# Logistic Regression

The ASTA team

## Contents



# 1 Introduction to logistic regression

## 1.1 Binary response

- We consider a binary response $y$ with outcome 1 or 0. This might be a code indicating whether a person is able or unable to perform a given task.
- Furthermore, we are given an explanatory variable $x$, which is numeric, e.g. age.
- We shall study models for
$$P(y = 1 \,|\, x)$$
i.e. the probability that a person of age $x$ is able to complete the task.
- We shall see methods for determining whether or not age actually influences the probability, i.e. is $y$ independent of $x$?

## 1.2 A linear model

$$P(y = 1 \,|\, x) = \alpha + \beta x$$

is simple, but often inappropiate. If $\beta$ is positive and $x$ sufficiently large, then the probability exceeds 1.

# 2 Simple logistic regression

## 2.1 Logistic model

Instead we consider the **odds** that the person is able to complete the task

$$\texttt{Odds}(y = 1 \,|\, x) = \frac{P(y = 1 \,|\, x)}{P(y = 0 \,|\, x)} = \frac{P(y = 1 \,|\, x)}{1 - P(y = 1 \,|\, x)}$$

which can have any positive value.

**The logistic model** is defined as:

$$\texttt{logit}(P(y = 1 \,|\, x)) = \log(\texttt{Odds}(y = 1 \,|\, x)) = \alpha + \beta x$$

The function $\texttt{logit}(p) = \log(\frac{p}{1-p})$ - i.e. **log of odds** - is termed **the logistic transformation**.

Remark that log odds can be any number, where zero corresponds to $P(y = 1 \,|\, x) = 0.5$. Solving $\alpha + \beta x = 0$ shows that at age $x_0 = -\alpha/\beta$ you have fifty-fifty chance of solving the task.

## 2.2 Logistic transformation

```
import numpy as np
from scipy.special import logit, expit

p = np.arange(0.1, 1.0, 0.2)  # stop is exclusive, so use 1.0
p
```

```
## array([0.1, 0.3, 0.5, 0.7, 0.9])
```

```
l = logit(p)
l.round(3)
```

```
## array([-2.197, -0.847,  0.   ,  0.847,  2.197])
```

```
expit(l)
```

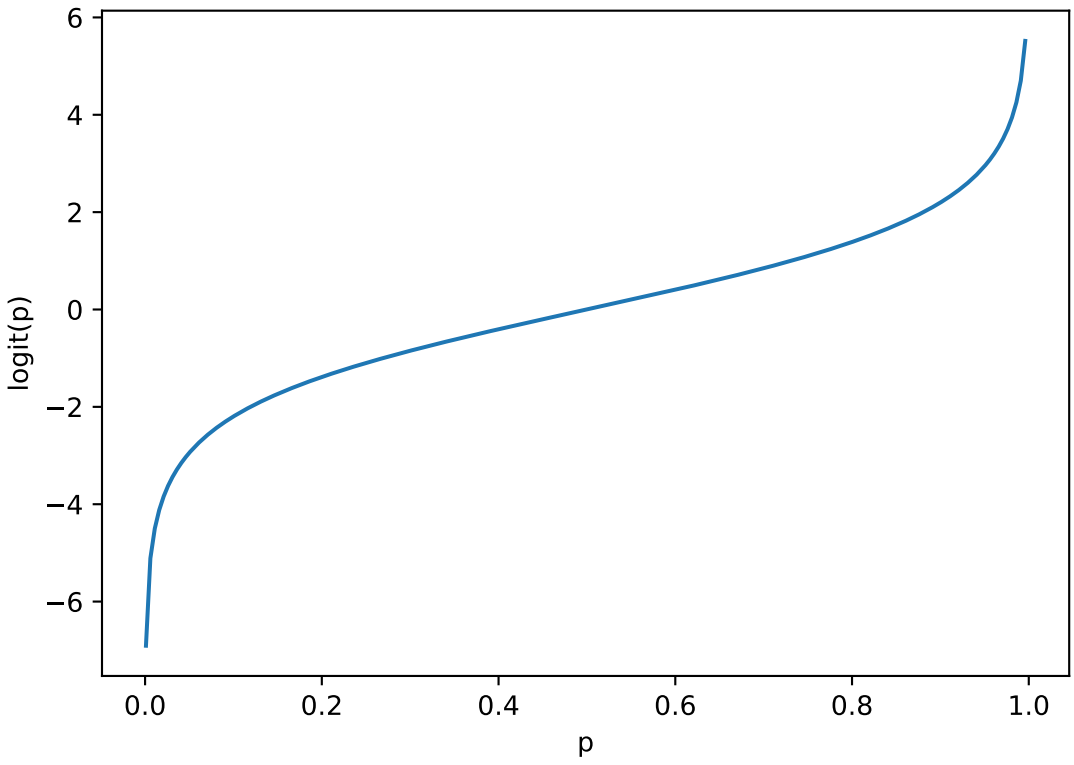
```
## array([0.1, 0.3, 0.5, 0.7, 0.9])
```

- The inverse logistic transformation `expit()` applied to the transformed values can recover the original probabilities:

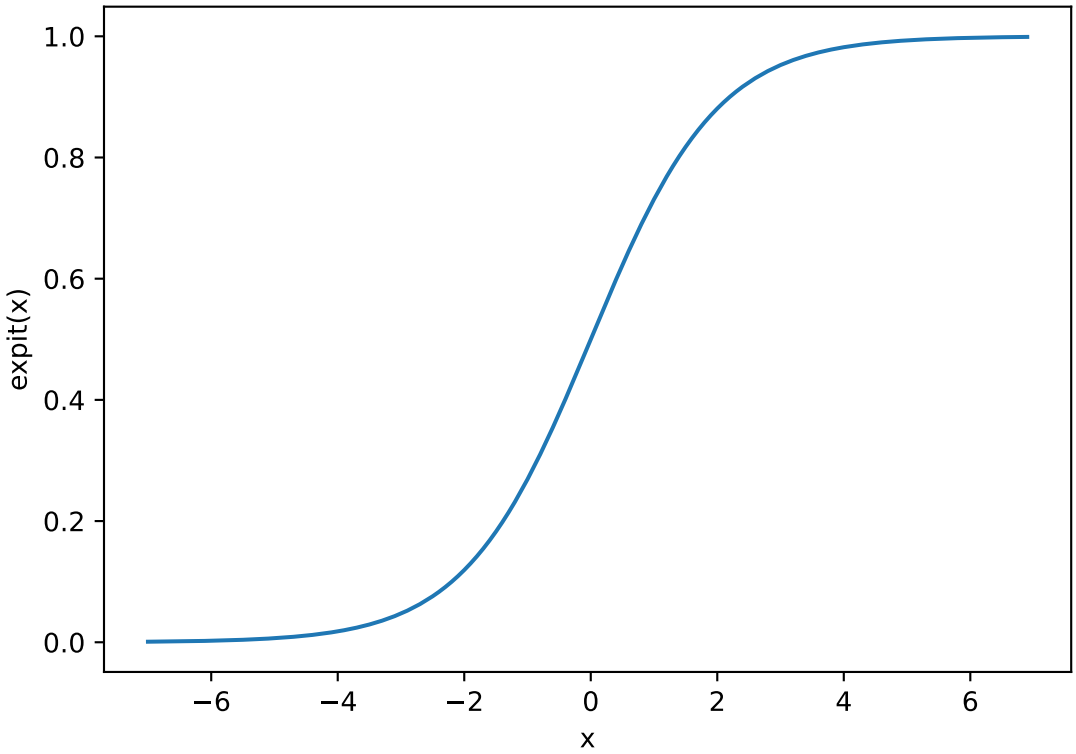
Plot of logistic function and inverse logistic

```
import matplotlib.pyplot as plt

p = np.arange(0.001, 0.999, 0.005)
fit = plt.plot(p, logit(p), label='logit(p)')
plt.xlabel('p')
plt.ylabel('logit(p)')
plt.show()
```




```
x = np.arange(-7, 7, 0.1)
fig = plt.plot(x, expit(x), label='inverse logit (expit)')
plt.xlabel('x')
plt.ylabel('expit(x)')
```




## 2.3 Odds-ratio

Interpretation of $\beta$:

What happens to odds, if we increase age by 1 year?

Consider the so-called **odds-ratio**:

$$\frac{\texttt{Odds}(y = 1 \,|\, x + 1)}{\texttt{Odds}(y = 1 \,|\, x)} = \frac{\exp(\alpha + \beta(x + 1))}{\exp(\alpha + \beta x)} = \exp(\beta)$$

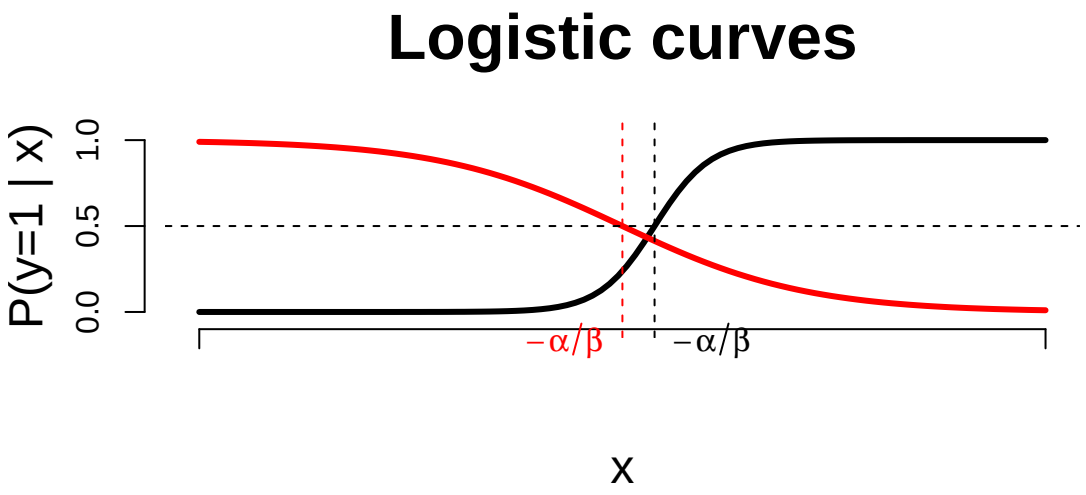
where we see, that $\exp(\beta)$ equals the odds for age $x + 1$ relative to odds at age $x$.

This means that when age increase by 1 year, then the relative change

$$\frac{\exp(\alpha + \beta(x + 1)) - \exp(\alpha + \beta x)}{\exp(\alpha + \beta x)}$$

in odds is given by $100(\exp(\beta) - 1)\%$.

## 2.4 Simple logistic regression




Examples of logistic curves for $P(y = 1|x)$. The black curve has a positive $\beta$-value (=10), whereas the red has a negative $\beta$ (=-3).

In addition we note that:

- Increasing the absolute value of $\beta$ yields a steeper curve.
- When $P(y = 1 \,|\, x) = \frac{1}{2}$ then logit is zero, i.e. $\alpha + \beta x = 0$.

This means that at age $x = -\frac{\alpha}{\beta}$ you have 50% chance to perform the task.

## 2.5 Example: Credit card data

We shall investigate if income is a good predictor of whether or not you have a credit card.

- Data structure: For each level of income, we let `n` denote the number of persons with that income, and `credit` how many of these that carries a credit card.

```
import pandas as pd

creInc = pd.read_csv("https://asta.math.aau.dk/datasets?file=income-credit.csv", sep=',')
creInc.head(6)
```

```
##    Income   n  credit
## 0      12   1       0
## 1      13   1       0
## 2      14   8       2
## 3      15  14       2
## 4      16   9       0
## 5      17   8       2
```

## 2.6 Example: Fitting the model

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

modelFit = smf.glm(formula='credit + I(n - credit) ~ Income',
                   data=creInc,
                   family=sm.families.Binomial()).fit()
modelFit.summary()
```

```
## <class 'statsmodels.iolib.summary.Summary'>
## """
##                      Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:     ['credit', 'I(n - credit)']   No. Observations:                   24
## Model:                                     GLM   Df Residuals:                       22
## Model Family:                         Binomial   Df Model:                           1
## Link Function:                           Logit   Scale:                         1.0000
## Method:                                   IRLS   Log-Likelihood:                -27.417
## Date:                         Mon, 03 Nov 2025   Deviance:                       39.276
## Time:                                 18:11:51   Pearson chi2:                     32.3
## No. Iterations:                              5   Pseudo R-squ. (CS):             0.6698
## Covariance Type:                     nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     -3.5179      0.710     -4.953      0.000      -4.910      -2.126
## Income         0.1054      0.026      4.030      0.000       0.054       0.157
## ==============================================================================
## """
```

- The response has the form `credit + I(n - credit)`.
- We need to use the function `glm` (generalized linear model).
- The argument `family=sm.families.Binomial()` tells the function that the data has binomial variation. Leaving out this argument will lead Python to believe that data follows a normal distribution (linear regression).
- The `params` extracts the coefficients (estimates of parameters) from the model:

```
modelFit.params
```

```
## Intercept   -3.517947
## Income       0.105409
## dtype: float64
```

## 2.7 Test of no effect

```
modelFit.summary2().tables[1]
```

```
##               Coef.  Std.Err.         z         P>|z|    [0.025    0.975]
## Intercept -3.517947  0.710336 -4.952513  7.326117e-07 -4.910179 -2.125714
## Income     0.105409  0.026157  4.029788  5.582714e-05  0.054141  0.156677
```

Our model for dependence of odds of having a credit card related to income($x$) is

$$\texttt{logit}(x) = \alpha + \beta x$$

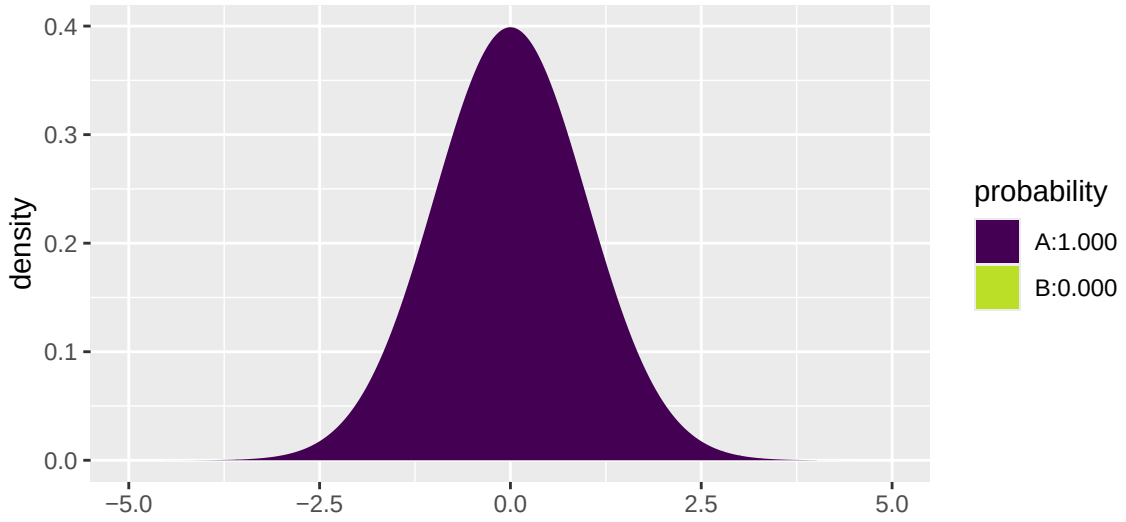
The hypothesis of no relation between income and ability to obtain a credit card corresponds to

$$H_0: \quad \beta = 0$$

with the alternative $\beta \neq 0$. Inspecting the summary reveals that $\hat{\beta} = 0.1054$ is more than 4 standard errors away from zero.

---

With a z-score equal to 4.03 we get the tail probability




```
from scipy.stats import norm

ptail = 2 * (1 - norm.cdf(4.03))
ptail
```

```
## np.float64(5.577685288105094e-05)
```

Which is very significant - as reflected by the p-value.

## 2.8 Confidence interval for odds ratio

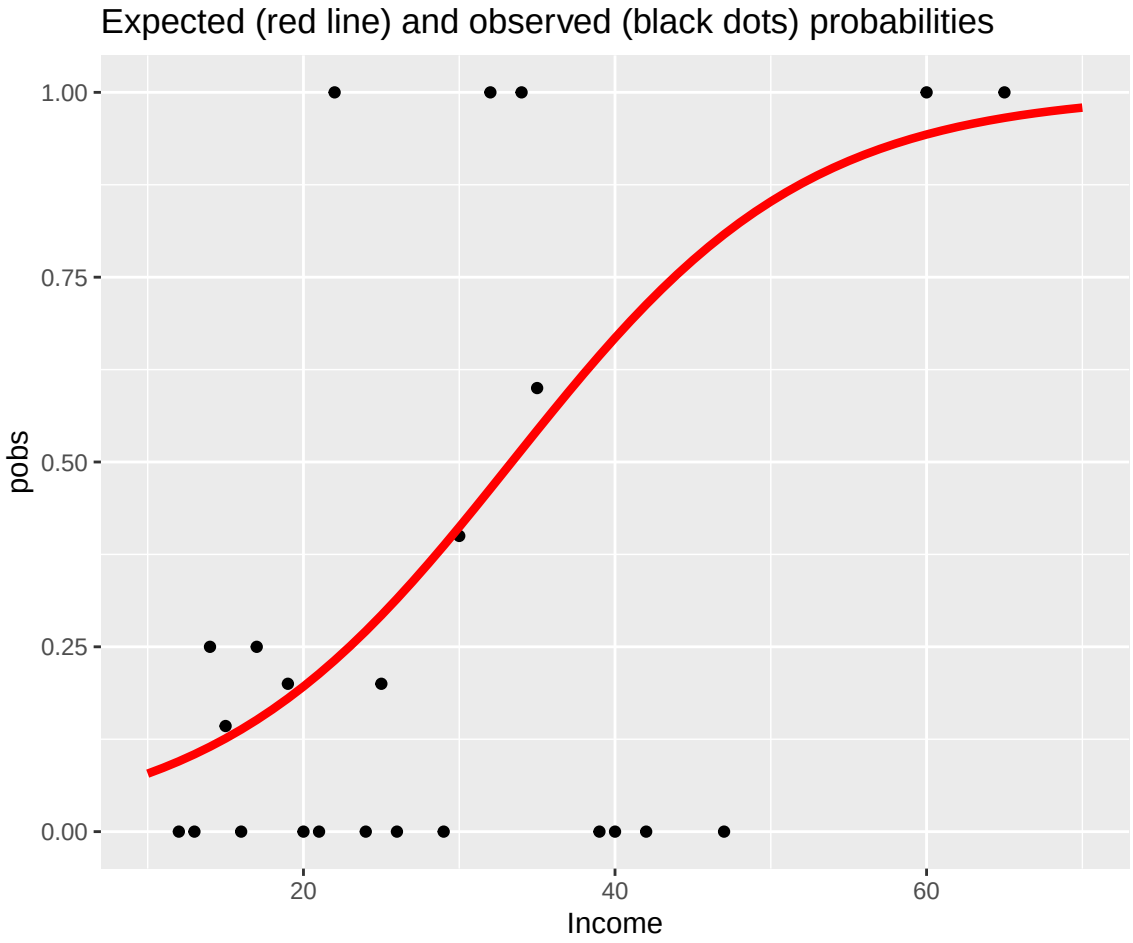
From the summary:

- $\hat{\beta} = 0.10541$ and hence $\exp(\hat{\beta}) - 1 = 0.11$. If income increases by 1000 euro, then odds increases by 11%.
- Standard error on $\hat{\beta}$ is 0.02616 and hence a 95% confidence interval for log-odds ratio is $\hat{\beta} \pm 1.96 \times 0.02616 = (0.054; 0,157)$.
- Corresponding interval for odds ratio: $\exp((0.054; 0,157)) = (1.056; 1.170)$,

  i.e. the increase in odds is - with confidence 95% - between 5.6% and 17%.

## 2.9 Plot of model predictions against actual data

```
## Ignoring unknown labels:
## * ylab : "Probability of credit card"
## * xlab : "Income"
```




- Tendency is fairly clear and very significant.
- Due to low sample size at some income levels, the deviations are quite large.

# 3 Multiple logistic regression

## 3.1 Several numeric predictors

We generalize the model to the case, where we have $k$ predictors $x_1, x_2, \ldots, x_k$. Where some might be dummies for a factor.

$$\texttt{logit}(P(y = 1 \,|\, x_1, x_2, \ldots, x_k)) = \alpha + \beta_1 x_1 + \cdots + \beta_k x_k$$

Interpretation of $\beta$-values is unaltered: If we fix $x_2, \ldots, x_k$ and increase $x_1$ by one unit, then the relative change in odds is given by $\exp(\beta_1) - 1$.

## 3.2 Example

Wisconsin Breast Cancer Database covers 683 observations of 10 variables in relation to examining tumors in the breast.

- Nine clinical variables with a score between 0 and 10.
- The binary variable `Class` with levels `benign/malignant`.
- By default `Python` orders the levels lexicografically and chooses the first level as reference ($y = 0$). Hence `benign` is reference, and we model odds of `malignant`.

We shall work with only 4 of the predictors, where two of these have been discretized.

---

```
BC = pd.read_csv("https://asta.math.aau.dk/datasets?file=BC0.dat", sep=' ')
BC.head(6)
```

```
##    nuclei  cromatin  Size.low  Size.medium  Shape.low      Class
## 0       1         3      True        False       True     benign
## 1      10         3     False         True      False     benign
## 2       2         3      True        False       True     benign
## 3       4         3     False        False      False     benign
## 4       1         3      True        False       True     benign
## 5      10         9     False        False      False  malignant
```

### 3.3 Global test of no effects

First we fit the model `mainEffects` with main effect of all predictors - remember the notation `~ .` for all predictors. Then we fit the model `noEffects` with no predictors.

```
BC['Class_numeric'] = (BC['Class'] == 'malignant').astype(int)
BC = BC.rename(columns={
    'Size.low': 'Size_low',
    'Size.medium': 'Size_medium',
    'Shape.low': 'Shape_low'
})
mainEffects = smf.glm('Class_numeric ~ nuclei + cromatin + Size_low + Size_medium + Shape_low', data=BC
noEffects = smf.glm('Class_numeric ~ 1', data=BC, family=sm.families.Binomial()).fit()
```

First we want to test, whether there is any effect of the predictors, i.e the null hypothesis

$$H_0: \quad \beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta_5 = 0$$

### 3.4 Example

We test the hypothesis that all $\beta_i = 0$ using a $\chi^2$-test.

```
from scipy.stats import chi2

ll_noEffects = noEffects.llf
ll_mainEffects = mainEffects.llf
test_stat = -2 * (ll_noEffects - ll_mainEffects)
df = mainEffects.df_model - noEffects.df_model
p_value = 1 - chi2.cdf(test_stat, df)
print(f"Test statistic: {test_stat}")
```

```
## Test statistic: 749.2859276261477
```

```
print(f"Degrees of freedom: {df}")
```

```
## Degrees of freedom: 5
```

```
print(f"P-value: {p_value}")
```

```
## P-value: 0.0
```

`mainEffects` is a much better model.

The test statistic is the Deviance (749.29), which should be small.

It is evaluated in a chi-square with 5 (the number of parameters equal to zero under the nul hypothesis) degrees of freedom.

The 95%-critical value for the $\chi^2(5)$ distribution is 11.07 and the p-value is in practice zero.

## 3.5 Test of influence of a given predictor

```
mainEffects.summary2().tables[1].round(4)
```

```
##                         Coef.  Std.Err.       z   P>|z|  [0.025  0.975]
## Intercept             -0.7090    0.8570 -0.8274  0.4080 -2.3887  0.9706
## Size_low[T.True]      -3.6154    0.8081 -4.4739  0.0000 -5.1992 -2.0315
## Size_medium[T.True]   -2.3773    0.7188 -3.3073  0.0009 -3.7861 -0.9685
## Shape_low[T.True]     -2.1490    0.6054 -3.5496  0.0004 -3.3356 -0.9624
## nuclei                 0.4403    0.0823  5.3483  0.0000  0.2790  0.6017
## cromatin               0.5058    0.1444  3.5025  0.0005  0.2228  0.7888
```

For each predictor $p$ can we test the hypothesis:

$$H_0: \quad \beta_p = 0$$

- Looking at the z-values, there is a clear effect of all 5 predictors. Which of course is also supported by the p-values.

## 3.6 Prediction and classification

```
BC['pred'] = mainEffects.predict()
```

- We add the column `pred` to our dataframe `BC`.
- `pred` is the final model's estimate of the probability of `malignant`.

```
BC[['Class', 'pred']].head(6)
```

```
##        Class      pred
## 0     benign  0.010817
## 1     benign  0.944507
## 2     benign  0.016702
## 3     benign  0.928883
## 4     benign  0.010817
## 5  malignant  0.999738
```

Not good for patients 2 and 4.

---

We may classify by `round(BC$pred)`:

- 0 to denote `benign` (probability `BC$pred` less than 0.5)
- 1 to denote `malignant` (probability `BC$pred` more than 0.5)

```
BC['pred_class'] = np.where(BC['pred'] > 0.5, "pred_malignant", "pred_benign")
tab = pd.crosstab(BC['Class'], BC['pred_class'])
tab
```

```
## pred_class  pred_benign  pred_malignant
## Class
## benign              433              11
## malignant            11             228
```

11+11=22 patients are misclassified.

---

```
malignant_preds = BC.loc[BC['Class'] == 'malignant', 'pred']
malignant_preds_sorted = malignant_preds.sort_values().head(5)
malignant_preds_sorted
```

```
## 440    0.034703
## 216    0.036964
## 63     0.088544
## 474    0.189870
## 55     0.205024
## Name: pred, dtype: float64
```

There is a malignant woman with a predicted probability of malignancy, which is only 3.5%.

If we assign all women with predicted probability of malignancy above 5% to further investigation, then we only miss two malignant.

```
BC['pred_class'] = np.where(BC['pred'] > 0.05, "pred_malignant", "pred_benign")
tab = pd.crosstab(BC['Class'], BC['pred_class'])
tab
```

```
## pred_class  pred_benign  pred_malignant
## Class
## benign              394              50
## malignant             2             237
```

The expense is that the number of false positive increases from 11 to 50.

---

```
BC['pred_class'] = np.where(BC['pred'] > 0.1, "pred_malignant", "pred_benign")
tab = pd.crosstab(BC['Class'], BC['pred_class'])
tab
```

```
## pred_class  pred_benign  pred_malignant
## Class
## benign              417              27
## malignant             3             236
```

- If we instead set the alarm to 10%, then the number of false positives decreases from 50 to 27.
- But at the expense of 3 false negative (instead of 2 as before).