2

Linear, Polynomial and Logistic Regression

and the difference between them

Chapter outline

1. Linear regression
2. Polynomial regression
3. Logistic regression

1. Linear regression

A linear classifier is based on linear regression.

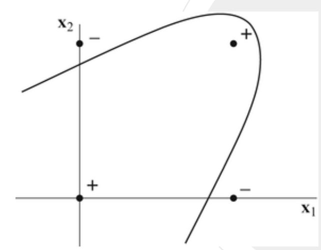
We have seen **linear regression** in the "Introduction to Artificial Intelligence" subject.

The best fit line

We can select the points situated above the line as elements of class 1, and the others as elements of class 2.

2. Polynomial regression

In some cases, it's not possible to separate the two classes with a straight line!



In this case, we're dealing with, for example, the **polynomial** classifier.

2. Polynomial regression (next)

A polynomial classifier is based on the **polynomial regression**.

How it works?

For simplicity, lets begin by two Boolean attributes x_1, x_2. The **second-order** polynomial is:

$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_1^2 + w_4 \cdot x_1 \cdot x_2 + w_5 \cdot x_2^2 = 0$$

$$\Leftrightarrow \sum_{k=0,l=0}^{k=2,l=2} w_i \cdot x_1^k \cdot x_2^l = 0$$

where $k + l \leq 2, \quad i = 0, 1, \ldots$

To generalize, we use **r^th order** polynomial:

$$\sum_{k=0,l=0}^{k=r,l=r} w_i \cdot x_1^k \cdot x_2^l = 0$$

2. Polynomial regression (next)

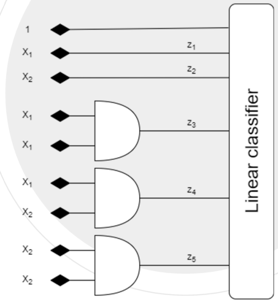
The goal is to **find weights** that separate positive examples from negative ones.

Next step is to convert the polynomial to Linear Classifier.

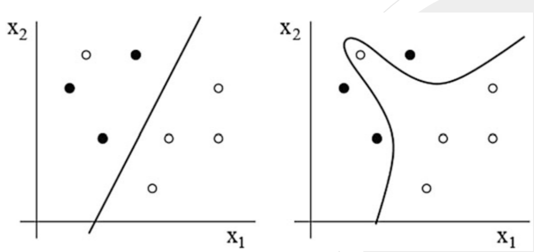
$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_1^2 + w_4 \cdot x_1 \cdot x_2 + w_5 \cdot x_2^2 = 0$$

$$\Longrightarrow \boldsymbol{w_0 + w_1 \cdot z_1 + w_2 \cdot z_2 + w_3 \cdot z_3 + w_4 \cdot z_4 + w_5 \cdot z_5 = 0}$$

Than we can use linear regression.



2. Polynomial regression (next)

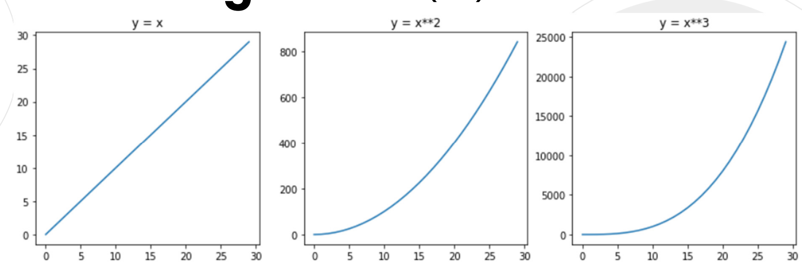


Limitations of polynomial classifier:

Overfitting!!!

2. Polynomial regression (next)

Exp.



Linear regression is just a **first-degree polynomial**.

Polynomial regression uses higher-degree polynomials.

Both of them are **linear models**, but the first results in a **straight line**, the latter gives a **curved line**.

2. Polynomial regression (next)

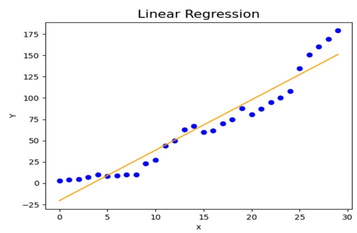
Exp. 1.

```
x = np.arange(0, 30)
y = [3, 4, 5, 7, 10, 8, 9, 10, 10, 23, 27,
44, 50, 63, 67, 60, 62, 70, 75, 88, 81, 87,
95, 100, 108, 135, 151, 160, 169, 179]
# Linear Regression
model = LinearRegression()
model.fit(x.reshape(-1,1), y)
Y_pred = model.predict(x.reshape(-1,1))
```

Linear Regression even failed to fit the training data well.

The polynomial is : $y = b_0 + b_1 x$

```
plt.scatter(x, y, color='blue')
plt.plot(x, Y_pred, color='orange')
plt.title("Linear Regression", size=16)
plt.xlabel('x')
plt.ylabel('Y')
plt.show()
```



2. Polynomial regression (next)

This problem is also called as **underfitting**.

To overcome the underfitting, we should a higher degree polynomial, for example 2nd order $(y = b_0 + b_1 x + b_2 x^2)$. We introduce new features vectors just by adding power to the original feature vector.

```
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(x.reshape(-1, 1))
    poly_features:
    [[  0.    0.]
     [  1.    1.]
     [  2.    4.]
     [  3.    9.]
     [  4.   16.]
     [  5.   25.]
     ...
```


2. Polynomial regression (next)

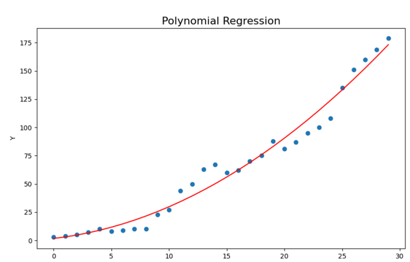
$y = b_0 + b_1 x + b_2 x^2$

We can now create the polynomial regression model.

```
poly_reg_model = LinearRegression()
```

Don't forget that polynomial regression is a linear model.

```
poly_reg_model.fit(poly_features, y)
y_predicted = poly_reg_model.predict(
poly_features)
# Plot the result
plt.figure(figsize=(10, 6))
plt.title("Polynomial Regression", size=16)
plt.scatter(x, y)
plt.plot(x, y_predicted, c="red")
plt.xlabel('x')
plt.ylabel('Y')
plt.show()
```



2. Polynomial regression (next)

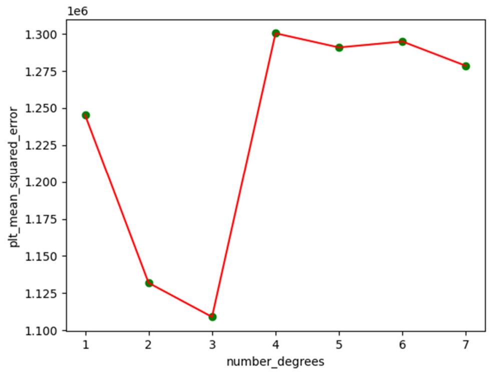
Exp. 2.

What if we have **multiple features**?

```
df = pd.read_csv('brooklyn_listings.csv')
df = df[['price', 'bathrooms', 'sqft']].dropna()
# Remove missing values
x_values = df[['bathrooms', 'sqft']].values
y_values = df['price'].values
number_degrees = [1, 2, 3, 4, 5, 6, 7]
plt_mean_squared_error = []
for degree in number_degrees:
  poly_model = PolynomialFeatures(degree=degree)
  poly_x_values =
  poly_model.fit_transform(x_values)
  X_train, X_test, y_train, y_test =
  train_test_split(poly_x_values, y_values,
  test_size=0.3, random_state=42)
  regression_model = LinearRegression()
  regression_model.fit(X_train, y_train)
  y_pred = regression_model.predict(X_test)
  plt_mean_squared_error.append(mean_squared_error
  (y_true=y_test, y_pred=y_pred, squared=False))

plt.scatter(number_degrees,
plt_mean_squared_error, color="green")
plt.plot(number_degrees, plt_mean_squared_error,
color="red")
plt.xlabel('number_degrees')
plt.ylabel('plt_mean_squared_error')
plt.show()
```

2. Polynomial regression (next)



3. Logistic regression

To complete what we have seen in the "Introduction to Artificial Intelligence" subject about regression, let's take a look at **logistic regression**.

Logistic regression is a supervised learning algorithm used to solve **classification** problems where the dependent variables (Y) are either binary or discrete (0 or 1).

It is a **predictive** analysis algorithm which works on the concept of **probability** called **Odd**.

Odd, is the ratio of **something occurring** to **something not occurring**.

It is **different** from probability as the probability is the ratio of something occurring to everything that could possibly occur.

3.1. Types of logistic Regression (next)

There are two kinds of logistic regression:

- **Binomial**: there can be only two possible values of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial**: there can be 3 or more possible **unordered** values of the dependent variable, such as "cat", "dogs", "sheep", … When the dependent variables are **ordered** (such as "low", "Medium", or "High"), multinomial is called **ordinal regression**.

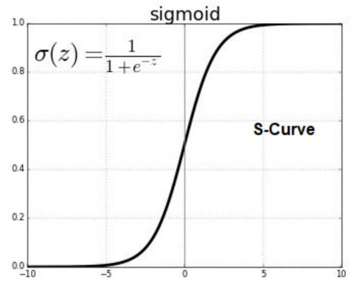
The logistic regression uses either **sigmoid function** or **SoftMax function** depending on the desired output.

3.1. Types of logistic Regression (next)

The **sigmoid function** is represented by:

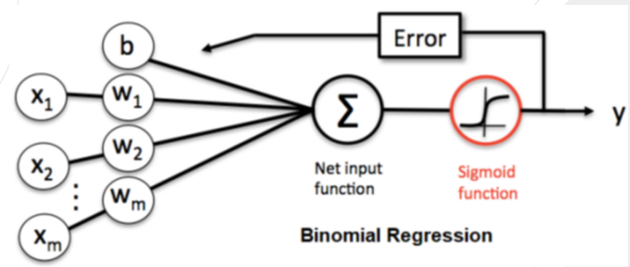
$f(z_i) = \frac{1}{1+e^{-z_i}} \in [0,1]$

for $i \in \{1, 2, \dots, n\}$



It uses the concept of **threshold** levels, with values above the threshold level **rounded up to 1** and values below the threshold level **rounded down to 0**.

3.1. Types of logistic Regression (next)



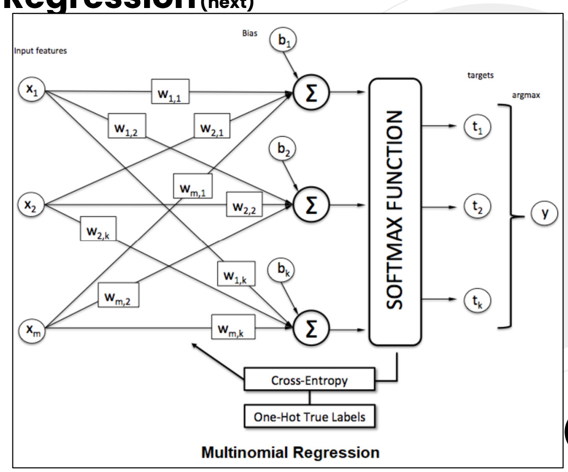
3.1. Types of logistic Regression (next)

The **SoftMax function** is represented by:

$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \in [0,1]$

for $i \in \{1, 2, \dots, k\}$

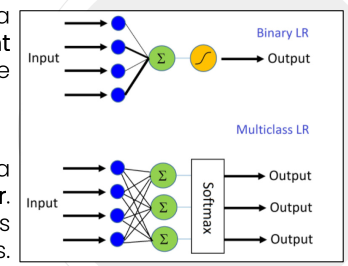
and k is the number of classes.



3.1. Types of logistic Regression (next)

We apply **sigmoid function** when we are building a classifier for a problem with **more than one right answer**. All right answers are classified in the same class (**Binomial regression**).

We apply **SoftMax function** when we are building a classifier for problems with **only one right answer**. Each right answer is classified in a different class from the classes of the other right answers. (**Multinomial regression**)



3.2. Difference between Linear and Logistic Regression

Linear Regression

- Linear regression is used to predict the **continuous** dependent variable using a given set of independent variables.
- The output must be **continuous** value, such as price, age, etc.
- Find best fit **line**

Logistic Regression

- Logistic regression is used to predict the **categorical** dependent variable using a given set of independent variables.
- Output must be **categorical** value such as "dog" or "cat", "car", etc.
- Find best fit **S-Curve**

3.3. Binomial regression: How it works?

Lets have the independent variables: $X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$

The dependent variable $y_i = \begin{cases} 0 & if\ class\ 1 \\ 1 & if\ class\ 2 \end{cases}$

$\Rightarrow$ **Binomial Regression.**

In logistic regression we apply multi-linear function to the input variables:

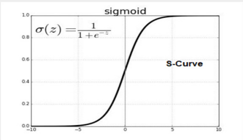
$$z_i = b_i + \sum_{j=1}^{m} w_{ij} \cdot x_{ij} \qquad i = \in \{1, \dots, n\}$$

$$\Leftrightarrow z_i = w_i \cdot X_i + b_i$$

3.3. Binomial regression: How it works? (next)

Next, we use the sigmoid function where the input is z_i.

$$\sigma(z_i) = \begin{cases} 1 & when\ z_i \to \infty \\ 0 & when\ z_i \to -\infty \\ \in [0,1] & \end{cases}$$



Now, we can define $\sigma(z_i)$ as the probability that the output is placed in class 2 $(y_i = 1) \Rightarrow p(X_i) = \sigma(z_i)$ (eq.1)

So,

the probability that the output is placed in class 1 $(y_i = 0)$ is $1 - p(X_i) = 1 - \sigma(z_i)$

3.3. Binomial regression: How it works? (next)

To define the **logistic regression equation** we have to define the **Odd** (the ratio of **something occurring** to **something not occurring**).

$$\frac{p(X_i)}{1-p(X_i)} = \frac{\sigma(z_i)}{1-\sigma(z_i)} = \frac{\frac{1}{1+e^{-z_i}}}{1-\frac{1}{1+e^{-z_i}}} = e^{z_i}$$

$$\Rightarrow \frac{p(X_i)}{1-p(X_i)} = e^{z_i}$$

If we apply the log we find:

$log\left(\frac{p(X_i)}{1-p(X_i)}\right) = z_i = w_i \cdot X_i + b_i$ (eq.2)

3.3. Binomial regression: How it works? (next)

So, the probability that the output is placed in class 2 ($y_i = 1$) is calculated by combining (eq.1) and (eq.2) as follows :

$$p(X_i) = \sigma(z_i) = \frac{1}{1+e^{-z_i}} = \frac{e^{z_i}}{1+e^{z_i}} = \frac{e^{w_i \cdot X_i + b_i}}{1+e^{w_i \cdot X_i + b_i}}$$

$$\Rightarrow \boldsymbol{p(X_i) = \frac{e^{w_i \cdot X_i + b_i}}{1+e^{w_i \cdot X_i + b_i}} = \frac{1}{1+e^{-(w_i \cdot X_i + b_i)}}}$$

which is called the Sigmoid Logistic Regression Equation.

3.3. Binomial regression: How it works? (next)

Exp. (binomial regression)

```
def probability (log_reg_model, X):
    w = log_reg_model.coef_       # Coefficients
    b = log_reg_model.intercept_  # Bias
    z = w * X + b
    odds = numpy.exp(z)  # e^z
    pX = odds / (1 + odds)
    return pX
X_train  =  numpy.array([3.78,  2.44,  2.09,  0.14,  1.72,  1.65,  4.92,  4.37,  4.96,  4.52,  3.69,
5.88]).reshape(-1, 1)
y_train = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
X_test = numpy.array([3.98, 1.44, 0.09, 5.14, 2.72]).reshape(-1, 1)
y_test = numpy.array([1, 0, 0, 1, 1])
log_reg = linear_model.LogisticRegression()
log_reg.fit(X_train, y_train)
print("Probabilities:\n", probability(log_reg, X_test))
```

Result:

```
Probabilities:
 [[0.67168934]
 [0.0558433 ]
 [0.00891433]
 [0.91166714]
 [0.26076382]]
```


3.3. Binomial regression: How it works? (next)

Exp. (next)

Python can predict output classes instead using our `probability()` function:

```
y_pred = log_reg.predict(X_test.reshape(-1, 1))
print(y_pred)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

Result:

```
y_pred:  [1 0 0 1 0]
Logistic Regression model
accuracy (in %): 80.0
```


3.4. Multinomial regression: How it works?

Lets have the independent variables: $X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$

The dependent variable $y_i \in \{1, 2, .., k\}$

$\Rightarrow$ Multinomial Regression.

In logistic regression we apply multi-linear function to the input variables:

$$z_i = b_i + \sum_{j=1}^{m} w_{ij} \cdot x_{ij} \qquad i \in \{1, \dots, n\}$$

$$\Leftrightarrow z_i = w_i \cdot X_i + b_i$$

3.4. Multinomial regression: How it works? (next)

The probability that the output is placed in class 'c' ($y = c$) is calculated by:

$$p(x_i) = softmax(z_i) = \frac{e^{z_i}}{\sum_{p=1}^{k} e^{z_p}} = \frac{e^{w_i \cdot X_i + b_i}}{\sum_{p=1}^{k} e^{w_p \cdot X_p + b_p}}$$

which is called the SoftMax Logistic Regression Equation.

3.4. Multinomial regression: How it works?

Exp. (Multinomial regression)

```
dataset = read_csv("iris.csv")
x = dataset.values[:, 0:4]
y = dataset.values[:, 4]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)
log_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs')
log_reg.fit(x_train, y_train)
y_pred = log_reg.predict(x_test)
acc = accuracy_score(y_test, y_pred)
print("Multinomial Logistic Regression model accuracy (in %):", acc*100)
```

Method to find the best Parameters (w) that give the least error in predicting the output.
`liblinear, lbfgs, newton-cg, sag, saga`
We'll look at some of them in the following sections.

Result:
```
Multinomial Logistic
Regression model accuracy
(in %): 96.66666666666667
```


3.4. Multinomial regression: How it works? (next)

Exp. (Multinomial regression) (next)

```
cm = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1, 2])
cm_display.plot()
plt.show()
```