Python programming and data analysis

Lecture 6

Introduction to Machine Learning

Robert Szmurło

e-mail: robert.szmurlo@ee.pw.edu.pl 2018Z

Lecture outline

1. Introduction to Machine Learning
2. Simple Python perceptron class implementaion for binary classification
3. Linear and polynomial regression with SciKit Learn
4. Logistics regression with SciKit Learn
5. Titanic Example

Next lecture

- Continue Introduction to Machine Learning with Python

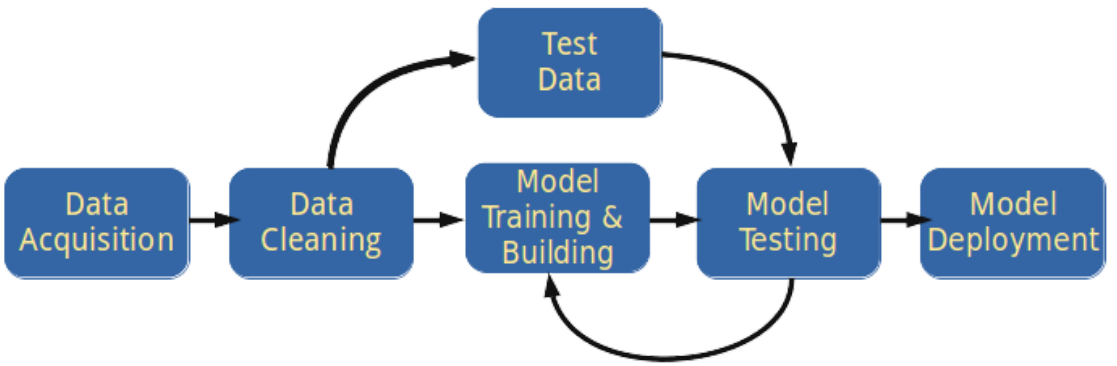
Introduction to Machine Learning

https://en.wikipedia.org/wiki/Machine_learning

Machine learning (ML) gives computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed by using statistical techniques.

Algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions

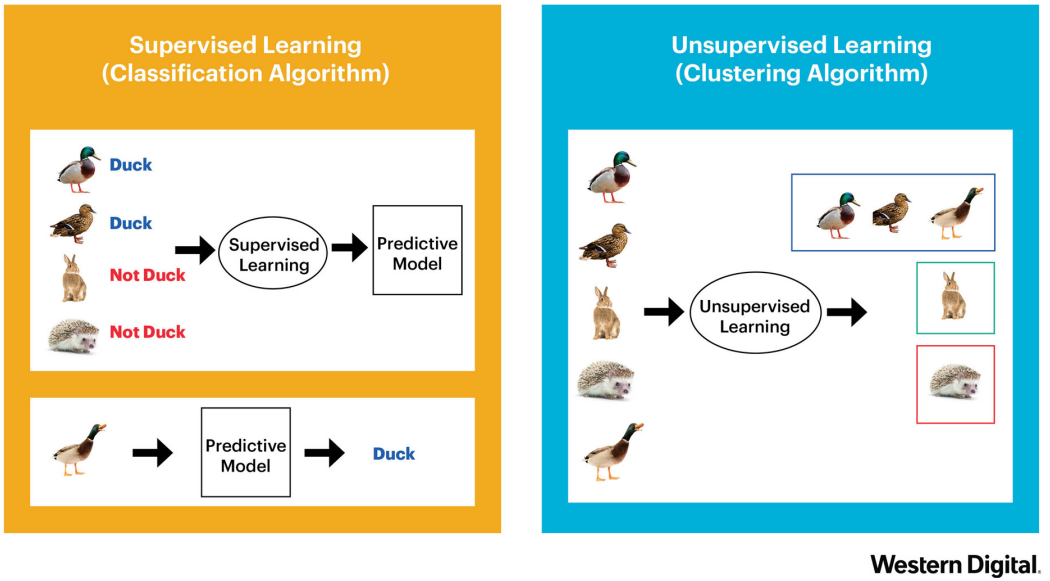
- Train and Test iteratively changing parametres to tune the model to test data set



3 types of machine learning

https://blog.westerndigital.com/machine-learning-pipeline-object-storage/supervised-learning-diagram/

- Labeled data - **supervised learning**
- Unlabelled dataset - **unsupervised learning**



Supervised learning

https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

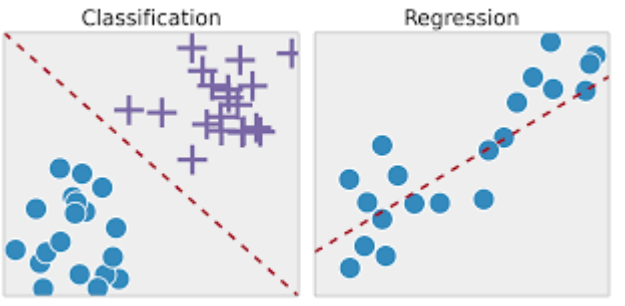
The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Recap:

- labeled data - predict a label based on a set of features
- trained on labeled exmaples (desired output is known)
- its goal is to infer the natural structure present within a set of data points

Supervised learning - continued

https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d



Supervised learning problems can be further grouped into regression and classification problems.

- **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively.

Some popular examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems.

Unsupervised Machine Learning

Unsupervised learning is where you only have input data (`x`) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are:

- k-means for clustering problems.

Semi-Supervised Machine Learning

Problems where you have a large amount of input data (X) and only some of the data is labeled (Y) are called semi-supervised learning problems.

These problems sit in between both supervised and unsupervised learning.

A good example is a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

Many real world machine learning problems fall into this area. This is because it can be expensive or time-consuming to label data as it may require access to domain experts. Whereas unlabeled data is cheap and easy to collect and store.

You can use unsupervised learning techniques to discover and learn the structure in the input variables.

You can also use supervised learning techniques to make best guess predictions for the unlabeled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.

Recap

- Supervised: All data is labeled and the algorithms learn to predict the output from the input data.
- Unsupervised: All data is unlabeled and the algorithms learn to inherent structure from the input data.
- Semi-supervised: Some data is labeled but most of it is unlabeled and a mixture of supervised and unsupervised techniques can be used. Do you have any questions about supervised, unsupervised or semi-supervised learning? Leave a comment and ask your question and I will do my best to answer it.

Three main issues

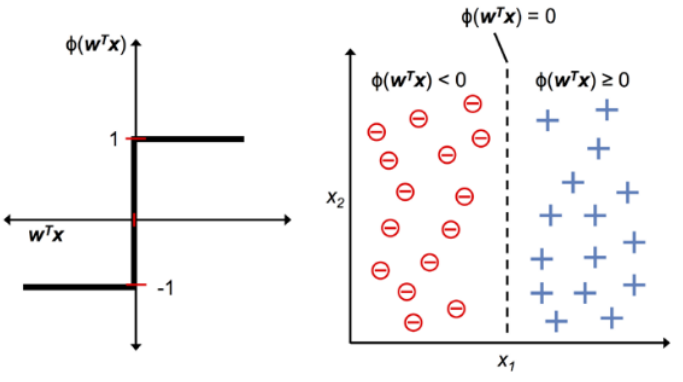
http://people.csail.mit.edu/torralba/shortCourseRLOC/slides/introduction.ppt

- Representation
  - How to represent an object category
- Learning
  - How to form the classifier, given training data
- Recognition
  - How the classifier is to be used on novel data

The formal definition of an artificial neuron

S.Raschka, V.Mirjalili, Python Machine Learning, Machine learning and deep learning with Python

- We can put the idea behind artificial neurons into the context of a binary classification task where we refer to our two classes as 1 (positive class) and -1 (negative class).
- A decision function (φ(z)) that takes a linear combination of certain input values x and a corresponding weight vector w, where z is the so-called net input $z = w_1x_1 + \ldots + w_mx_m$.



$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

- In the perceptron algorithm, the decision function φ(·) is a variant of a unit step function:

$$\phi(z) = \begin{cases} 1 & \text{if} \quad z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

The perceptron learning rule

The whole idea behind the Rosenblatt's thresholded perceptron model is to use a reductionist approach to mimic how a single neuron in the brain works: it either fires or it doesn't. Thus, Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $x^{(i)}$: a. Compute the output value `\(\hat{y}\)` . b. Update the weights.

$$w_j := w_j + \Delta w_j$$

The value of Δw_j , which is used to update the weight w_j , is calculated by the perceptron learning rule:

$$\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$$

Where η is the learning rate (typically a constant between 0.0 and 1.0), y is the true class label of the ith training sample, and $\hat{y}$`^{(i)}` is the predicted class label. It is important to note that all weights in the weight vector are being updated simultaneously, which means that we don't recompute the $\hat{y}$`^{(i)}` before all of the weights Δw_j are updated.

The convergence of the perceptron

It is important to note that the convergence of the perceptron is only guaranteed if the two classes are linearly separable and the learning rate is sufficiently small.

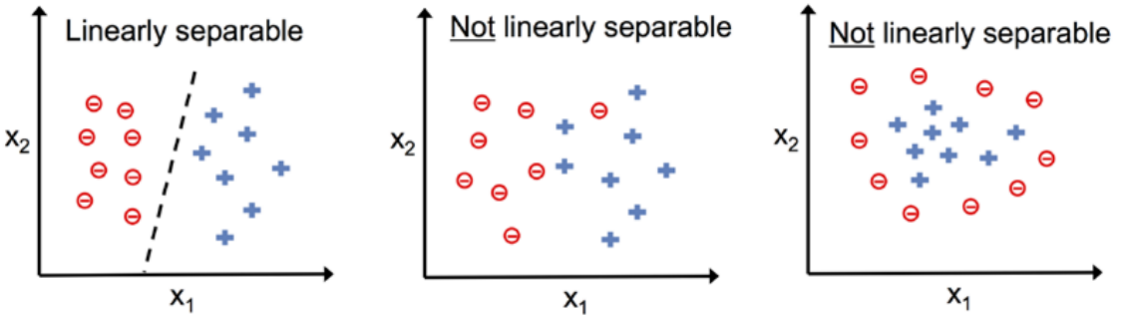
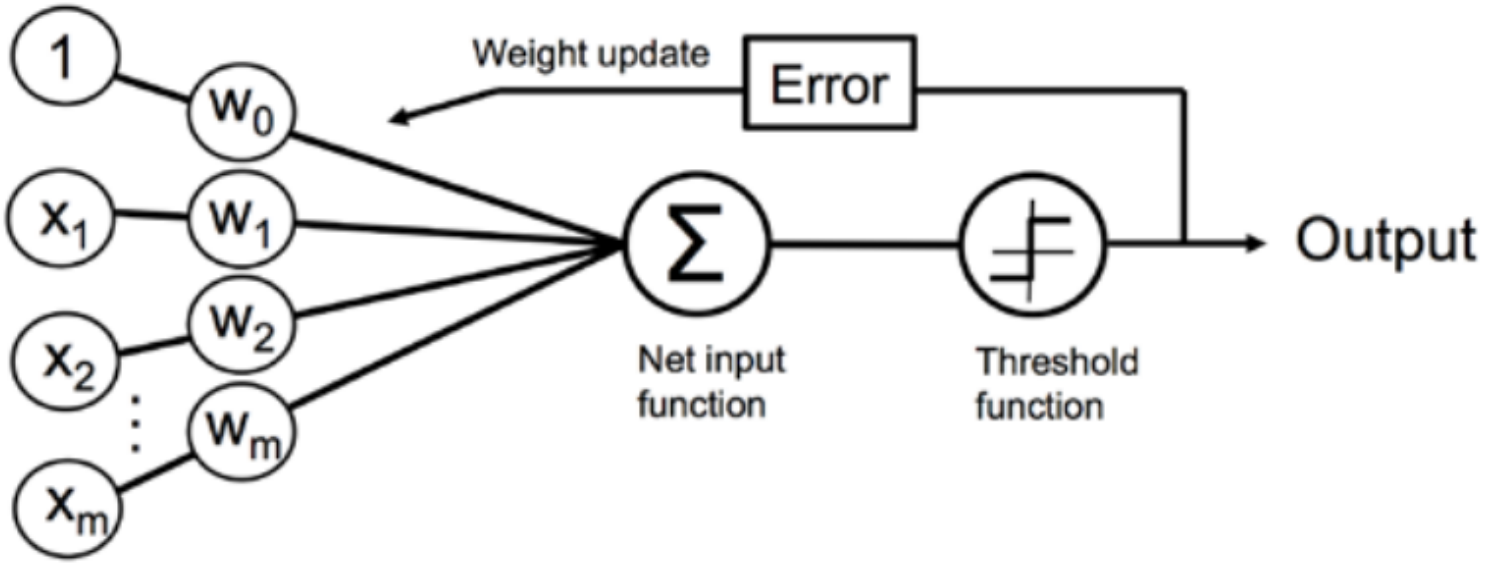


Diagram illustrating the general concept of the perceptron



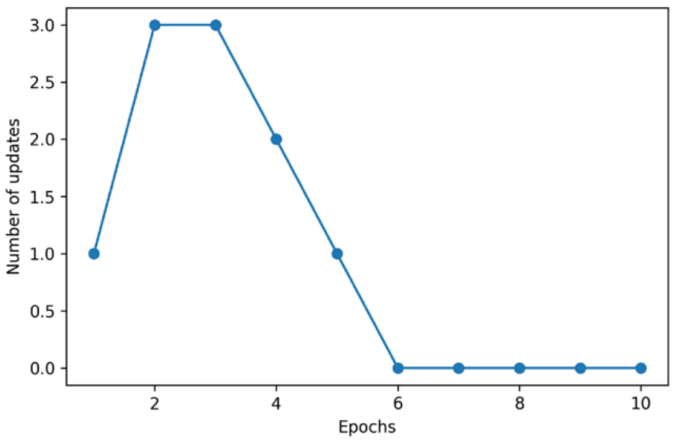
During the learning phase, this output is used to calculate the error of the prediction and update the weights.

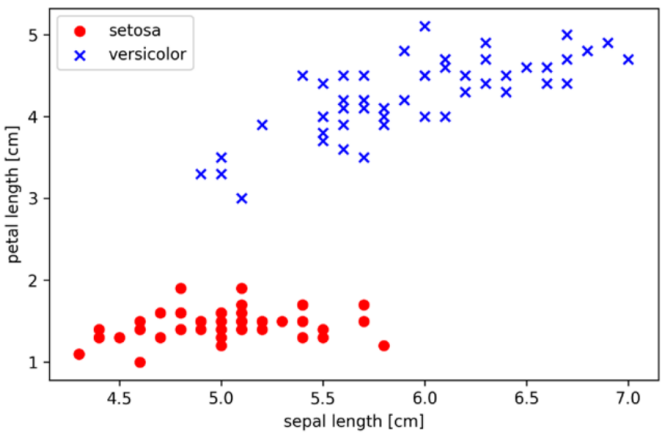
An object-oriented Python perceptron API

```
class Perceptron(object):
  def __init__(self, eta=0.01, n_iter=50, random_state=1):
    self.eta = eta
    self.n_iter = n_iter
    self.random_state = random_state

  def fit(self, X, y):
    rgen = np.random.RandomState(self.random_state)
    self.w_ = rgen.normal(loc=0.0, scale=0.01,
    size=1 + X.shape[1])
    self.errors_ = []
    for _ in range(self.n_iter):
    errors = 0
    for xi, target in zip(X, y):
    update = self.eta * (target - self.predict(xi))
    self.w_[1:] += update * xi
    self.w_[0] += update
    errors += int(update != 0.0)
    self.errors_.append(errors)
    return self

  def net_input(self, X):
    return np.dot(X, self.w_[1:]) + self.w_[0]
  def predict(self, X):
    return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Training a perceptron model on the Iris dataset

```
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1),
ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()

plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```





A small convenience function to visualize the decision boundaries

```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
  # setup marker generator and color map
  markers = ('s', 'x', 'o', '^', 'v')
  colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
  cmap = ListedColormap(colors[:len(np.unique(y))])
  # plot the decision surface
  x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
  x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
  xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
  np.arange(x2_min, x2_max, resolution))
  Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
  Z = Z.reshape(xx1.shape)
  plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
  plt.xlim(xx1.min(), xx1.max())
  plt.ylim(xx2.min(), xx2.max())
  # plot class samples
  for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0],
    y=X[y == cl, 1],
    alpha=0.8,
    c=colors[idx],
    marker=markers[idx],
    label=cl,
    edgecolor='black')
```

Scikit process

- Every model is exposed via an *Estimator* object
- We import the model with the general syntax:

```
from sklearn.[family] import [Model]
```

- where `[family]` and `[Model]` are replaced with appropriate families and models
- for example:

```
from sklearn.linear_model import LinearRegression
```

Scikit - Estimator parameters

- filled in with default values during instantiation, and can be changed during instantiation
- Hint! Check `[shift-tab]` in Jupiter Notebook
- Example:

```
model = LinearRegression()
print(model)
```

Additional parametrs

```
LinearRegression(copy_X=True, fit_intercept=True, normalize=True)
```

Scikit - Fitting example data

- **First split the data into a training set and a test set**

Input:

```
import numpy as np
from sklearn.cross_validation import train_test_split
np.random.seed(20)
X = np.random.randn(6,2)
y = range(6)
print(f'X = {X}')
print(f'y= {list(y)}')
X_train, X_test, Y_train, Y_test = ...
        train_test_split(X,y,test_size=0.35)
print(f'X_train = {X_train}')
print(f'Y_train = {Y_train}')
print(f'X_test = {X_test}')
print(f'Y_test = {Y_test}')
```

Output:

```
X = [[ 0.88389311  0.19586502]
 [ 0.35753652 -2.34326191]
 [-1.08483259  0.55969629]
 [ 0.93946935 -0.97848104]
 [ 0.50309684  0.40641447]
 [ 0.32346101 -0.49341088]]
y= range(0, 6)

X_train = [[ 0.93946935 -0.97848104
 [ 0.35753652 -2.34326191]
 [-1.08483259  0.55969629]]
Y_train = [3, 1, 2]
X_test = [[ 0.88389311  0.19586502]
 [ 0.50309684  0.40641447]
 [ 0.32346101 -0.49341088]]
Y_test = [0, 4, 5]
```

test_size : float, int, or None (default is None) If float, should be between 0.0 and 1.0 and represent

Scikit - Training model

- We can now train/fit our model on the training dataset

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train)
```

- X_train - **features**
- Y_train - **training labels**

Scikit - Testing model

- Now, we can test our model on a test the set
- Since we are dealing with supervised model we have labels
- We will check how our model will deal with the test data. This means we will check if it will correctly assign labels to feature sets with the `predict()` method

```
Y_predicted = model.predict(X_test)
Y_predicted

array([4.04549276, 3.78269805, 2.71179806])
```

Comparing it to test data labels:

```
Y_test

[0, 4, 5]
```

it is rather far away from the desired solution, because the RegressionModel is definitely not suitable to random data analysis.

Scikit - improved data

If we artificially improve the data introducing a strong relation between the labels (y) and features, simply filling in sequence of growing numbers in column 2:

```
np.random.seed(20)
X = np.random.randn(6,2)
y = range(6)
X[:,1] = range(600, 606)
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.35)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train)
Y_predict = model.predict(X_test)
print(f'Y_predict = {Y_predict}')
print(f'Y_test = {Y_test}')
```

We shall get:

```
Y_predict = [0. 4. 5.]
Y_test = [0, 4, 5]
```

Which is a perfect match.

Scikit - uniform interface

Scikit tries to have a uniform interface to models. Given that we have a scikit model instance with a variable name `model`, we have the following standarized functions on all **Estimators**:

- `model.fit(X,y)` - fit the model to training data for supervised machine learning applications (the data with labels),
- `model.fit(X)` - to fit for unsupervised learning applications.

For **supervised estimators**:

- `model.predict(X_new)` - assuming that he model is trained, it will predict labels for given feature sets
- `model.predict_proba()` - used for classification problems, returns the probability for each observation that it belongs to each categorical label,
- `model.score()` - (supported by most estimators), scores are between 0 and 1, and those with the larger score indicating a better fit

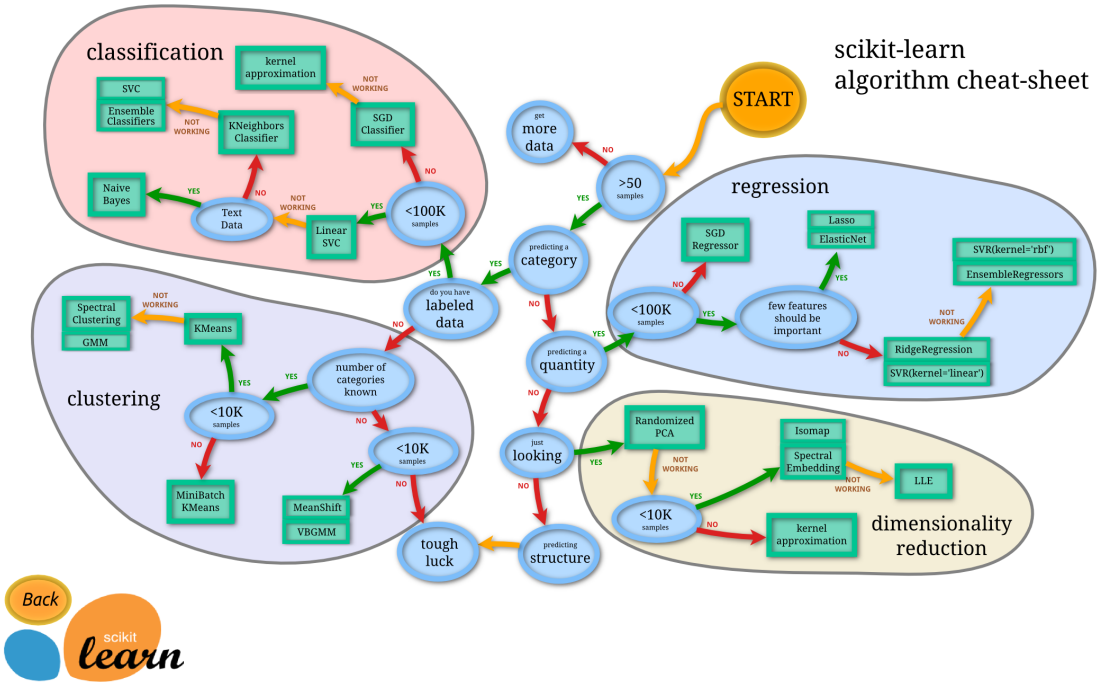
For **supervised estimators**:

- `model.predict()` - predict labels in clustering algorithms,
- `model.tarnsform()` - transforms new data into the new basis, this accepts `X_new`, and returns the new representation of the data based on the Unsupervised model

Choosing a model

http://www.datasciguide.com/content/scikit-learn-algorithm-cheat-sheet/

Scikit cheat-sheet, a walk through guide which to pick...

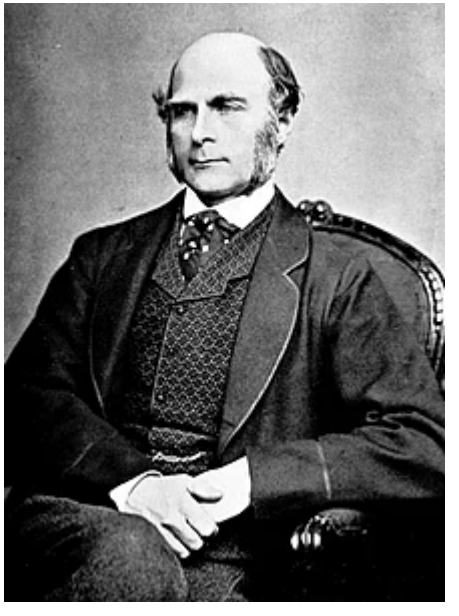


Linear regression

http://www-bcf.usc.edu/~gareth/ISL/

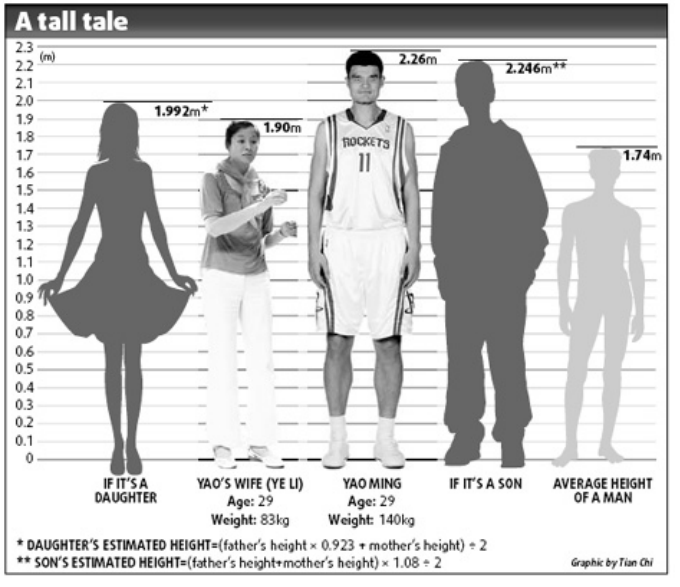
Though it may seem somewhat dull compared to some of the more modern statistical learning approaches described in later chapters of this book, linear regression is still a useful and widely used statistical learning method. Moreover, it serves as a good jumping-off point for newer approaches: as we will see in later chapters, many fancy statistical learning approaches can be seen as generalizations or extensions of linear regression.

- History of linear Regression
  - Francis Galton (1822-1911) - created a theory of correlation and regression to mean value
  - He discovered that mens son tended to be roughly as tall as the father, however the sons height tended to be closer to the average height of all people

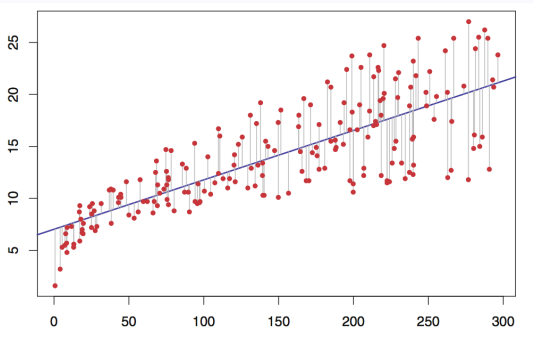


Machine learning for linear regression - the idea

- Now lets presume that knowing the height of a father we want to predict height of his son
- We can take multiple men and their son's heights. By doing this we take a data and label it (with the height)



- If we minimize the vertical distance between all the data points and our line we will determine the best line going as close as possible to the set.
  - sum of squared errors, sum of absolute errors,
- We will use SciKit-Learn in Python for doing this.



Simple Linear Regression

'Let we regress Y on X':

$$\mathbf{Y} \approx \beta_0 + \beta_1 \mathbf{X}$$

for example:

$$\text{sales} \approx \beta_0 + \beta_1 \text{TV}$$

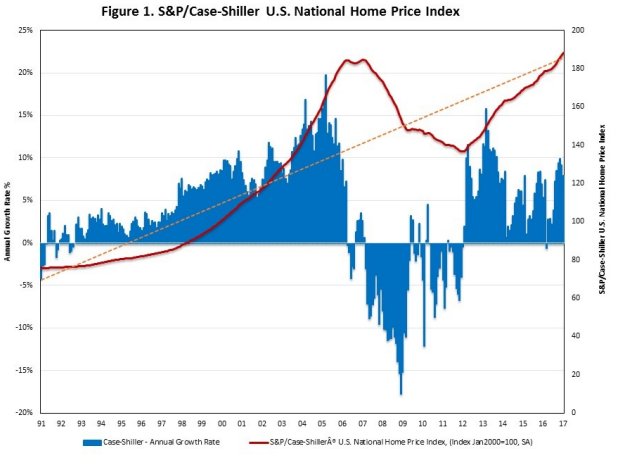
```
sklearn.model_selection import train_test_split
```

- We will work on an example of linear regression related to housing prices based off of existing features
- This is going to be an artificial dataset
- Our goal is to help predicting housing prices in the USA.

Housing prices data overview

https://seekingalpha.com/article/4060386-u-s-real-estate-market-trends-characteristics-outlook

The data contains the following columns:



- 'Avg. Area Income': Avg. Income of residents of the city house is located in.
- 'Avg. Area House Age': Avg Age of Houses in same city
- 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
- 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
- 'Area Population': Population of city house is located in
- 'Price': Price that the house sold at
- 'Address': Address for the house

(slump economic housing bubble)

Data exploration

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

df = pd.read_csv('L06_USA_Housing.csv')
df.head()
df.columns
```

```
df.info()
df.describe()
sns.pairplot(df)
fig,ax = plt.subplots(nrows=1,ncols=1, figsize=(8,6), dpi=150)
sns.distplot(df['Price'], bins=100, ax=ax)
```

Heatmap of the correlation between columns

```
sns.heatmap(df.corr(), ax=ax)
```

Learning and Testing linear regression

https://scikit-learn.org/stable/modules/linear_model.html

```
X = df[]
```

- Create a model *Target value is expected to be a linear combination of the input variables. In mathematical notion, if $\hat{y}$ is the predicted value.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

  - Across the `linear_model` module, the vector $w(w_1, \ldots, w_p)$ as `coef_` and w_0 as `intercept_`.

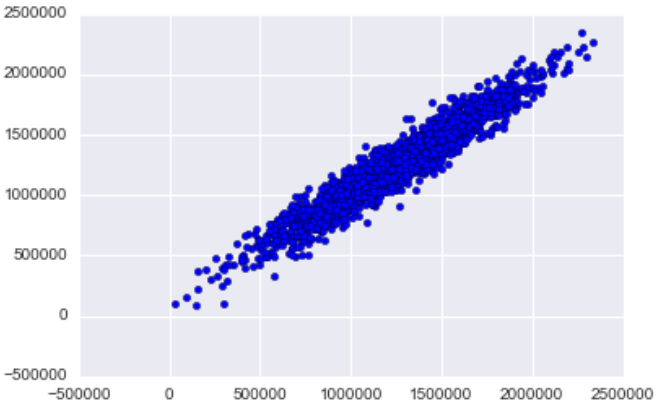
LinearRegression - Ordinary Least Squares

LinearRegression fits a linear model with coefficients $w(w_1, \ldots, w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w {||Xw - y||_2}^2$$

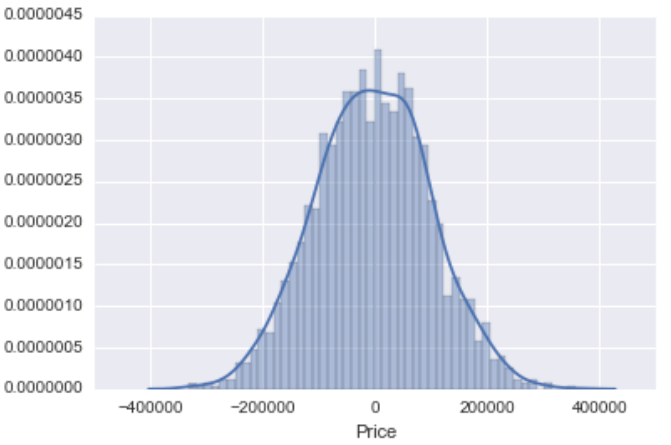
Visually check if the model was corrected

There is one quick way you can visually analyze this which is just by doing a scatterplot. So you can say something like Piazzi thoughts scatter from that plot limb and compare Whitehurst versus the predictions you just made. And if they line up something like this you know you've done a pretty good job. It means that a perfect straight line would be perfectly correct predictions. So a little bit off the sort of straight line is actually a very good job.



Histogram of residuals

```
sns.distplot((y_test-predictions),bins=50);
```



- a histogram of the distribution of our residuals
- the residuals are the difference between the actual values we test and the predicted values
- Notice here that our residuals look to be normally distributed.
- That's a very good sign. It means your model was a correct choice for the data.
- If this is not normally distributed and you have some sort of weird behavior going on you may want to look back at your data and check to see if a linear regression model was the correct choice for the data set.

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

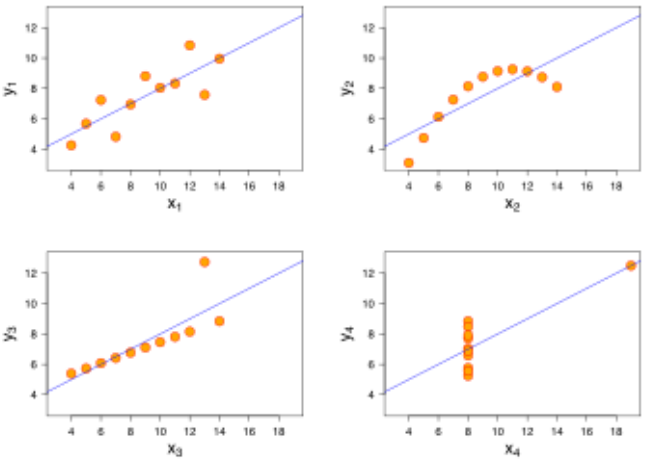
- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

Anscombe's quartet

https://en.wikipedia.org/wiki/Anscombe%27s_quartet

Anscombe's quartet comprises four datasets that have nearly identical simple descriptive statistics, yet appear very different when graphed. Each dataset consists of eleven (x,y) points. They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it and the effect of outliers on statistical properties.



PROPERTY	VALUE	ACCURACY
Mean of x	9	exact
Sample variance of x	11	exact
Mean of y	7.50	to 2 decimal places
Sample variance of y	4.125	±0.003
Correlation between x and y	0.816	to 3 decimal places
Linear regression line	y = 3.00 + 0.500x	to 2 and 3 decimal places, respectively
Coefficient of determination of the linear regression	0.67	to 2 decimal places

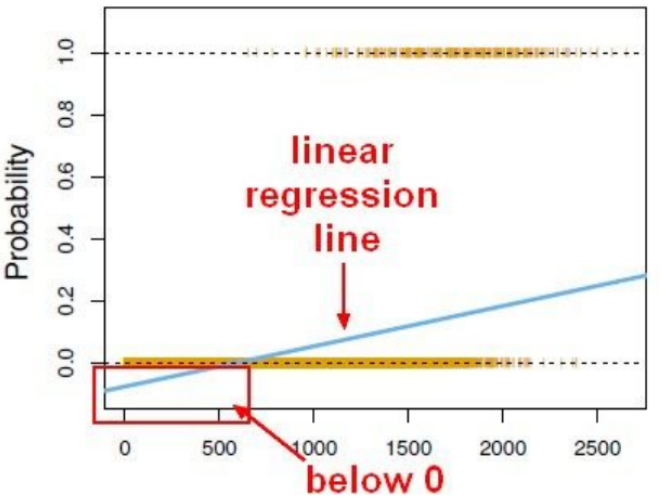
Other metrics

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()


# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
```

Logistics Regression

Method for Classification

- We want to predict to which category belongs 'a new observation'
- Examples:
  - Spam versus 'Ham' emails
  - Disease diagnosis (has cancer or not?)

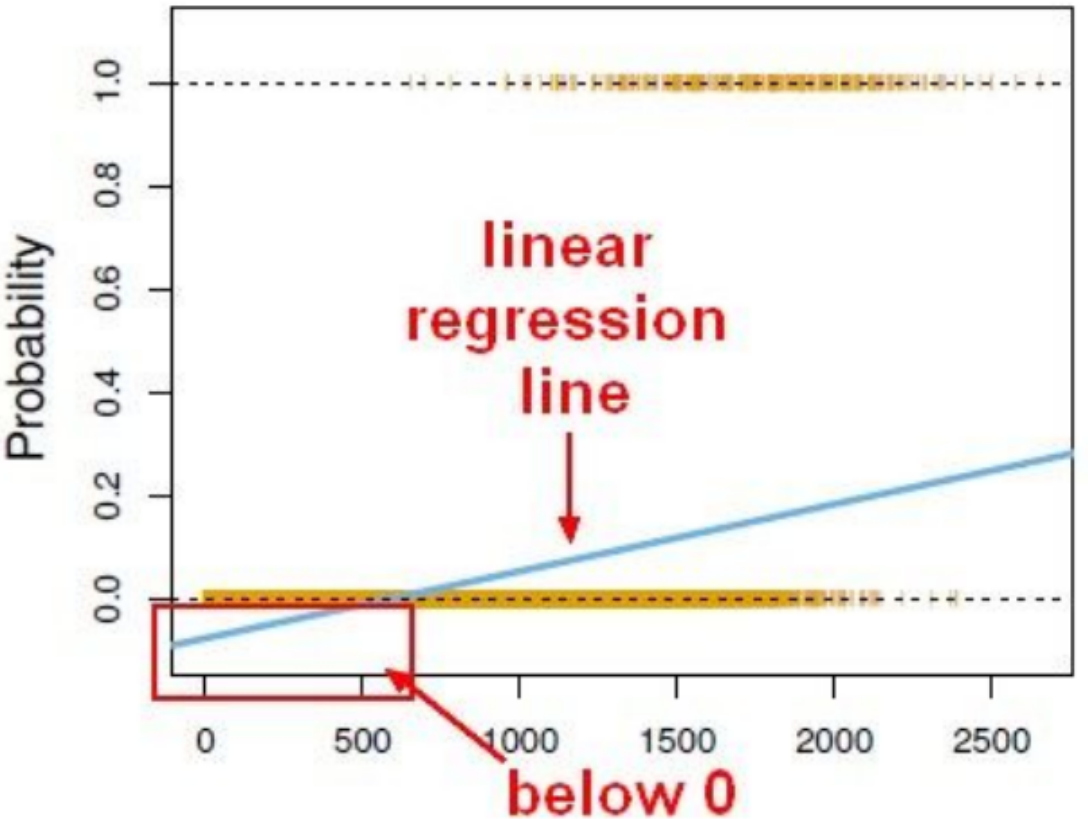


- Binary classification - 0 and 1 (The convention for binary classification is to have two classes 0 and 1.)

- We can’t use a normal linear regression model on binary groups. It won’t lead to a good fit

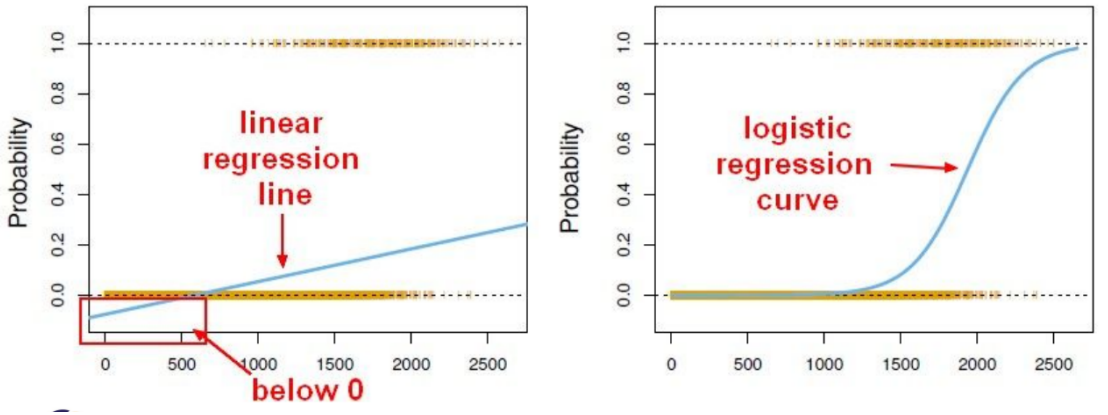
Logistics Regression

- Instead we can transform our linear regression to a logistic regression curve.



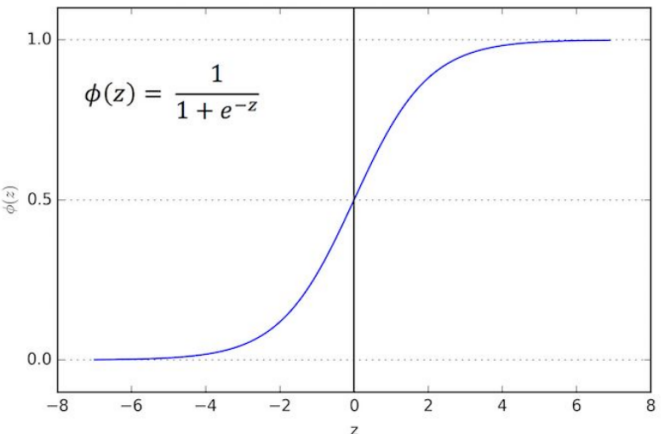
Logistics Regression

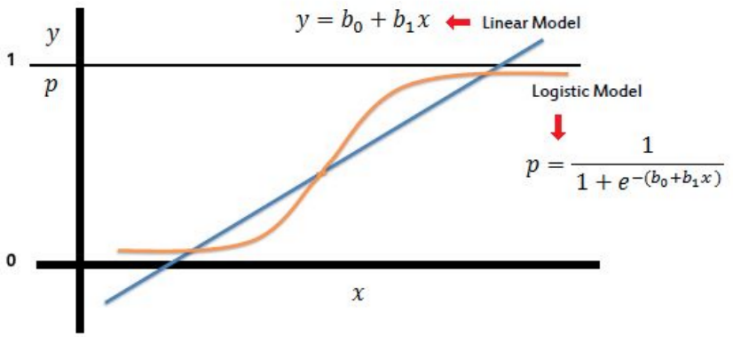
- Instead we can transform our linear regression to a logistic regression curve.



Logistics Regression

- The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1
- This means we can take our Linear Regression Solution and place it into the Sigmoid Function.
- This results in a probability from 0 to 1 of belonging in the 1 class.





Model Evaluation

- After you train a logistic regression model on some training data, you will evaluate your model's on some test data.
- You can use a confusion matrix to evaluate classification models.
- For example, imagine testing for disease

N=165	PREDICTED: NO	PREDICTED: YES
Actual: NO	50	10
Actual: YES	5	100

Accuracy:

- Overall, how often is it correct?
- (TP + TN) / total = 150/165 = 0.91

Misclassification (Error Rate):

- Overall, how often is it wrong?
- (FP + FN) / total = 15/165 = 0.09

Example: Test for presence of disease

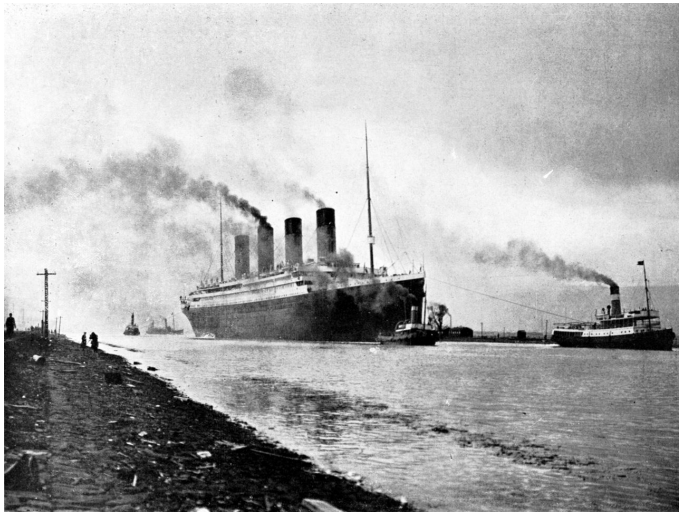
- NO = negative test = False = 0
- YES = positive test = True = 1

Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

Titanic example

https://www.kaggle.com/c/titanic
https://www.kaggle.com/helgejo/an-interactive-data-science-tutorial



One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Titanic example

Plan:

- Exploratory Data Analysis - fill in Missing Data

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

- Data cleaning
- Converting categorical features

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

- Building a Logistic Regression model

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

- Evaluation

```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

Resources

https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12
https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471
https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/
https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

Thank you