Logistic Regressions - Part 1

Parkinson's disease doesn't announce itself loudly—it whispers. Subtle tremors in the voice, changes in pitch, and micro-pauses in speech may reveal early signs long before a patient notices symptoms. But can a machine detect these patterns better than a human ear? The answer is yes. With the help of *logistic regression*, we can classify patients into two groups: "Parkinson's detected" or "no Parkinson's detected."

In previous posts, we've discussed regression models that predict a continuous regressand. But which model should we use when the goal is sorting data into predefined categories? This is where *classification models* come in. These models analyze input data, identify key patterns, and apply this knowledge to assign new, unseen data points to the appropriate category or *class*[1]. Among classification models, logistic regression is one of the simplest yet powerful tools.

What is Logistic Regression?

Logistic regression, also known as a logit model, estimates the probability of an event occurring[2]. Using a mathematical *sigmoid function,* it transforms predictors into a probability score between 0 and 1, which we'll learn more about later. In our Parkinson's classification task, logistic regression predicts whether a patient belongs to one of two classes: "Parkinson's detected" or "no Parkinson's detected" based on their speech patterns.

Logit models aren't a one-size-fits-all model; they're versatile enough to handle various types of classification problems. The choice of model depends on the structure of the outcome variable:

1. A binary logistic model is used when the outcome variable has two possible outcomes.
2. A nominal or multinomial logistic model is used when the outcome variable has three or more possible outcomes.
3. An ordinal logistic model is used when the outcome variable has three or more ranked outcomes (e.g., 1 = good, 2 = better, 3 = best)

[1] Chris Drummond, "Classification," Encyclopedia of Machine Learning and Data Mining, Springer, 2017
[2] IBM. "Logistic Regression." *Ibm.com*, 16 Aug. 2021, www.ibm.com/think/topics/logistic-regression.

4. Logistic regression models can also be tailored for more specific uses like conditional logistic regressions and Firth logistic regressions.

To decide which logistic regression to use for the Parkinson's problem, let's take a look at the data.

Our dataset contains 195 observations and 24 variables, with the outcome variable 'status' indicating the presence (1) or absence (0) of Parkinson's. This binary structure aligns perfectly with the requirements of a binary logistic regression model. Before diving into the model-building process, it's crucial to ensure that the dataset meets the key assumptions for logistic regression. Adhering to these assumptions lays the foundation for accurate and unbiased predictions. Let's review these assumptions.

Assumptions of Logit

1. Predicted Variable Structure

The outcome variable of a dataset must be categorical - not continuous - for logistic regression. As 'status' is binary with values of 0 and 1, it fulfills the categorical requirement.

2. Independence of Observations

Each data point should be independent, meaning no observations are overly correlated. High correlation misleads the estimated impact of each variable, causing unstable coefficients and, ultimately, unreliable predictions.

3. Absence of Multicollinearity

Multicollinearity occurs when two or more independent model variables are highly correlated, making it difficult to isolate their individual effects on the outcome variable. To check for multicollinearity, let's build a *correlation matrix* - a plot that shows us the degree of correlation between each variable[3], the code for which is found in my code file:

[3] For a better understanding of each variable, refer to the metadata provided by the data source

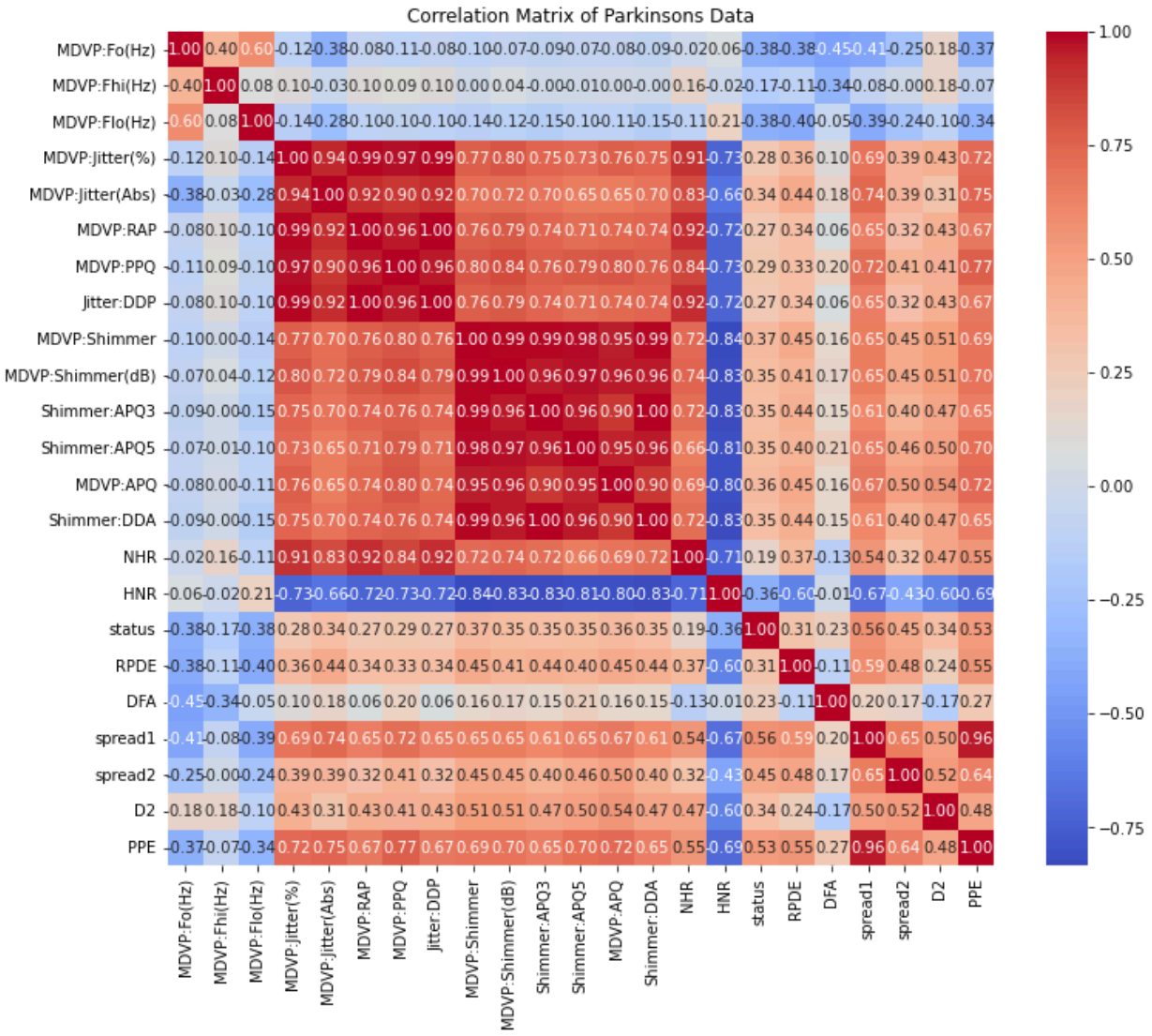


Figure 1

Oh no! The correlation matrix has revealed highly correlated variables in our dataset. To address this, we set a conservative correlation threshold of *0.95,* removing features with correlations exceeding this limit to retain as much data as possible. This approach resulted in the removal of 9 variables: 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'MDVP:APQ', 'Shimmer:DDA', and 'PPE'. Our new correlation matrix with 14 continuous variables looks a lot better:

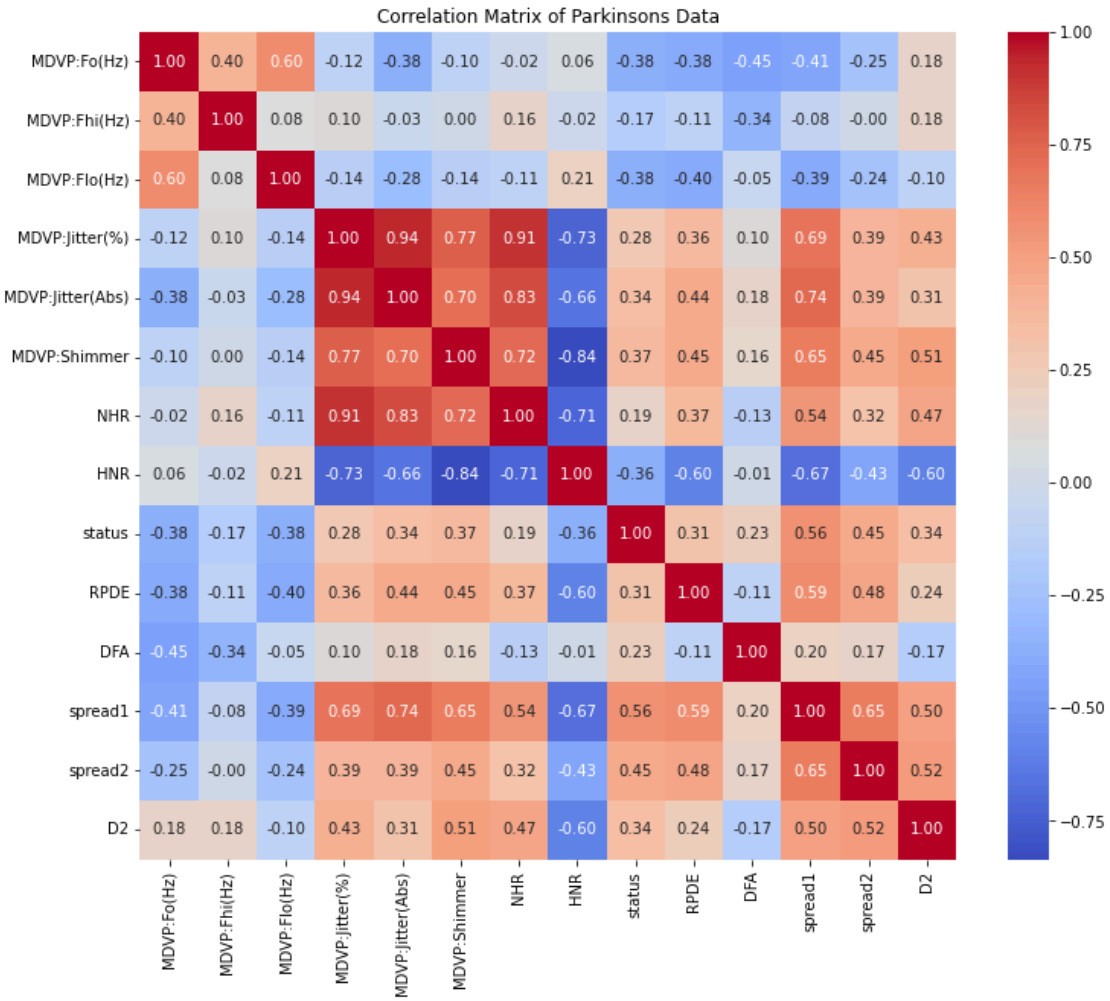


Figure 2

The updated correlation matrix shows no high-correlation pairs, meeting this assumption.

4. Linearity of Independent Variables with Log-Odds

For logistic regression, continuous predictors must have a linear relationship with the *log-odds* of the regressand. But what is a log-odd? Log-odds, a.k.a *logit*, are the natural logarithm of the odds of an event occurring. The odds of an event are calculated by the ratio of the probability of an event occurring to the probability of it not occurring. Mathematically, log-odds are expressed as:

$$ln(\frac{(P(Y=1)}{1-P(Y=1)}),$$ where $P(Y=1)$ is the probability an event occuring.

Using log-odds allows logistic regression to create a linear relationship between predictors and the outcome, simplifying estimation and interpretation. The model coefficients show how the log-odds change with a one-unit increase in a predictor, consistently scaling the odds by a fixed

amount when a predictor unit increases. To test this assumption, we use scatter plots or the *Box-Tidwell test*[4]. The code for generating scatter plots is found in my code file. Below are scatter plots visualizing the relationships for each continuous predictor:

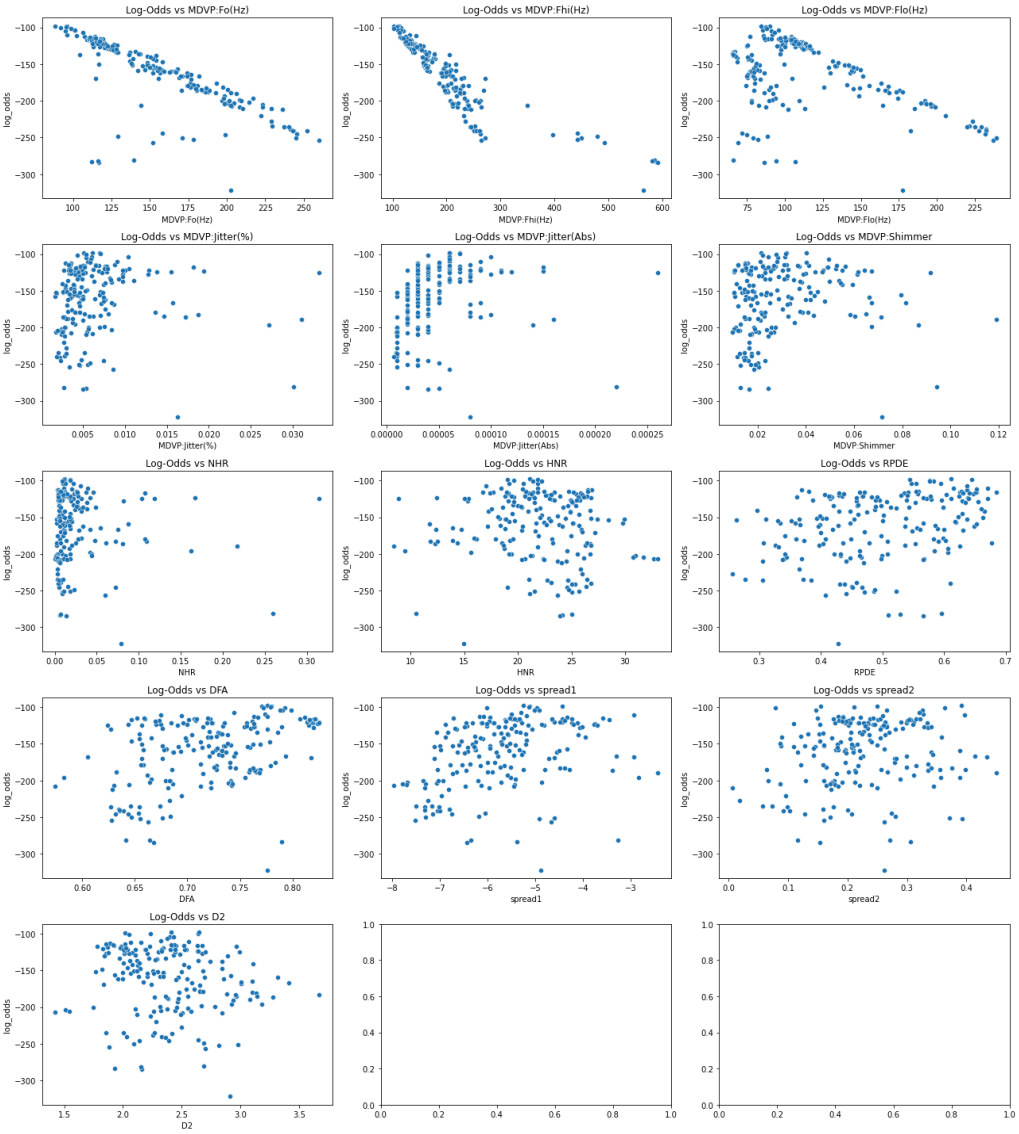


Figure 3 - last two plots are empty as intended

The results are mixed; some variables like MDVP:Fo(Hz), MDVP:Fhi(Hz), and MDVP:Flo(Hz) have strong negative linear relationships with log-odds while other predictors show scattered or non-linear patterns. Should we abandon ship? Not necessarily. Logistic regression is fairly robust to minor violations, so we'll proceed cautiously.

[4] If you're interested in using this approach, here's a resource that can guide you

5. **Sufficient Sample Size with Balanced Classes**

A sufficient sample size ensures reliable estimates. A general rule of thumb[5] here is using the Events Per Variable (EPV) Rule: "10 events per variable" rule. With 14 predictors and over 140 observations, this criterion is met in our dataset.

Additionally, class distribution should be reasonably balanced. In our case, the 'status' variable has a 25/75 split, which is manageable:

0s	1s
48	147

6. **No Influential Outliers**

Extreme outliers can distort predictions. Cook's distance[6] helps detect influential outliers, enabling you to evaluate their impact and decide a course of action. Our dataset does not have any outliers.

With the dataset meeting key assumptions, we're ready to dive into building the logistic regression model to detect Parkinson's disease. Let's get started!

Can We Build It? Yes, We Can!

With Python, initializing and training a logistic regression model is remarkably straightforward—it takes just two lines of code:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

This code has initialized and trained a logit model. But what's going on inside the 'black box'? In machine learning, when we call something a 'black box, ' we mean that the internal process is hidden from us. Though scikit-learn makes it easy to use this function, we still want to understand what's happening inside.

[5] Peduzzi, P., Concato, J., Kemper, E., Holford, T. R., & Feinstein, A. R. (1996). A simulation study of the number of events per variable in logistic regression analysis. Journal of Clinical Epidemiology, 49(12), 1373-1379. https://doi.org/10.1016/S0895-4356(96)00236-3

[6] "Cook's Distance - MATLAB & Simulink." Www.mathworks.com, www.mathworks.com/help/stats/cooks-distance.html.

Math in the Black Box

The goal of logistic regression is to predict the class of future observations given multiple predictors ($X_1, X_2, ..., X_m$) and a target variable (Y).

Key Notations and Methods

Before diving into the math, here are some fundamental terms and concepts:

- The number of predictors in this dataset is denoted as m
- The number of observations in this dataset is denoted as n
- P(Y|X) - Conditional probability, the probability of Y given X
- Joint probability - the probability of more than one event, assuming they are independent. Joint probability is calculated by multiplying individual probabilities.
- $\prod_{i=1}^{n}$ - $\prod$ denotes repeated multiplication of terms, just like $\sum$ denotes repeated addition. The term indicated the multiplication of all terms from 1 to n
- You should be familiar with operations on logarithms, especially multiplication and exponent calculations.
- You should be familiar with matrix operations for the computation aspect of logit models.

To understand logistic regression, we need to have a robust understanding of sigmoid functions first.

The Logistic Sigmoid Function

A Sigmoid function, also known as the probabilistic function, is any S-shaped function. A *logistic sigmoid* function is a special case of the sigmoid function that maps any real number to a range between 0 and 1, making it ideal for probability estimation. It is mathematically defined as:

$$S(z) = \frac{1}{1 + e^{-z}}$$

Where input, z, is a linear combination of all m number of predictors and coefficients:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_m X_m$$

$S(z)$ can be interpreted as the probability of the positive class ($Y=1$) given the input z. Visually, sigmoid logistic functions look like this:

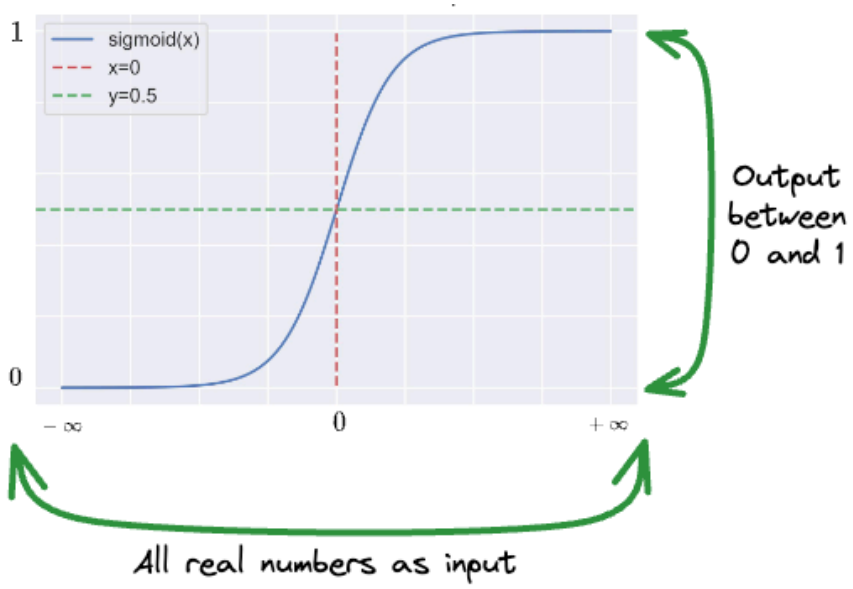


Figure 4[7]

The link between the logistic sigmoid function and probability is rooted in the log-odds transformation. Let's derive it step by step. We start with the logistic function, where we assume the probability of the positive class is p:

$$p = \frac{1}{1 + e^{-z}}$$

Let's solve for z, the :

$$\Rightarrow 1+e^{-z} = \frac{1}{p}$$

$$\Rightarrow e^{-z} = \frac{1}{p} - 1$$

$$\Rightarrow \frac{1}{e^{z}} = \frac{1-p}{p}$$

$$\Rightarrow e^{z} = \frac{p}{1-p}$$

Taking the natural logarithm on both sides:

$$\Rightarrow \ln(e^{z}) = \ln(\frac{p}{1-p})$$

[7] https://www.dailydoseofds.com/why-do-we-use-sigmoid-in-logistic-regression/

$$\Rightarrow z = \ln\left(\frac{p}{1-p}\right) \quad [1]$$

Does this formula look familiar? That's right, it's the log-odds formula! This proof shows that the sigmoid function is the *inverse* of the logit function. This fundamental relationship allows us to model probabilities while ensuring outputs remain within valid probability bounds of 0 and 1.

- -

Deep Dive

If you think the derivation of the sigmoid function from log-odds is neat, boy, do I have a surprise for you!

The sigmoid function is tied to the binomial distribution by mirroring the logistic cumulative distribution function (CDF), ensuring its outputs align with probability axioms. This connection gives logistic regression a strong statistical foundation, as the sigmoid naturally arises from the Bernoulli distribution's canonical link function[8] in generalized linear models. Unlike other bounded functions, the sigmoid's grounding in probability theory ensures consistency and efficiency in parameter estimation, making it ideal for modeling binary outcomes.

This is why, if we modeled odds directly without applying the natural log, we'd have an exponential model rather than a linear one, making estimation harder.

- -

Now that we know what logistic sigmoid functions are and why we use them, we will extrapolate this knowledge to logistic regressions.

Estimating Coefficients

Given an observation, we estimate the probability of Y=1 using the logistic function. The input is a linear combination of predictors and their corresponding coefficients:

$$S(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_m X_m) = \frac{1}{1 + e^{-(\beta 0 + \beta 1x1 + \beta 2x2 + \ldots + \beta px p)}} \quad [2]$$

Using our log-odds derivation, we express this in another form:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_m X_m = \ln\left(\frac{p(y=1)}{1-p(y=1)}\right) \quad [3]$$

This equation shows that the linear combination of predictors directly models the log-odds of Y=1. We will use this equation in the computation section to calculate *z*.

[8] https://www.cs.cmu.edu/~epxing/Class/10708-16/note/10708_scribe_lecture5.pdf

Unlike linear regression, which minimizes squared errors, logistic regression estimates coefficients by maximizing the likelihood of observing the given data, a process called *Maximum Likelihood Estimation (MLE)*.

Maximum Likelihood Estimation (MLE)

MLE finds the best coefficients β by maximizing the *likelihood function*. The likelihood function represents the probability of observing a label based on the input features X and the model parameters (β). To compute this probability, the logistic regression model first predicts individual conditional probabilities P(Y|X) using the sigmoid function. For a single observation, where *z* is the linear combination of all predictors and coefficients, the probability of Y = 1 is:

$$\mathrm{P}(y_i = 1 \mid x_i) = \frac{1}{1 + e^{-z}}$$

For Y = 0, the probability is:

$$\mathrm{P}(y_i = 0 \mid x_i) = 1 - \frac{1}{1 + e^{-z}}$$

For multiple independent observations, the likelihood function $L(\beta)$ calculates the joint probability of observing all outcomes in *Y*. Since we assume observations are independent (asm 2), the likelihood function for the entire dataset is the product of individual probabilities:

$$L(\beta) = \prod_{i=1}^{n} \mathrm{P}(Y_i \mid X_i) \qquad [4]$$

y_i can be either 0 or 1, so the expression can be written as:

$$L(\beta) = \prod_{i=1}^{n} p(x_i)^{y_i} \, (1 - p(x_i))^{1-y_i} \qquad [5]$$

This equation may look complicated, but it's just a mathematical trick to handle both possible values of y_i in a single expression. Let's break it down step by step:

If y_i =1 (positive class), then:

- $p(x)^1 = p(x)$, meaning the first term is included.
- $(1 - p(x))^{1-1} = (1 - p(x))^0 = 1$, meaning the second term does not contribute.

The final result simplifies to just $\mathrm{P}(y_i = 1 \mid x_i)$, which is the correct probability for this case.

If $y_i = 0$ (negative class), then:

- $p(x)^0 = 1$, meaning the first term does not contribute.
- $(1 - p(x))^{1-0} = (1 - p(x))$, meaning the second term is included.

The final result simplifies to just $P(y_i = 0 \mid x_i)$, which is the correct probability for this case.

Remember, $p(x)$ is short-hand for the sigmoid function applied to the linear combination of predictors: $\frac{1}{1 + e^{-(\beta 0 + \beta 1 x1 + \beta 2 x2 + \ldots + \beta p x p)}}$. To make the likelihood function computationally more convenient, we build a *logarithmic likelihood function* by taking the natural logarithm of equation 5 to get:

$$\ell(\beta) = \sum_{i=1}^{1} [y_i \, ln \, p(x_i) + 1 - y_i \, ln \, (1 - p(x_i))] \qquad [6]$$

Let me quickly explain what happened in this step:

- To multiply 2 logs of the same base, they must be added. Therefore, $\prod$ turns into $\sum$.
- Ln of $p(x_i)^{y_i}$ turns into $y_i \, ln \, p(x_i)$.
- Lastly, the ln of $L(\beta)$ turns into the $\ell(\beta)$ notation.

Gradient Ascent

Once we have simplified the Maximum Likelihood Estimation (MLE) problem, we maximize the log-likelihood function to find the best coefficients (β). Among many optimization algorithms, *gradient ascent* is one of the most popular techniques. Gradient ascent is an iterative method that adjusts the coefficients step-by-step in the direction that increases the log-likelihood.

Deep Dive

Here are the key features of gradient ascent you should know to better understand the algorithm:

a. *Gradient*: A gradient is the slope of a function at a specific point. Using partial derivatives, it measures the magnitude and the direction of change.

   Think of the gradient as the steepness of a hill—gradient ascent helps you find the fastest way to the top.

b. *Learning rate*: controls the speed of adjustments in a gradient ascent. If it's too high, the model might overshoot and miss the best values; if it's too low, learning takes too long.

Here's how gradient ascent works step-by-step:

1. Start with random guesses: The model begins with random values for its coefficients, often initialized to zeros.
2. Compute Predicted Probabilities: The sigmoid function calculates probabilities p for each observation.
3. Calculate gradients: Determine how the log-likelihood function $l(\beta)$ changes with respect to each coefficient.
4. Make adjustments: Update coefficients in the direction that increases $l(\beta)$ using the learning rate.
5. Repeat until Convergence (until it's good enough): Repeat steps 2–4 until changes in β become very small or a maximum number of iterations is reached.

- -

Here's an illustration of how gradient ascent works, where each red dot represents an adjusted iteration as per gradient ascent

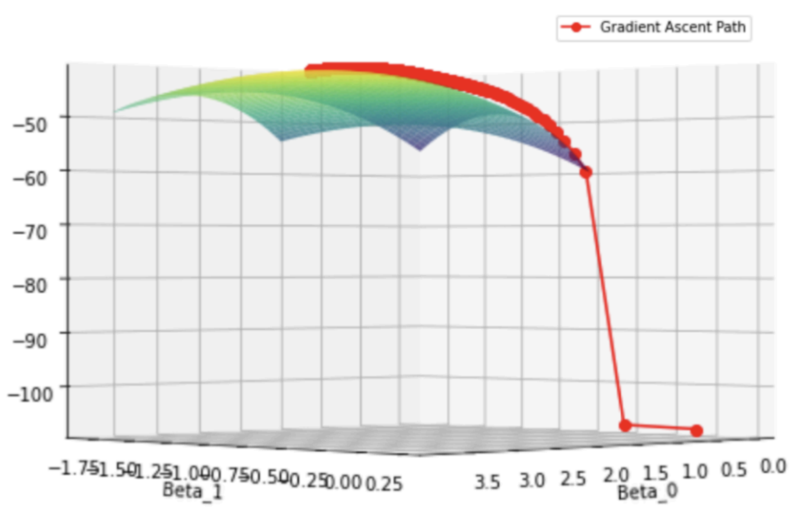


Figure 5

Iteratively adjusting coefficients using gradient ascent allows us to maximize the log-likelihood function and find optimal values for β. This results in a model that best fits the data.

So far, we've explored the mathematical foundation of logistic regression. Now, let's put theory into practice. We'll build a logistic regression model in Python to classify patients as having Parkinson's or not.

Before diving into the code, note that the dataset has been feature-engineered. Highly correlated variables are removed (as discussed in Part 1), and non-relevant columns are omitted. We'll split the dataset: 80% for training and 20% for testing. For details, check out the code file.

Computational Mathematics

You'll need to know some basic linear algebra for this next part. To express the regression equation in matrix form, we define three key matrices:

- Design Matrix (X) - also known as a regressor matrix, contains all m predictor variables in the Parkinson's dataset and an additional column for the intercept term. Dimensions:

$$X = n \times (m+1)$$

- Target vector (y) - contains all observed Y values. Dimensions:

$$y = n \times 1$$

- Coefficient vector (β) - an array of regression coefficients (including β_0). Dimensions:

$$\beta = (m+1) \times 1$$

Now, we plug these into the logistic sigmoid function to calculate the predicted probability of Y = 1, or 'Parkinson's detected'. Recall that z is the linear combination of all predictors and their coefficients:

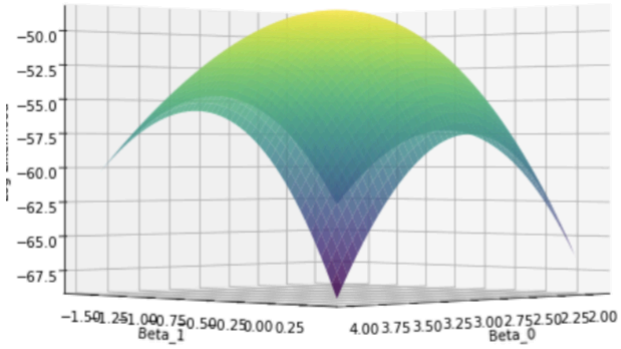
$$S(z) = \frac{1}{1 + e^{-\beta X}}$$

Where z is the dot product of the coefficient vector and the design matrix, here's how it looks in Python:

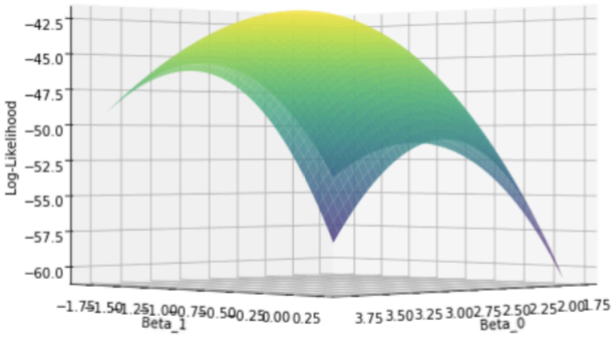
```
z = np.dot(X, beta)
sigmoid_p = 1 / (1 + np.exp(-z))
```

```

Next, we sum the natural log of these probabilities to calculate the the logarithmic likelihood:

$$\ell(\beta) = \sum_{i=1}^{1} [y_i \ln p(x_i) + 1 - y_i \ln (1 - p(x_i))]$$

Since taking the log of 0 is undefined, we introduce a small, negligible *epsilon* value to prevent errors. This ensures that in cases where the probability of the positive class is 0, we don't end up with log(0):

```
```
epsilon = 1e-10  #[.000000001]
log_likelihood = np.sum(y * np.log(sigmoid_p + epsilon) + (1 - y) * np.log(1 - sigmoid_p + epsilon))
```

To visualize how the log-likelihood function behaves, let's plot its surface. Since we're dealing with multiple coefficients, I've selected the intercept and the first feature to create a 3D surface plot. This surface represents how the log-likelihood changes with different $\beta 1$ values:



Log-Likelihood Surface (Entire Dataset)



Log-Likelihood Surface (Training Dataset)

Figures 6 & 7

With the computed log-likelihood in hand, we now optimize β to find the maximum likelihood estimation. Since no closed-form solution exists, we iteratively adjust β using the gradient of the log-likelihood function to maximize $\ell(\beta)$. To compute the gradient, we calculate the derivative of the log-likelihood function with respect to each coefficient in the β vector. I won't get into the derivation steps here, but I've linked some resources in the footnote[9] if you're curious. The result of this calculus step is:

$$\nabla \ell(\beta) = X^{\top}(y - p)$$

[9] https://web.stanford.edu/~jurafsky/slp3/5.pdf (Pages 16, 22-23), https://www.youtube.com/watch?v=FuGXKzJpL_Y

This gradient, $\nabla \ell(\beta)$, indicates how sensitive the function is to changes in β. It tells us which way is up on the log-likelihood function and how to adjust β to increase the log-likelihood:

```
error = y - p
gradient = np.dot(X.T, error)
```

To maximize the likelihood function, gradient ascent is used to iteratively adjust the coefficients in the direction that increases the log-likelihood, moving along the gradient. The gradient ascent rule is:

$$\beta_{\text{new}} = \beta_{\text{old}} + \alpha \cdot \nabla \ell(\beta)$$

The optimized coefficient, β_{new}, is calculated by adding a minor adjustment - the product of the gradient and the learning rate, α- to the previous coefficient, β_{old}. This iterative process pushes β in the direction that maximizes the log-likelihood. The process stops when the log-likelihood change between iterations falls below a *tolerance* value you set - indicating convergence - or when the maximum number of iterations is reached. In Python, this implementation is:

```
log_likelihoods = []
for i in range(max_iter):
     beta += learning_rate * gradient
     ll = log_likelihood(X, y, beta, lambda_reg)
     log_likelihoods.append(ll)
     if i > 0 and abs(log_likelihoods[-1] - log_likelihoods[-2]) < tolerance:
     opt_beta = log_likelihoods[i]
     return opt_beta
```

Rather than looping through each coefficient manually, a more efficient method leverages NumPy's matrix operations to update all coefficients simultaneously. And now, we finally have our optimized coefficients!

Below is a visual aid for understanding gradient ascent. Each red dot represents an updated iteration dictated by gradient ascent. As the ascent path moves closer to the maximum point, beta adjustment reduces until convergence.

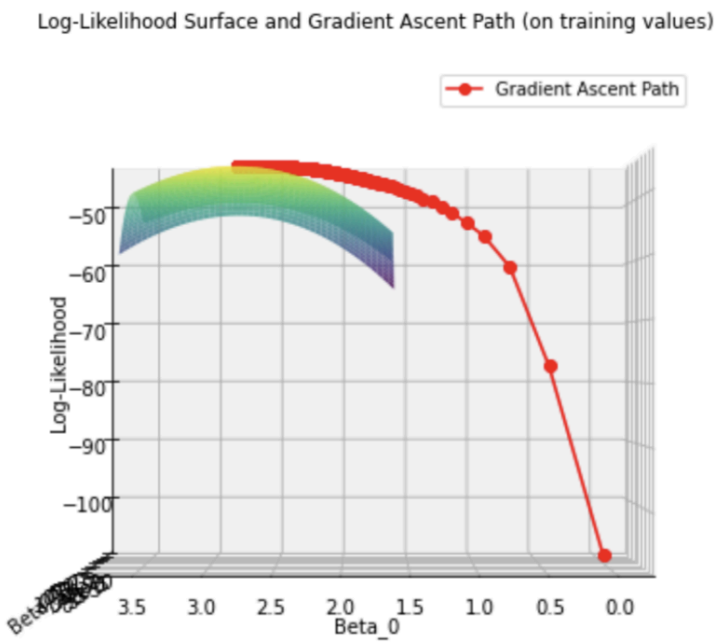


Figure 8

By understanding the math behind logistic regressions, you gain deeper insight into how the model estimates coefficients. While Python makes it easy to implement regression, knowing what happens inside the "black box" allows you to fine-tune models for better accuracy and adapt them to custom needs.

Predictions

We've optimized β; we can now classify new patients!

To classify new observations, we once again defer to the sigmoid formula to calculate the probability of 'Parkinson's detected'. However, this time, we plug in the optimized coefficient values derived from MLE, as opposed to random values. By default, we use a decision threshold of 0.5:

- p >= 0.5: class prediction is 1,
- p < 0.5: the class prediction is 0.

In Python, this process looks like:

```
```

```
z = np.dot(X, optimized_beta)
probabilities = sigmoid(z)
predictions = (probabilities >= 0.5).astype(int)
```

```
Different thresholds can be adjusted according to circumstance (like handling imbalanced datasets).

## Comparing Manual and Scikit-Learn Models

If you've been following along with the code, you'll notice we built two logistic regression models—one manually and one using scikit-learn. But how do they compare? Let's take a look at the first five $\beta$ values:

| $\beta$ | Scikit Learn Model | Manually Built Model | Difference |
|---|---|---|---|
| $\beta_0$ | 2.204015 | 2.649518 | 0.445504 |
| $\beta_1$ | -0.007363 | -0.230054 | 0.222691 |
| $\beta_2$ | -0.170670 | -0.057623 | 0.113048 |
| $\beta_3$ | -0.152301 | -0.049406 | 0.102895 |
| $\beta_4$ | -0.165924 | 0.045464 | 0.211388 |

The optimized coefficients differ, possibly due to differences in regularization or the optimization algorithms used. Our manual implementation follows gradient ascent, while scikit-learn may apply L2 regularization (by default) or use a different solver, such as L-BFGS.
Even though the coefficient estimates are not identical, what truly matters is how the models perform in classification. Let's compare their predictions:

```
```
print("Match %:", accuracy_score(y_pred, test_classes)*100)
```
```

Match %: 100.0

Despite different coefficient values, both models classify every observation identically. This demonstrates an important principle of logistic regression: different optimization methods and regularization techniques may produce slightly different coefficients, yet still lead to the same classification results
```

Evaluation

Now that our model predicts classes, how do we know it performs well? We use key evaluation metrics to measure its success.

Evaluating a logistic regression is far simpler than evaluating continuous-variable models. Since predictions are binary, we focus on right vs. wrong classifications. We are also interested in the breakdown of each accurate and inaccurate prediction. To do this, we build a *confusion matrix*. A confusion matrix is a table that compares actual outcomes with predicted outcomes. Take a look:

		Predicted Class	
		Negative (0)	Positive (1)
Actual Class	Negative (0)	True Negatives **(TN)**	False Positives **(FP)**
	Positive (1)	False Negatives **(FP)**	True Positives **(TP)**

- True Negative (TN): Predicted negative and actually negative.
- True Positive (TP): Predicted positive and actually positive.
- False Positive (FP): Predicted positive but actually negative.
- False Negative (FN): Predicted negative but actually positive.

This tool is critical for calculating additional metrics like *accuracy, precision*, *recall*, and *F1-score* to assess overall performance. Let's take a look at them:

1. **Accuracy**: measures the proportion of correct predictions out of all predictions made. It gives a sense of how well the model performs. Here is the accuracy formula:

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

2. **Precision**: indicates how many of the predictions labeled as 'positive' are correct. It is useful when tweaking models to minimize false positives.

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

3. **Recall**: measures how many actual positive cases were correctly identified by the model. It is essential when minimizing false negatives is critical.

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

4. **F1-Score**: is the mean of precision and recall, balancing their trade-offs. It is useful when a single metric is required to evaluate both false positives and negatives.

$$\text{F1-Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5. **Support**: is not an evaluation metric, but rather additional information Python's classification_report() function outputs. It refers to the number of actual occurrences of each class in the dataset.

The Python implementation of these evaluation metrics is relatively simple:

```
```
confusion_matrix(actual_y_values, predicted_y_values)
accuracy_score(actual_y_values, predicted_y_values)
classification_report(actual_y_values, predicted_y_values)
```
```

Interpreting the Model

Here is the confusion matrix of our model:

$$\begin{bmatrix} 8 & 2 \\ 1 & 28 \end{bmatrix}$$

It tells us that of the 39 new patients, the model:

- predicted that 8 patients didn't have Parkinson's, and they didn't (TN)
- predicted that 28 patients had Parkinson's, and they did (TP)
- predicted that 2 patients had Parkinson's when they didn't (FP)
- predicted that 1 patient did not have Parkinson's, when they did (FN)

Correctly diagnosing 36 out of 39 patients is a strong result. On the other hand, falsely diagnosing 2 patients with Parkinson's disease is not a pleasant experience, but it ends well. However, a false negative (failing to detect Parkinson's when it's present) is a crucial mistake that hospitals cannot afford. This is why models should assist, not replace, medical professionals.

The confusion matrix has given us an understanding of the model's behavior. Let's use it to calculate other metrics:

1. Accuracy: 92.31%

With a score above 90%, this is considered excellent for most applications, indicating that the model performs well overall. However, accuracy alone may not be a sufficient metric for model evaluation.

2. Precision: 93%

Precision reflects how many positive predictions are correct. A score of 93% is excellen, especially in scenarios where minimizing false positives is necessary, such as Parkinson's disease detection.

3. Recall: 96.55%

Recall measures how well the model identifies actual positives. A recall above 95% indicates that the model captures most of the true positives. However, this rate is too low for fatal medical diagnostics. This is why even with the best models, doctors should not and do not rely solely on the model.

4. F-1: 95%

An F1-score of 95% indicates that our model achieves a strong balance between precision and recall, making it highly reliable for balanced performance.

With an impressive set of evaluation scores, our model performs strongly in classifying Parkinson's patients. These metrics prove that logistic regression is a great choice for Parkinson's disease detection and that our model works well. Kudos!

That said, every model has room for improvement. One question that lingers is why our model performed so well despite breaking the fourth assumption (linearity of independent variables with log-odds) of logistic regression. One possible explanation is that the dataset inherently exhibits strong class separability, making it easier for logistic regression to distinguish between positive and negative cases—even if the relationship between predictors and log-odds isn't perfectly linear. Additionally, features like MDVP:Fo(Hz), MDVP:Fhi(Hz), and MDVP:Flo(Hz), which show a linear or near-linear relationship with log-odds, may be contributing significantly to the model's predictive power.

Model Comparison: Linear vs. Logistic

In my *Inside the Black Box* series, I've covered linear regression and just wrapped up a nonlinear one - logistic regression. It's helpful to compare the two fundamental models: Understanding these differences ensures that you choose the right tool for the job.

Feature	Linear	Logistic
Problem type	Prediction	Classification
Outcome variable	Binary	Categorical
Equation/ Relationship	Linear equation/ Linear relationship	Logistic Sigmoid Function/ Non-linear relationship
Method to estimate coefficients	Ordinary Least Squares (OLS)	Maximum Likelihood Estimation (MLE)
Optimizing coefficients	Closed-form solution of OLS	Gradient Ascent
Output type	Predicted value of the regressand	Probability of a class/ event
Sensitivity to Outliers	High	Low
Evaluation	MAE, MSE, RMSE, R^2 score	Accuracy, precision, recall, F1

Despite their differences, linear and logistic regression share common foundations. Let's compare their assumptions to understand their similarities and distinctions better.

Assumptions	Linear	Logistic
Linearity	✅	
Independence of observations	✅	✅
Absence of Multicollinearity	✅	✅
Homoscedasticity	✅	
Normality of error distribution	✅	
Sufficient Sample Size with Balanced Classes		✅
Linearity of Independent Variables with Log-Odds		✅

Conclusion

Logistic regression is a powerful and foundational tool for both statisticians and machine learning practitioners. It bridges the gap between simple predictive models and more complex

machine learning techniques, offering interpretability and strong performance in classification tasks. Through careful feature selection and model optimization, we achieved highly accurate predictions for Parkinson's disease detection. Beyond just implementing a model, understanding its inner workings— from the assumptions to the optimization process—allows for better decision-making, fine-tuning, and adaptation to real-world challenges. Mastering logistic regression sets the stage for tackling more advanced machine learning algorithms and classification problems with confidence.

In this post, we've learned:

- What logistic regression is
- Assumptions of logistic regressions and how to assess your dataset for them
- MLE as a method to estimate coefficients in logistic regressions
- Use gradient ascent to optimize coefficients
- How to build a logistic regression using the scikit-learn library in Python
- How to build a logistic regression manually in Python
- How to build a gradient ascent function in Python manually
- How to predict values using scikit and manually in Python
- How to evaluate the model using a confusion matrix, accuracy, precision, recall, and F1 metrics with scikit
- How to interpret model evaluations
- As a bonus: How to create the 3D plots of likelihood surfaces for data visualization

That's a wrap on logistic regression! Next, we'll dive deeper into more advanced machine learning algorithms - stay tuned!