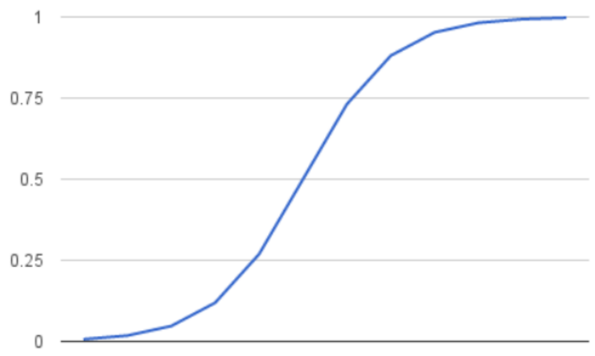
Logistic Regression

Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.



Logistic Function

Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

Logistic Regression Predicts Probabilities

Logistic regression models the probability of the default class (e.g. the first class).For example, if we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

$$P(sex=male|height)$$

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

$$P(X) = P(Y=1|X)$$

Note that the probability prediction must be transformed into a binary values (0 or 1) in order to actually make a probability prediction. More on this later when we talk about making predictions.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

$$p(X) = e^(b0 + b1*X) / (1 + e^(b0 + b1*X))$$

I don't want to dive into the math too much, but we can turn around the above equation as follows (remember we can remove the e from one side by adding a natural logarithm (ln) to the other):

$$ln(p(X) / 1 – p(X)) = b0 + b1 * X$$

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class (it's historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So we could instead write:

$$\ln(odds) = b0 + b1 * X$$

Because the odds are log transformed, we call this left hand side the log-odds or the probit. It is possible to use other types of functions for the transform (which is out of scope_, but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit link function.

We can move the exponent back to the right and write it as:

$$odds = e^{(b0 + b1 * X)}$$

All of this helps us understand that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

Let’s make this concrete with a specific example.

Let’s say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.

We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150). We will use EXP() for e, because that is what you can use if you type this example into your spreadsheet:

$$y = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$$

$$y = exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))$$

$$y = 0.0000453978687$$

Or a probability of near zero that the person is a male.

In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

0 if p(male) < 0.5

1 if p(male) >= 0.5

Now that we know how to make predictions using logistic regression, let's look at how we can prepare our data to get the most from the technique.

Implementing Logistic Regression with Python

For implementing Logistic Regression we will be working with a fake advertising data set, indicating whether or not a particular internet user clicked on an Advertisement. We will try to create a model that will predict whether or not they will click on an ad based off the features of that user.

This data set contains the following features:

- 'Daily Time Spent on Site': consumer time on site in minutes
- 'Age': customer age in years
- 'Area Income': Avg. Income of geographical area of consumer
- 'Daily Internet Usage': Avg. minutes a day consumer is on the internet
- 'Ad Topic Line': Headline of the advertisement
- 'City': City of consumer
- 'Male': Whether or not consumer was male
- 'Country': Country of consumer
- 'Timestamp': Time at which consumer clicked on Ad or closed window
- 'Clicked on Ad': 0 or 1 indicated clicking on Ad

Import Libraries:

```
import pandas as pd
import numpy as numpy
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data:

```
ad_data=pd.read_csv('advertising.csv')
```

```
ad_data.head()
```

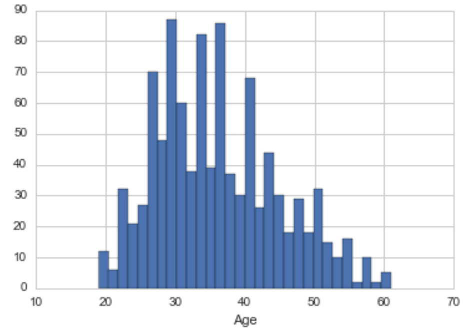
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

Exploratory Data Analysis

Let's use seaborn to explore the data!

```
ad_data['Age'].plot.hist(bins=30)
plt.xlabel('Age')
```

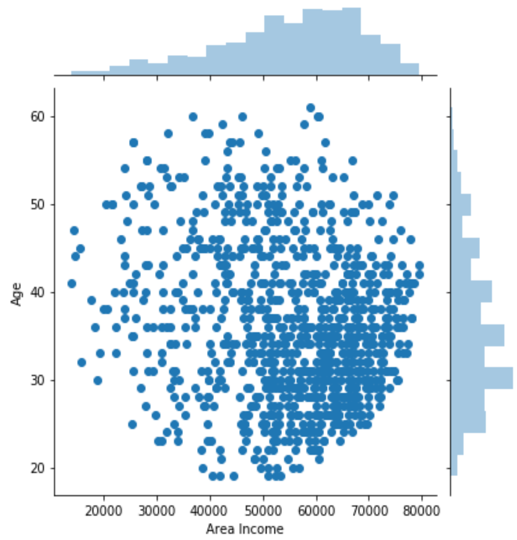
```
<matplotlib.text.Text at 0x11a05b908>
```



Create a jointplot showing Area Income versus Age

```
sns.jointplot(x='Area Income',y='Age',data=ad_data)
```

```
<seaborn.axisgrid.JointGrid at 0x1f1b2739c48>
```



Now it's time to do a train test split, and train our model!

Split the data into training set and testing set using train_test_split

```
from sklearn.model_selection import train_test_split
```

```
X=ad_data[['Daily Time Spent on Site', 'Age', 'Area Income',
         'Daily Internet Usage','Male']]
y=ad_data['Clicked on Ad']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=101)
```

** Train and fit a logistic regression model on the training set.**

```
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression()
```

```
lm.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Predictions and Evaluations

```
predictions=lm.predict(X_test)
```

** Create a classification report for the model.**

```
from sklearn.metrics import  classification_report
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       171
           1       0.99      0.97      0.98       159

    accuracy                           0.98       330
   macro avg       0.98      0.98      0.98       330
weighted avg       0.98      0.98      0.98       330
```