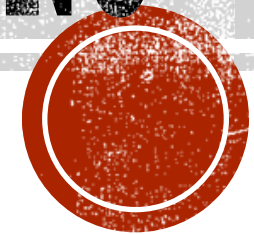
LOGISTIC REGRESSION
LOSS FUNCTION & PROGRAMMING



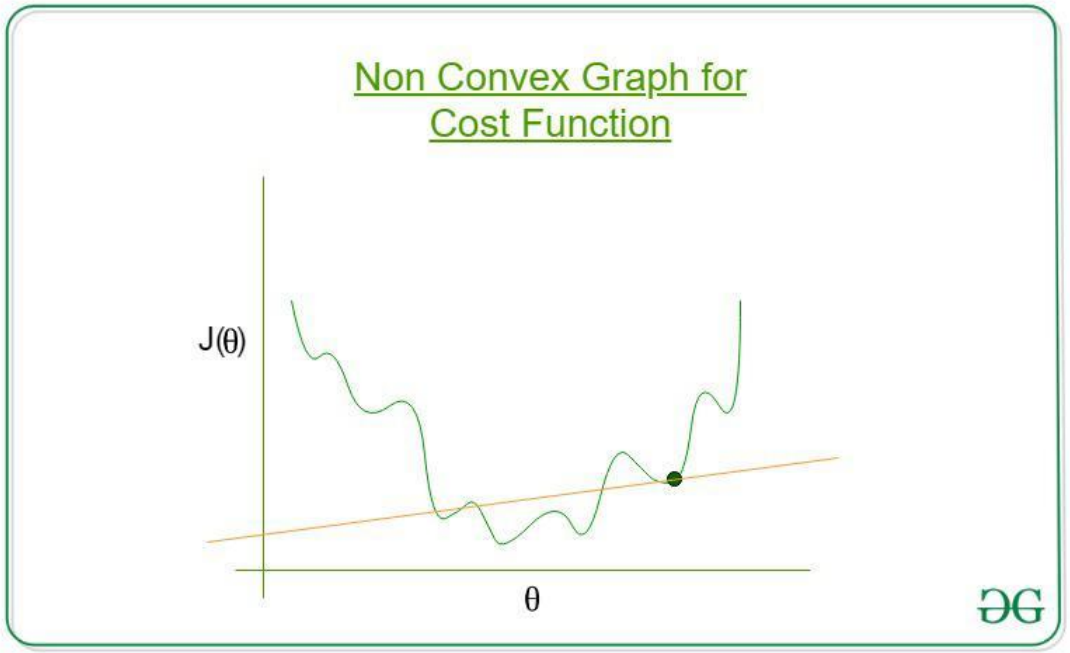
Dr. Brian Mc Ginley

LOSS FUNCTION

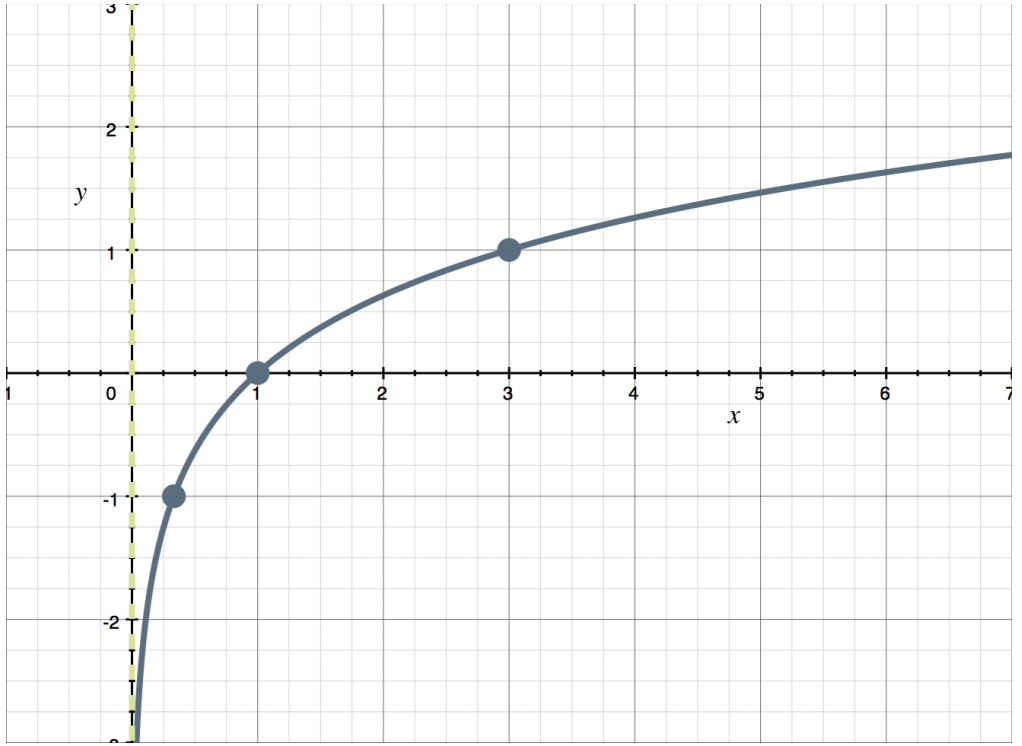
- To train a logistic regression model, we have to figure out what weights (w) minimise the cost/loss. We have to choose the cost function
- Mean Squared Error performs very poorly for Logistic Regression

$$\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - f_w(x^{(i)})\right)^2$$

- it would end up being a non-convex function with many local minimum (making it difficult to find the global minimum).



LOG LOSS FUNCTION



- Instead use this Cost function (which models the loss for 1 datapoint):

$$L(f_w(\boldsymbol{x}), y) = \begin{cases} -log\big(f_w(\boldsymbol{x})\big) & if\ y = 1 \\ -log\big(1 - f_w(\boldsymbol{x})\big) & if\ y = 0 \end{cases}$$

- Let's combine the above equation into one:

$$L(f_w(\boldsymbol{x}), y) = -\mathrm{y} * \log\big(f_w(\boldsymbol{x})\big) - (1 - \mathrm{y})\log(1 - f_w(\boldsymbol{x}))$$

- Note that this equation is still only for 1 sample

LOG LOSS FUNCTION

- Of course we need to calculate and average the error over all samples. Therefore, we use the following as our Loss Function for logistic regression

multiple samples

$$L(\boldsymbol{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m} -y_i * \log\left(f_{\boldsymbol{w}}(x_i)\right) - (1-y_i)\log(1-f_{\boldsymbol{w}}(x_i)\right]$$

- The function is convex which greatly simplifies minimisation. We need to find the optimum parameters $\boldsymbol{w}$ that minimises the loss on the training set. Then we can use the model to predict new outputs on unseen data.
- The algorithm to solve the problem and find the $\boldsymbol{w}$ is the same as others and we'll see it later.

PROGRAMMING

```
from sklearn.linear_model import LogisticRegression
```

- It has some options, but we'll mostly just keep the default

```
penalty: {'l1', 'l2', 'elasticnet', None },default='l2'
```

- is a hyperparameter for it, the default is good - I'm just pointing it out, I'll explain later.

```
maxiter: int, default=100
```

- I've mentioned a few times about how the learning is doing things over and over again. Here we can control the max number of times it will do this process. Sometimes you might need more.

PROGRAMMING

- Now to talk about the outputs. Like LinearRegression it has

```
model.coef_
model.intercept_
model.score(X_test, y_test)        #Gives the "accuracy" score of the model
y_pred=model.predict(X_test)
```

- A new one it has is

```
model.classes_
```

- This gives you a list of class labels known to the model. e.g. your y's are

```
y = np.array([0,1,2,1,1,1,1,1,0,2,1]) #or a pandas series equivalent
```

- then model.classes will be [0, 1, 2]. If your y's are

```
y = np.array(["cat","dog","hamster","dog","dog","dog","cat"," hamster"])
```

- then model.classes will be ["cat","dog","hamster"] instead.

model.classes_ gives a list of classes known to model

PROGRAMMING

- A new method it has is

```
model.predict_proba(X_test)
```

- This one gives probability estimates for all the classes. Maybe you want to know how confident the model is in its predictions - but be aware, models can be over-confident!
- The .predict I mentioned on the previous slide uses thresholds to make the decisions.
- You may also want the probabilities for some performance metrics when trying to make decisions about models.