

Assignment 2

Implementation of Linear Perceptron Algorithm

13500576

Project link: <https://colab.research.google.com/drive/1YwrXamrtPjY2EfRImTW-KORDNvAybiTH?usp=sharing>

Introduction:

A linear perceptron is a supervised classification model. The models produced by this method are based on the assumption that the target variable 'y' to be predicted in a dataset is the weighted sum of its other attributes (linear combination of other attributes). Each k-dimensional data point can be represented as a vector (inner product of data and associated weight vector). This data vector is then compared against a decision boundary or a 'Threshold'.

The input of the algorithm is considered as weighted sum of data attributes $a = x_1 \cdot w_1 + x_2 \cdot w_2 + \dots + x_K \cdot w_K$. $x_K > 0$

The output of this comparison gives us the predicted target value 'y' which can be '+1' or '-1' depending upon the classification.

It is considered that $y(\text{predicted}) \cdot y(\text{actual}) = +1$ for correct predictions (since correctly classified outputs will have same sign).

Linear Perceptron:

All the code snippets have been taken from colab notebook **ML A2.ipynb** and the line indexes are corresponding to the original notebook.

Overall Perceptron concept:

Calculates the sums of weighted parameters towards axis = 1. This means the $w_1x_1 + w_2x_2 + \dots + w_kx_k$ for one record. The output of this is stored in variable **hval**. The output with correct sign of hval is considered as the prediction of the perceptron model.

```
def perceptron_predict(W,X):
    # function accepts 2 arguments W and X
    # here W is the weight vector(5 vectors)
    # X is an array of data with n records * 5 attributes
    hval = (W*X).sum(axis=1) # takes summation of all weighted
    attributes in an entry
    pred = np.sign(hval)
    return pred
```

Homogenous list creation:

While creating an array to store data of k dimensions, an additional dummy variable 'Xo'(say) is added according to the algorithm to increase the dimensionality to (k+1). This extra dimension is then used

to adjust training errors and improve the accuracy of the model. The conventional value of dummy variable is taken as '1'. In the following code, dummy variable is being added (line 3 – **np.concatenate**) to the training data. This leads the data to convert from 4 dimensions to 5-dimensional data. This ultimately leads to creation of a new homogenous array **X_train_homo**(line 5).

```
1. #create a homogenous variable list
2. ons_ = np.ones((X_train.shape[0],1))
3. X_train_homo = np.concatenate([X_train, ons_], axis = 1)
4. ons_ = np.ones((X_test.shape[0],1))
5. X_test_homo = np.concatenate([X_test, ons_] ,axis =1)
```

Accuracy:

One of the primary measures of a good perceptron model is the accuracy. Output of the target variable is generated in TRUE/FALSE or 0/1 pattern (Binary). Line 25 stores the ratio of count of correctly predicted values (TRUE) to the total number of training samples. This ratio is nothing but how accurate the model was to predict y for training data. Line 29 **accu_valid** compares the output to actual value of the target 'y' to get the validation or verification of accuracy on testing data.

```
25. accu_train = np.count_nonzero(is_pred_corr) / y_train.size
26. print(f"Training accuracy {accu_train}")

28. pred_valid = perceptron_predict(W,X_test_homo)
29. accu_valid = np.count_nonzero(pred_valid == y_test) / y_test.size
30. print(f"Validation accuracy {accu_valid}")
```

Error Computation:

The incorrectly classified samples need to be identified to train them for better accuracy. This function identifies the indexes of the data records which are incorrectly classified. The output of this function will be 0 or False in case of an error. All these indexes are stored in the array **error_indexes**

```
#indexes in the array where prediction is wrong
error_indexes = np.nonzero(is_pred_wrong)[0]
```

Weight update:

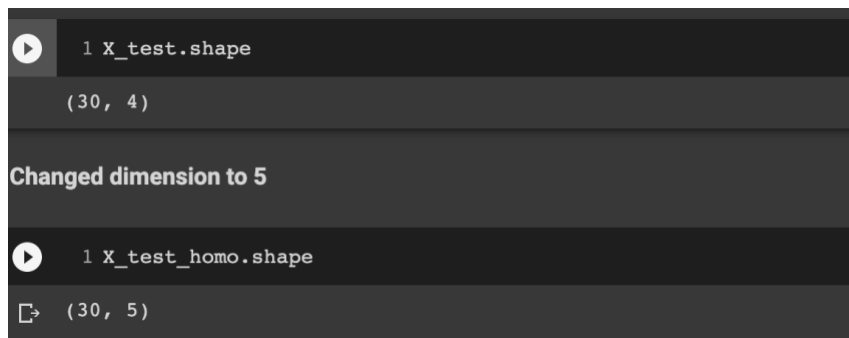
The weights need to be updated for all the incorrectly classified samples. This is done by changing the current weights towards the direction of the actual value of the target (**moving x towards target**). However, no change is applied to correctly classified values. The loop starts from 1st incorrect prediction and keeps on updating until there are no incorrect values to update.

Wnew = Wold +y*x(towards the required direction of x)

```
if len(error_indexes) > 0: # Iterate until there are no more errors
left to update
    next_i = error_indexes[0]
    W_update = X_train_homo[next_i] * y_train[next_i] # updating in the
direction of actual y
    W += W_update # adding updates to previous weight
else:
    break
```

Model Evaluation:

This implementation concerns with the use of Iris dataset which comes preloaded with the Python library sklearn. The dataset contains information about the parameters of the iris flower. It consists of 4 attributes sepal length, petal length, sepal width and petal width. The flower can be classified into 3 categories Versicolor, Sentosa and Virginica. For this implementation only two categories have been considered to make it a binary classification (Sentosa and Versicolor).



```
1 X_test.shape
(30, 4)

Changed dimension to 5

1 X_test_homo.shape
(30, 5)
```

Figure 1: Data set with dummy parameter

Data Manipulation

As mentioned above, the data has been classified into two categories only. This was achieved by assigning ‘-1’ to versicolor category and ‘+1’ to Sentosa category. This can also be called as a ‘Decision boundary’ or a threshold. If the value of “hval” >0 then it will be classified as Sentosa, otherwise Versicolor.

Another manipulation to the dataset was addition of another dimension by introducing a dummy variable with value 1.

Experiment Design and Evaluation

1. First the data is randomly split using the scikit library in a 30/70 split.(30% train)
2. The value of Random is set to 42 to shuffle data before splitting.
3. Homogenous array created
4. Sum of weights stored in “hval”
5. Error indexes identified
6. Weights updated till end of indexes.

```
Help X

Init signature: Perceptron(*args, **kwargs)
Docstring:
Perceptron

Read more in the :ref:`User Guide <perceptron>`.

Parameters
-----

penalty : {'l2', 'l1', 'elasticnet'}, default=None
    The penalty (aka regularization term) to be used.

alpha : float, default=0.0001
    Constant that multiplies the regularization term
    if regularization is
    used.

fit_intercept : bool, default=True
    Whether the intercept should be estimated or not.
    If False, the
    data is assumed to be already centered.

max_iter : int, default=1000
    The maximum number of passes over the training
    data (aka epochs).
    It only impacts the behavior in the ``fit``
    method, and not the
    :meth:`partial_fit` method.

.. versionadded:: 0.19

tol : float, default=1e-3
    The stopping criterion. If it is not None, the
    iterations will stop
    when (loss > previous_loss - tol).

.. versionadded:: 0.19

shuffle : bool, default=True
```

Figure 2:Hyperparameters

Evaluation Results:

By constantly updating the W for every incorrectly classified record, the accuracy improves with each iteration. To prevent any indexing errors a loop has been used to only iterate until there are values left to correct.

```
> Training accuracy 0.6
> Validation accuracy 0.8
> Training accuracy 0.5285714285714286
> Validation accuracy 0.43333333333333335
> Training accuracy 0.5285714285714286
> Validation accuracy 0.46666666666666667
> Training accuracy 0.4714285714285714
> Validation accuracy 0.56666666666666667
> Training accuracy 0.5285714285714286
> Validation accuracy 0.43333333333333335
> Training accuracy 0.4714285714285714
> Validation accuracy 0.6333333333333333
> Training accuracy 0.5285714285714286
> Validation accuracy 0.43333333333333335
> Training accuracy 1.0
> Validation accuracy 1.0
```

Conclusion

Perceptron was successfully implemented for iris dataset. The model gets better by subsequently adjusting the weight towards the actual target. The final output is able to classify between Sentosa and versicolor with a decent level of accuracy.