All Single layer Perceptron Algorthm performed by Matthew McKernan, ID: 17321381

We divided the work so Matthew McKernan completed all things related to single-layer perceptron and Anagha completed all work for multi-layer perceptron.

Algorithm and design decisions for Single layer Perceptron Algorthm:

Data preparation:
We checked the value counts to make sure the data was relatively symmetric. We split data into train, test and validation. We normalised the data feature data. We converted yes/no to 1/0 so we could use it to train the weights later on.

Perceptron algorithm

I created several functions to do some of the perceptron processes.

Threshold algorithm:

The threshold algorithm outputs yes or no depending on if the weighted sum of features and weights is greater than some step value.

Weighted sum Algorithm:

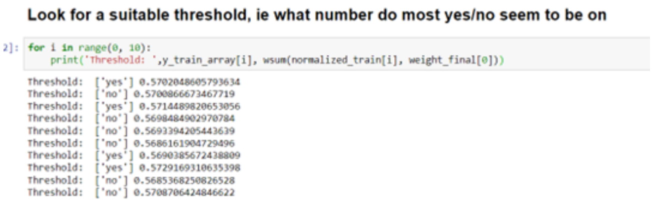
An algorithm that inputs the features row and weights and find the sum of the dots products

Train weights on training data using formula w = w + x*r*(y_actual-y_predicted):

It went through each row of the training data and updated the weights using the formula w = w + x*r*(y_actual-y_predicted). The weights were updated if they were different from the actual data. I used a correction term of 0.5. I tried out several correction terms and corresponding thresholds and this correction term seemed to give good results.

Find what value to use to divide yes/no:

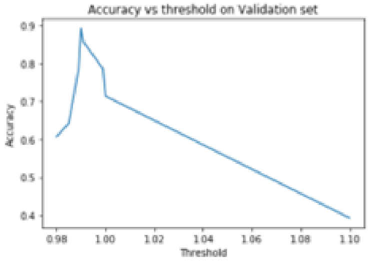
I compared the weighted sum of several training rows and the actual output for that row. I picked a number that seemed to divide most of the yes/no outputs for the threshold. Later on I used a learning curve to find the best value for the threshold. The best value I found was 0.99.



Look for a suitable threshold, ie what number do most yes/no seem to be on

```
2]: for i in range(0, 10):
        print('Threshold: ',y_train_array[i], wsum(normalized_train[i], weight_final[0]))
```

```
Threshold:  ['yes'] 0.5702048605793634
Threshold:  ['no'] 0.5700866673467719
Threshold:  ['yes'] 0.5714489820653056
Threshold:  ['no'] 0.5698484902970784
Threshold:  ['no'] 0.5693394205443639
Threshold:  ['no'] 0.5686161904729496
Threshold:  ['yes'] 0.5690385672438809
Threshold:  ['yes'] 0.5729169310635398
Threshold:  ['no'] 0.5685368250826528
Threshold:  ['no'] 0.5708706424846622
```

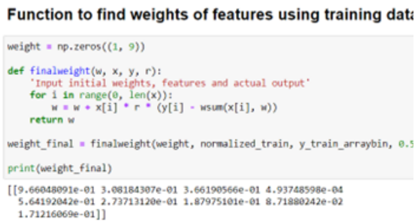


The perceptron algorithm:

The algorithm inputs the trained weights, feature data and what step size to use for the threshold. It outputs an array containing yes/no for each row. For each row of feature data, find the weighted sum, see if the weighted sum is above or below a threshold and output yes/no to an array.

Testing, results and Conclusions:

What features hold the most weight:



Function to find weights of features using training dat

```
weight = np.zeros((1, 9))

def finalweight(w, x, y, r):
    'Input initial weights, features and actual output'
    for i in range(0, len(x)):
        w = w + x[i] * r * (y[i] - wsum(x[i], w))
    return w

weight_final = finalweight(weight, normalized_train, y_train_arraybin, 0.5

print(weight_final)
```

```
[[9.66048091e-01 3.08184307e-01 3.66190566e-01 4.93748598e-04
  5.64192042e-01 2.73713120e-01 1.87975101e-01 8.71880242e-02
  1.71216069e-01]]
```

Feature	year	temp	humidity	Rainfall	drought_code	buildup_index	day	month	wind_speed
Weight	0.966	0.308	0.366	4.93748598e-04	0.5641	0.273	0.188	0.087	0.171

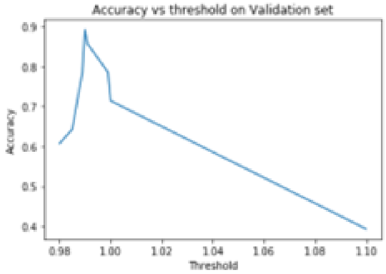
Rainfall was the worst predictor of wildfires. Year was the best predictor of wildfires, but it couldn't have caused fires directly. I assume it is following a trend of greater wildfires from global warming over time.

Accuracy of model:

I tested the accuracy of the model on training, validation and testing sets. I used classification report, confusion matrix and accuracy score. The accuracy scores were 0.69, 0.89 and 0.78 for testing, validation and testing sets respectively.

Accuracy Curve:

I plotted an accuracy curve to find the best threshold to divide yes/no data. I used the validation set. The most accurate value seemed to be a threshold of 0.99. If the weighted sum of a row is greater than this it's classified as yes and if it's less than this it's classified a no.



Comparison of my algorithm with reference:

I used a reference algorithm from Sklearn. The following table compares the accuracies of the two algorithms.

	My algorithm	Reference
Train	Accuracy: 0.69	Accuracy: 0.87
Validation	Accuracy: 0.89	Accuracy: 0.75

Test	Accuracy: 0.78	Accuracy: 0.56

My algorithm from scratch outperformed the reference algorithm for validation and testing sets.

All the results are from random state = 5

I repeated this for 4 other random states , random states =1,2, 3,4

The testing accuracies for my algorithm were: 0.51, 0.42, 0.48, 0.42

The testing accuracies for reference algorithm: 0.87, 0.66, 0.86, 0.81

My accuracies were very low because I would need to change my threshold for each random state. The model will work for any inputted data but not for any random state. If I had more time I could repeat my process and find an accuracy curve for each random state so find a good threshold for each of the random state.