Logistic Regression

07 March 2018 14:03

List of Topics covered:

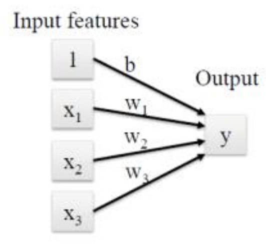
Theory:

Why Logistic Regression is called a regression?

- A linear regression function determines the strength of the relationship between a dependent variable (Y) and a set of independent variables (X1, X2, etc.,).
- The relationship is expressed as a linear weighted sum of input independent variables.
  Y = w0+w1X1

  W1 → slope or regression coefficient → indicates how much of a change/ impact will variation in X have on the observed Y

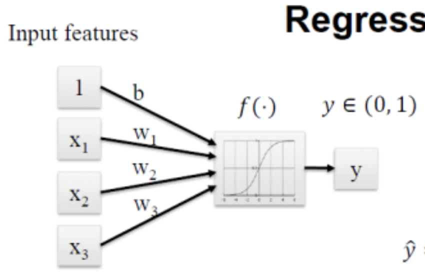
Linear Regression

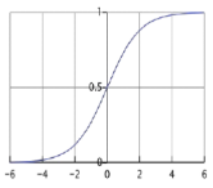


$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

- Yhat is continuous in Linear
- In Logistic Regression, the application of logistic function over the linear weighted sum of independent variables transforms the output Y into a categorical output.

Linear models for classification: Logistic Regression





$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

- Yhat is categorical in Logistic regression

In simple words, it predicts the probability of occurrence of an event by fitting weighted sum of independent variables to a logistic function. Hence, it is also known as **logistic regression**. Since, it predicts the probability, the output values lies between 0 and 1 (as expected).

In other words,
When a binary outcome variable is modeled using logistic regression, it is assumed that the logit transformation of the outco me variable has a linear relationship with the predictor variables

$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

Applying logit transformation on both sides,
Logit (yhat) = bhat + w1hat . x1 + …..+wnhat . xn

What is a Logistic Function?

- Logistic function is nothing but a sigmoid function of z

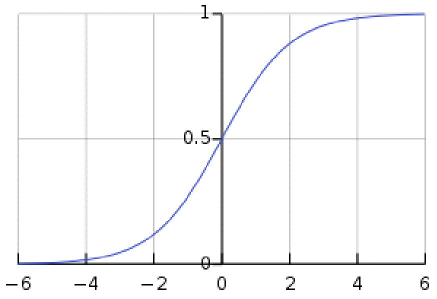
$$z = W_0 + W_1 Studied + W_2 Slept$$

$$P(class = 1) = \frac{1}{1 + e^{-z}}$$

$$S(z) = \frac{1}{1+e^{-z}}$$

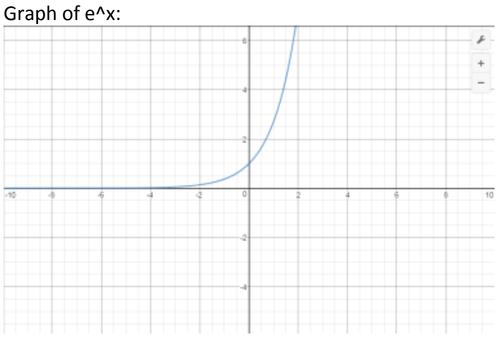
Note

- $s(z)$ = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithm's prediction e.g. mx + b)
- e = base of natural log

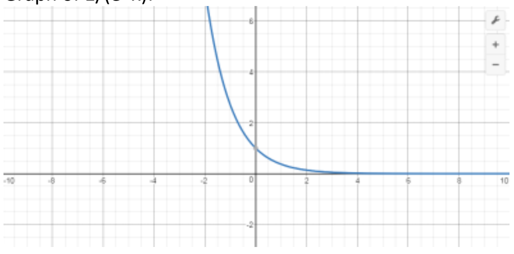


As z approaches negative infinity, S(z) approaches zero; As z approaches positive infinity, S(z) approaches 1

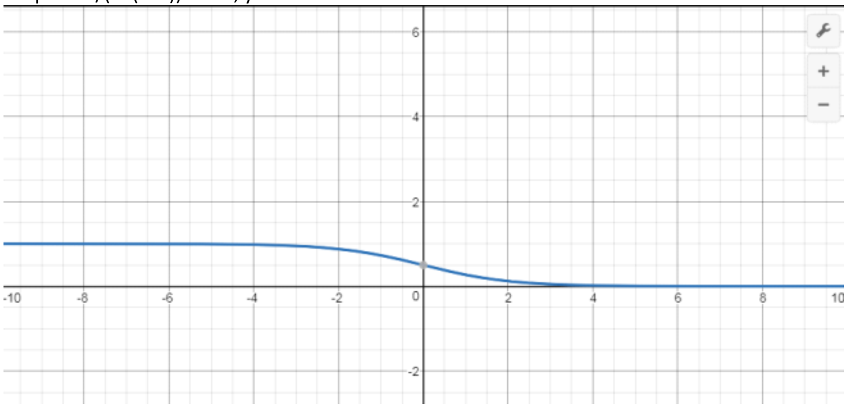
Graph of e^x:



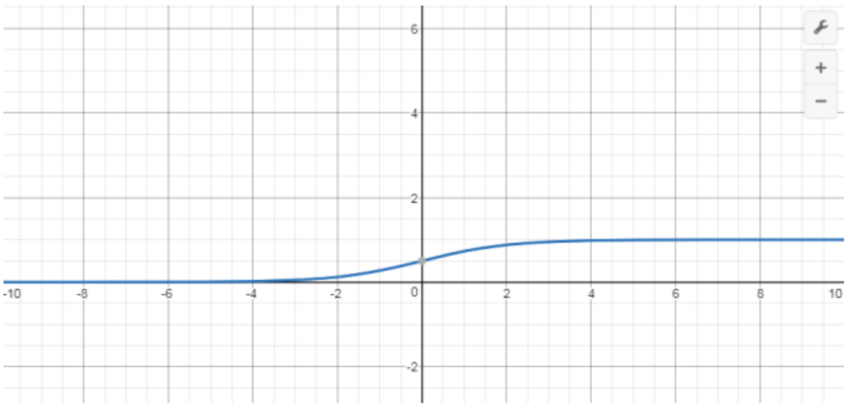
Graph of 1/(e^x):



Graph of 1/(1+(e^x)): Here, y is between 0 and 1

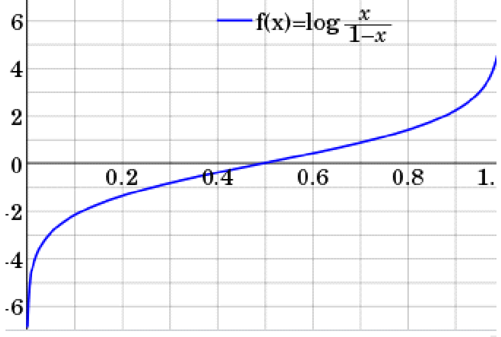


Graph of 1/(1+(e^-x)): Here, y is between 0 and 1 - a perfect sigmoidal function:



About Logit function:

the **logit function** gives the log-odds, or the logarithm of the odds p/(1 − p).



Plot of logit(p) in the domain of 0 to 1, where the base of logarithm is e

[illegible]

- Posterior probability = $\frac{\text{Prior probability x Likelihood}}{\text{Evidence}}$

dual predictors. To do so, they will want to examine the
ie criterion for each uni t change in the predictor

- $\Pr(y \mid X) = \frac{\Pr(y) \times \Pr(X \mid y)}{\Pr(X)}$

In logistic regression, the significance of coefficients are
assessed via likelihood ratio test

Likelihood function:
Probability of Evidence (X) given a hypothesis (y)

- Prior probabilities: Pr(y) for all y in Y
- Likelihood: Pr(x_i | y) for all features x_i and labels y in Y

In logistic regression, the regression coefficients ($\hat{\beta}_0, \hat{\beta}_1$) are calculated via the general method of maximum likelihood. For a simple logistic regression, the maximum likelihood function is given as

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})).$$

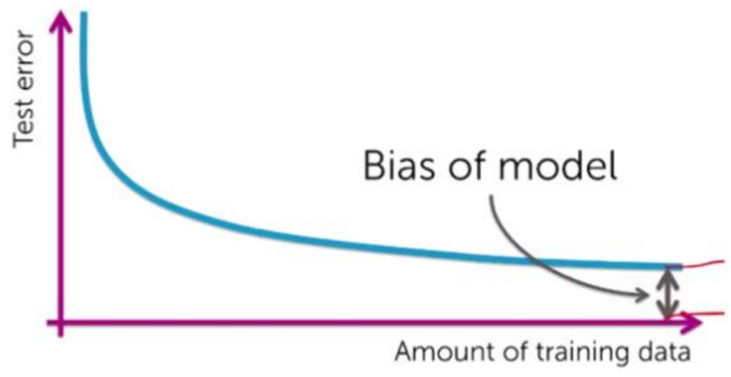
https://stats.stackexchange.com/questions/304988/understanding-the-logistic-regression-and-likelihood

In statistics, **maximum likelihood estimation** (**MLE**) is a method of estimating the parameters of a statistical model, given observations. MLE attempts to find the parameter values that maximize the likelihood function, given the observations.

Bias - Variance Trade Off

Learning curves

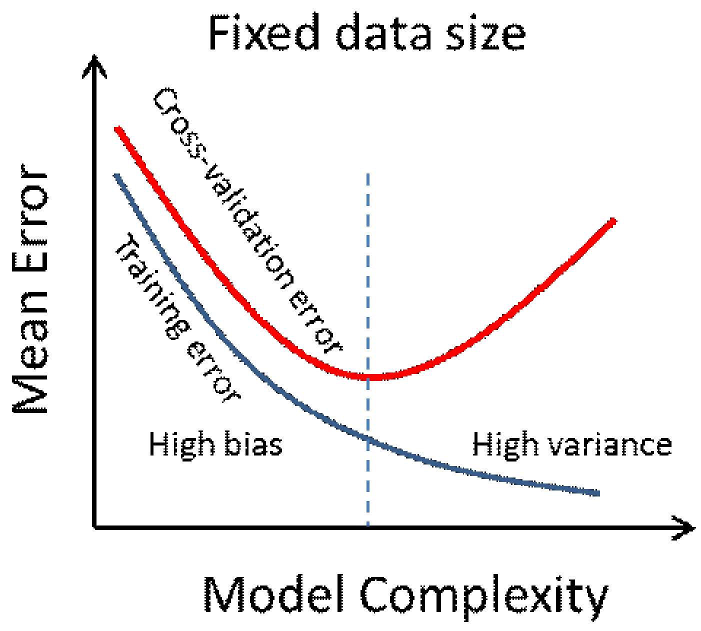
- More data means lower test error.



- Limit: even with infinite data test error will not go to 0. In other words, there will always be bias.
- The higher is the bias, the higher is the test error.

- The *bias* is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- The *variance* is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

- More complex models tend to have less bias



Regularization:
Penalty parameter that prevents overfitting by restricting the model complexity. Regularization parameter serves as **a degree of importance that is given to miss-classifications**

In Scikit-learn:

- **The strength of regularization is determined by C**
- **Larger values of C: less regularization**
  – *Fit the training data as well as possible*
  – *Each individual data point is important to classify correctly*
- **Smaller values of C: more regularization**
  – *More tolerant of errors on individual data points*

So, intuitively, as **C grows larger** the **less wrongly classified examples are allowed** (or the highest the price we pay in the loss function). Then when C tends to infinite the solution tends to the hard-margin (allow no miss-classification). When C tends to 0 (without being 0) the more the miss-classifications are allowed.

Cross Validation also prevents Overfitting:
In *k*-fold cross-validation, the original sample is randomly partitioned into *k* equal sized subsamples. Of the *k* subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated *k* times (the *folds*), with each of the *k* subsamples used exactly once as the validation data. The *k* results from the folds can then be averaged to produce a single estimation.

For example, setting *k = 2* results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d_0 and validate on d_1, followed by training on d_1 and validating on d_0.

Also in practice - Doing cross validation on the trained dataset only … saving test set for final model evaluation

- In practice:
  - Create an initial training/test split
  - Do cross-validation on the training data for model/parameter selection
  - Save the held-out test set for final model evaluation

Cost Function and Gradient Descent:

if Y being whether a student will pass or not,

$$z = W_0 + W_1 Studied + W_2 Slept$$

$$P(class = 1) = \frac{1}{1 + e^{-z}}$$

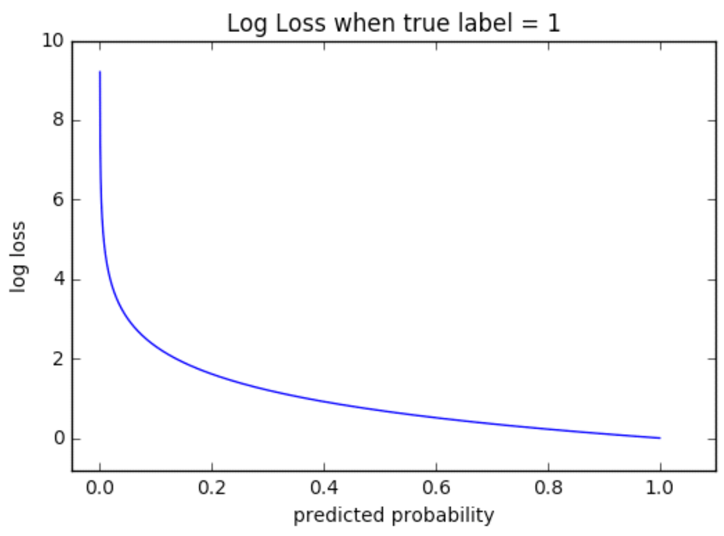
If the model returns .4 it believes there is only a 40% chance of passing. If our decision boundary was .5, we would categori ze this observation as "Fail.""

The **cost function** quantifies the error, as it compares the model's response with the correct answer. **Gradient descent** is a learning algorithm, that given the value of the cost/loss function finds new values for the parameters that result in a smaller error. **Descent** comes from **minimizing** the error. The **gradient** part refers to how the algorithm updates the model's parameters at each training step.

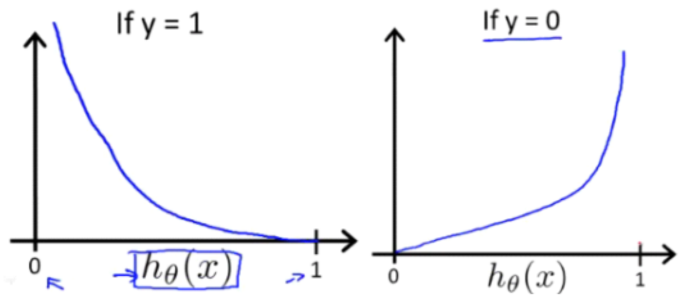
Similar to Mean Squared Error in Linear Regression,
Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 a nd 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual ob servation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \quad \text{if y} = 1$$
$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \quad \text{if y} = 0$$

h theta is predicted probability value (between 0 and 1) for the particular value of x



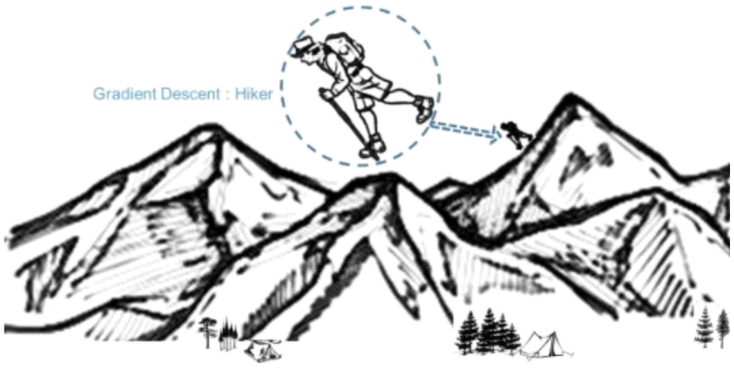
The benefits of taking the logarithm reveal themselves when you look at the cost function graphs for y=1 and y=0. These smooth monotonic functions [7] (always increasing or always decreasing) make it easy to calculate the gradient and minimize cost. Image from Andrew Ng's slides on logistic regression [1].



The key thing to note is the cost function penalizes confident and wrong predictions more than it rewards confident and right predictions! The corollary is increasing prediction accuracy (closer to 0 or 1) has diminishing returns on reducing cost due to the logistic nature of our cost function.

Gradient Descent Optimization and Trekking

Gradient descent works similar to a hiker walking down a hilly terrain. Essentially, this t
loss functions for machine learning algorithms. The idea with the ML algorithms, as alre
to the bottom-most or minimum error point by changing ML coefficients β_0 , β_1 and β_2 .
spread across the x, and y-axis on the physical world. Similarly, the error function is sp
of machine learning algorithm i.e. β_0 , β_1 and β_2 .



The idea is to find the values of the coefficients such that the error becomes minimum.

Gradient Descent and Differential Calculus

The mathematical equivalent of human ability to identify downward slope is differentiation of a function also called gradients. The differentiation of the landscape Captain Kirk was walking is:

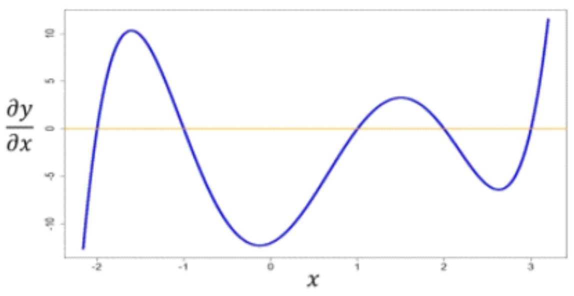
$$\frac{\partial y}{\partial x} = x^5 - 3x^4 - 5x^3 + 15x^2 + 4x - 12$$

Now, if you insert x=3.1 in this equation you will get the $\frac{\partial y}{\partial x}$ as 4.83. The positive value indicates it's an upslope ahead. This mean Captain Kirk, or the pointer for gradient descent algorithm, needs to walk backward or toward the lower values of x. Similarly, at x=2.9, the $\frac{\partial y}{\partial x}$ = -3.27. The -ve value means it is a downslope ahead, and Captain Kirk will walk forward or towards the higher values of x.

At x=3, the global minima, $\frac{\partial y}{\partial x}$ =0. This means there is no slope or you have reached a flat plane. As it turns out, the above polynomial equation can be simplified to this. It was a trick question from the alien.

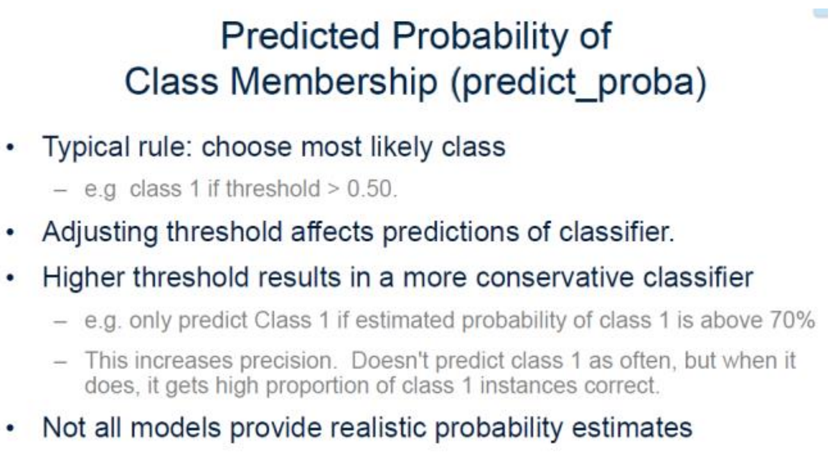
$$\frac{\partial y}{\partial x} = (x-1)(x+1)(x-2)(x+2)(x-3)$$

We can find minimum values of this function without gradient descent by equating this equation to 0. $\frac{\partial y}{\partial x}$ =0 for, x = -2, -1, 1, 2, and 3. So, if Captain Kirk knew some math he could have avoided all the walking.



Let's use this knowledge to find the machine learning prediction solution to our market research data.

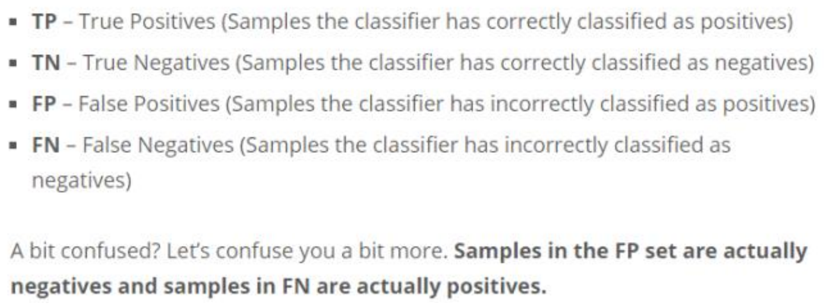
Decision functions:



Predicted Probability of Class Membership (predict_proba)

- Typical rule: choose most likely class
  - e.g class 1 if threshold > 0.50.
- Adjusting threshold affects predictions of classifier.
- Higher threshold results in a more conservative classifier
  - e.g. only predict Class 1 if estimated probability of class 1 is above 70%
  - This increases precision. Doesn't predict class 1 as often, but when it does, it gets high proportion of class 1 instances correct.
- Not all models provide realistic probability estimates

Classification Accuracy Metrics:



- **TP** – True Positives (Samples the classifier has correctly classified as positives)
- **TN** – True Negatives (Samples the classifier has correctly classified as negatives)
- **FP** – False Positives (Samples the classifier has incorrectly classified as positives)
- **FN** – False Negatives (Samples the classifier has incorrectly classified as negatives)

A bit confused? Let's confuse you a bit more. **Samples in the FP set are actually negatives and samples in FN are actually positives.**

Accuracy: Overall, how often is the classifier correct? (TP+TN)/total
Misclassification Rate: Overall, how often is it wrong? (FP+FN)/total
equivalent to 1 minus Accuracy
also known as "**Error Rate**"

True Positive Rate: When it's actually yes, how often does it predict yes? TP/actual yes = TP/(TP+FN)
also known as "**Sensitivity**" or "**Recall**"

True Negative Rate: also known as "**Specificity**" : When it's actually No, how often does it predict No?
TN/actual no = TN/(TN+FP)

False Positive Rate: When it's actually No, how often does it predict yes? FP/actual No = FP/(TN+FP)

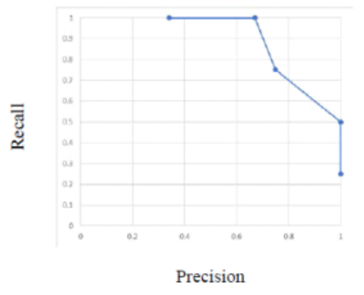
1-Specificity = False Positive Rate
Precision: Out of the total yes predicted, how many are predicted right? TP/(TP+FP)

F1 Score: Harmonic mean of Precision and Recall = 2 *Precision* Recall / (Precision + Recall)

Varying the Decision Threshold

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

Classifier score threshold	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25

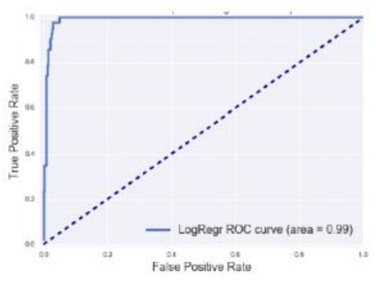


ROC Curves

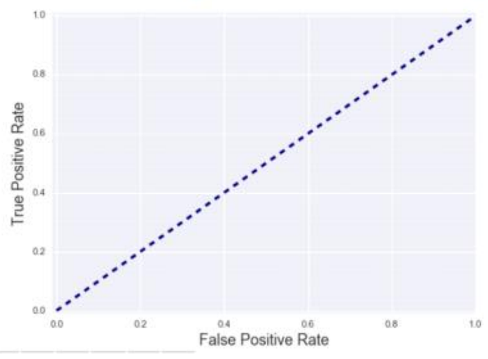
X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- **The "ideal" point**
- **False positive rate of zero**
- **True positive rate of one**

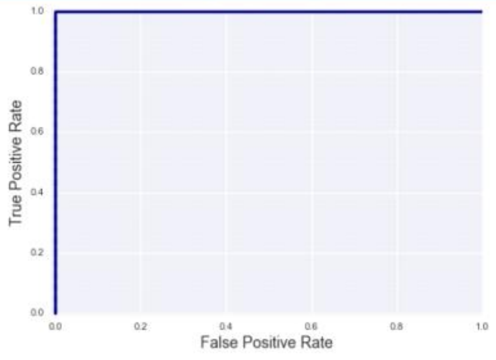
"Steepness" of ROC curves is important:
- **Maximize the true positive rate**
- **while minimizing the false positive rate**



ROC curve examples: random guessing



ROC curve examples: perfect classifier



Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems.

The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random

ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

A commonly used graph that summarizes the performance of a classifier over all possible thresholds. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class

Metrics for Multiclass Classification:
Accuracy = #correctly_classified_samples/#all_samples

The solution is to **reduce a multiclass classification problem to many binary classification problems**. If we have **K** classes, we deal with **K binary classification problems**. We consider each class to be the positive one and the rest of the classes the negative one. Let's build a utility function that computes **TP, TN, FP and FN** since we'll need these values later on.

- **macro**: the averaged per class precisions
- **weighted**: similar to macro but weighted by the number of samples for each class
- **micro**: `GLOBAL_TP / (GLOBAL_TP + GLOBAL_FP)`

1. Micro-average Method

In Micro-average method, you sum up the individual true positives, false positives, and false negatives of the system for different sets and the apply them to get the statistics. For example, for a set of data, the system's

```
True positive (TP1)  = 12
False positive (FP1) = 9
False negative (FN1) = 3
```

Then precision (P1) and recall (R1) will be $57.14\% = \frac{TP1}{TP1+FP1}$ and $80\% = \frac{TP1}{TP1+FN1}$

and for a different set of data, the system's

```
True positive (TP2)  = 50
False positive (FP2) = 23
False negative (FN2) = 9
```

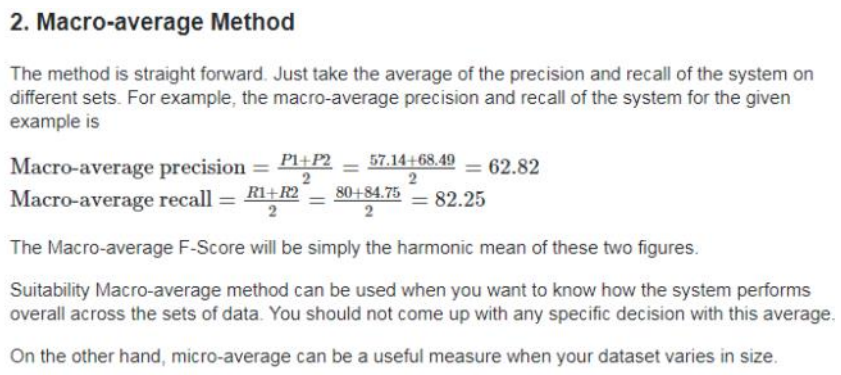
Then precision (P2) and recall (R2) will be 68.49 and 84.75

Now, the average precision and recall of the system using the Micro-average method is

$$\text{Micro-average of precision} = \frac{TP1+TP2}{TP1+TP2+FP1+FP2} = \frac{12+50}{12+50+9+23} = 65.96$$

$$\text{Micro-average of recall} = \frac{TP1+TP2}{TP1+TP2+FN1+FN2} = \frac{12+50}{12+50+3+9} = 83.78$$

The Micro-average F-Score will be simply the harmonic mean of these two figures.



2. Macro-average Method

The method is straight forward. Just take the average of the precision and recall of the system on different sets. For example, the macro-average precision and recall of the system for the given example is

$$\text{Macro-average precision} = \frac{P1+P2}{2} = \frac{57.14+68.49}{2} = 62.82$$
$$\text{Macro-average recall} = \frac{R1+R2}{2} = \frac{80+84.75}{2} = 82.25$$

The Macro-average F-Score will be simply the harmonic mean of these two figures.

Suitability Macro-average method can be used when you want to know how the system performs overall across the sets of data. You should not come up with any specific decision with this average.

On the other hand, micro-average can be a useful measure when your dataset varies in size.

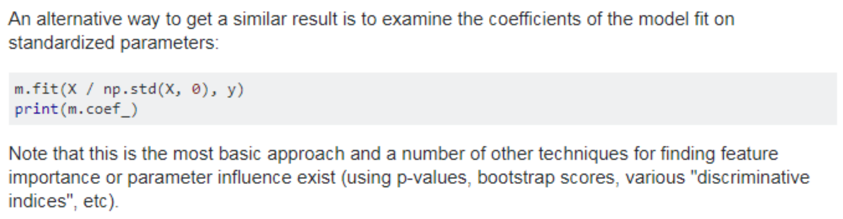
A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric. **In a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance (i.e you may have many more examples of one class than of other classes)**.

Conclusion on Classification Metrics:
The F1-score is considered the most *"complete"* score, being a combination of precision and recall. **It is also the least intuitive one.**

How to find the importance of Predictors in a Logistic Regression?

By Coefficient method:

- Approximately put - computing the magnitude of the coefficient of the predictor variable gives the importance of that variable to the model
- Accurately put - Coefficient of the variable * Std deviation of the variable (to give a normalized score across all variables)



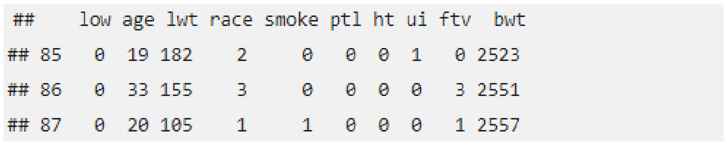
An alternative way to get a similar result is to examine the coefficients of the model fit on standardized parameters:

```
m.fit(X / np.std(X, 0), y)
print(m.coef_)
```

Note that this is the most basic approach and a number of other techniques for finding feature importance or parameter influence exist (using p-values, bootstrap scores, various "discriminative indices", etc).

Other methods by which importance of a predictor can be analyzed:
https://www.mwsug.org/proceedings/2009/stats/MWSUG-2009-D10.pdf

Interpreting the Logistic Regression Summary Output in R:
https://analyticsdataexploration.com/logistic-regression-output-interpretation-in-r/



```
##    low age lwt race smoke ptl ht ui ftv  bwt
## 85   0  19 182    2     0   0  0  1   0 2523
## 86   0  33 155    3     0   0  0  0   3 2551
## 87   0  20 105    1     1   0  0  0   1 2557
```

Low = 0 if birthweight > 2.5Kg
Low = 1 if birthweight <2.5Kg

```
## Call:
## glm(formula = low ~ age + lwt, family = binomial(link = "logit"),
##     data = DataFrame)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1352  -0.9088  -0.7480   1.3392   2.0595
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.748773   0.997097   1.754   0.0795 .
## age         -0.039788   0.032287  -1.232   0.2178
## lwt         -0.012775   0.006211  -2.057   0.0397 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 234.67  on 188  degrees of freedom
## Residual deviance: 227.12  on 186  degrees of freedom
## AIC: 233.12
##
## Number of Fisher Scoring iterations: 4
```

- **Intercept Coefficients interpretation (b0, b1 and b2)**

1. Intercept Coefficient(b0)=1.748773

2. lwt coefficient(b1) =-0.012775

Interpretation: The increase in logit score per unit increase in weight(lwt) is -0.012775

age coefficient(b2) =-0.039788

Interpretation: The increase in logit score per unit increase in age is -0.039788

- **p-value interpretation**

3. p-value for lwt variable=0.0397

Interpretation: According to z-test,p-value is 0.0397 which is comparatively low which implies its unlikely that there is "no relation" between lwt and target variable i.e low.Star next to p-value in the summary shows that lwt is significant variable in predicting low variable.

4. p-value for age=0.2178

Interpretation: According to z-test,p-value is 0.2178 which is comparatively high which implies its unlikely that there is "any relation" between age and target variable i.e low.

- **Logit score Calculation**

5. Let's consider a random person with age =25 and lwt=55.Now let's find the logit score for this person
b0 + b1*x1 + b2*x2= 1.748773-0.039788*25-0.012775*55=0.05144(approx).

6. So logit score for this observation=0.05144

- **Odds ratio calculation**

7. Now let's find the probability that birthwt <2.5 kg(i.e low=1).See the help page on birthwt data set (type ?birthwt in the console)

8. Odds value=exp(0.05144) =1.052786

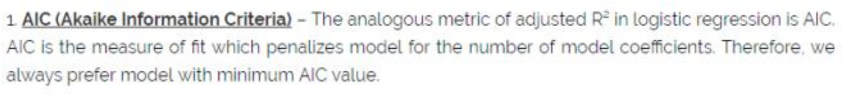
- **Probability Calculation**

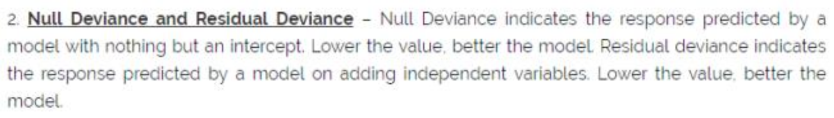
9. probability(p) = odds value / odds value + 1
p=1.052786/2.052786=0.513(approx.)

p=0.513

Interpretation

0.513 or 51.3% is the probability of birth weight less than 2.5 kg when the mother age =25 and mother's weight(in pounds)=55


1. **AIC (Akaike Information Criteria)** – The analogous metric of adjusted R^2 in logistic regression is AIC. AIC is the measure of fit which penalizes model for the number of model coefficients. Therefore, we always prefer model with minimum AIC value.


2. **Null Deviance and Residual Deviance** – Null Deviance indicates the response predicted by a model with nothing but an intercept. Lower the value, better the model. Residual deviance indicates the response predicted by a model on adding independent variables. Lower the value, better the model.

3. Number of Fisher Scoring iterations is a derivative of Newton-Raphson algorithm which proposes how the model was estimated. In your case, it can be interpreted as, Fisher scoring algorithm took 4 iterations to perform the fit. This metric doesn't tell you anything which you must know. It just confirms the model convergence. That's it

Boosting:
Used in Ensemble learning
Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance[1] in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones.

Steps in performing a Logistic Regression:
1. Identifying the independent variables and one dependent Variable [if there is more than one dependent variable, it becomes multi-label classification
2. Identifying and imputing Missing and Outlier Values appropriately (impute via mean, median, mode, etc.,)
3. Encode categorical data into binary ones or zero values; Avoid Multicollinearity
   (deviation: categorical variable is otherwise known as nominal variable; note: there is no order to the categories in a data; If ordered, it becomes ordinal variable)
4. Create the Logistic Classifier Object (with the needed parameters) - More depth later on how to tune model parameters
5. Train on the Training dataset (or complete input dataset)
6. Predict on test/validation dataset (or unlabelled new dataset)
7. Evaluate the accuracy of the model - More depth later on how to tune model parameters
   - ☐ Sensitivity/Recall/TPR, Specificity, Precision, Precision-Recall curve (F-score), AUC (FPR (X-axis) vs TPR (Y-axis)) area under the curve graph
   - ☐ Confusion Matrix
   - ☐ Decision Functions (predicted Probabilities)
   - ☐ Cross Validation (GridSearch CV, Randomized Search CV)

8. Optimize/ Tuning the model
   Cost/Loss Function and Gradient Descent to minimize the Cost Function

Source Links:

Logistic Regression - Theory Basics (with Python Codes to build from scratch):
https://www.analyticsvidhya.com/blog/2015/10/basics-logistic-regression/

Logistic Regression Detailed theory:

http://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
https://github.com/bfortuner/ml-cheatsheet/blob/master/docs/logistic_regression.rst

building-a-logistic-regression-in-python-step-by-step using Scikit Learn
https://datascienceplus.com/building-a-logistic-regression-in-python-step-by-step/
https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Logistic%20Regression%20in%20Python%20-%20Step%20by%20Step.ipynb

Implementing Logistic Regression and Stochastic Gradient from Scratch
https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/

https://www.quora.com/What-are-gradient-descent-and-cost-function-in-logistic-regression

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Multiclass Classification Model Metrics:
https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin/16001#16001
http://nlpforhackers.io/classification-performance-metrics/

Gradient Descent - Logistic Regression
http://ucanalytics.com/blogs/gradient-descent-logistic-regression-simplified-step-step-visual-guide/

How to interpret odds ratio in Logistic Regression:
https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faq-how-do-i-interpret-odds-ratios-in-logistic-regression/

Logistic Regression - Interpretation of the summary output:
https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/

A small detour:

What does the base of natural log e denote?
e is a mathematical constant, approximately equal to **2.71828**

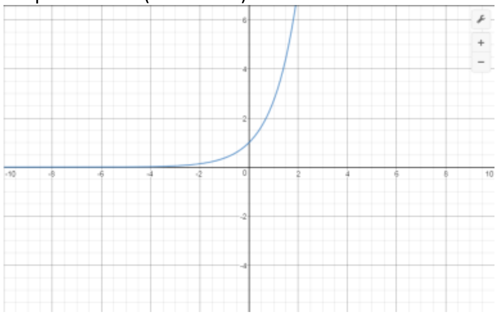
$Natural Logarithm$

$log_e(x) = ln(x)$

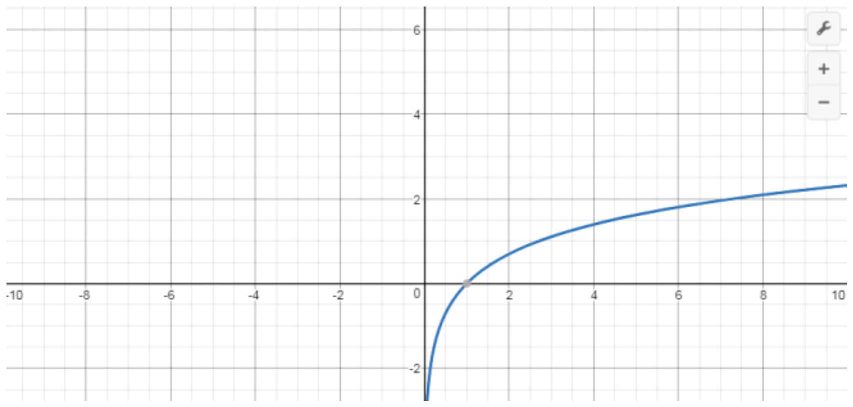
In particular, ln, which is $\log_e$; and using log for $\log_{10}$, we have these properties:

$$\begin{aligned} \log(xy) &= \log(x) + \log(y) & \ln(xy) &= \ln(x) + \ln(y) \\ \log\left(\frac{x}{y}\right) &= \log(x) - \log(y) & \ln\left(\frac{x}{y}\right) &= \ln(x) - \ln(y) \\ \log(x^a) &= a\log(x) & \ln(x^a) &= a\ln(x) \\ \log(10^x) &= x & \ln(e^x) &= x \\ 10^{\log(x)} &= x & e^{\ln(x)} &= x \end{aligned}$$

Graph of ln(x): Simple Logic y=ln(x) ==> e^y = x; y the inverse function of e^x
Graph of e^x or (2.71828^x)



For ln(x), Just interchange x and y axis, since it is the inverse of the above graph
Hence, contrary to the above chart, y axis will have negative values and x axis will have values tending to infinity


6
4
2
0
-2
-10
-8
-6
-4
-2
2
4
6
8
10