Artificial Intelligence Diploma

Machine Learning Session 6

NeuroTech

Agenda

1. Features Selection.
2. Filter Methods
3. Wrapper Methods
4. Embedded Methos
5. Classification
6. Logistic Regression

Feature Selection

NeuroTech

Feature Selection

- **Feature selection** is a crucial step in data preprocessing and machine learning. It involves choosing a subset of relevant features (variables or attributes) from a larger set of available features to build a more efficient and accurate predictive model. Effective feature selection can improve model performance, reduce overfitting, and make the model more interpretable. Here's everything you need to know about feature selection:

- **Why Feature Selection:**

  - Reduces the dimensionality of the data, which can lead to faster training and prediction times.
  - Helps prevent overfitting, where a model learns noise in the data rather than true patterns.
  - Improves model interpretability, as a reduced feature set is easier to understand.
  - Reduces the risk of the curse of dimensionality, which can make models less robust.

Feature Selection

2. Types of Feature Selection:

- **Filter Methods:** These methods use statistical techniques to evaluate the relevance of features before the model is trained. Common metrics include correlation, chi-squared, and mutual information. Features are selected based on their individual characteristics and their relationship with the target variable.

- **Wrapper Methods:** Wrapper methods evaluate different subsets of features by training and testing the model with each subset. Common algorithms include forward selection, backward elimination, and recursive feature elimination (RFE). These methods can be computationally expensive but can provide the best feature subset for a specific model.

- **Embedded Methods:** Embedded methods incorporate feature selection as an integral part of the model training process. Techniques like L1 regularization (Lasso), decision trees, and random forests perform feature selection while building the model.

Feature Selection Methods

Filter Methods

Feature Selection

- **Filter methods** are a category of feature selection techniques used to evaluate the relevance of features based on their statistical properties, without involving a machine learning model. These methods are applied before the modeling process and are particularly useful when you have a large dataset with many features. Filter methods are computationally efficient and can help reduce dimensionality while selecting the most informative features. Here are the details of filter methods:

- **Common Metrics:**
  - Filter methods use various statistical metrics to assess the importance of each feature. Common metrics include:
    1. Correlation
    2. Chi-Squared
    3. Variance Threshold

Feature Selection

Correlation Coefficient

- Correlation is a measure of the linear relationship between 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that good variables correlate highly with the target. Furthermore, variables should be correlated with the target but uncorrelated among themselves.
- If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only needs one, as the second does not add additional information. We will use the Pearson Correlation here.

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Correlation matrix
cor = dataframe.corr()

# Plotting Heatmap
plt.figure(figsize = (10,6))
sns.heatmap(cor, annot = True)
```

Feature Selection

We need to set an absolute value, say 0.5, as the threshold for selecting the variables. If we find that the predictor variables are correlated, we can drop the variable with a lower correlation coefficient value than the target variable. We can also compute multiple correlation coefficients to check whether more than two variables correlate. This phenomenon is known as multicollinearity.

Feature Selection

Chi-square Test

- The Chi-square test is used for categorical features in a dataset. We calculate Chi-square between each feature and the target and select the desired number of features with the best Chi-square scores. In order to correctly apply the chi-squared to test the relation between various features in the dataset and the target variable, the following conditions have to be met: the variables have to be categorical, sampled independently, and values should have an *expected frequency greater than 5*.

```
1  from sklearn.feature_selection import SelectKBest
2  from sklearn.feature_selection import chi2
3
4  # Convert to categorical data by converting data to integers
5  X_cat = X.astype(int)
6
7  # Three features with highest chi-squared statistics are selected
8  chi2_features = SelectKBest(chi2, k = 3)
9  X_kbest_features = chi2_features.fit_transform(X_cat, Y)
10
11 # Reduced features
12 print('Original feature number:', X_cat.shape[1])
13 print('Reduced feature number:', X_kbest_features.shape[1])
```

```
Original feature number: 8
Reduced feature number: 3
```

Feature Selection

Variance Threshold

- The variance threshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e., features with the same value in all samples. We assume that features with a higher variance may contain more useful information, but note that we are not taking the relationship between feature variables or feature and target variables into account, which is one of the drawbacks of filter methods.

```
1  from sklearn.feature_selection import VarianceThreshold
2
3  # Resetting the value of X to make it non-categorical
4  X = array[:,0:8]
5
6  v_threshold = VarianceThreshold(threshold=0)
7  v_threshold.fit(X)  # fit finds the features with zero variance
8  v_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

Wrapper Methods

NeuroTech

Feature Selection

- Wrapper methods for feature selection are a category of techniques used to evaluate the performance of different feature subsets by training and testing a machine learning model with each subset. These methods wrap around the machine learning algorithm and use its performance as a criterion for selecting the best features. Unlike filter methods, which rely on statistical measures, wrapper methods consider the impact of feature subsets on the model's predictive performance. They can be computationally expensive but tend to yield more accurate feature selections.

- There are several popular wrapper methods for feature selection, including:
  1. **Forward Selection**
  2. **Backward Elimination**
  3. **Exhaustive Feature Selection**

Feature Selection

Forward Feature Selection

- This is an iterative method wherein we start with the performing features against the target features. Next, we select another variable that gives the best performance in combination with the first selected variable. This process continues until the preset criterion is achieved.

```
# Forward Feature Selection
from mlxtend.feature_selection import SequentialFeatureSelector
ffs = SequentialFeatureSelector(lr, k_features='best', forward = True, n_jobs=-1)
ffs.fit(X, Y)
features = list(ffs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Feature Selection

Backward Feature Elimination

- This method works exactly opposite to the Forward Feature Selection method. Here, we start with all the features available and build a model. Next, we the variable from the model, which gives the best evaluation measure value. This process is continued until the preset criterion is achieved.

```
# Backward Feature Selection
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector
lr = LogisticRegression(class_weight = 'balanced', solver = 'lbfgs', random_state=42, n_jobs=-1, max_iter=500)
lr.fit(X, Y)
bfs = SequentialFeatureSelector(lr, k_features='best', forward = False, n_jobs=-1)
bfs.fit(X, Y)
features = list(bfs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Feature Selection

Exhaustive Feature Selection

- This is the most robust feature selection method covered so far. This is a brute-force evaluation of each feature subset. This means it tries every possible combination of the variables and returns the best-performing subset.

```python
# Exhaustive Feature Selection
from mlxtend.feature_selection import ExhaustiveFeatureSelector

# import the algorithm you want to evaluate on your features.
from sklearn.ensemble import RandomForestClassifier

# create the ExhaustiveFeatureSelector object.
efs = ExhaustiveFeatureSelector(RandomForestClassifier(),
           min_features=4,
           max_features=8,
           scoring='roc_auc',
           cv=2)

# fit the object to the training data.
efs = efs.fit(X, Y)

# print the selected features.
selected_features = x_train.columns[list(efs.best_idx_)]
print(selected_features)

# print the final prediction score.
print(efs.best_score_)
```

Embedded Methods

Feature Selection

- Embedded methods for feature selection are techniques that perform feature selection as an integral part of the model training process. Unlike filter methods, which are independent of the machine learning algorithm, and wrapper methods, which use a separate model to evaluate feature subsets, embedded methods select features while training the model itself. These methods are particularly useful when you want to optimize model performance and reduce overfitting by selecting the most relevant features during the training process. Some popular embedded methods for feature selection include:

- **L1 Regularization (Lasso):** L1 regularization adds a penalty term to the model's cost function that encourages some feature weights to become exactly zero. As a result, Lasso regression automatically selects a subset of the most important features while reducing the impact of less relevant features.

- **L2 Regularization (Ridge):** While L2 regularization (Ridge) doesn't exactly eliminate features like L1 regularization, it can reduce the impact of less important features by shrinking their coefficients. Ridge regression tends to keep all features but down weights irrelevant.

Feature Selection

- **Elastic Net:** Elastic Net combines L1 and L2 regularization, allowing you to benefit from both feature selection and feature weight shrinkage. It provides a trade-off between Lasso and Ridge regularization.

- **Decision Trees (Random Forest, XGBoost, LightGBM):** Tree-based models can calculate feature importance scores based on how often features are used for splitting nodes. Features with higher importance scores are typically more relevant for the model's decision-making process.

Feature Selection

- **Ridge regression** adds "*squared magnitude*" of coefficient as penalty term to the loss function. Here the *highlighted* part represents L2 regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Here, if *lambda* is zero then you can imagine we get back OLS. However, if *lambda* is very large then it will add too much weight and it will lead to under-fitting. Having said that it's important how *lambda* is chosen. This technique works very well to avoid over-fitting issue.

Feature Selection

- **Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds "*absolute value of magnitude*" of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Again, if *lambda* is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.

Feature Selection

Ridge vs Lasso

o Ridge forces parameters to be small and Ridge is computationally easier because it is differentiable

o Lasso tends to set coefficients exactly equal to zero

- *This is useful as a sort of 'automatic feature selection' mechanism*
- *Leads to 'sparse models*
- *Serves a similar purpose to stepwise features selection*
- *Sparse models will benefit from lasso*
- *Dense models will benefit from ridge*

o In general L1(Lasso) penalties are better at recovering sparse signals

o L2 penalties are better at minimizing prediction error. Each out-of-sample (new/test data) difference is called a prediction error instead of residual.

Feature Selection

Random Forest Importance

- Random Forests is a kind of Bagging Algorithm that aggregates a specified number of decision trees. The tree-based strategies used by random forests naturally rank by how well they improve the purity of the node, or in other words, a decrease in the impurity (**Gini impurity**) over all trees. Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of the trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.

Go to Notebook

NeuroTech

Classification

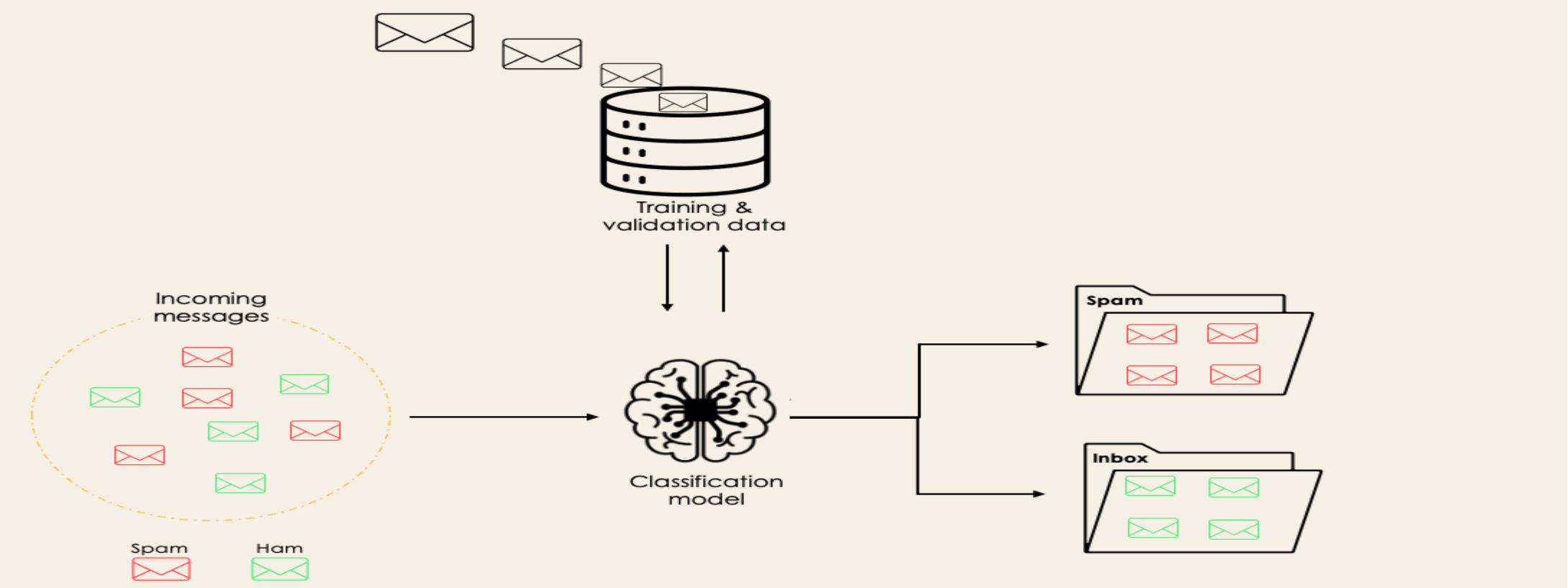
NeuroTech

Classification

- **Classification** is a fundamental and widely used technique in machine learning and data analysis. It is a type of supervised learning that involves categorizing or labeling data into predefined classes or categories based on the features or attributes of the data. The goal of classification is to build a model that can automatically assign new, unseen data points to one of these predefined classes.

- Classification plays a pivotal role in various real-world applications, ranging from spam email detection and medical diagnosis to image recognition and sentiment analysis. It enables machines to make decisions and predictions by learning patterns and relationships within the data.

Classification



Logistic Regression

NeuroTech

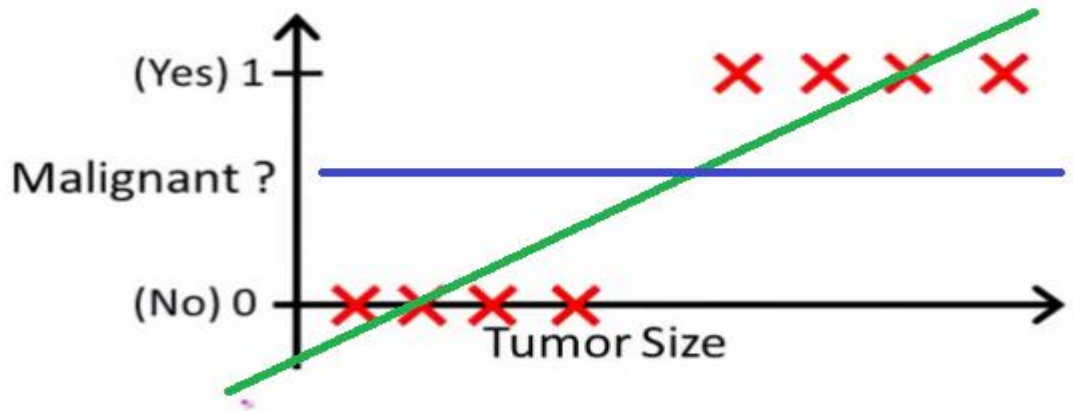
Logistic Regression

- **Logistic Regression** is a statistical and machine learning model used for binary classification problems, where the goal is to predict one of two possible outcomes (usually labeled as 0 and 1, or "negative" and "positive"). It is a well-established and interpretable model widely used in various fields, including healthcare, marketing, finance, and natural language processing.

- **Linear Regression vs. Logistic Regression:**
  - Logistic Regression is a generalized linear model, but it's distinct from linear regression. In linear regression, you predict a continuous outcome, while in logistic regression, you predict the probability of a binary outcome.

Logistic Regression

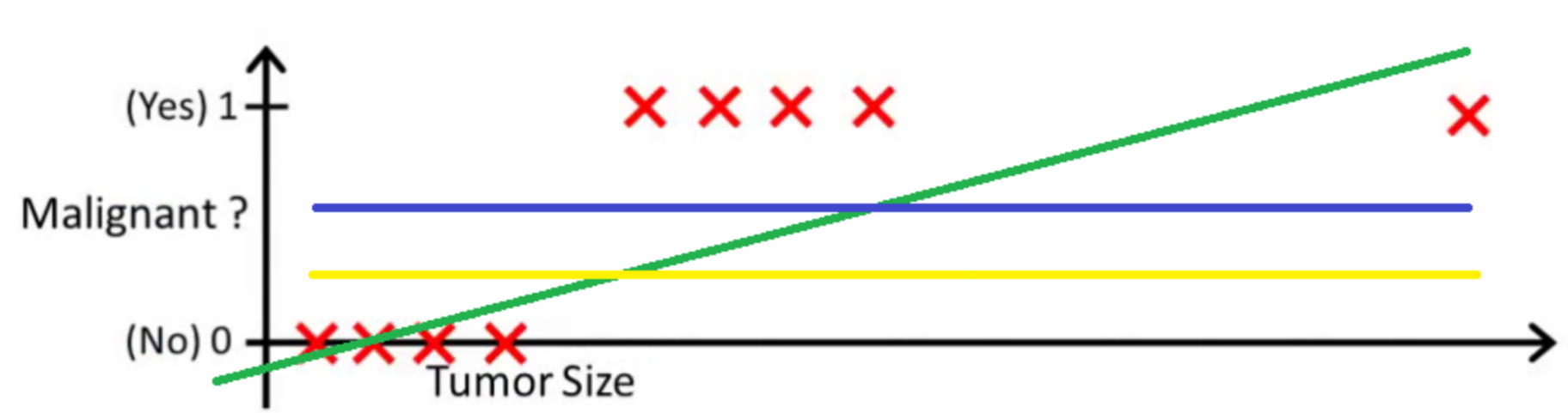
- **Why do we use Logistic Regression rather than Linear Regression?**
- if we use linear regression to find the best fit line which aims at minimizing the distance between the predicted value and actual value, the line will be like this:



- Here the threshold value is 0.5, which means if the value of h(x) is greater than 0.5 then we predict malignant tumor (1) and if it is less than 0.5 then we predict benign tumor (0).

Logistic Regression

- Everything seems okay here but now let’s change it a bit, we add some outliers in our dataset, now this best fit line will shift to that point. Hence the line will be somewhat like this:



Logistic Regression

- Do you see any problem? The blue line represents the old threshold and the yellow line represents the new threshold which is maybe 0.2 here. To keep our predictions right we had to lower our threshold value. Hence we can say that linear regression is prone to outliers. Now here if h(x) is greater than 0.2 then only this regression will give correct outputs.

- Another problem with linear regression is that the predicted values may be out of range. We know that probability can be between 0 and 1, but if we use linear regression this probability may exceed 1 or go below 0.

- To overcome these problems we use Logistic Regression, which converts this straight best fit line in linear regression to an S-curve using the sigmoid function, which will always give values between 0 and 1. How does this work and what's the math behind this will be covered in a later section?

Logistic Regression

- The formula of logistic function:

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

- We all know the equation of the best fit line in linear regression is:

$$y = \beta_0 + \beta_1 x$$

- Let's say instead of y we are taking probabilities (P). But there is an issue here, the value of (P) will exceed 1 or go below 0 and we know that range of Probability is (0-1). To overcome this issue we take ***"odds"*** of P:

$$P = \beta_0 + \beta_1 x$$

$$\frac{P}{1 - P} = \beta_0 + \beta_1 x$$

Logistic Regression

- We know that odds can always be positive which means the range will always be (0,+∞). Odds are nothing but the ratio of the probability of success and probability of failure. Now the question comes out of so many other options to transform this why did we only take ***'odds'***? Because odds are probably the easiest way to do this, that's it.

- The problem here is that the range is restricted and we don't want a restricted range because if we do so then our correlation will decrease. By restricting the range we are actually decreasing the number of data points and of course, if we decrease our data points, our correlation will decrease. It is difficult to model a variable that has a restricted range. To control this we take the ***log of odds*** which has a range from (-∞,+∞).

$$\log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x$$

Logistic Regression

- If you understood what I did here then you have done 80% of the maths. Now we just want a function of P because we want to predict probability right? not log of odds. To do so we will multiply by ***exponent*** on both sides and then solve for P.

$$\exp[\log(\frac{p}{1-p})] = \exp(\beta_0 + \beta_1 x)$$

$$e^{\ln[\frac{p}{1-p}]} = e^{(\beta_0 + \beta_1 x)}$$

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

$$p = e^{\left(\beta_0 + \beta_1 x\right)} - pe^{\left(\beta_0 + \beta_1 x\right)}$$

Logistic Regression

$$p = p[\frac{e^{(\beta_0+\beta_1 x)}}{p} - e^{(\beta_0+\beta_1 x)}]$$

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

$$p[1 + e^{(\beta_0+\beta_1 x)}] = e^{(\beta_0+\beta_1 x)}$$

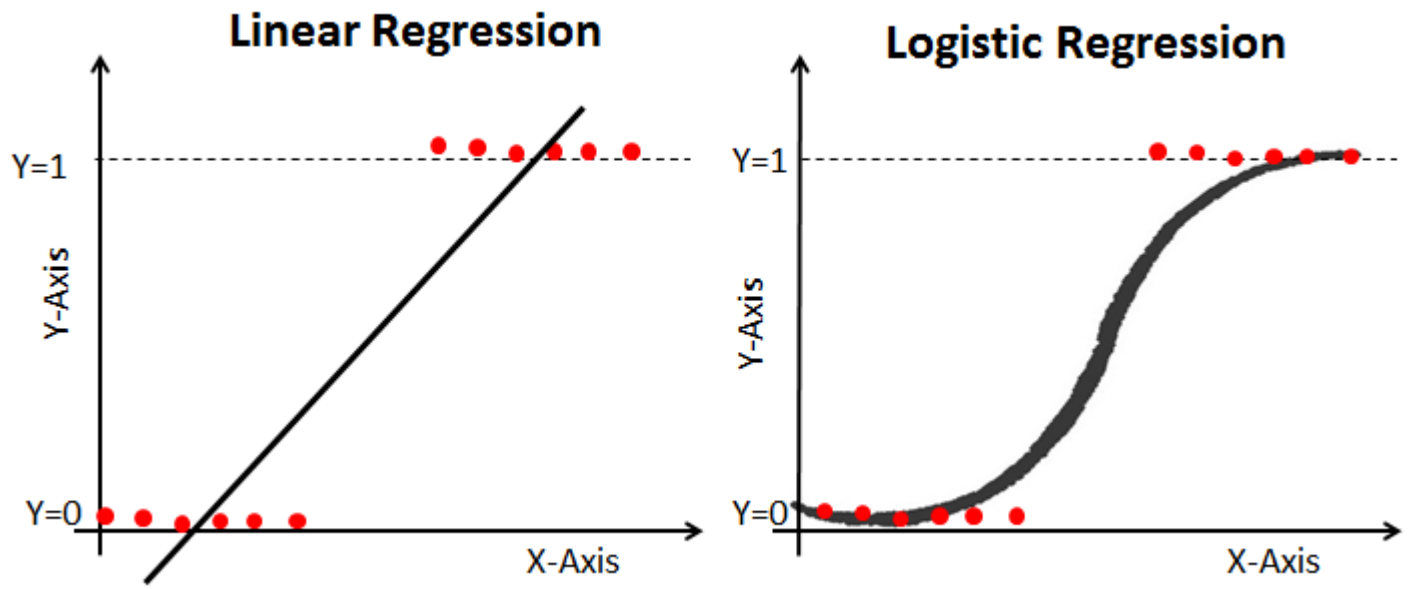
$$p = \frac{e^{(\beta_0+\beta_1 x)}}{1 + e^{(\beta_0+\beta_1 x)}}$$

Now dividing by $e^{(\beta_0+\beta_1 x)}$, *we will get*

$$p = \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$ *This is our sigmoid function.*

Logistic Regression

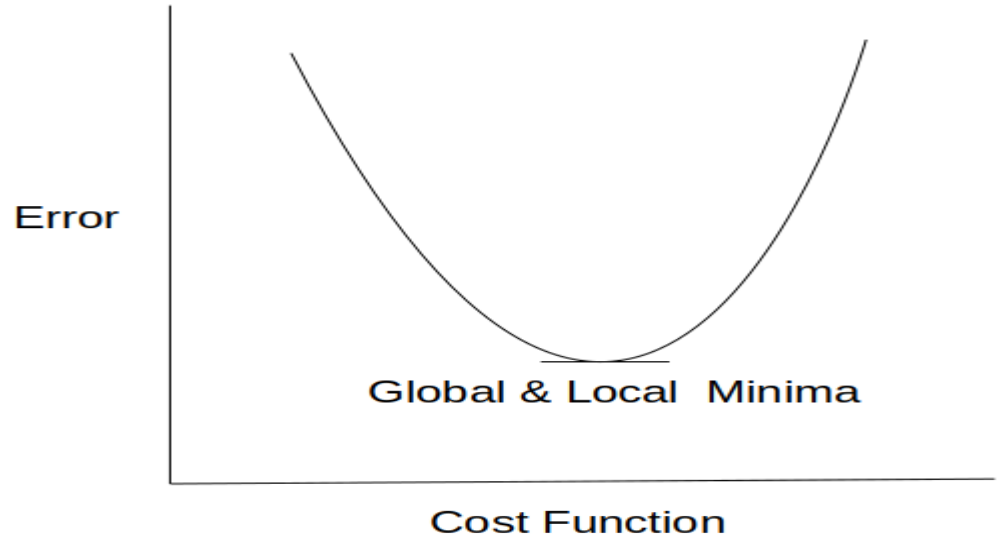
- Now we have our logistic functioan, also called a sigmoid function. The graph of a sigmoid function is as shown below. It squeezes a straight line into an S-curve.



Logistic Regression

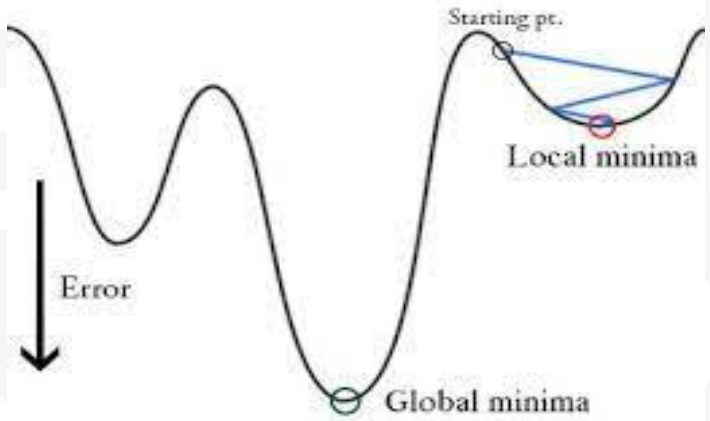
Cost Function in Logistic Regression

- In linear regression, we use the Mean squared error which was the difference between y_predicted and y_actual and this is derived from the maximum likelihood estimator. The graph of the cost function in linear regression is like this:



Logistic Regression

- In logistic regression Yi is a non-linear function ($\hat{Y}=1/1+ e^{-z}$). If we use this in the Previous MSE equation then it will give a non-convex graph with many local minima as shown



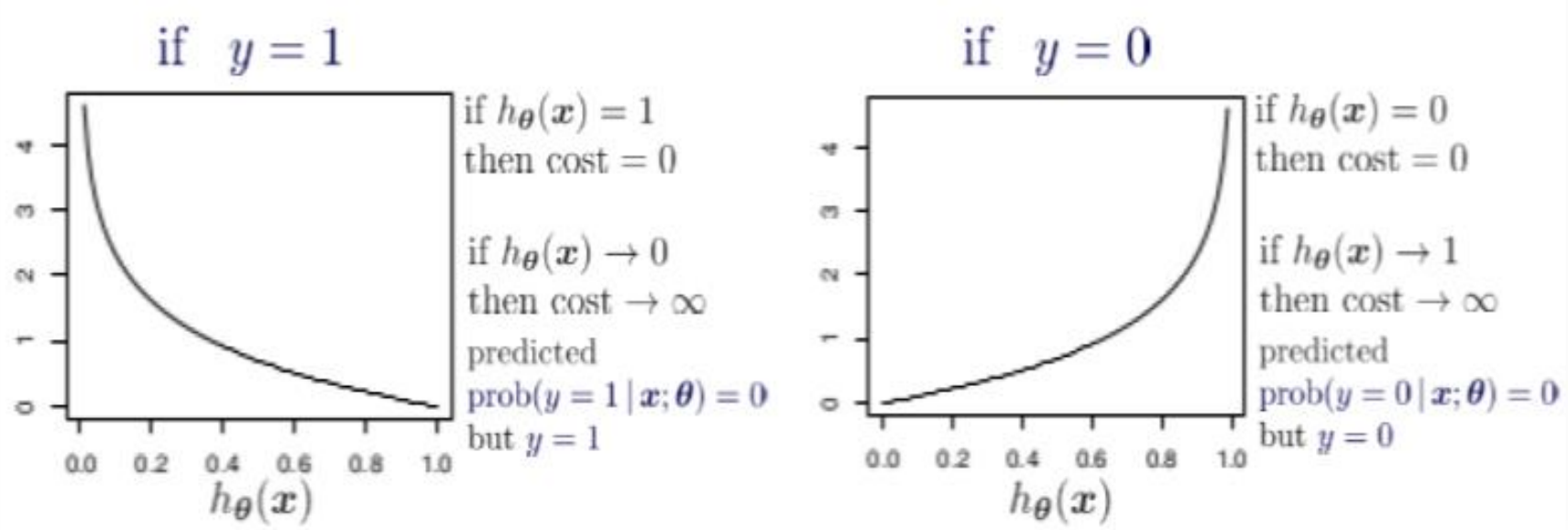
Logistic Regression

- The problem here is that this cost function will give results with local minima, which is a big problem because then we'll miss out on our global minima and our error will increase.
- In order to solve this problem, we derive a different cost function for logistic regression called ***log loss*** which is also derived from the *maximum likelihood estimation* method

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -(y_i * \log (\hat{Y}_i) + (1- y_i) * \log (1-\hat{Y}_i))$$

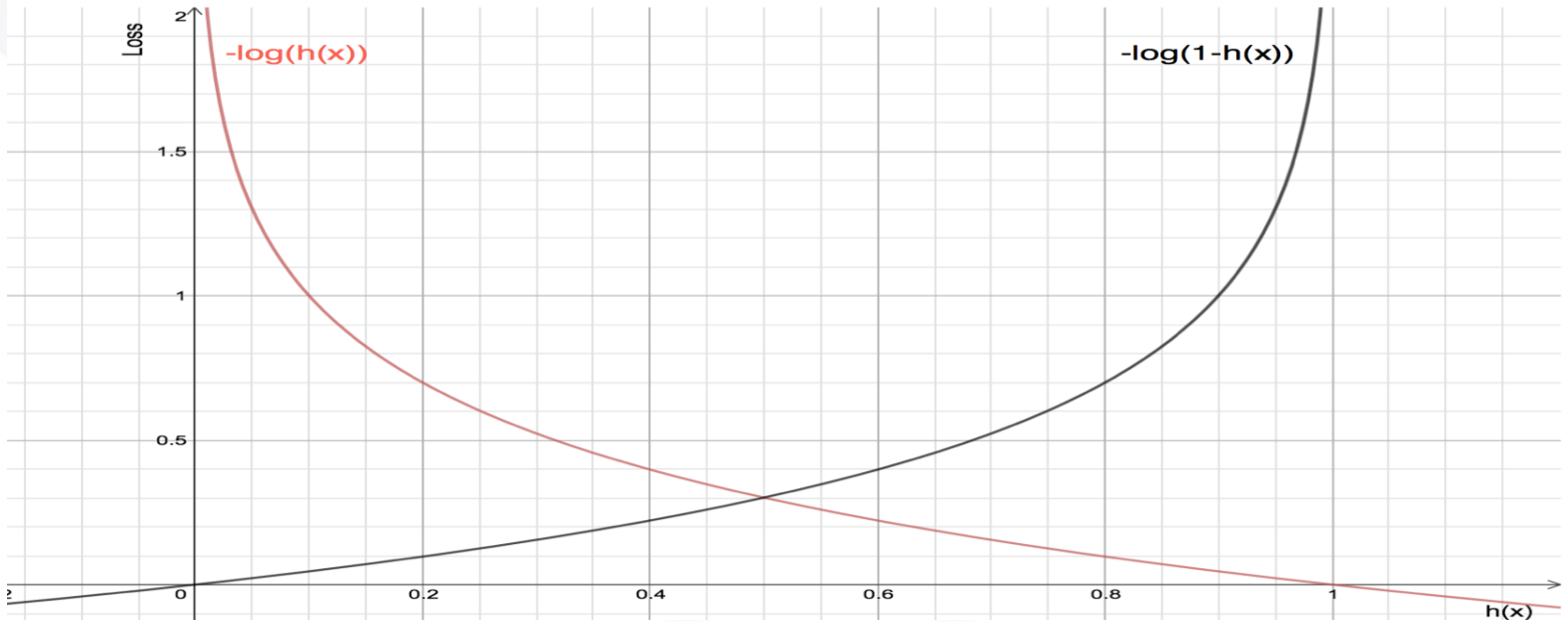
Logistic Regression

- The graph of cost function when y=1 and y=0



Logistic Regression

- If we combine both the graphs, we will get a convex graph with only 1 local minimum and now it'll be easy to use gradient descent here.



Logistic Regression

The red line here represents the 1 class (y=1), the right term of cost function will vanish. Now if the predicted probability is close to 1 then our loss will be less and when probability approaches 0, our loss function reaches infinity.

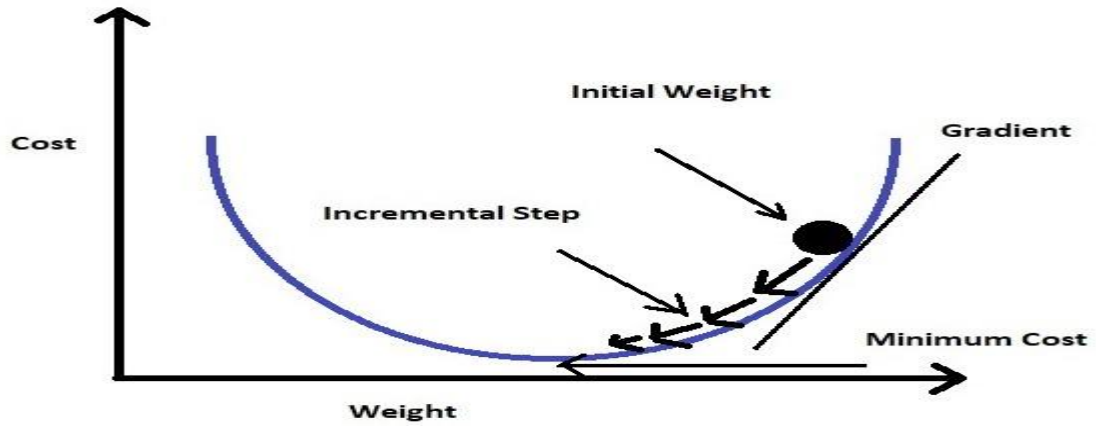
The black line represents 0 class (y=0), the left term will vanish in our cost function and if the predicted probability is close to 0 then our loss function will be less but if our probability approaches 1 then our loss function reaches infinity.

$$\text{Cost}(\text{h}_\Theta(x), y) = \begin{cases} -log(h_\Theta(x)) & if \quad y = 1 \\ -log(1 - h_\Theta(x)) & if \quad y = 0 \end{cases}$$

This cost function is also called log loss. It also ensures that as the probability of the correct answer is maximized, the probability of the incorrect answer is minimized. Lower the value of this cost function higher will be the accuracy.

Logistic Regression

- **Gradient Descent Optimization:**
  - In Linear regression session, we will try to understand how we can utilize *Gradient Descent* to compute the minimum cost.
  - Gradient descent changes the value of our weights in such a way that it always converges to minimum point or we can also say that, it aims at finding the optimal weights which minimize the loss function of our model. It is an iterative method that finds the minimum of a function by figuring out the slope at a random point and then moving in the opposite direction.



Logistic Regression

At first gradient descent takes a random value of our parameters from our function. Now we need an algorithm that will tell us whether at the next iteration we should move left or right to reach the minimum point. The gradient descent algorithm finds the slope of the loss function at that particular point and then in the next iteration, it moves in the opposite direction to reach the minima. Since we have a convex graph now we don't need to worry about local minima. A convex curve will always have only 1 minima.
We can summarize the gradient descent algorithm as:

$$\theta_{new} = \theta_{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

Go to Notebook

NeuroTech

Any Question?

NeuroTech

Thanks

NeuroTech