Assignment 2

Principles of Machine Learning

Name: Rutwik Borole | Id: 22224253 |Program: CSD1

In this report we can see the use of a supervised learning algorithm known as perceptron. Perceptron is a single layer neural network and multilayer perceptron is known as neural networks. Perceptron is a linear classifier; it helps us in classifying a given input data.

Perceptron can be seen as one single unit of an artificial neural network, basically a simplified model of a biological neuron. Our perceptron has 5 parts, i.e inputs, weights, net input function, activation function and the output.

The algorithm for my perceptron model goes as follows:

We supply our input features, in our case of wildfire data set, our input features are temp, humidity, rainfall, drought_code, buildup_index and wind speed.

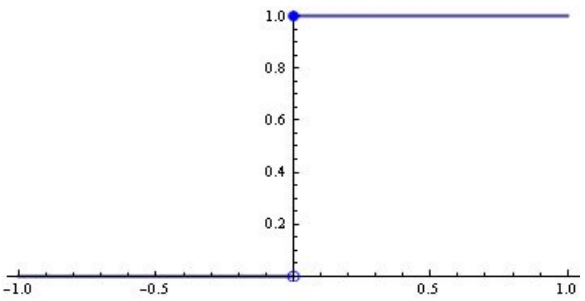
These input features are then multiplied by weights, we start with weights zero and apply a learning rate of 0.01.

We make use of a linear function $f(w,b) = w^T.x + b$, *where w is weight, x is our input values, and b is the bias.*

Then we make use of the input step function which is defined as

$$g(a) = \begin{cases} 1 & if\ a >= 0 \\ 0 & if\ a < 0 \end{cases}$$

`Basically, this function says that if the output is greater than or equal to zero, then the model will classify it as 1 (yes) and if our output is less than zero, our model will classify it as 0 (no). It can be viewed graphically below:



We can observe from the above graph that the threshold is zero, for a>= 0, g(a)=1 and for a<0, g(a) = 0.

Now for setting out weights and bias, we make use of the perceptron update rule, this rule is quite similar to the gradient descent update rule.

So, we look at each training sample x_i and for each sample we apply the update step, this is defined as, new weight is equal to the old weight plus the delta weight.

$w_n = w_o + \delta w$,

where the delta weight is equal to alpha times the actual label minus the predicted label times x.

$\delta w = \alpha \, . \, (y_i - \hat{y}_i) \, . \, x_i$ *,where alpha(α) is the learning rate*

For implementing our algorithm, I am using jupyter notebook, and for reading our data, I have converted the wildfire dataset into a csv file.

For testing our data, first I have divided the wildfire dataset into training and testing set. I am using the Sklearn model selection library for achieving this split. We can split the dataset into 2 parts, i.e 2/3 part for training and 1/3 part for testing. Also, to do it 5 times, I am using a simple for loop to test and train the dataset. From the wildfire dataset, we have the first column as the label (yes/no), followed by the 9 attributes which help determine if there will be a fire or not. I have removed the following attributes from the dataset which aren't a good measure for determining the label, year, month & day.

After executing our algorithm on the wildfire dataset 5 times we get accuracy:

Perceptron classification accuracy 1st time - 0.9117647058823529

	precision	recall	f1-score	support
0	0.72	0.96	0.82	50
1	0.97	0.78	0.86	86
accuracy			0.85	136
macro avg	0.84	0.87	0.84	136
weighted avg	0.88	0.85	0.85	136

Perceptron classification accuracy 2nd time - 0.8088235294117647

	precision	recall	f1-score	support
0	0.79	0.98	0.88	51
1	0.99	0.85	0.91	85
accuracy			0.90	136
macro avg	0.89	0.91	0.89	136
weighted avg	0.91	0.90	0.90	136

```
Perceptron classification accuracy 3rd time - 0.7794117647058824
              precision    recall  f1-score   support

           0       0.81      1.00      0.89        51
           1       1.00      0.86      0.92        85
    accuracy                           0.91       136
   macro avg       0.90      0.93      0.91       136
weighted avg       0.93      0.91      0.91       136
```

```
Perceptron classification accuracy 4th time - 0.8382352941176471
              precision    recall  f1-score   support

           0       0.76      0.98      0.86        49
           1       0.99      0.83      0.90        87
    accuracy                           0.88       136
   macro avg       0.87      0.90      0.88       136
weighted avg       0.91      0.88      0.88       136
```

```
Perceptron classification accuracy 5th time - 0.8529411764705882
              precision    recall  f1-score   support

           0       0.76      0.98      0.85        51
           1       0.99      0.81      0.89        85
    accuracy                           0.88       136
   macro avg       0.87      0.90      0.87       136
weighted avg       0.90      0.88      0.88       136
```

Using a reference implementation from the Sklearn perceptron class, we get the accuracy:

```
Perceptron classification accuracy - 0.8897058823529411
              precision    recall  f1-score   support

           0       0.94      0.85      0.89        74
           1       0.84      0.94      0.89        62

    accuracy                           0.89       136
   macro avg       0.89      0.89      0.89       136
weighted avg       0.89      0.89      0.89       136
```

Appendix

Appendix A

Code snippet of perceptron algorithm

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

class Alg_Pecerp:
    def __init__(self, l_rate=0.01, iters=100):
        self.iters = iters
        self.learn = l_rate
        self.wt = None
        self.bias = None
        self.activationfunction = self.unitstepfunction

#defining our fit fucntion
    def fit(self, X_value, y_value):
        samples, features = X_value.shape

#parameters for our init function
        self.bias = 0
        self.wt = np.zeros(features)

        label = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.iters):

            for index, xi in enumerate(X):

                linear_func_value = np.dot(xi, self.wt) + self.bias
                predicted_y_value =
self.activationfunction(linear_func_value)

# Implementing our perceptron update rule
                update = self.learn * (label[index] - predicted_y_value)

                self.wt = self.wt + update * xi
                self.bias =  self.bias + update

#defining our predict fucntion
    def predict(self, X):
        linear_func_value = np.dot(X, self.wt) + self.bias
        predicted_y_value = self.activationfunction(linear_func_value)
        return predicted_y_value

#defining our unit step function
    def unitstepfunction(self, x):
        return np.where(x >= 0, 1, 0)

if __name__ == "__main__":
    def accu(y_true, y_pred):
        accu = np.sum(y_true == y_pred) / len(y_true)
        return accu

#Reading our data and testing it 5 times
df = pd.read_csv('wildfire.csv')
X = df.drop(columns=['fire','year'])
X = X.to_numpy()
y = df['fire']
y = df['fire'].map({'yes': 1, 'no': 0})

for i in range(1,6):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=np.random)
```

```
    p = Alg_Pecerp(l_rate=0.01, iters=100)
    p.fit(X_train, y_train)
    predictions = p.predict(X_test)
    print("Perceptron classification accuracy",i, accu(y_test,
predictions))

prediction = pd.DataFrame(predictions,
columns=['predictions']).to_csv('prediction.csv')
```

#Reference implementation of perceptron from Sklearn library:

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

df = pd.read_csv('wildfire.csv')
X = df.drop(columns=['fire','year'])
X = X.to_numpy()
y = df['fire']
y = df['fire'].map({'yes': 1, 'no': 0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=20)
p = Perceptron(random_state=42)
p.fit(X_train, y_train)

predictions = p.predict(X_train)
predictions_test = p.predict(X_test)
train_score = accuracy_score(predictions, y_train)
from sklearn.metrics import classification_report
print(classification_report(p.predict(X_train), y_train))
print("Accuracy on data: ", train_score)
```