Top Most Used Machine Learning Algorithms in Python

**Machine Learning Algorithms in Python :**

|  |
| --- |
| 1. **Linear regression** |
| 1. **Decision tree** |
| 1. **Logistic regression** |
| 1. **(SVM) Support Vector Machines.** |
| 1. **Naive Bayes** |

**1. Linear regression**

It is one of the most popular Supervised Machine Learning algorithms in Python that maintains an observation of continuous features and based on it, predicts an outcome. It establishes a **relationship between dependent and independent variables** by fitting a best line. This **best fit line is represented by a linear equation *Y=a\*X+b*,** commonly called the regression line.

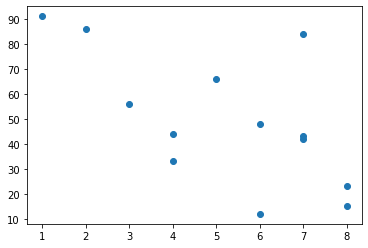
In this equation, ***Y=a\*X+b***

|  |
| --- |
| * **Y – Dependent variable** |
| * **a- Slope** |
| * **X – Independent variable** |
| * **b- Intercept** |

The regression line is the line that **fits best in the equation to supply a relationship between the dependent and independent variables**. When it runs on a single variable or feature, we call it**simple linear regression** and when it runs on different variables, we call it **multiple linear regression.** This is used to estimate the cost of houses, total sales .

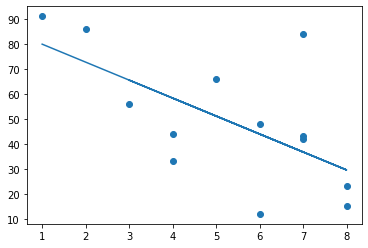
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **import** **matplotlib.pyplot** **as** **plt**  x = [**6**,**7**,**8**,**4**,**6**,**7**,**8**,**3**,**5**,**7**,**4**,**2**,**1**]  y = [**12**,**42**,**15**,**44**,**48**,**84**,**23**,**56**,**66**,**43**,**33**,**86**,**91**]  plt.scatter(x, y)  plt.show() |

Output:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **matplotlib.pyplot** **as** **plt**  **from** **scipy** **import** stats  x = [**6**,**7**,**8**,**4**,**6**,**7**,**8**,**3**,**5**,**7**,**4**,**2**,**1**]  y = [**12**,**42**,**15**,**44**,**48**,**84**,**23**,**56**,**66**,**43**,**33**,**86**,**91**]  slope, intercept, r, p, std\_err = stats.linregress(x, y)  **def** **myfunc**(x):  **return** slope \* x + intercept  mymodel = list(map(myfunc, x))  plt.scatter(x, y)  plt.plot(x, mymodel)  plt.show() |

Output :



**2. Decision Trees**

A decision tree is built by repeatedly asking questions to the partition data. The aim of the decision tree algorithm is to **increase the predictiveness at each level of partitioning so that the model is always updated with information about the dataset.**

Even though it is a **Supervised Machine Learning algorithm**, it is used mainly for **classification rather than regression**. decision tree by comparing important features with a conditional statement. As it descends to the left child branch or right child branch of the tree, depending on the result, the features that are more important are closer to the root. The good part about this machine learning algorithm is that**it works on both continuous dependent and categorical variables.**

**3. Logistic regression**

A **supervised machine learning algorithm in Python**that is used in estimating discrete values in binary, e.g: 0/1, yes/no, true/false. This is based on a set of independent variables. This algorithm is used to **predict the probability of an event’s occurrence by fitting that data into a logistic curve or logistic function.**This is why it is also called logistic regression.

Logistic regression, also called as Sigmoid function, takes in any real valued number and then maps it to a value that falls between 0 and 1. This algorithm finds its use in finding spam emails, website or ad click predictions and customer churn.

Sigmoid Function is defined as,

|  |
| --- |
| **f(x) = L / 1+e^(-x)** |
| * x: domain of real numbers |
| * L: curve’s max value |

|  |  |
| --- | --- |
| 1  2  3  4  5    6  7  8  9  10  11  12  13  14 | **import** **numpy**  **from** **sklearn** **import** linear\_model  #Reshaped for Logistic function.  X = numpy.array([**4.28**, **3.24**, **1.17**, **1.27**, **1.52**, **3.86**, **4.12**, **4.37**, **4.96**, **4.52**, **3.69**, **5.88**]).reshape(-**1**,**1**)  y = numpy.array([**0**, **0**, **0**, **0**, **0**, **0**, **1**, **1**, **1**, **1**, **1**, **1**])  logr = linear\_model.LogisticRegression()  logr.fit(X,y)  log\_odds = logr.coef\_  odds = numpy.exp(log\_odds)  **print**(odds) |

Output :

[[3.14625066]]

**4. Support Vector Machines (SVM)**

This is one of the most important machine learning algorithms in Python which is mainly used for **classification but can also be used for regression tasks**. In this algorithm, each data item is plotted as a point in n-dimensional space, where **n denotes the number of features you have, with the value of each feature as the value of a particular coordinate.**

SVM does the **distinction of these classes by a decision boundary.** For e.g: If length and width are used to classify different cells, their observations are plotted in a 2D space and a line serves the purpose of a decision boundary. If you use 3 features, your decision boundary is a plane in a 3D space. SVM is highly effective in cases where the number of dimensions exceeds the number of samples.

**5. Naive Bayes**

Naive Bayes is a **supervised machine learning algorithm used for classification tasks**. This is one of the reasons it is also called a Naive Bayes Classifier. It assumes that features are independent of one another and there exists no correlation between them. But as these assumptions hold no truth in real life, this algorithm is called ‘naive’.

This algorithm works on Bayes’ theorem which is:

|  |
| --- |
| **p(A|B) = p(A) . p(B|A) / p(B)** |

In this,

|  |
| --- |
| * p(A): Probability of event A |
| * p(B): Probability of event B |
| * p(A|B): Probability of event A given event B has already occurred |
| * p(B|A): Probability of event B given event A has already occurred |

The Naive bayes classifier calculates the probability of a class in a given set of features, p( yi I x1, x2, x3,…xn). As this is put into the ***Bayes’ theorem***, we get :

**p( yi I x1, x2…xn)= p(x1,x2,…xn I yi). p(yi) / p(x1, x2….xn)**

As the **Naive Bayes’ algorithm** assumes that features are independent, p( x1, x2…xn I yi) can be written as :

**p(x1, x2,….xn I yi) = p(x1 I yi) . p(x2 I yi)…p(xn I yi)**

p(x1 I yi) is the**conditional probability for a single feature** and can be easily estimated from the data.

|  |  |
| --- | --- |
| 1  2  3  4 | %matplotlib inline  **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **seaborn** **as** **sns**; sns.set() |

|  |  |
| --- | --- |
| 1  2  3 | **from** **sklearn.datasets** **import** make\_blobs  X, y = make\_blobs(**100**, **2**, centers=**2**, random\_state=**2**, cluster\_std=**1.5**)  plt.scatter(X[:, **0**], X[:, **1**], c=y, s=**50**, cmap='RdBu'); |

Output :

