Logistic regression

- Logistic **regression** is a linear **classification** technique that builds upon linear regression.
- It models **the probability of belonging to class 1, given a feature vector**:

$$P(y_i = 1|\vec{x}_i) = \sigma(w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}) = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$$

designed for binary classification

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$0 < \sigma(x) < 1$

$y_i \in \{0, 1\}$

- By default, the `predict` method of a fit `LogisticRegression` model predicts $T = 0.5$ to the predicted probability.

In [3]:
```
model_logistic_multiple.predict(pd.DataFrame([{
    'Glucose': 150,
    'BMI': 25,
}]))
```

Out[3]:
```
array([0])
```

- We can access the predicted **probabilities** using the `predict_proba` method.

In [4]:
```
model_logistic_multiple.predict_proba(pd.DataFrame([{
    'Glucose': 150,
    'BMI': 25,
}]))
```

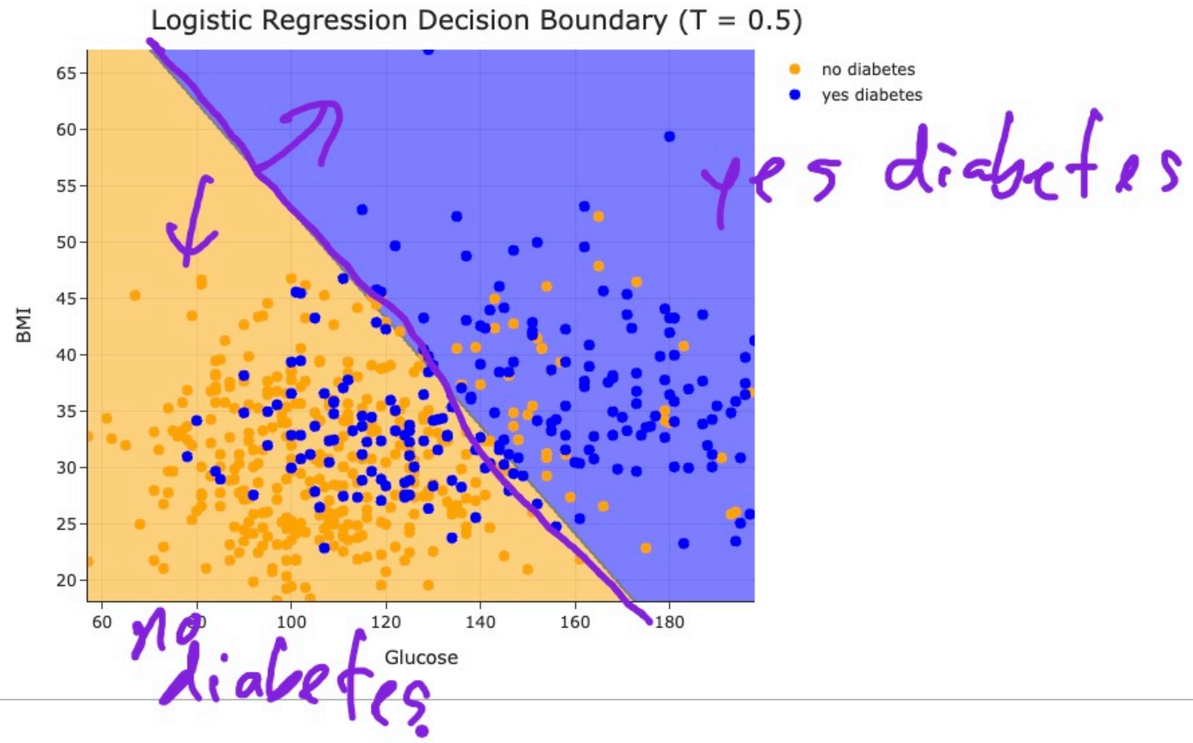
$$P(y_i = 1 \mid \vec{x}_i) = \sigma(w_0 * + w_1 * \cdot 150 + w_2 * \cdot 25)$$

Out[4]:
```
array([[0.58, 0.42]])
```

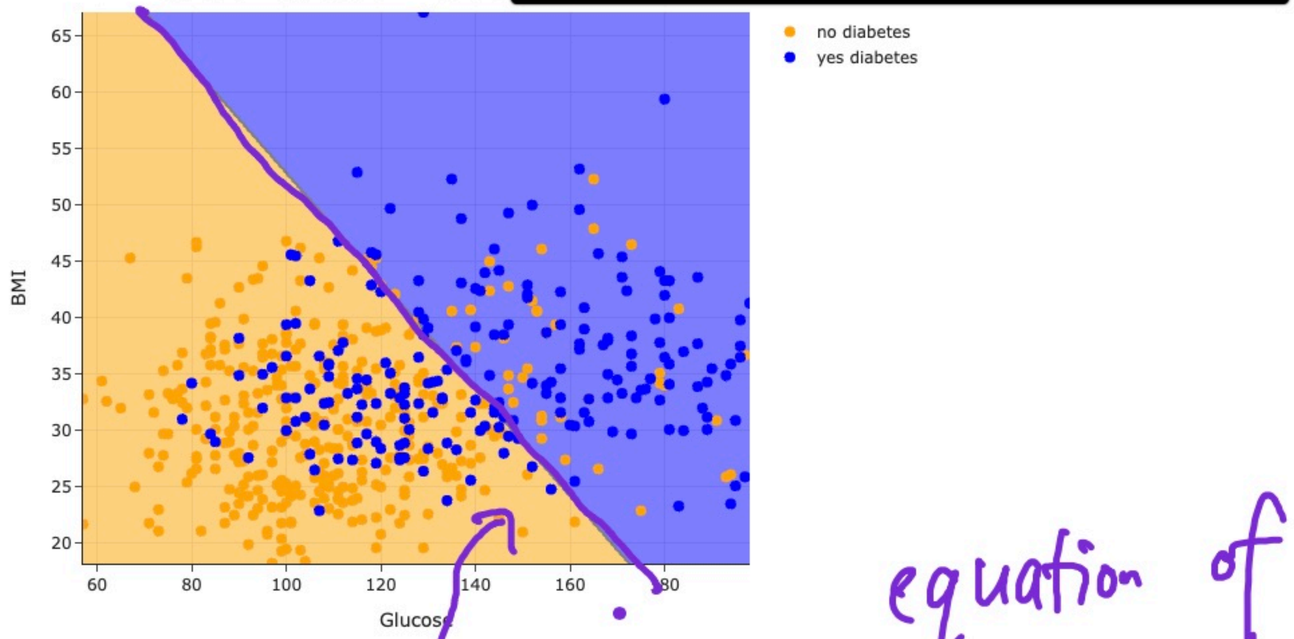
The decision boundary in the feature space

- After choosing $T = 0.5$, what does the resulting **decision boundary** look like, in a $d = 2$ dimensional plot?

```
In [5]: 1 util.show_decision_boundary(model_logistic_multiple, X_train, y_train, title='Logistic Regression Decision Boundary (T = 0.5
```



- Note that unlike the decision boundaries for k-Nearest Neighbors and decision trees, this decision boundary is **linear**. Specifically, it is the line:



equation of this line

- Note that unlike the decision boundaries for ~~k-Nearest~~ Neighbors and decision trees, this decision boundary is **linear**. Specifically, it is the line:

$$\sigma(-7.85 + 0.04 \cdot \text{Glucose}_i + 0.08 \cdot \text{BMI}_i) = 0.5$$

- **Important**: Since $\sigma(0) = 0.5$, we can write the above as:

$$-7.85 + 0.04 \cdot \text{Glucose}_i + 0.08 \cdot \text{BMI}_i = 0$$

Question 🤔 (Answer at practicaldsc.org/q)

Which expression describes the **odds ratio**, $\frac{P(y_i = 1 | \vec{x}_i)}{P(y_i = 0 | \vec{x}_i)}$, in the logistic regression model?

- A. $\vec{w} \cdot \text{Aug}(\vec{x}_i)$
- B. $-\vec{w} \cdot \text{Aug}(\vec{x}_i)$
- C. $e^{\vec{w} \cdot \text{Aug}(\vec{x}_i)}$
- D. $\sigma(\vec{w} \cdot \text{Aug}(\vec{x}_i))$
- E. None of the above.

$\text{odds}(p) = \frac{p}{1-p}$

$\frac{P(y_i=1|\vec{x}_i)}{P(y_i=0|\vec{x}_i)} = \frac{e^{d_i}}{e^{d_i}+1} \cdot \frac{e^{d_i}+1}{1} = e^{d_i} = e^{\vec{w} \cdot \text{Aug}(\vec{x}_i)}$

$P(y_i=1|\vec{x}_i) = \sigma(\vec{w} \cdot \text{Aug}(\vec{x}_i)) = \sigma(d_i) = \frac{1}{1+e^{-d_i}} \cdot \frac{e^{d_i}}{e^{d_i}} = \frac{e^{d_i}}{e^{d_i}+1}$

$\rightarrow P(y_i=0|\vec{x}_i) = 1 - \frac{e^{d_i}}{e^{d_i}+1} = \frac{e^{d_i}+1}{e^{d_i}+1} - \frac{e^{d_i}}{e^{d_i}+1} = \frac{1}{e^{d_i}+1}$

Question 🤔 (Answer at practicaldsc.org/q)

Which expression describes $P(y_i = 0 | \vec{x}_i)$ in the logistic regression model?

- A. $\sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$
- B. $-\sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$
- C. $\sigma\left(-\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$
- D. $1 - \log\left(1 + e^{\vec{w} \cdot \text{Aug}(\vec{x}_i)}\right)$
- E. $1 + \log\left(1 + e^{-\vec{w} \cdot \text{Aug}(\vec{x}_i)}\right)$

$P(y_i = 1 | \vec{x}_i) = \dfrac{1}{1 + e^{-d_i}} = \sigma(d_i)$

$P(y_i = 0 | \vec{x}_i)$ (from last slide) $= 1 - P(y_i = 1 | x_i)$

$= 1 - \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$

$= 1 - \dfrac{1}{1 + e^{-d_i}}$

$= \dfrac{1 + e^{-d_i}}{1 + e^{-d_i}} - \dfrac{1}{1 + e^{-d_i}} = \dfrac{e^{-d_i}}{1 + e^{-d_i}} \cdot \dfrac{e^{d_i}}{e^{d_i}}$

$= \dfrac{1}{e^{d_i} + 1} = \sigma(-d_i)$

```
    For each P(y_i = 1 | x_i), return 1 if >= T and 0 if < T.'''
    probs = model_logistic_multiple.predict_proba(X)[:, 1]
    return (probs >= T).astype(int)
```

- Now, we can choose any threshold we'd like, and compute the a

In [7]:
```
predict_thresholded([[150, 25]], 0.5)
```

Out[7]: array([0])

$p(y_i = 1 | \vec{x}_i) = 0.42$ from a few slides ago.

In [8]:
```
predict_thresholded([[150, 25]], 0.4)
```

Out[8]: array([1])

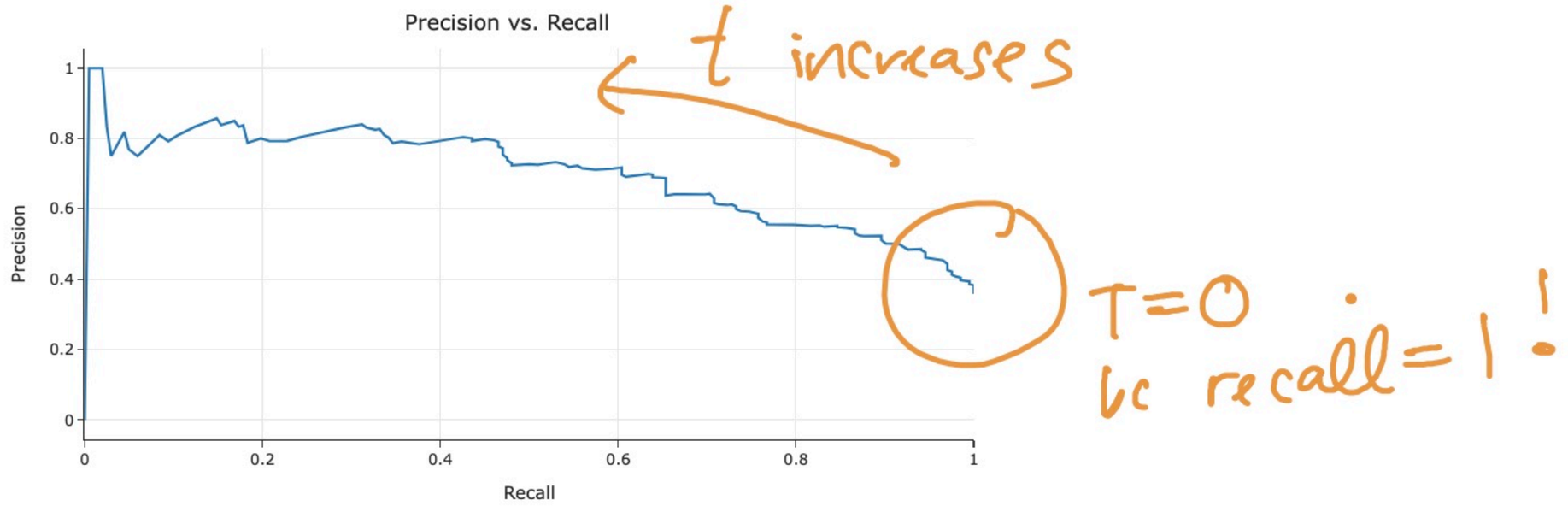
In [ ]:
```
predict_thresholded(X_train, 0.4)
```

In [ ]:
```
# Training accuracy for the threshold T = 0.4
```

Precision vs. recall

- We can visualize how precision and recall vary **together**.

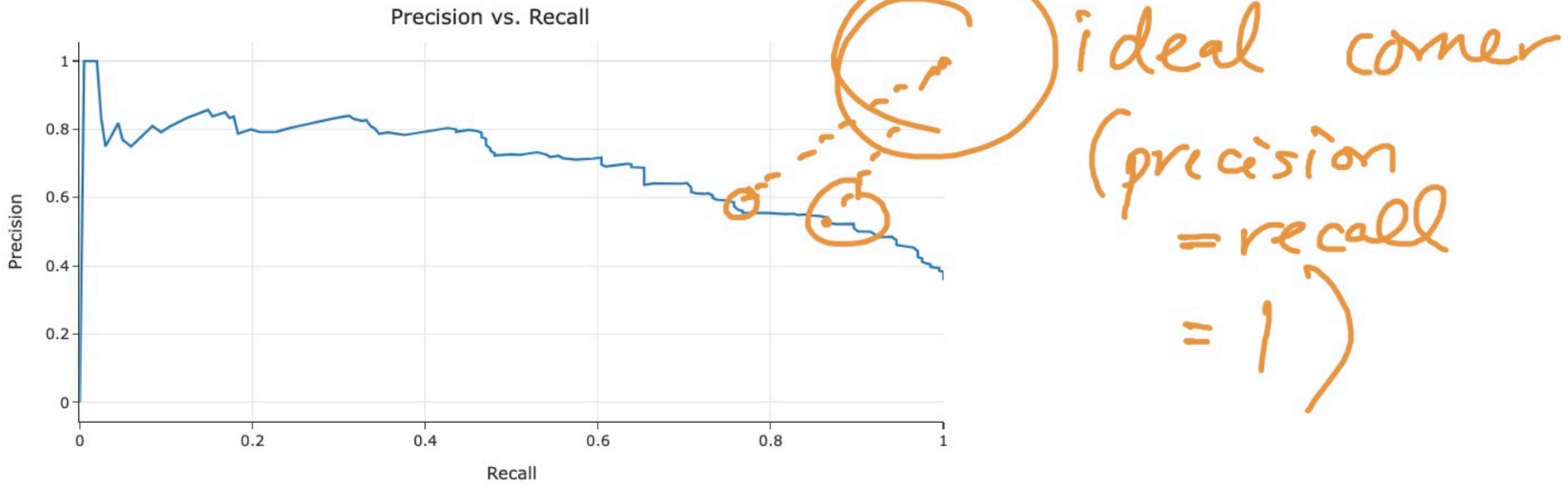
In [19]:
```
1 util.pr_curve(X_train, y_train)
```



Precision vs. recall

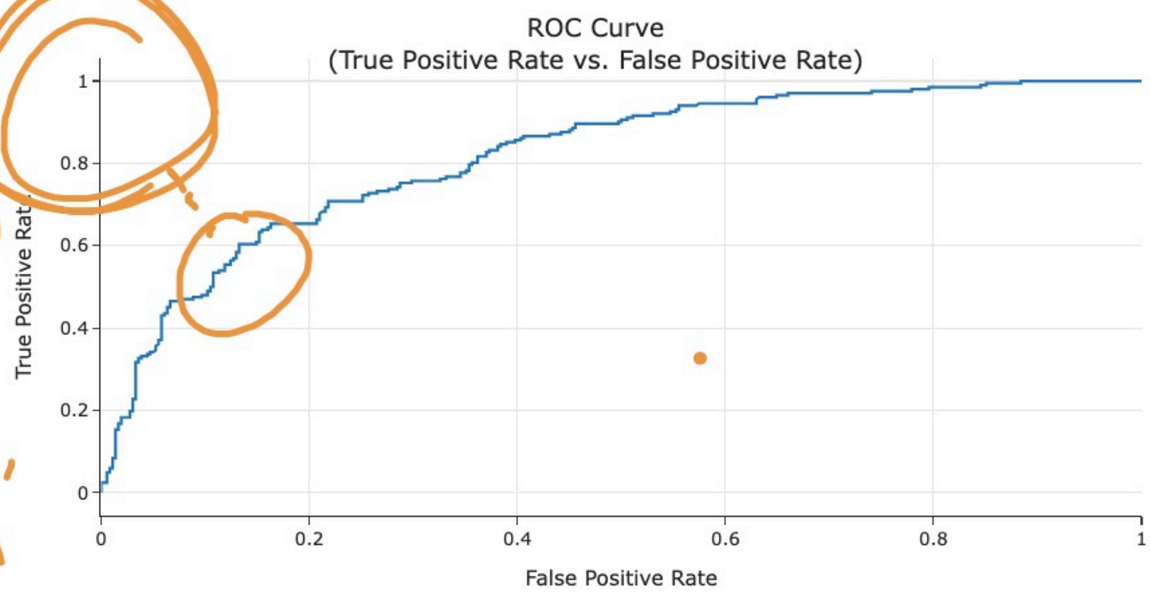
- We can visualize how precision and recall vary **together**.

In [19]:
```
1 util.pr_curve(X_train, y_train)
```



The ROC curve for our classifier looks like:

In [20]:
```
1 util.draw_roc_curve(X_train, y_train)
```



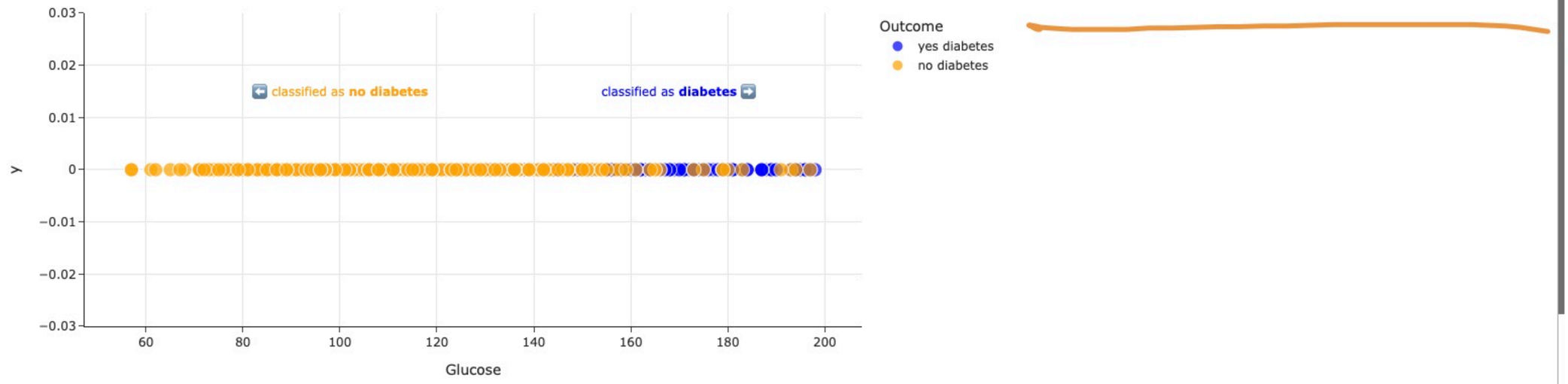
Feature space

- Suppose we're using d features as inputs to our classifier. Consider a visualization of the features in d-dimensional space.

- Example: $d = 1$.

crucially, don't draw y as a separate axis

In [21]:
```
1 util.show_one_feature_plot_in_1D(X_train, y_train, thres=False)
```

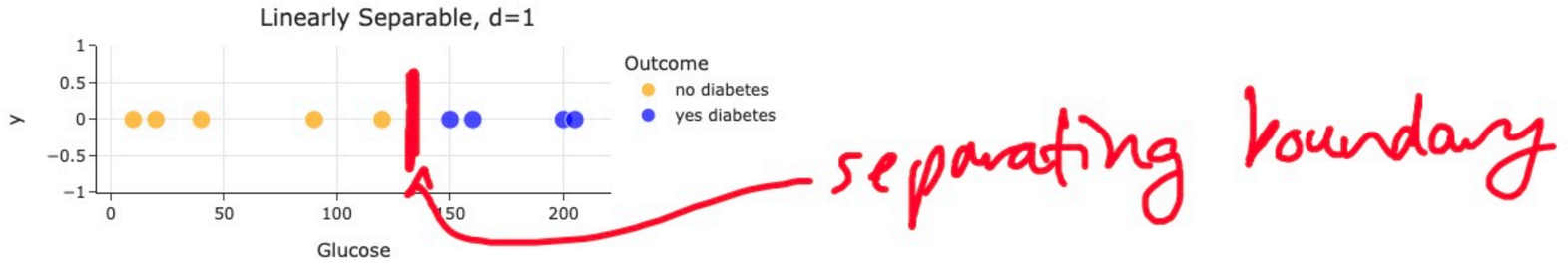


Linear separability

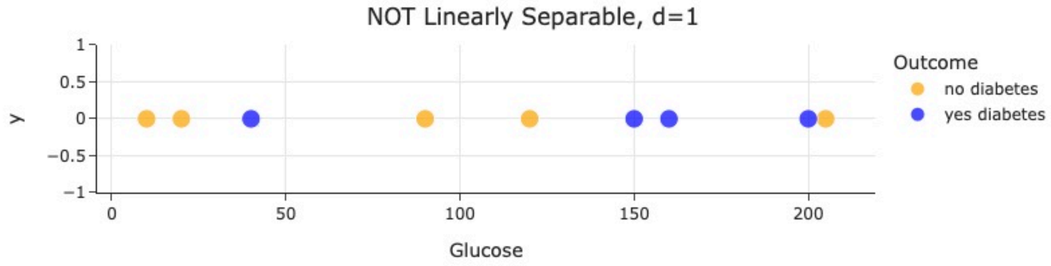
- A dataset is **linearly separable** if a line, plane, or hyperplane can be drawn in d-dimensional space that **perfectly separates** the two classes.

- Example: $d = 1$.

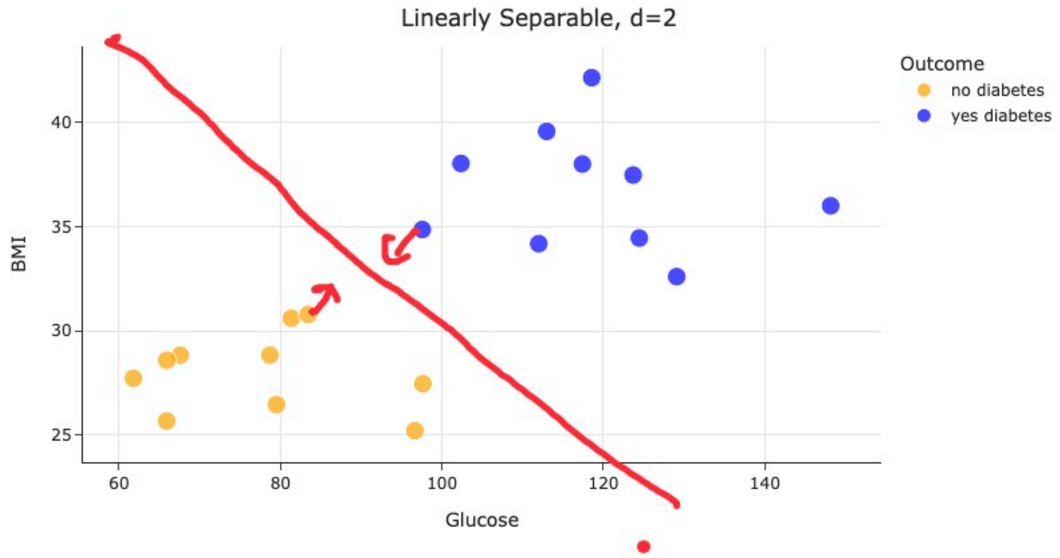
In [23]:
```
1 util.lin_sep_1D()
```



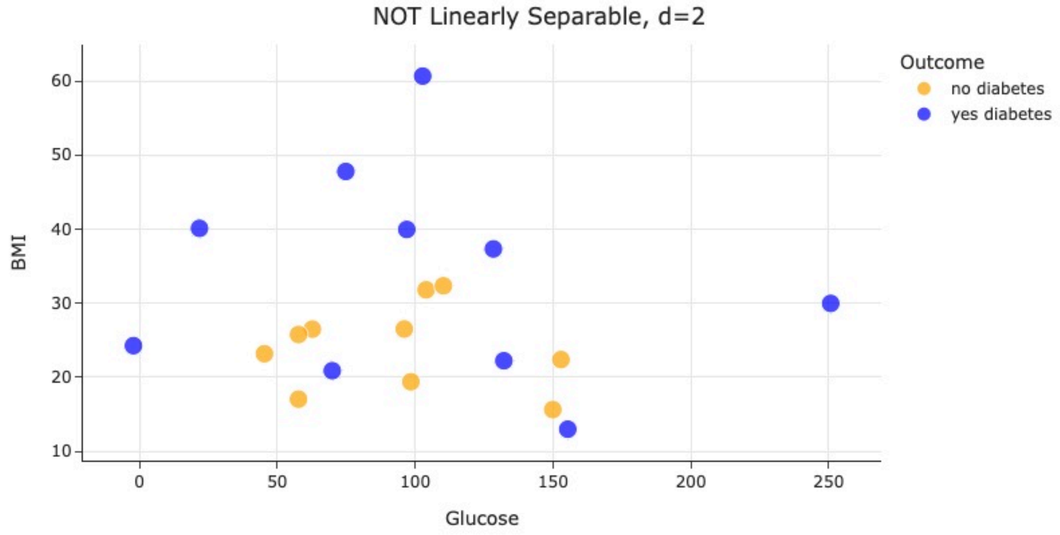
In [24]:
```
1 util.non_lin_sep_1D()
```



In [25]:
```
1 util.lin_sep_2D()
```

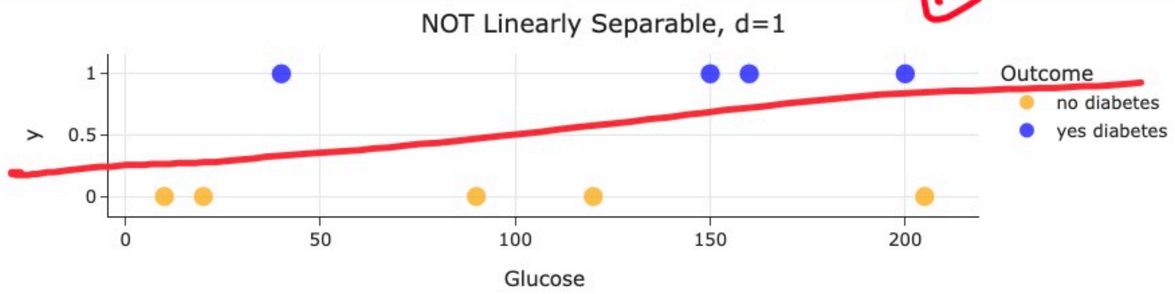


In [26]:
```
1 util.non_lin_sep_2D()
```



- Why is the dataset below **not** linearly separable?

In [27]:
```
1 util.bad_example_1D()
```



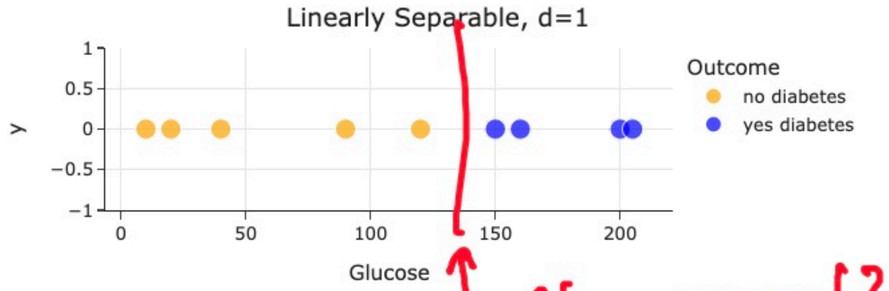
not a plot in the feature space!

not a 1D plot!
must be
d=1 dimensional
to test for
linear sep.

Linear separability and decision boundaries

- By definition, if a dataset is linearly separable, then there exists a **linear decision boundary** that achieves 100% training accuracy.

In [28]:
```
1 util.lin_sep_1D()
```



Glucose = 130

the line $x = 10$

- Above, any value of c in $(120, 150)$ would make the **decision boundary**

$$\text{Glucose} = c$$

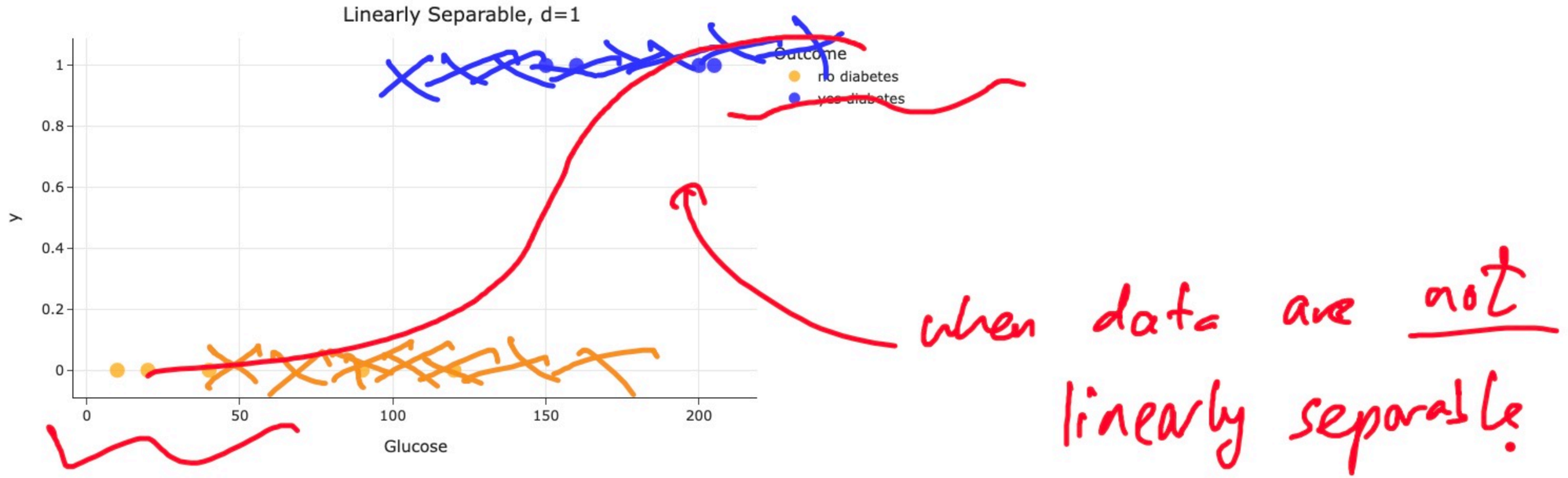
achieve 100% training accuracy.

- Why would the optimal w_1^* below tend to ∞?

  See the annotated slides for more details.

$$P(y_i = 1|\text{Glucose}_i) = \sigma(w_0 + w_1 \cdot \text{Glucose}_i) = \frac{1}{1 + e^{-(w_0 + w_1 \cdot \text{Glucose}_i)}}$$

In [30]:
```
1 util.lin_sep_1D_elevated()
```



- Why would the optimal w_1^* below tend to ∞?

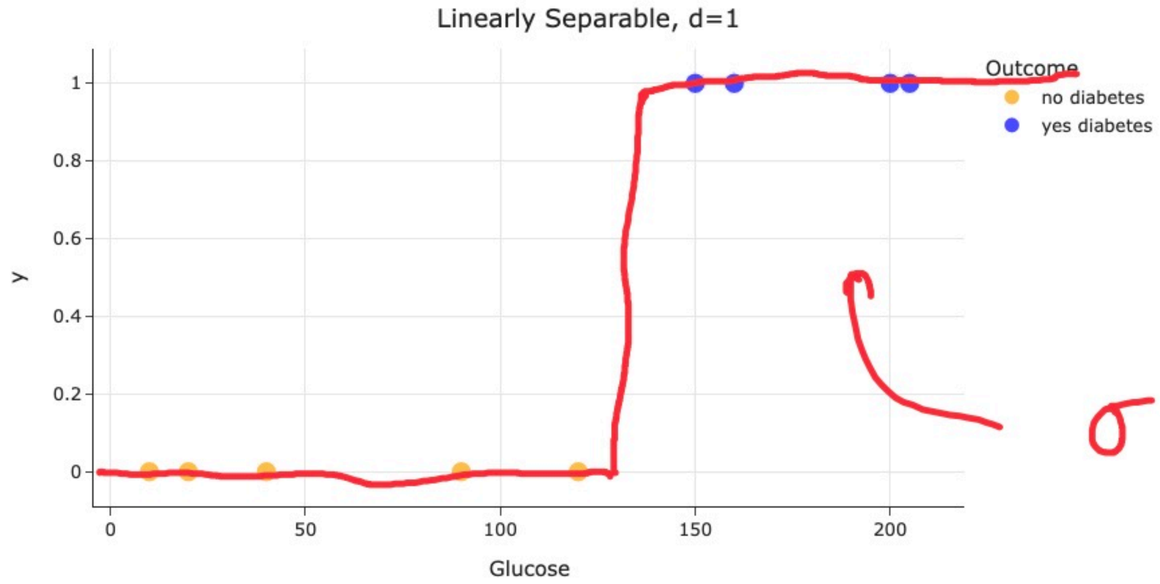
  See the annotated slides for more details.

$$P(y_i = 1 | \text{Glucose}_i) = \sigma(w_0 + w_1 \cdot \text{Glucose}_i) = \frac{1}{1 + e^{-(w_0 + w_1 \cdot \text{Glucose}_i)}}$$

steepness of

$w_1 \to \infty$

In [30]:
```
1 util.lin_sep_1D_elevated()
```



$\sigma(\cdot) \to$ a step function

- Why would the optimal w_1^* below tend to ∞?

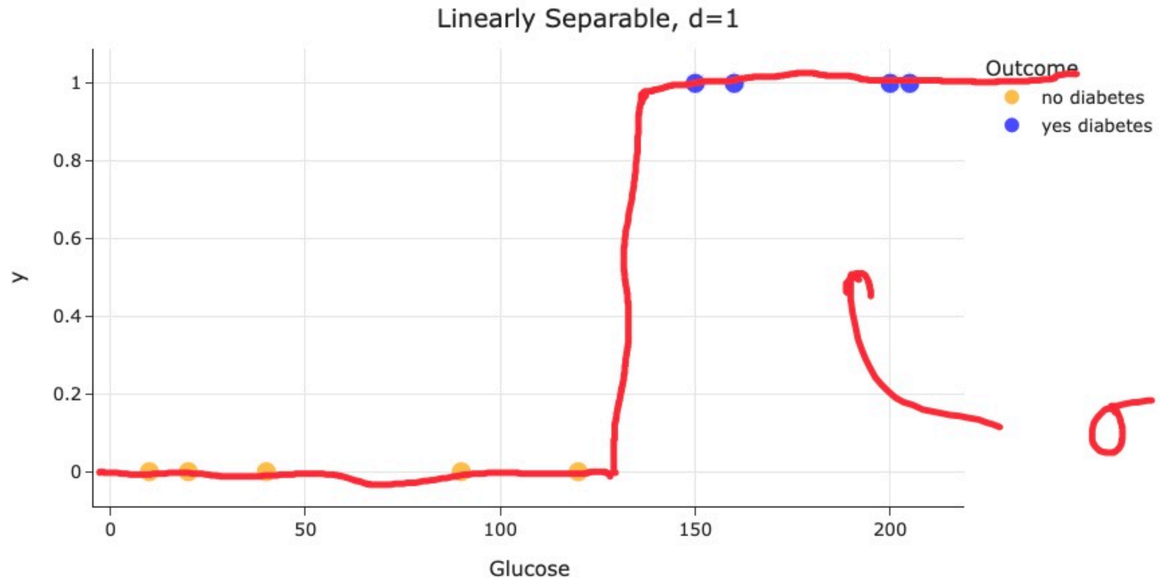
  See the annotated slides for more details.

$$P(y_i = 1 | \text{Glucose}_i) = \sigma(w_0 + w_1 \cdot \text{Glucose}_i) = \frac{1}{1 + e^{-(w_0 + w_1 \cdot \text{Glucose}_i)}}$$

steepness of

$w_1 \to \infty$

In [30]:
```
1 util.lin_sep_1D_elevated()
```



$\sigma(\cdot) \to$ a step function

From binary to multiclass classification

- In binary classification, there are only two possible classes, typically either 0 or 1.

$$y_i \in \{0, 1\}$$

- In multiclass classification, there can be any finite number of classes, or **labels**. They need not be numbers, either.

$$y_i \in \{\text{Adelie}, \text{Chinstrap}, \text{Gentoo}\}$$

$C = \{$ Adelie, Chinstrap, Gentoo $\}$

- **Important**: Let C be the set of possible classes for our classification problem, and let $|C|$ be the number of classes total.

$|C| = 3$

Multinomial logistic regression

- **Multinomial** logistic regression, also known as **softmax regression**, models the probability of belonging to **any class, given a feature vector** $\vec{x}_i$.

  Think of it as a generalization of logistic regression.

$$p_{\text{Adelie}} = P(y_i = \text{Adelie} | \vec{x}_i) = \frac{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)}}{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Gentoo}} \cdot \text{Aug}(\vec{x}_i)}}$$

Multinomial logistic regression

- **Multinomial** logistic regression, also known as **softmax regression**, models the probability of belonging to **any class, given a feature vector** $\vec{x}_i$.
  Think of it as a generalization of logistic regression.

$$p_{\text{Adelie}} = P(y_i = \text{Adelie}|\vec{x}_i) = \frac{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)}}{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Gentoo}} \cdot \text{Aug}(\vec{x}_i)}}$$

$$p_{\text{Chinstrap}} = P(y_i = \text{Chinstrap}|\vec{x}_i) = \frac{e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)}}{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Gentoo}} \cdot \text{Aug}(\vec{x}_i)}}$$

same denominator!

$$\underbrace{p_j = P(y_i = j|\vec{x}_i) = \frac{e^{\vec{w}_j \cdot \text{Aug}(\vec{x}_i)}}{\sum_{k \in C} e^{\vec{w}_k \cdot \text{Aug}(\vec{x}_i)}}}$$

$$p_{\text{Adelie}} = P(y_i = \text{Adelie}|\vec{x}_i) = \frac{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)}}{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Gentoo}} \cdot \text{Aug}(\vec{x}_i)}}$$

$$p_{\text{Chinstrap}} = P(y_i = \text{Chinstrap}|\vec{x}_i) = \frac{e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)}}{e^{\vec{w}_{\text{Adelie}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Chinstrap}} \cdot \text{Aug}(\vec{x}_i)} + e^{\vec{w}_{\text{Gentoo}} \cdot \text{Aug}(\vec{x}_i)}}$$

$\vec{x}_i =$ (Bill Length$_i$, Body Mass$_i$)

$$\underbrace{p_j = P(y_i = j|\vec{x}_i) = \frac{e^{\vec{w}_j \cdot \text{Aug}(\vec{x}_i)}}{\sum_{k \in C} e^{\vec{w}_k \cdot \text{Aug}(\vec{x}_i)}}}_{\text{in general}}$$

Suppose $z \in \mathbb{R}^n$. Then, the softmax of z is defined element-wise as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

- For example, suppose $\vec{z} = \begin{bmatrix} -5 \\ 2 \\ 4 \end{bmatrix}$. Then:

$$\sigma(\vec{z}) = \begin{bmatrix} \sigma(\vec{z})_1 \\ \sigma(\vec{z})_2 \\ \sigma(\vec{z})_3 \end{bmatrix} = \begin{bmatrix} \frac{e^{-5}}{e^{-5}+e^{2}+e^{4}} \\ \frac{e^{2}}{e^{-5}+e^{2}+e^{4}} \\ \frac{e^{4}}{e^{-5}+e^{2}+e^{4}} \end{bmatrix} = \begin{bmatrix} 0.0001 \\ 0.1192 \\ 0.8807 \end{bmatrix}$$

note the constant denominator!

$e^{-5} + e^{2} + e^{4}$ all 3 have same denominator!

ll sum to 1.

- The `LogisticRegression` class supports multinomial logistic regression.

In [36]:
```
1 model_log = LogisticRegression(multi_class='multinomial')
2 model_log.fit(X_train, y_train)
```

Out[36]:

LogisticRegression

```
LogisticRegression(multi_class='multinomial')
```

- In total, the fit model has $3 \times 2 = 6$ coefficients and $3 \times 1 = 3$ intercepts.

In [37]:
```
1 model_log.coef_
```

Out[37]:
```
array([[-0.85,  0.  ],
       [ 0.84, -0.01],
       [ 0.02,  0.01]])
```

In [38]:
```
1 model_log.intercept_
```

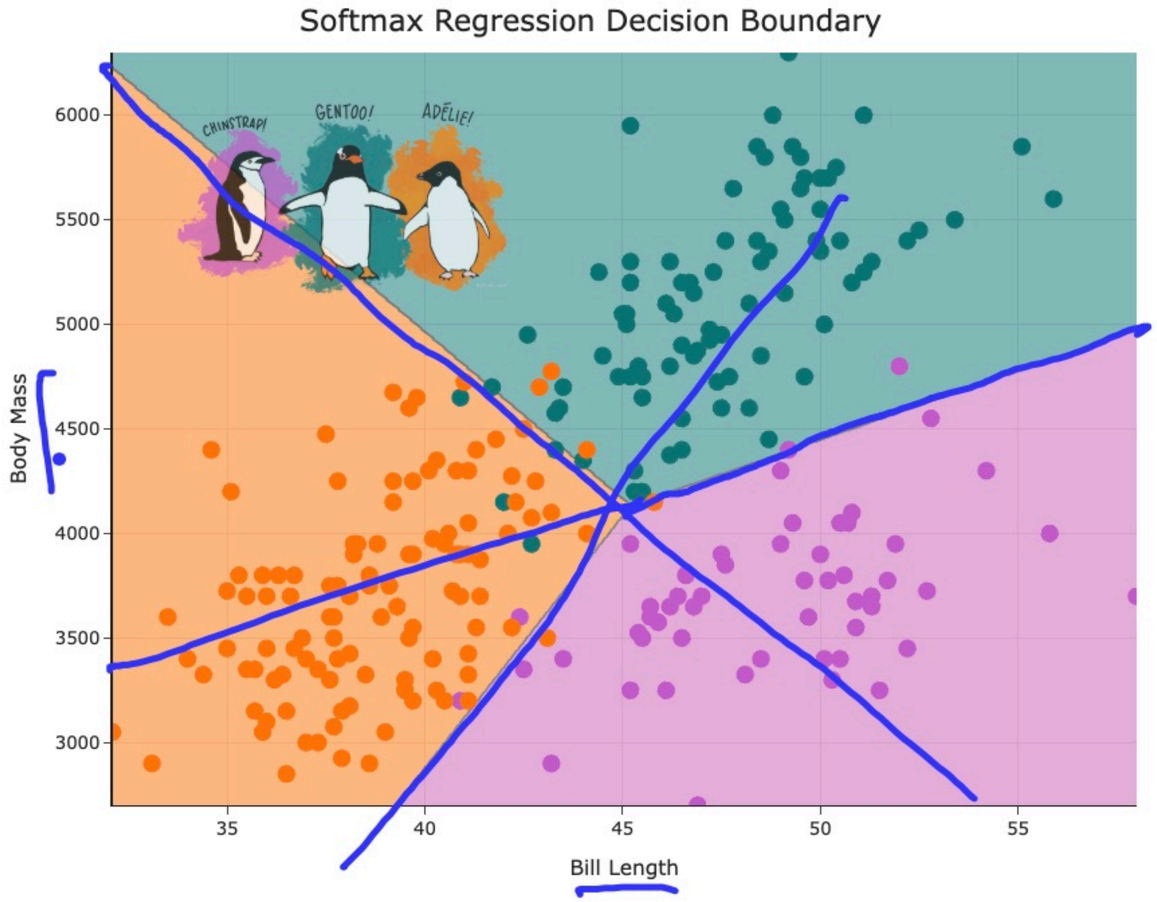
Out[38]:
```
array([ 36.4 , -10.96, -25.43])
```

In [39]:
```
1 model_log.classes_
```

Out[39]:
```
array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

What does this model *look* like?

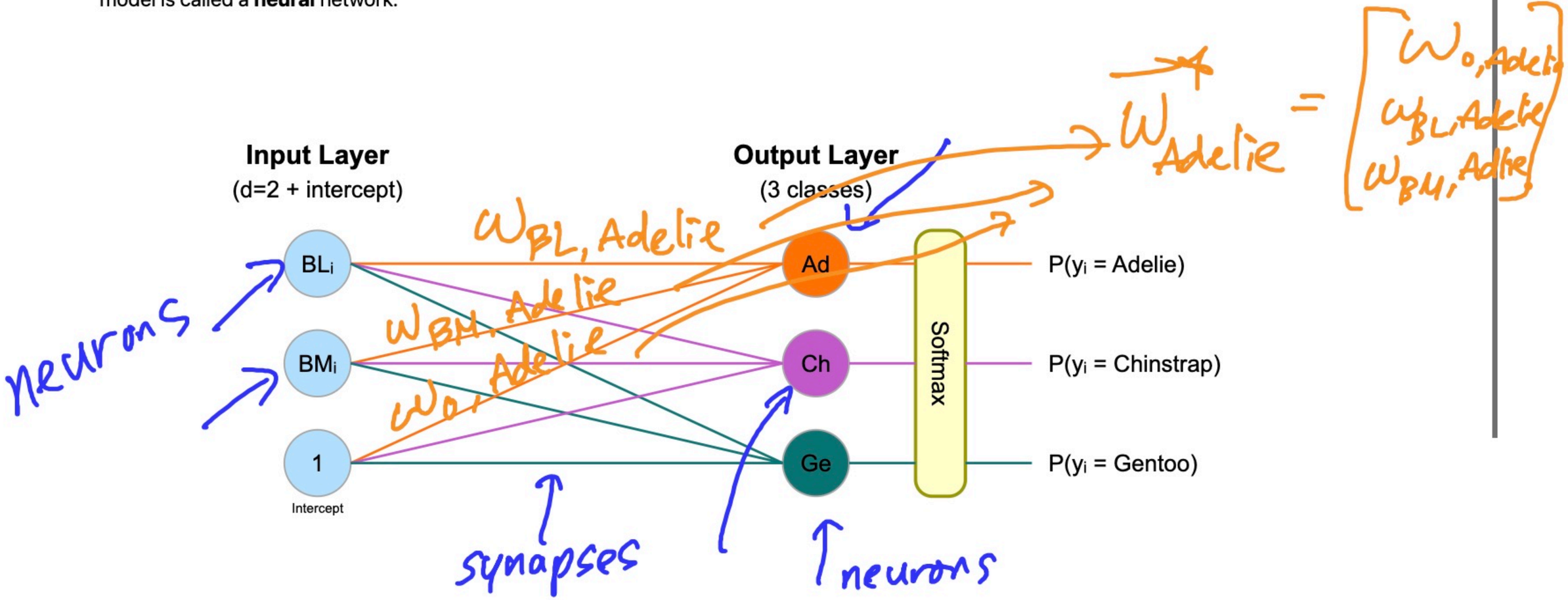
```
In [41]: 1 util.penguin_decision_boundary(model_log, X_train, y_train, title="Softmax Regression Decision Boundary")
```



Neural networks

- Softmax regression is an example of a **neural network**.

  Our brains are made up of **neurons** connected by "links", called synapses. The model diagram below loosely resembles this structure, which is why the model is called a **neural** network.



Neural networks

- Softmax regression is an example of a **neural network**.

  Our brains are made up of **neurons** connected by "links", called synapses. The model diagram below loosely resembles this structure, which is why the model is called a **neural** network.