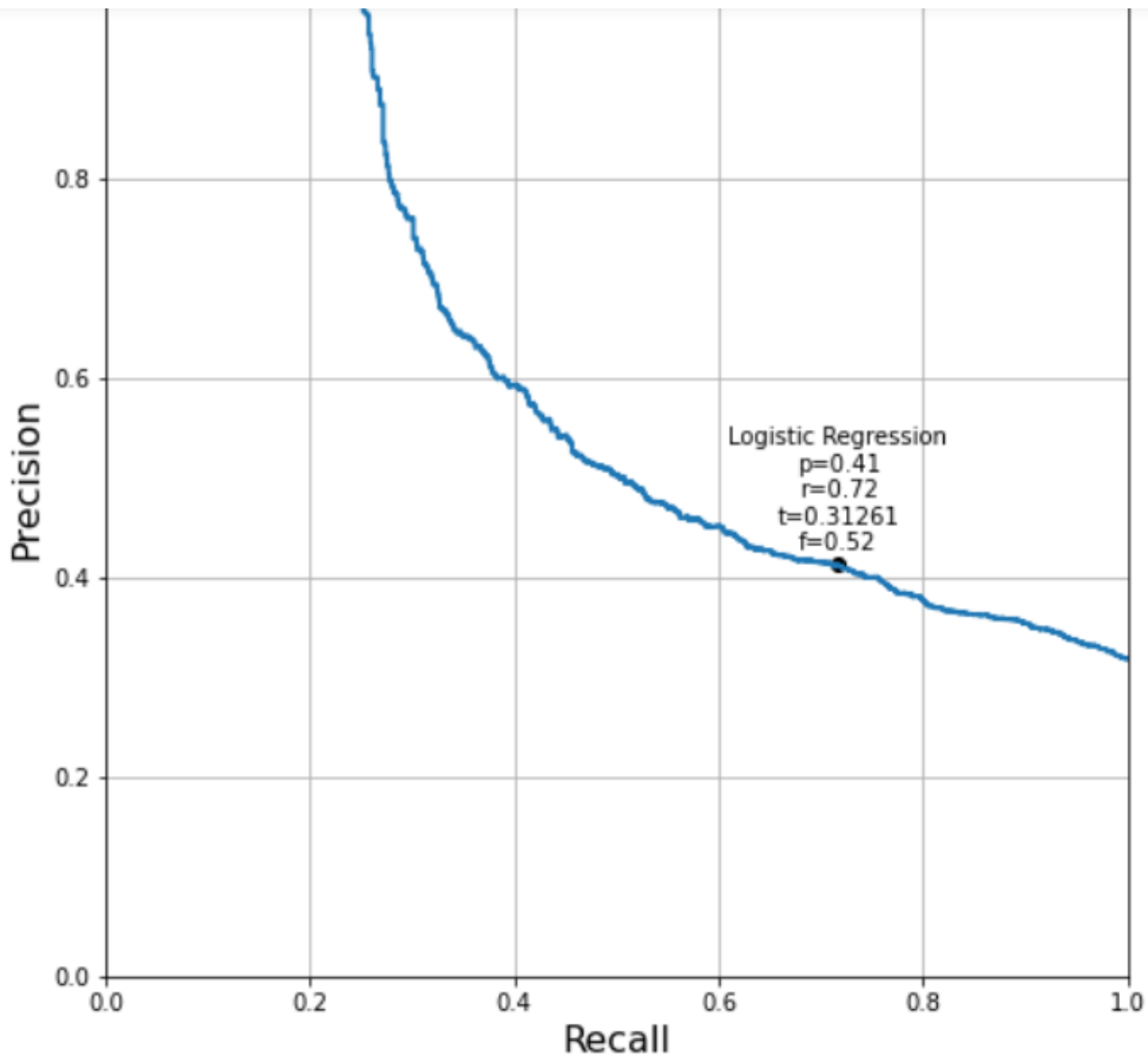
[Open in app](#)

Image by Author

Here we can see that the optimum (i.e. the maximum) F1 score is 0.5237148537347345 with the optimum precision and recall being 0.41 and 0.72 respectively. The threshold that would produce this outcome from the algorithm is 0.3126109044627986

Automating the Threshold Calculation Take 2

This second attempt at extending the `LogisticRegression` class added a new method for optimising for the F1 score to produce the optimal precision and threshold values -



[Open in app](#)

Now we can easily use the new version of the class to tell me what the threshold needs to be to optimise the f1 score and then to use that value to fine-tune the algorithm for the optimum precision and recall ...





Open in app

(0.3126109044627986, 0.5237148537347345)

Logistic Regression (Threshold Optimized for F1 Score = 0.52371)

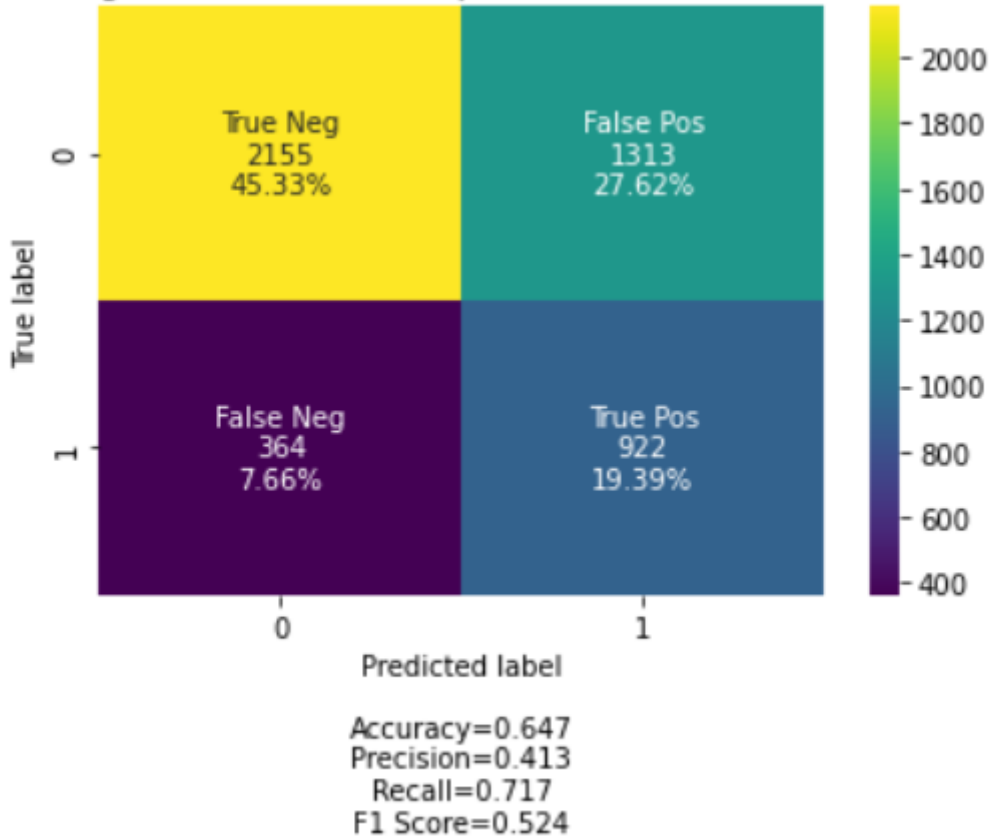


Image by Author



[Open in app](#)

Now imagine I took the new model back to the management team at the bank and they said “Well, that is an improvement but we absolutely have to have a recall of 90% for this model to be effective”.

Remember, recall is $TP/(TP+FN)$ and at the moment this model has $922 / (364+922) = 71.7\%$.

We would need to explain to the management team that recall and precision are trade offs so if the model is tuned for 90% recall the precision must decrease but if that were acceptable, how could this be achieved?

Automating the Threshold Calculation Take 3

This third attempt adds two new methods to calculate the required threshold for a given specific recall (and precision) which can now be used to achieve the performance requested by the management team -





Open in app

(0.19142214362243234, 0.35382262996941893)



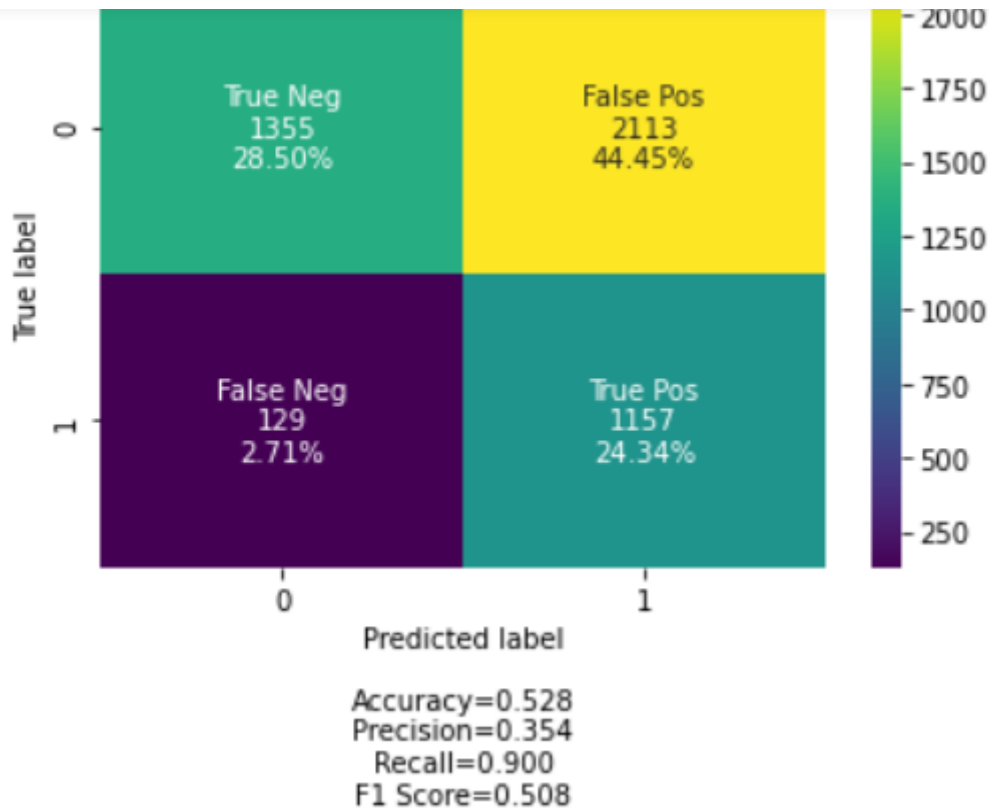
[Open in app](#)

Image by Author

We can see that the recall requested by the management team has been achieved and that as a result the precision has been traded off from 41.3% to 35.4%.

Automating the Threshold Calculation Take 4

At this point, I was getting rather carried away and I wanted a version of the optimisation that would implement a custom cost function.

Let's suppose the management team now give us the following new information -

- Every true negative means we can sell £10,000 of additional credit products to customers who will not default.
- Every false positive costs us £1,000 because we could have engaged those customers but instead, we avoided them.
- Every false negative costs us £1,500 because we failed to intervene and stop those customers defaulting.



[Open in app](#)

And with a bit more work we can modify our class to work out the threshold required to optimise this cost function and then apply it -





Open in app

... which can be used as follows -



[Open in app](#)

(0.4324142856743824, -40033500)

The new confusion matrix looks like this ..

Logistic Regression (Threshold Set using Cost Function 'default_cost_function')

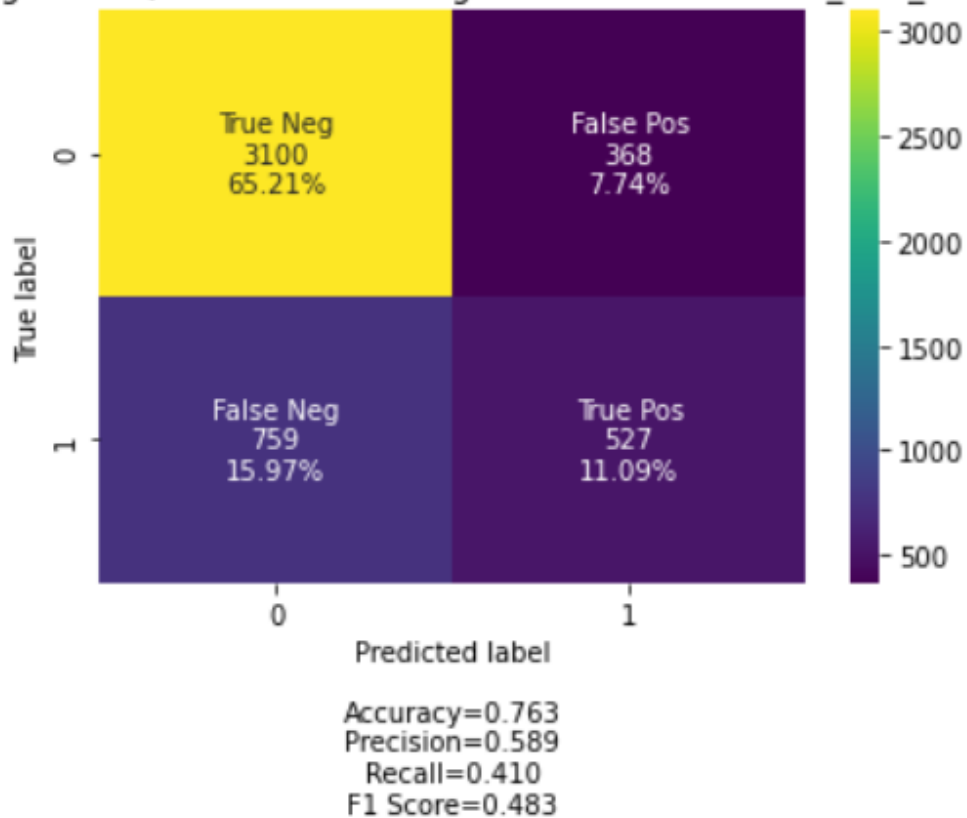


Image by Author

... and here is the plot of our custom cost function values vs. the threshold values ...



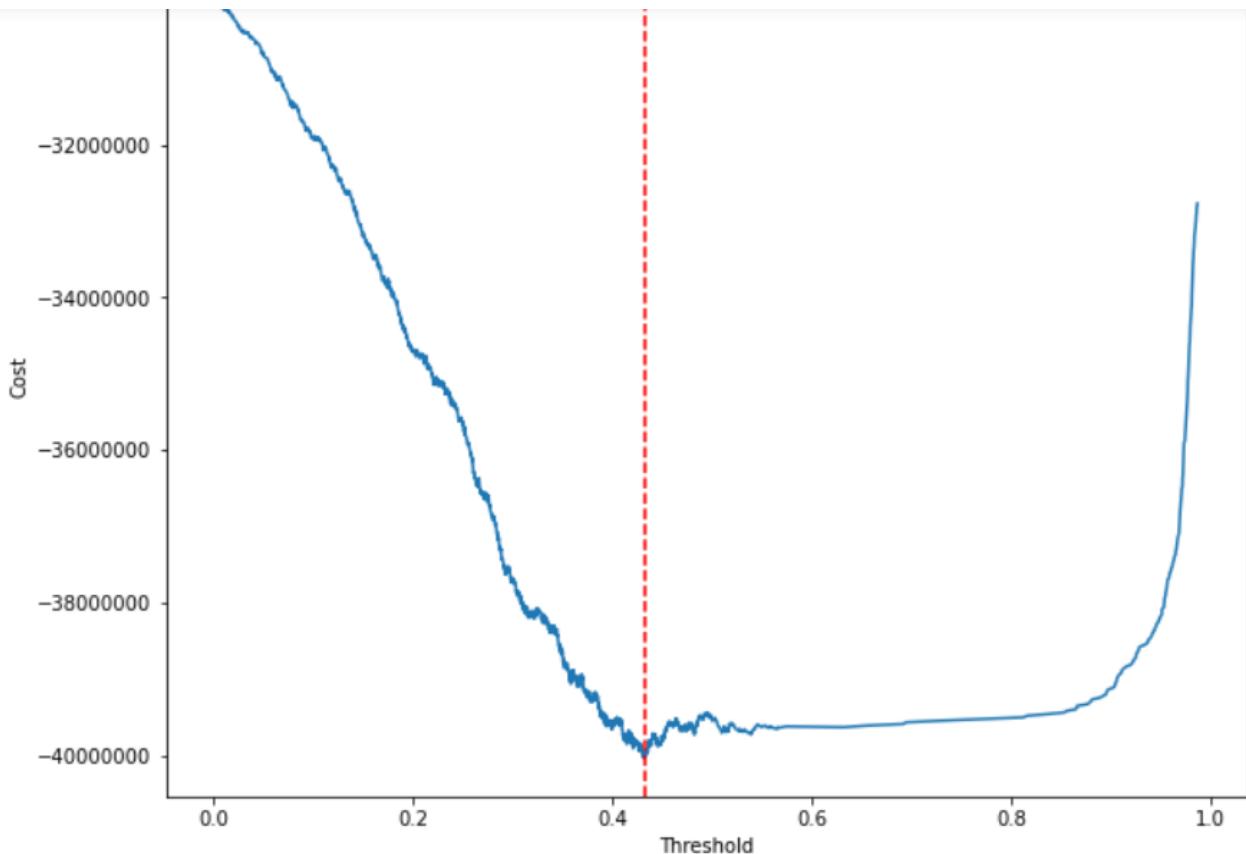
[Open in app](#)

Image by Author

Automating the Threshold Calculation — Final Version

Given that I had realised that the threshold for optimal accuracy could be calculated using a very similar approach to that for a custom cost function, I thought I would add that into the final version.

After all, many of the data science competitions do evaluate based on accuracy even though I had come to consider that it is not always, and in fact not often, the best measure of effectiveness of a classification algorithm.

That being said when I plugged the threshold for optimum accuracy calculated using the training data into the test data it lifted the accuracy measure of the predictions by a whole 1% and in a data science competition that might just make a difference!

The final version of the `LogisticRegressionWithThreshold` class adds a new method to calculate the threshold needed to optimize accuracy -





Open in app



[Open in app](#)

(0.5654501621491699, 0.7968026924694994)

Here is the confusion matrix for the optimal (highest) accuracy ...

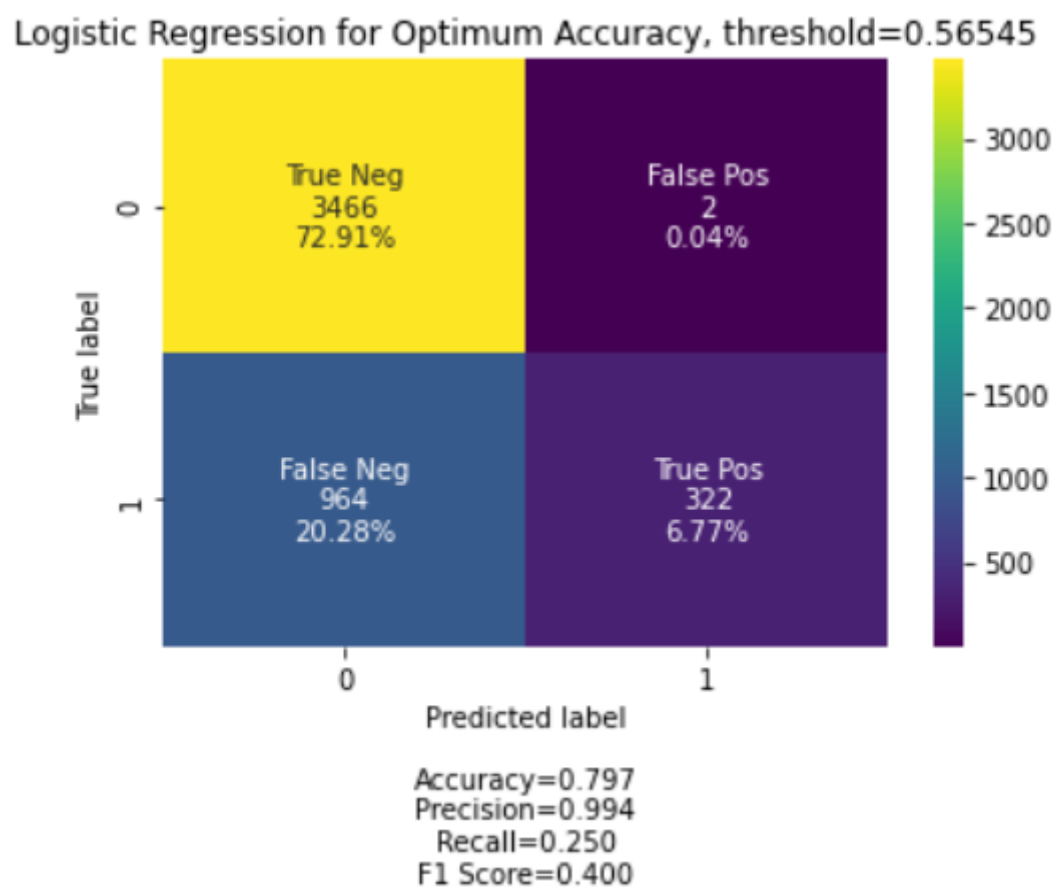


Image by Author



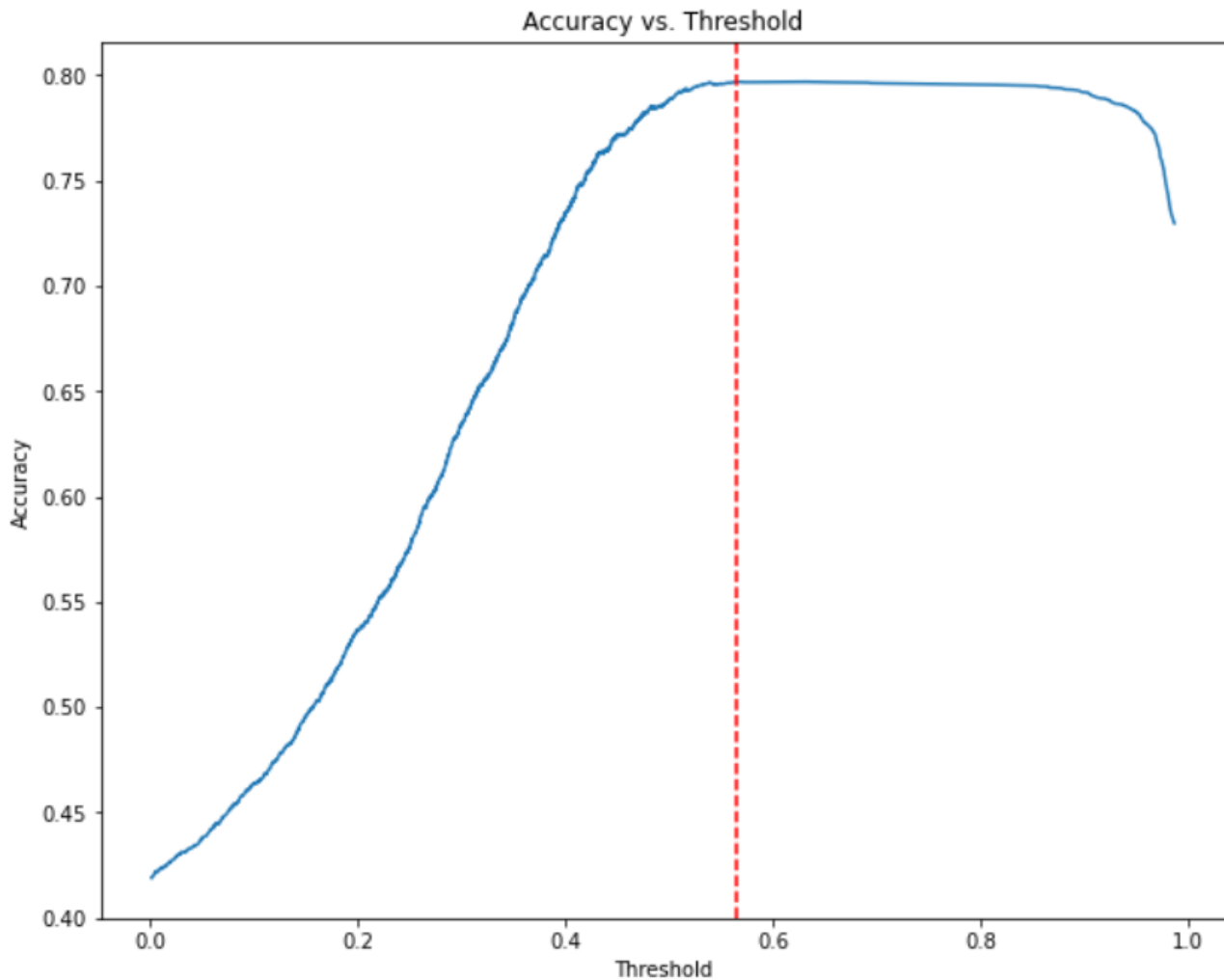
[Open in app](#)

Image by Author

Conclusion

We have seen that there are many ways to optimise a logistic regression which incidentally can be applied to other classification algorithms. These optimisations include finding and setting thresholds for the optimisation of precision, recall, f1 score, accuracy, tpr — fpr or custom cost functions.

We have also seen that the chosen optimisation is heavily dependent on the desired business outcomes.

We have seen that the `LogisticRegression` class in `sklearn.linear_model` does not have a way to set the threshold so that the algorithm can be optimised but that we can use the `LogisticRegressionWithThreshold` class developed in this article to give



[Open in app](#)

grahamharrison68/Public-Github

Repository for GH public projects. Contribute to grahamharrison68/Public-Github development by creating an...

github.com

Thank you for reading!

If you enjoyed reading this article, why not check out my other articles at <https://grahamharrison-86487.medium.com/>?

Also, I would love to hear from you to get your thoughts on this piece, any of my other articles or anything else related to data science and data analytics.

If you would like to get in touch to discuss any of these topics please look me up on LinkedIn — <https://www.linkedin.com/in/grahamharrison1> or feel free to e-mail me at GHarrison@lincolncollege.ac.uk.

Enjoy the read? Reward the writer.^{Beta}

Your tip will go to Graham Harrison through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Sign up for The Variable

By Towards Data Science





Open in app



Get this newsletter

