perceptron

September 15, 2022

Lab 4: The monolayer perceptron

Goals:

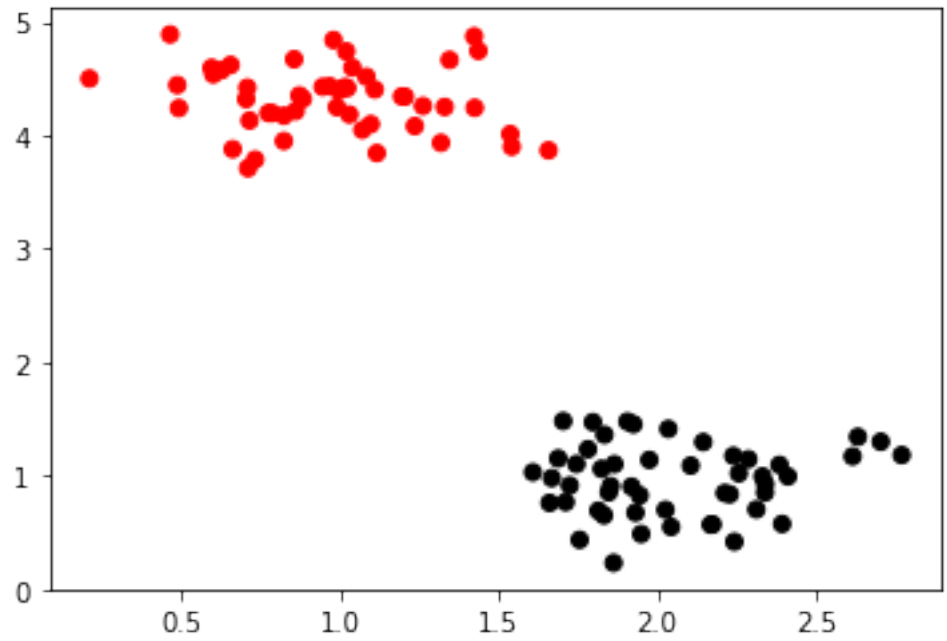
- Implement your first classifier
- Understand the concept of model complexity

Let's a generate a simple toy dataset.

[121]:
```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=100, centers=2, n_features=2, cluster_std=0.3,␣
 ↪random_state=0)

labels = y.copy()
y[y==0] = -1

print("Shape of the data", X.shape)
print("labels:", y)
```

```
Shape of the data (100, 2)
labels: [ 1  1 -1 -1  1 -1 -1  1 -1  1 -1  1 -1 -1  1  1  1 -1  1  1 -1 -1  1 -1
  1 -1  1 -1 -1  1  1 -1  1  1  1 -1  1  1 -1  1 -1 -1 -1 -1  1  1  1  1
 -1 -1  1  1 -1 -1 -1 -1 -1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1
  1 -1 -1 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1 -1  1  1  1  1  1  1 -1 -1
  1  1 -1  1]
```

[122]:
```
colors = ["red", "black"]
scatter_colors = np.array(colors)[labels]

plt.figure()
plt.scatter(X[:, 0], X[:, 1], color=scatter_colors)
plt.show()
```



A monolayer perceptron is a geometric method that performs classification using a linear decision function of the form:

$$f(x) = \langle w, x \rangle + \theta$$

or with a different notation:

$$f(x) = w^\top x + \theta$$

0.0.1 Question 1

Given a sample x, a prediction is then given by taking the sign of f:

$ prediction(x) = 1$ if $f(x) \geq 0$ and -1 if $f(x) < 0$.

Complete the function below that performs gradient descent to optimize the parameters w and θ.

The loss function that we minimize is:

$$l_i(w) = (-y_i \sum_{i=1}^{n} x_i w)_+$$

Hint: Take a look at the lectures slides to refresh your memory on how to update the parameters

[44]:
```
def gradient_descent(X, y, step_size=0.01, n_iter=10000):
    n_samples, n_features = X.shape
    # do initialization
    w = np.zeros(n_features)
    theta = 0.
```

```
    for t in range(n_iter):
        for ii in range(n_samples):
            # for each sample, if its an error do gradient step
    return w
```

[ ]:

0.0.2 Question 2

The funciton f is parametrized using one slope vector w and the offset scalar θ. Can you think of a change of variable and change to the data X such that this parametrization is reduced to only one parameter w to get a function of the form:

$$f(x) = w^\top x$$

0.0.3 Question 3

Implement this change in your model and update the gradient descent function accordingly.

[ ]:

0.0.4 Question 4

Write a function that uses the learned parameters to make predictions.

[ ]:
```
def predict(x, w):

    return
```

0.0.5 Question 5

Time to write a proper classifier object that following the sklearn API. We import the sklearn BaseEstimator model and customize it with our gradient descent function. Complete the following cell:

[ ]:
```
from sklearn.base import BaseEstimator

class Perceptron(BaseEstimator):
    def __init__(self, step_size=0.1, n_iter=1000):
        self.step_size = step_size
        self.n_iter = n_iter

    def fit(self, X, y):
        # TODO
```

```
    def predict(self, X):
        # TO DO
```

0.0.6 Question 6

Time to properly evaluate our model. Divide the data into train-test. Create an instance of your perceptron, train it on the data and compute a test accuracy score.

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

[ ]:
```

```

0.0.7 Question 7:

Complete the code below to plot the accuracy curve for a list of `train_size`. What do you notice ?

[ ]:
```
train_sizes = np.linspace(0.1, 0.9, 10)

accuracies = []

for test_size in train_sizes:
    accuracies.append(train_size)
    ##
```