Perceptron algorithm

The Perceptron algorithm is a **binary classification** machine learning algorithm.

A perceptron is a classification model that consists of a set of weights, or scores, one for every feature, and a threshold.

The weighted sum of the input of the model is called the **activation.**

Activation = Weights * Inputs + Bias

If the activation is above 0.0, the model will **output 1.0**; otherwise, it will **output 0.0.**

Predict 1: If Activation > 0.0

Predict 0: If Activation <= 0.0

The Perceptron is a **linear classification algorithm**. This means that it learns a decision boundary that separates two classes using a line (**called a hyperplane**) in the feature space.

The coefficients of the model are referred to as **input weights** and are trained using the **stochastic gradient descent** optimization algorithm.

The weights of the model are then updated to reduce the errors for the example. This is called the **Perceptron update rule**. This process is repeated for all examples in the training dataset, called an **epoch**.

Model weights are updated with a small proportion of **the error each batch**, and the proportion is controlled by a hyperparameter called **the learning rate**, typically set to a small value.

weights (t + 1) = weights(t) + learning_rate * (expected_i – predicted_) * input_i

Training is stopped when the error made by **the model falls to a low level or no longer improves, or a maximum number of epochs is performed.**

The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using **heuristics or hyperparameter tuning.**

<u>Pseudocode:</u>

Algorithm 5 $\textsc{PerceptronTrain}(\mathbf{D}, \mathit{MaxIter})$

$w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $\mathit{iter} = 1 \ldots \mathit{MaxIter}$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \ldots, w_D, b$

Algorithm 6 $\textsc{PerceptronTest}(w_0, w_1, \ldots, w_D, b, \hat{x})$

$a \leftarrow \sum_{d=1}^{D} w_d\, \hat{x}_d + b$ // compute activation for the test example
return $\textsc{sign}(a)$

1. Initialize our weight vector w with small random values

2. Until Perceptron converges:

(a) Loop over each feature vector xj and true class label di in our training set D

(b) Take x and pass it through the network, calculating the output value: yj = f(w(t) · xj)

(c) Update the weights w: wi(t +1) = wi(t) +α(dj −yj)xj,i for all features 0 <= i <= n

B. **Code:**

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import pandas as pd

df = pd.read_csv (r'train.data')
print (df)
df.head()


# In[3]:


import pandas as pd

df = pd.read_csv (r'test.data')
print (df)
df.head()


# In[4]:


import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[5]:


plt.scatter(df['5.0'],df['3.5'],df['1.3'])


# In[7]:

```

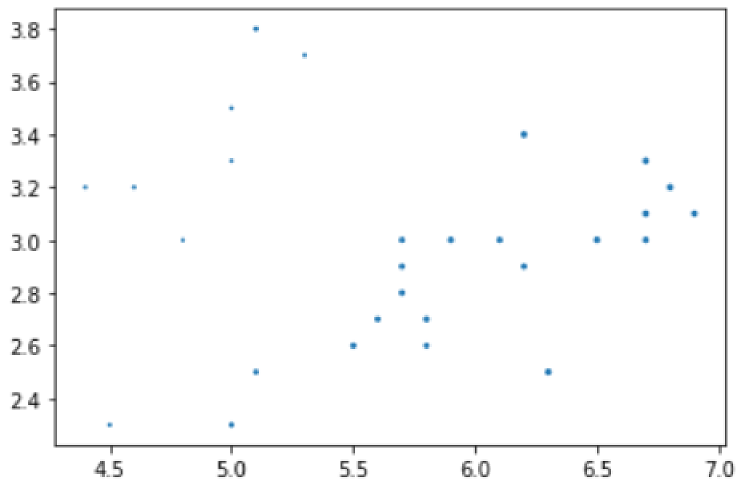
```python
plt.scatter(df['0.3'],df['class-1'])


# In[8]:


X = df[['5.0','3.5','1.3']]


# In[9]:


y= df['0.3']


# In[10]:


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)


# In[11]:


X_train


# In[12]:


X_test


# In[13]:


y_train


# In[14]:


y_test


# In[15]:


from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)


# In[16]:
```

```
X_test


# In[17]:


clf.predict(X_test)


# In[18]:


y_test


# In[19]:


clf.score(X_test, y_test)


# In[20]:


X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.3,random_state=10)
X_test


# In[ ]:
```

C: **Classes:**

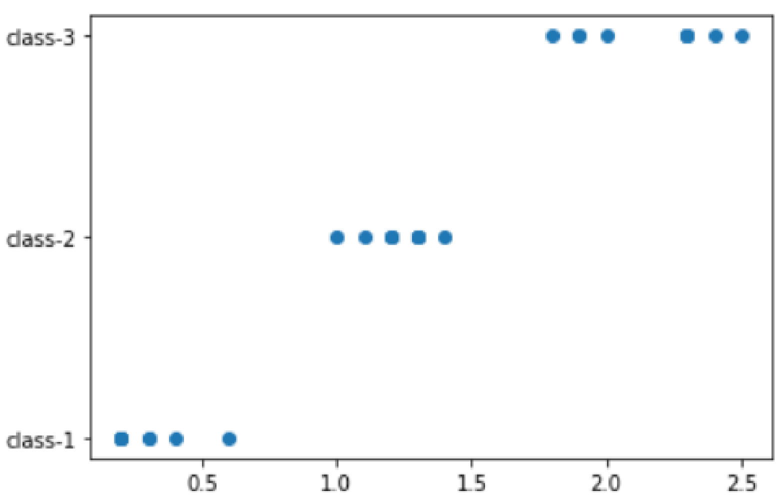
```
In [5]: plt.scatter(df['5.0'],df['3.5'],df['1.3'])
```

```
Out[5]: <matplotlib.collections.PathCollection at 0x20cadca8d00>
```




```
In [7]: plt.scatter(df['0.3'],df['class-1'])
```

```
Out[7]: <matplotlib.collections.PathCollection at 0x20cafe719c0>
```




```
In [8]: X = df[['5.0','3.5','1.3']]
```

D. **<u>train and test:</u>**

clf.score(X_test, y_test)

```
0.8668918391902904 --→ 0.3

0.9265018393202821 --→ 0.1 is better score
```