Logistic Regression in real-life: building a daily productivity classification model

Carolina Bento Just now · 11 min read ★

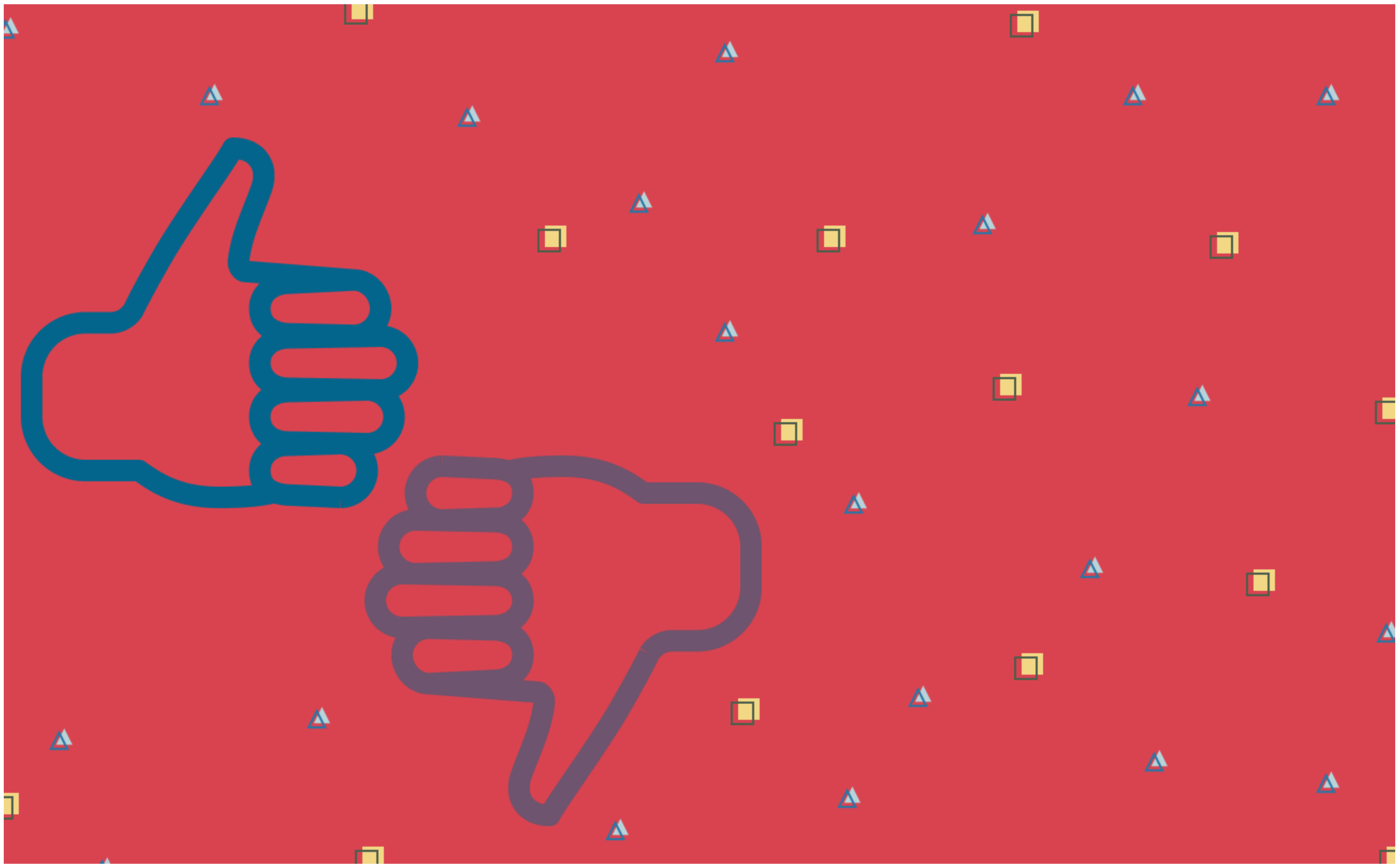


Image by author.

The name makes you think about Linear Regression, but it's not used to predict an unbounded, continuous outcome. Instead, it is a statistical classification model, it gives you the likelihood that an observation belongs to a specific class.

Logistic regression is used across many scientific fields. In Natural Language Processing (NLP), it's used to determine the sentiment of movie reviews, while in Medicine it can be used to determine the probability of a patient developing a particular disease.

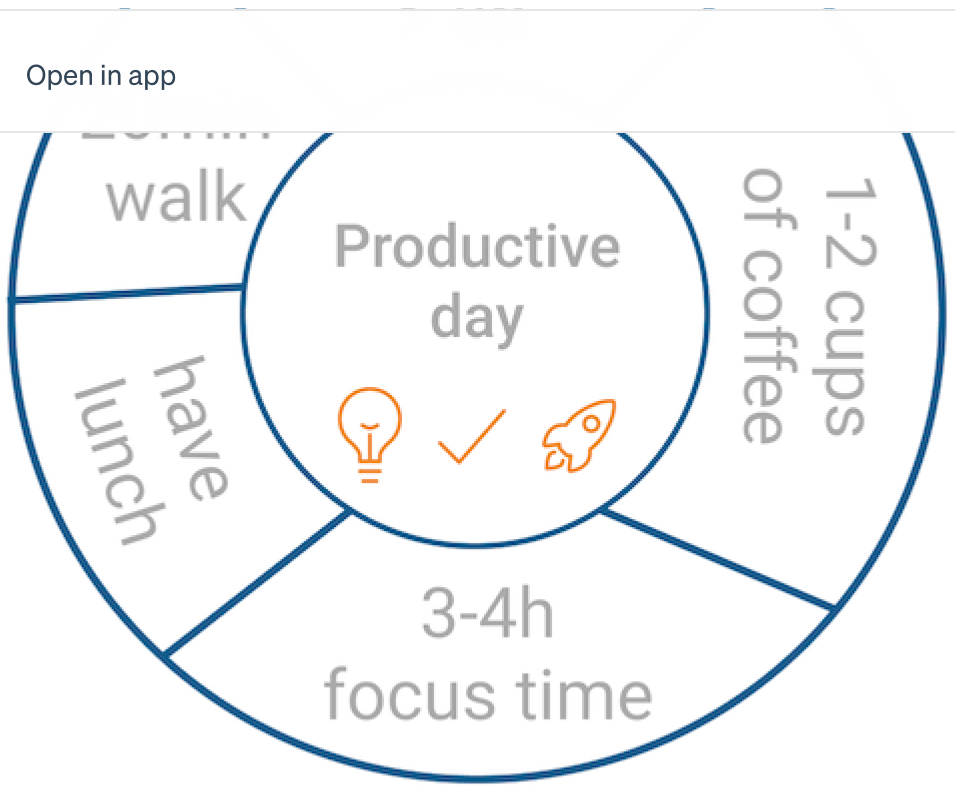
Classifying your daily productivity

Lately you've been interested in gauging your productivity. Not necessarily to quantify it to the smallest detail, but just to get a better sense if your day was really productive.

You've been asking yourself, at the end of each day, if the day was indeed productive. But that's just a potentially biased, qualitative data point. You want to find a more scientific way to go about it.

You've observed the natural ebbs and flows of your day, and realized that what impacts it the most is:

- **Sleep** you know that sleep, or lack thereof, has a big impact on your day.
- **Coffee** doesn't the day start after coffee?
- **Focus time** it's not always possible, but you try to have 3–4h of intently focused time to dive into projects.
- **Lunch** you've noticed the day flows smoothly when you have time for a proper lunch, not just snacks.
- **Walks** you've been taking short walks to get your steps in, relax a bit and muse about your projects.



What impacts your productivity.

You've heard about Machine Learning classification models and, since there are a lot of different models, with different levels of complexity and degree customization, you want to start with a simple one.

As you're thinking through which model to use, you realize these activities have a linear relationship with your overall productivity.

Then your first question is, *Could Linear Regression be the simplest, best tool for the job?*

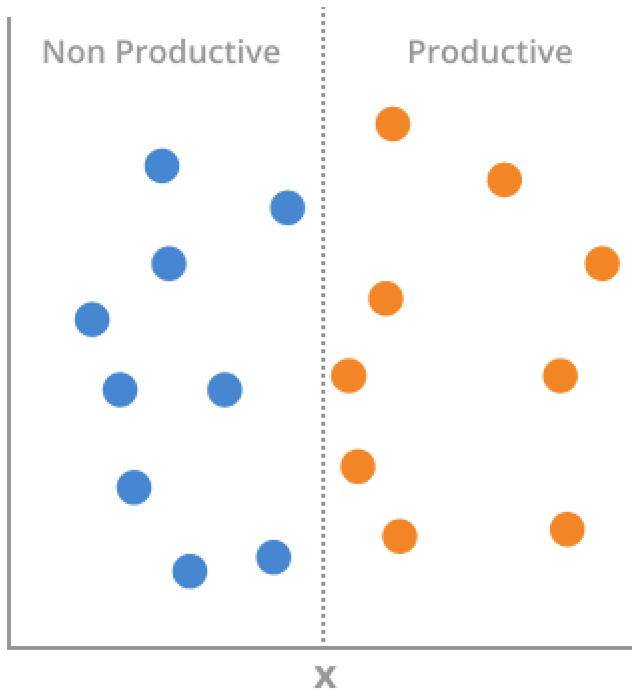
If it's a linear model, why not just use Linear regression?

In the end, you want to create a model that describes a linear relationship between a set of features, what impacts your productivity, and a target variable, a productive or non productive day.

So, why not keep things simple and use a Linear Regression model?

To classify your day as productive or not with a Linear Regression model, the first step is to pick an arbitrary threshold *x* and assign observations to each class based on a simple criteria:

Class *Productive* otherwise, i.e., all outcomes greater than *x*.



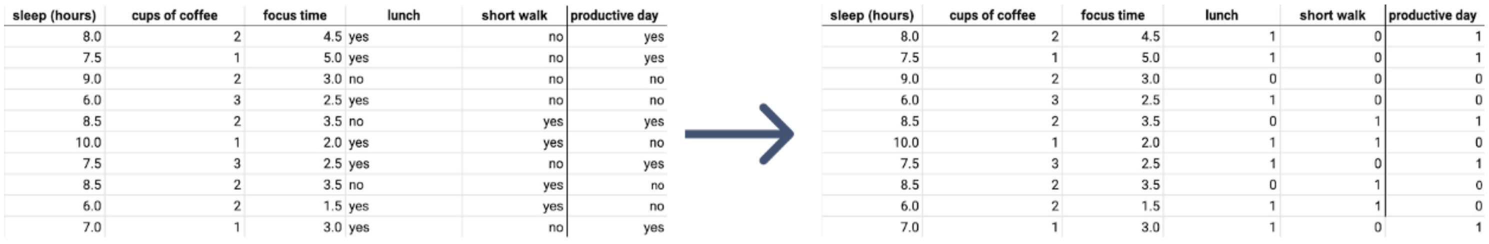
Using an arbitrary threshold x to determine which class the data belongs to.

This is straightforward!

You take the data and split it between training and testing sets, making sure to encode all categorical values into numerical ones.

> Linear Regression models don't know how to deal with categorical values, so you need to encode them.

And you can do this in way you like, as long as it's with an integer. In this case *No* becomes a 0 and *Yes* becomes a 1.



sleep (hours)	cups of coffee	focus time	lunch	short walk	productive day
8.0	2	4.5	yes	no	yes
7.5	1	5.0	yes	no	yes
9.0	2	3.0	no	no	no
6.0	3	2.5	yes	no	no
8.5	2	3.5	no	yes	yes
10.0	1	2.0	yes	yes	no
7.5	3	2.5	yes	no	yes
8.5	2	3.5	no	yes	no
6.0	2	1.5	yes	yes	no
7.0	1	3.0	yes	no	yes

sleep (hours)	cups of coffee	focus time	lunch	short walk	productive day
8.0	2	4.5	1	0	1
7.5	1	5.0	1	0	1
9.0	2	3.0	0	0	0
6.0	3	2.5	1	0	0
8.5	2	3.5	0	1	1
10.0	1	2.0	1	1	0
7.5	3	2.5	1	0	1
8.5	2	3.5	0	1	0
6.0	2	1.5	1	1	0
7.0	1	3.0	1	0	1

As for the arbitrary threshold, you've decided that:

- Outcomes less than or equal to zero are assigned to Class 0, i.e., a non productive day.
- Positive outcomes are assigned to Class 1, i.e., a productive day.

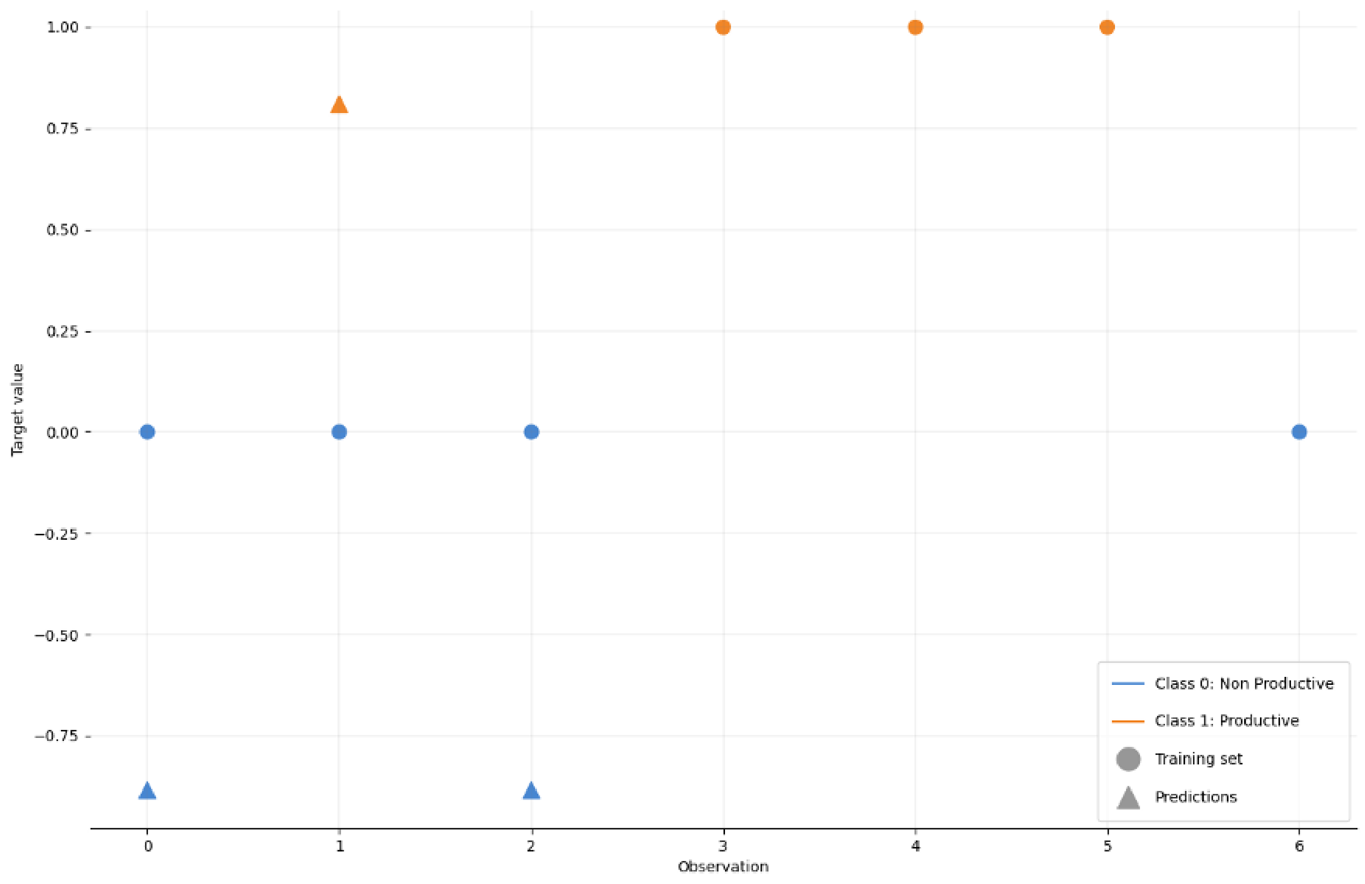
You run the data through a Linear Regression model using Python's SckitLearn and plot the outcomes along with their respective class.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from matplotlib.lines import Line2D
from sklearn.model_selection import train_test_split

def plot_results(train_targets, predictions):
    fig, ax = plt.subplots(figsize=(15, 10))
    # removing all borders except bottom
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)

    # adding major gridlines
    ax.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)

    training_colors = ['#4786D1' if target <= 0 else '#F28627' for
target in train_targets]
    prediction_colors = ['#4786D1' if target <= 0 else '#F28627' for
target in predictions]

    train_set_len = len(train_targets)
    predictions_len = len(predictions)

    plt.scatter(np.arange(0, train_set_len), train_targets,
color=training_colors, marker='o', s=[12 * train_set_len])
    plt.scatter(np.arange(0, predictions_len), predictions,
color=prediction_colors, marker='^', s=[40 * predictions_len])

    ax.set_xlabel('Observation')
    ax.set_ylabel('Target value')

    # Customizing symbols in the legend
    legend_items = [Line2D([0], [0], color='#4786D1', markersize=15),
        Line2D([0], [0], color='#F28627', markersize=15),
markerfacecolor='#979797', markeredgecolor='#979797', markersize=15)]

    # Adding some spacing between each legend row and padding
    ax.legend(handles=legend_items,
    labels=['Class 0: Non Productive', 'Class 1: Productive',
'Training set', 'Predictions'],labelspacing=1.5, borderpad=1)

    plt.show()

def fit_linear_regression(features, targets):
    train_features, test_features, train_targets, test_targets =
train_test_split(features, targets, test_size=0.25, random_state=123)

    model = linear_model.LinearRegression()
    fitted_model = model.fit(train_features, train_targets)
    predictions = fitted_model.predict(test_features)

    print('---Linear Regression')
    print('Coefficients: ' + str(fitted_model.coef_))
    print('Intercept: ' + str(fitted_model.intercept_))
    print('R-squared: ' + str(fitted_model.score(train_features,
np.array(train_targets).reshape(-1, 1))))

    plot_results(train_targets, predictions)

productivity_features = [[8.0, 2, 4.5, 1, 0],
                         [7.5, 1, 5.0, 1, 0],
                         [9.0, 2, 3.0, 0, 0],
                         [6.0, 3, 2.5, 1, 0],
                         [8.5, 2, 3.5, 0, 1],
                         [10.0, 1, 2.0, 1, 1],
                         [7.5, 3, 2.5, 1, 0],
                         [8.5, 2, 3.5, 0, 1],
                         [6.0, 2, 1.5, 1, 1],
                         [7.0, 1.0, 3.0, 1, 0]]

productivity_targets = [1, 1, 0, 0, 1, 0, 1, 0, 0, 1]

fit_linear_regression(productivity_features, productivity_targets)
```

Linear Regression: not the best tool for this job

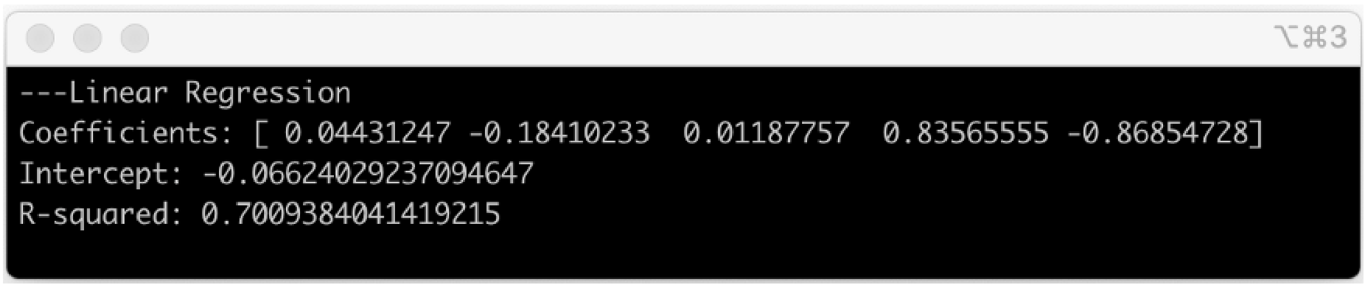
Plotting the targets, or outcomes, for the training set along with model predictions you can see that some points are incorrectly classified.

In some cases, the model didn’t do a god job at predicting the correct class.



Scatter plot of the target values for training (circle) and values predicted by the Linear Regression model (triangle). Color-coded according to their class: blue for Class 0 and orange for Class 1.

So, to get better idea of the model's performance you check its R-squared, which is a measure of how well these features help determine the outcome.



```
---Linear Regression
Coefficients: [ 0.04431247 -0.18410233  0.01187757  0.83565555 -0.86854728]
Intercept: -0.06624029237094647
R-squared: 0.7009384041419215
```

Output of the Linear Regression Model.

0.7 is not bad. Sure, this model has room for improvement, but it's not a complete flop!

1. Implies outcome values have a specific order.
2. Produces continuous, unbounded, outcomes.
3. Only explains how a single feature impacts the target.

1. Implies outcome values have a specific order

If you use only a threshold to assign classes, you're assuming the outcome has an implicit order.

For instance, you decided that all outcomes less than or equal to *0* belong to *Class 0,* while all other outcomes belong to *Class 1*.

So, with this approach, you're implicitly assuming that all observations from *Class 0* must come before all observations for *Class 1*.

You have control over this threshold, so you can pick any number you want. But the model will blindly use the threshold it is given when, in practice, the data may not follow that particular order.

As a consequence, the model tends to have lower accuracy.

2. Produces continuous, unbounded, outcomes

In classification problems the outcome variable is *categorical*, in this case, a productive or non productive day.

But you're fitting the data to a regression model that can only produce continuous outcomes.

The fact that outcomes are continuous is not a problem. The big limitation is that these outcomes are not restricted to values between 0 and 1.

And you just saw it in action. Even though the possible targets were 0 and 1, some predictions were exactly zero, others were positive but not exactly one. Some predictions were even negative!

variable and a set of features, but the model coefficients only tell you how a particular feature, independently and in isolation from other features, impacts the target.

> Linear Regression maps the relationship between the target and a particular feature, assuming all other features are fixed.

But in a classification task you want to measure how the entire feature set impacts the outcome, because all features *exist* at the same time, not in isolation[1].

Logistic Regression to the rescue!

Knowing these limitations, it's clear that Linear Regression is not the best tool for a classification task.

One viable option is to use **Logistic Regression**. It addresses Linear Regression's limitations while sharing a lot of its characteristics. For instance, both Logistic and Linear Regression are Generalized Linear Models, and describe a linear relationship between a target and the set of independent variables.

But Logistic Regression is more suitable for a binary classification task, because it:

- Predicts the likelihood of a Bernulli event, where the outcome can only be *yes/no* or *positive/negative*.
- Finds a function *g,* also called *link function*, that takes the likelihood of an observation being part of a class, and models its result as a linear combination of predictors.
- Assumes it's not the model's outcome that changes linearly as features change. It's the function *g* that varies linearly as features change.

For a model with only one feature, or predictor, the **link function** *g* can be described as:

$$g(P(Y = \text{Class } 1 \mid X = \text{ observation})) = \beta_0 + \beta_1 x_1$$

Logistic Regression uses the *link function*, but its outcome is a probability, which means it can only predict values between 0 and 1.

So the question becomes, *how does Logistic Regression turn the unbounded, continuous values of the linear combination into a probability?*

The logit function

To do this mathematical feat and restrict its outcome to be between 0 and 1, Logistic Regression uses the **logit function** as its link function to represent the **log-odds,** the logarithm of the odds ratio for a probability *p*.

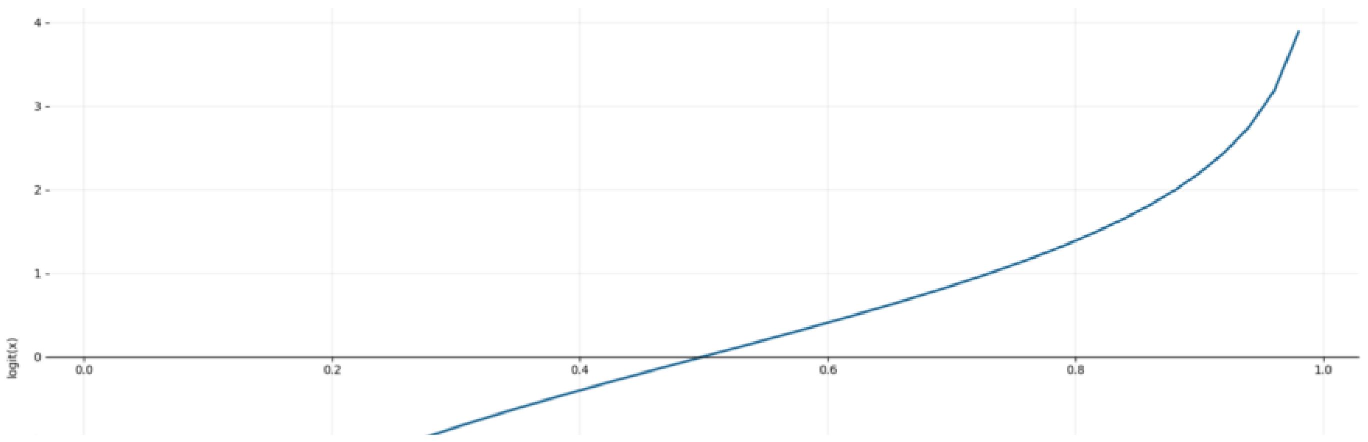
In this case, the log-odds of a productive day is the ratio of the probability of a productive day (*p*) to the probability of a non productive day (1- *p*).

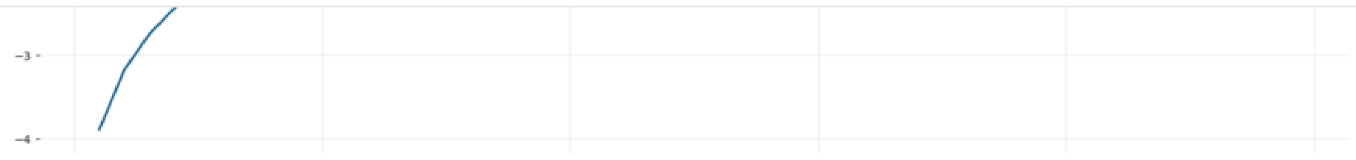
That's why the logit function is also known as the log-odds function.

$$g(p) \equiv \text{logit}(p) \Rightarrow g(p) = \log\left(\frac{p}{1-p}\right)$$

But why logit and not some other function?

The *logit* function is a sigmoid function. It asymptotes at 0 and 1 so, as model outcomes get closer to *-infinity* the result of *logit* gets closer to 0. On the other hand, as outcomes get closer to +*infinity*, the result of *logit* gets closer to 1.





Pot of a logit function.

The logit function great, because it keeps values between 0 and 1, but you're not interested in the *log-odds*.

In a binary class task, you want the actual probabilities.

The logistic function

In practice, you want to *extract* the probabilities from the log-odds.

Sounds like a difficult task but, the good news is, it only requires a couple mathematical transformations!

Here's how you can *extract* the probability of *success* (p) from the log-odds.

First you *remove* the logarithm and then, after some algebraic manipulation, you're able to *isolate* *p*.

$$g(\mathrm{P}(\mathrm{Y} = \text{Class } 1 \mid \mathrm{X} = \text{ observation})) = \beta_0 + \beta_1 x_1$$

$$g(p) = \beta_0 + \beta_1 x_1$$

logit function constraints results between 0 and 1

$$\operatorname{logit}(p) = \beta_0 + \beta_1 x_1$$

$$\log\left(\tfrac{p}{1-p}\right) = \beta_0 + \beta_1 x_1$$

remove the logarithm

$$\mathrm{e}^{\operatorname{logit}(p)} = \beta_0 + \beta_1 x_1$$

$$\mathrm{e}^{\log\left(\frac{p}{1-p}\right)} = \beta_0 + \beta_1 x_1$$

$$\tfrac{p}{1-p} = \mathrm{e}^{\beta_0+\beta_1 x_1}$$

$$p = \mathrm{e}^{\beta_0+\beta_1 x_1} \times (1-p)$$

$$p = \mathrm{e}^{\beta_0+\beta_1 x_1} - (p \times \mathrm{e}^{\beta_0+\beta_1 x_1})$$

$$\boxed{\begin{aligned}
&\frac{\mathrm{e}^{\beta_0+\beta_1 x_1}}{1+\mathrm{e}^{\beta_0+\beta_1 x_1}} \equiv \\
&\equiv \frac{1}{\frac{1}{\mathrm{e}^{\beta_0+\beta_1 x_1}} + \frac{\mathrm{e}^{\beta_0+\beta_1 x_1}}{\mathrm{e}^{\beta_0+\beta_1 x_1}}} \equiv \\
&\equiv \frac{1}{\frac{1}{\mathrm{e}^{\beta_0+\beta_1 x_1}}+1} \equiv \\
&\equiv \frac{1}{1+\mathrm{e}^{-(\beta_0+\beta_1 x_1)}}
\end{aligned}}$$

divide both sides by $\mathrm{e}^{\beta_0+\beta_1 x_1}$

$$p = \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}} \xrightarrow{\text{Equivalent expression}} p = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1)}}$$

At the end of this process the probability *p*, the probability of *success*, is a function of a linear combination.

Mathematically speaking, what you achieved is the inverse of the logit function, a function that's called **logistic function**.

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1)}} \equiv p(z) = \frac{1}{1+e^{-z}}$$

Definition of the logistic function.

A function of the linear combination *z*, in its short form.

If you're interested in the probability of *failure*, you can do an equivalent manipulation and isolate (1- *p*) instead of *p*.

Putting all of this together, now know you that Logistic Regression uses the logistic function to predict the likelihood that an observation belongs the *positive* class. A probability that is also a function of a linear combination, like the one used in Linear Regression.

The name Logistic Regression makes much more sense!

There's only one detail the model needs to take into account, before it can classify your day as productive or not.

Defining the decision boundary

For each observation, the Logistic Regression model calculates the probability that the observation belongs to the *positive* class, in this case, a productive day. It looks

$$\text{P}(Y = \text{Class Productive} \mid X = \textit{observation})$$

But you're using this model in a classification task, so it still has to make a decision about which class to assign to each observation.

The model needs some criteria to distinguish between observations that correspond to a productive day from the ones that correspond to a non productive day. It needs a **decision boundary**.

> The decision boundary is the imaginary border that separates the observations from the positive class from the observations from the negative class.

Depending on the problem at hand, you can set the decision boundary to be any probability you'd like. But typically, the decision boundary is set to 0.5.

So, if the model predicts the observation has a probability:

- **Lower or equal to 0.5**, the observation is assigned to the negative class, e.g., a non productive day.
- **Greater to 0.5**, the observation is assigned to the positive class, e.g., a productive day.

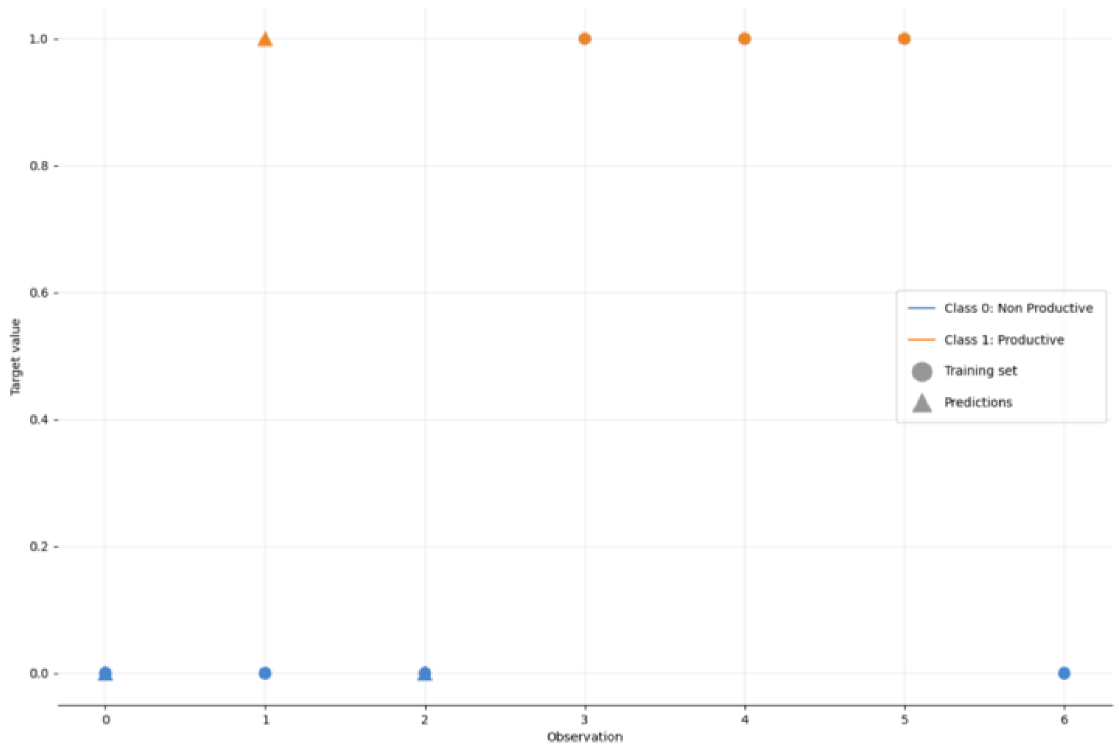
Is your day (really) productive?

To answer this question you can built onto what you had already done for Linear Regression, and turn to ScikitLearn once again to fit a Logistic Regression model.

```
from sklearn.linear_model import LogisticRegression

def fit_logistic_regression(features, targets):
    train_features, test_features, train_targets, test_targets = 
train_test_split(features, targets, test_size=0.25, random_state=123)
```

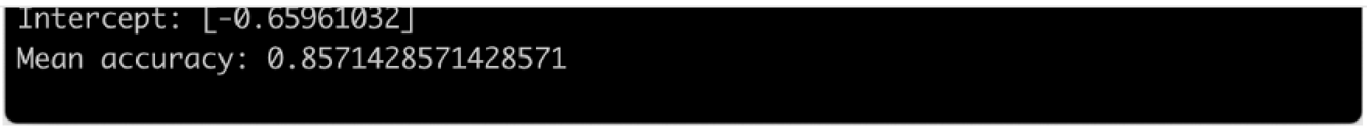
```
    print('---Logistic Regression')
    print('Coefficients: ' + str(fitted_model.coef_))
    print('Intercept: ' + str(fitted_model.intercept_))
    print('Mean accuracy: ' + str(fitted_model.score(train_features, 
np.array(train_targets).reshape(-1, 1))))

    plot_results(train_targets, predictions)

fit_logistic_regression(productivity_features, productivity_targets)
```

However, just by looking at the plot of the model outcomes, it's hard to tell if the model does a good job.



Scatter plot of the target values for training (circle) and values predicted by the Logistic Regression model (triangle). Color-coded according to their class: blue for Class 0 and orange for Class 1.

To measure performance you need to look at the model's **accuracy**.



```
Intercept: [-0.65961032]
Mean accuracy: 0.8571428571428571
```

Output of the Logistic Regression model.

An accuracy of 85% means your model correctly predicted 85% of the test observations.

Now you don't have to rely solely on a qualitative, self-evaluation to tell if your day was indeed productive!

You've worked through a quantitative, and reproducible, way of classifying your day's productivity.

In the process, you also got a better understanding about why Linear Regression is not the best model for a classification task, even when the relationship between features and targets is linear.

Hope you enjoyed learning about Logistic Regression.

Thanks for reading!

References

[1] Stoltzfus JC. Logistic regression: a brief primer. Acad Emerg Med. 2011 Oct;18(10):1099–104.

Images by author except where stated otherwise.

Thanks to Caitlin Kindig.

Data Science Logistic Regression Linear Regression Machine Learning Classification