Optional: Logistic Regression as Linear Regression with Log Odds

Module 2 | Chapter 2 | Notebook 4

This notebook is optional. You don't have to work through it to complete this training successfully. In this notebook you will learn more details about how linear and logistic regression are related.

Log Chances

Let's import the Data directly now in order to get started quickly.

In [1]:
```
# module import
import pandas as pd

# data gathering
df = pd.read_csv("social_media_train.csv", index_col=[0])

#look at data
df.head()
```

Out[1]:

	fake	profile_pic	ratio_numlen_username	len_fullname	ratio_numlen_fullname	sim_name_username
0	0	Yes	0.27	0	0.0	No match
1	0	Yes	0.00	2	0.0	Partial match
2	0	Yes	0.10	2	0.0	Partial match
3	0	Yes	0.00	1	0.0	Partial match
4	0	Yes	0.00	2	0.0	No match

Linear regression and Logistic regression basically use the same formula. The only difference is that the linear regression predicts target values with the formula, while logistic regression uses it to predict **log odds**. To understand this, we first need to look at what **log odds** actually are. Let's use a weather example to show the differences.

Suppose we want to know what the weather will be like this week, based on the weather of the same week last year. In the same week last year there were 3 days of rain and 4 days of sun. The **probability** of it being sunny on a given day is 4 (sunny days) out of 7 (weekdays), or 0.57.

In [2]:
```
probability_sunny = 4 / 7
print("Probability of sunny day:", round(probability_sunny, 2))
```

```
Probability of sunny day: 0.57
```

The **odds** are the probability divided by the converse-probability.

\begin{equation*} odds = \frac{probability}{Converse Probability} \end{equation*}
So to get the **chance** (*odds*) of sunny weather, you divide the probability of sunshine (0.57) by the probability of rain (0.43). The odds are 1.33.

In [3]:
```
probability_not_sunny = 1 - probability_sunny
print("Probability of *not* sunny day:", round(probability_not_sunny, 2))
odds_sunny= probability_sunny / probability_not_sunny
print("The odds of sunny day:", round(odds_sunny, 2))
```

```
Probability of *not* sunny day: 0.43
The odds of sunny day: 1.33
```

If the odds are higher than 1, like in our case, the probability of the case arising (sunny day) is higher than that of it not arising (rainy day). The reason is that the probability in the numerator is greater than the converse-probability in the denominator.

If the chance is 1, the probability and the converse-probability are equal. Then you can't say that one case or the other is more likely to happen. That's when the uncertainty is greatest.

If the chance is less than 1, the probability is smaller than the converse-probability. This is the case here, for example on rainy days.

Calculate the chance of having a rainy day. The probability is $\frac{3}{7}$.

In [4]:
```
probability_sunny = 3 / 7
probability_not_sunny = 1 - probability_sunny
odds_sunny= probability_sunny / probability_not_sunny
print("The odds of sunny day:", round(odds_sunny, 2))
```

```
The odds of sunny day: 0.75
```

Since the odds are less than 1, it's less likely that a rainy day will occur than that a rainy day will not occur.

So now how do you get from the odds to the **log odds**? It's very simple - you take the natural logarithm of the odds. For example, you can use `np.log()` for this. For a sunny day it would look like this.

In [6]:
```
import numpy as np
print("Log odds of having sunny day:", round(np.log(odds_sunny), 2))
```

```
Log odds of having sunny day: -0.29
```

Calculate the log odds for a rainy day.

In [8]:
```
print("Log odds of having sunny day:", round(np.log(probability_not_sunny /probability
```

```
Log odds of having sunny day: 0.29
```

As you may have already guessed, a positive log odds value means that the odds are in favor of the case arising. In our case, that a sunny day is more likely than a rainy day.

The interpretation of probabilities, odds and log odds all seem very similar. It looks like the difference lies mainly in the thresholds. This is also worth noting, so here's a short summary again:

x	probability of case arising	odds of case	log odds of the case
probability > converse-probability	$probability > 0.5$	$odds > 1$	$log(odds) > 0$
probability == converse-probability	$probability = 0.5$	$odds = 1$	$log(odds) = 0$
probability < converse-probability	$probability < 0.5$	$odds < 1$	$log(odds) < 0$

So why go to the effort to generate log odds from probabilities using a sigmoid transformation? The reason lies not so much in the thresholds, but rather in the distribution of the values. You can only use a linear regression to predict categories when the probabilities have been transformed into log odds.

So we call linear regression with log odds **logistic regression**. It does not directly predict categories (`1` or `0`), but log odds:

\begin{equation*} log\left(\frac{Probability}{1 - Probability}\right) = intercept + (slope \cdot Feature) \end{equation*}
These can then be easily converted into probabilities for categories and interpreted in this way. The formula for this is is changed to $Probability$:

\begin{equation*} Probability = \frac {1} {1 + e^{-(intercept + (slope \cdot Feature))}} \end{equation*}
This conversion turns the straight regression line into the typical S-curve of a logistic regression.

Congratulations: You have learned what log odds are and how logistic and linear regression are related. Next we'll look into this connection with a visual approach.

Curve and line

We can also visualize how the S-curve of logistic regression becomes the straight line of linear regression. It is best to first train a logistic regression model in order to generate predictions afterwards.

Instantiate a logistic regression model using `LogisticRegression` from the `sklearn.linear_model` module and name it `model_log`. Set the `solver` parameter to `'lbfgs'`. Then create a feature matrix `features_train` with only one feature, consisting of the `'ratio_numlen_username'` column of `df`. The target vector should be the `fake` column. Then fit the model to the data.

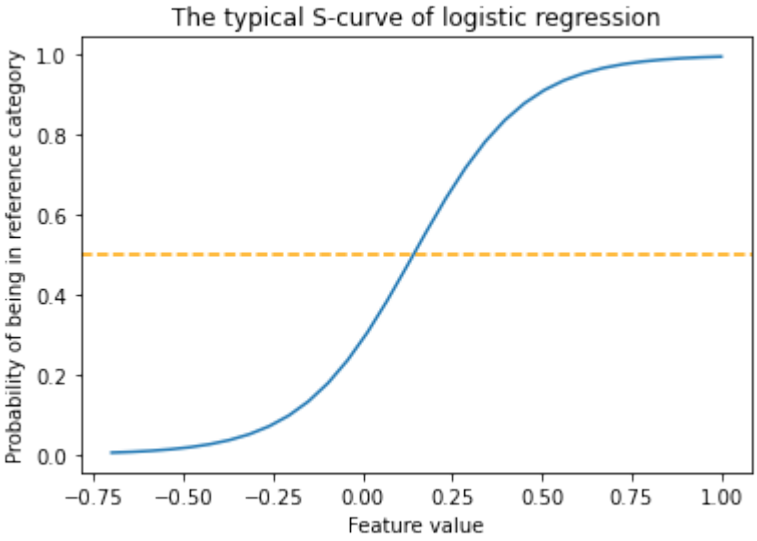
In [12]:
```
from sklearn.linear_model import LogisticRegression
model_log = LogisticRegression(solver='lbfgs')
features_train = df.loc[:,['ratio_numlen_username']]
target_train = df.loc[:,'fake']
model_log.fit(features_train, target_train)
```

Out[12]:
```
LogisticRegression()
```

By default, you predict categories using logistic regression with `my_model.predict()`. To get to the probabilities used for the category prediction, we'll use `my_model.predict_proba()`. We'll also use negative feature values to see a clear S-curve.

In [13]:
```
# create artificial feature values
features_pred = pd.DataFrame({'ratio_numlen_username': np.linspace(-0.7, 1, 32)})

# predict probabilities of artificial feature values
target_train_pred_proba = model_log.predict_proba(features_pred)

# module import for visualisation
import seaborn as sns

# line plot
ax = sns.lineplot(x=features_pred.iloc[:, 0],
            y=target_train_pred_proba[:, 1])

# labels
ax.set(title='The typical S-curve of logistic regression',
       xlabel='Feature value',
       ylabel='Probability of being in reference category')

# orange horizontal line
ax.axhline(0.5, ls='--', color = "orange")
```

Out[13]:
```
<matplotlib.lines.Line2D at 0x7f256c58cfa0>
```

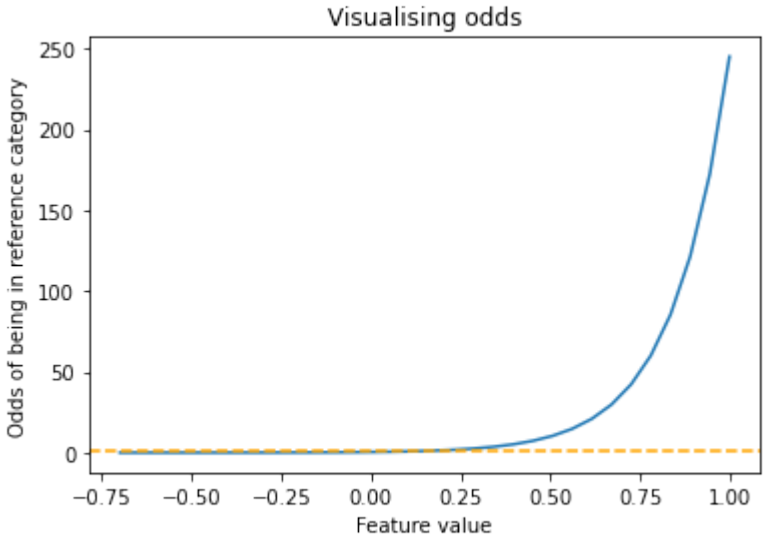


This S-curve is created when you predict probabilities. The categorical prediction (e.g. fake or not fake) refers to whether the probability value is below `0.5` (50%) or not. That's why we've drawn this threshold in as a dotted line.

The y-axis now displays the probability values. If you convert these values (`target_train_pred_proba`) into odds (`target_train_pred_odds`), the S-curve looks completely different.

In [15]:
```
# calculate odds
target_train_pred_odds = target_train_pred_proba[:, 1] / (1 - target_train_pred_proba[

# line plot
ax = sns.lineplot(x=features_pred.iloc[:, 0],
            y=target_train_pred_odds)

# labels
ax.set(title='Visualising odds',
       xlabel='Feature value',
       ylabel='Odds of being in reference category')

# orange horizontal line
ax.axhline(1, ls='--', color = "orange")
```

Out[15]:
```
<matplotlib.lines.Line2D at 0x7f256a46b100>
```

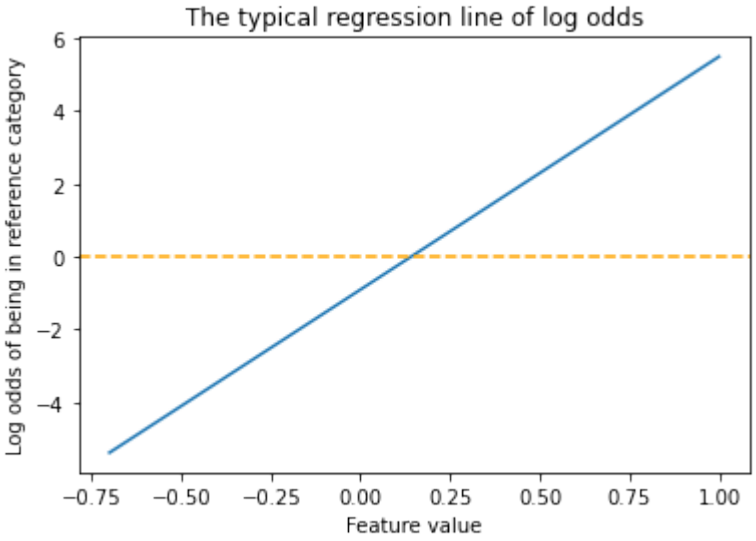


As you can see, the S-curve of the sigmoid function now becomes an exponential function. Remember that for odds the value 1 is critical. The categorical prediction (e.g. fake or not fake) is based on whether the probability value is below `1` or not. So we've drawn this threshold in as a dotted line.

Based on the formulas for linear and logistic regression, we've established that log odds are predicted with a regression line. If this is the case, we should see a straight line when visualizing the log odds.

In [17]:
```
# calculate log odds
target_train_pred_log_odds = np.log(target_train_pred_odds)

# line plot
ax = sns.lineplot(x=features_pred.iloc[:, 0],
            y=target_train_pred_log_odds)

# labels
ax.set(title='The typical regression line of log odds',
       xlabel='Feature value',
       ylabel='Log odds of being in reference category')

# orange horizontal line
ax.axhline(0, ls='--', color = "orange")
```

Out[17]:
```
<matplotlib.lines.Line2D at 0x7f256a392d30>
```



It's true. This line is a regression line as you learned in the lesson *Simple Linear Regression with sklearn (Module 1, Chapter 1)*. The special thing about it is the nature of the predicted values: Log odds So you now see log odds on the y-axis.

You saw above that the value zero is critical for log odds. So it's displayed here as a dotted line. The categorical prediction (e.g. fake or not fake) is based on whether the probability value is below `0` or not.

Remember this regression line of log odds if you don't know how to interpret a certain aspect of logistic regression. For example, the feature coefficients that the model learns from the training data represent slope values of this regression line of log odds. In *Simple Linear Regression with sklearn (Module 1, Chapter 1)* you learned that a change of 1 in the feature results in a change of the slope value in the predicted target value. In this case the target value is log odds and the principle still applies.

Negative feature coefficients therefore represent a negative influence of the feature on the probability of predicting the feature category. By the same token, the same applies to positive feature coefficients. A feature coefficient of about 0 means that the feature has virtually no effect on the predictions.

Congratulations: You have used a visual approach to see that logistic regression is basically just an alternative version of linear regression used for predicting probabilities. In the next notebook we'll prepare the categorical features to generate good predictions.

Remember:

- Logistic regression is a linear regression that predicts log odds.

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.
