**Machine Learning Algorithms**

**Dataset:** Supermarket Dataset

**Rows:** 1000

**Attributes:** 87 explanatory variables (categorical)

**Predictor:** Expenditure is High or Low

**Dataset:**

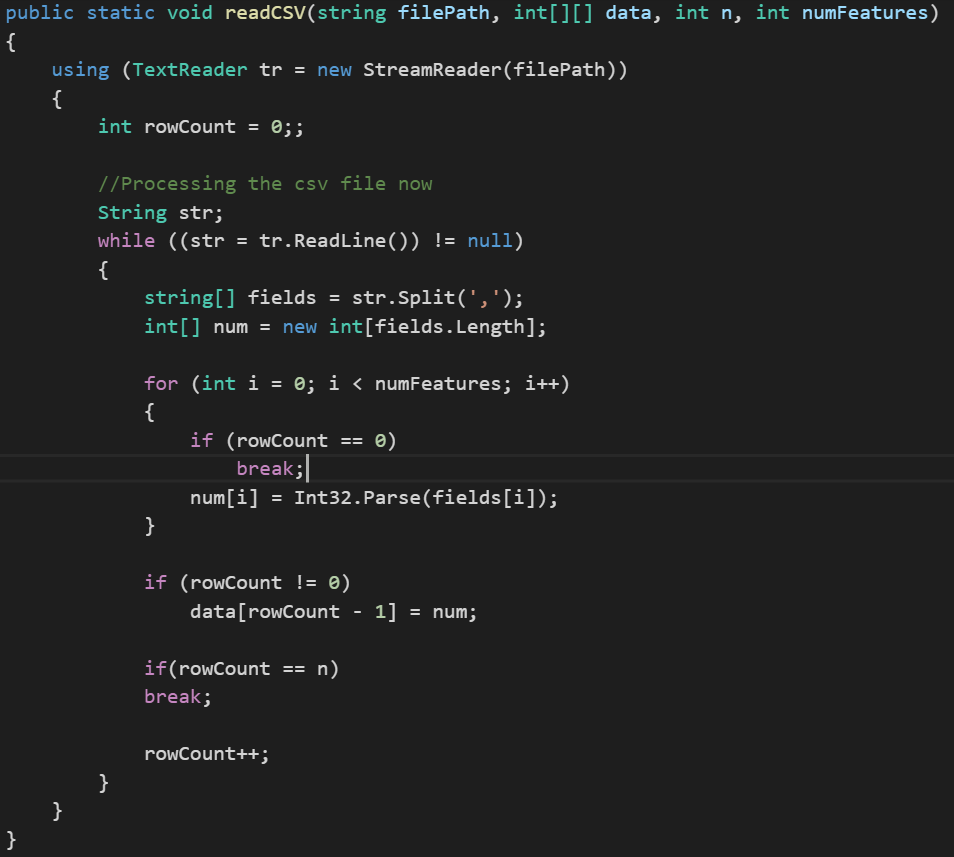
The dataset describes the purchase of a person on various departments and food items in a supermarket. Each person’s expenditure should be classified as Low or High. The expenditure pattern of 1000 person on 87 department or food items has been recorded.

**Winnow Algorithm**

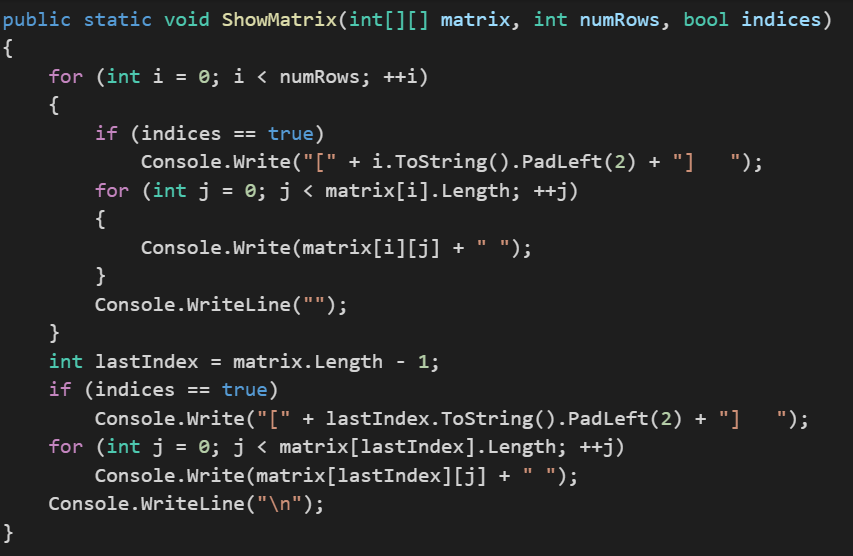
Since most of the explanatory variables are categorical, we can use the Winnow Algorithm. As per the Winnow Algorithm, train a binary classifier based on binary features, using a linear decision boundary. The prediction model assign weights to each future. To predict the state of an observation, check all the “**active**” features and sum up the weights assign to the feature. If the total is above a certain threshold, the result is true, otherwise it’s false. Winnow algorithm updates itself with learning process, it updates the weights to the feature when the model make mistakes. If the current model predicts the output correctly, don’t change anything. If it predicts true but should predict false, it is over-shooting, so weights that were used in the prediction (i.e. the weights attached to active features) are reduced i.e., divide the weights of every feature by alpha. Conversely, if the prediction is false but the correct result should be true, the active features are not used enough to reach the threshold, so they should be bumped up i.e., multiply the weights of every feature by alpha.

**Code Explanation:**

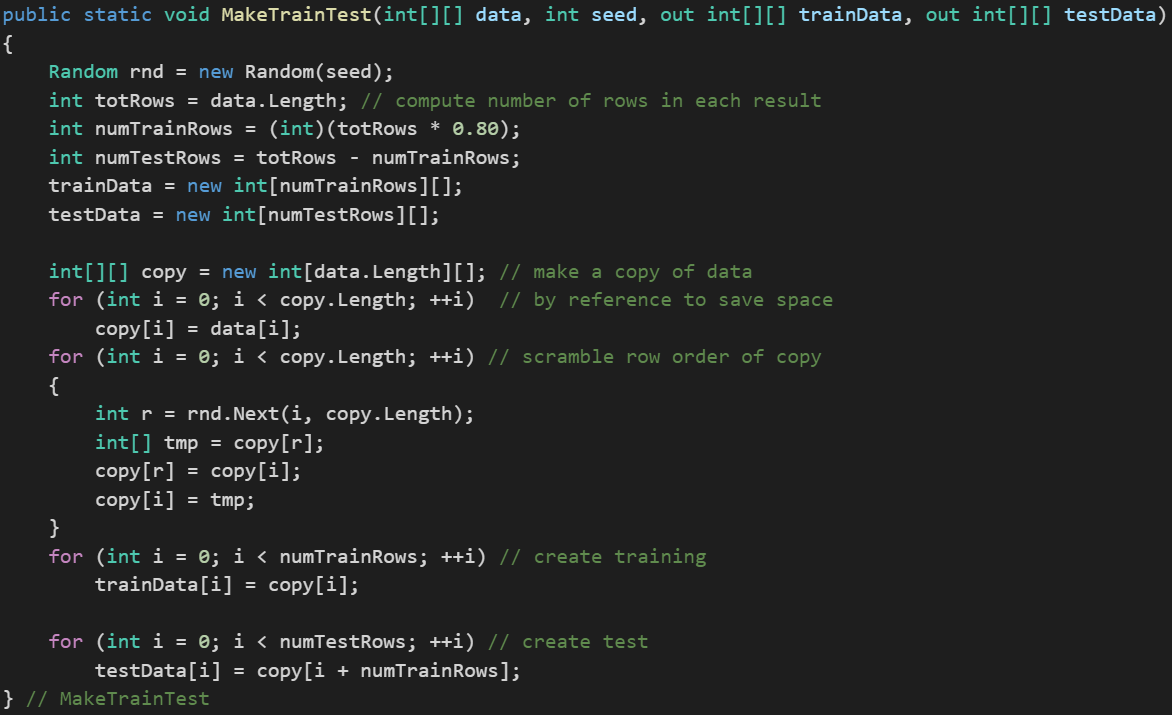
To run the algorithm, we are going to read the csv, **readCSV** function is to read the csv, and form a matrix, which is later used for training the model and later predicting the expenditure based on the observation or features given.



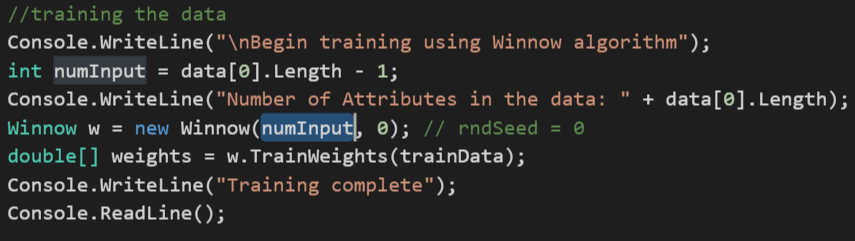
Once the dataset is formed, we can use **showMatrix function,** this function is to display first and last few rows of the dataset, it takes the dataset which we formed, the number of rows you want to display and boolean indices, do you want to show the index number of row or not.

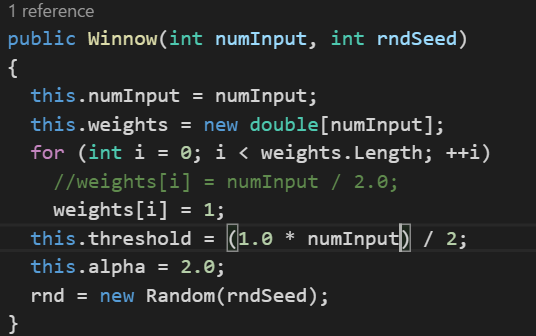


Once we have verified the dataset, we can now make the train and test from the dataset. For this we have **MakeTrainTest function,** this function make train and test data from the dataset, we split the data into 80% and 20%, 80% is to train the model based on Winnow Algorithm and rest 20% is to predict the accuracy of the algorithm. Before splitting the dataset, we set the seed value, seed value is a Random number which is used for generating the same trainData again. The function is going to output the result into **trainData** and **testData.**

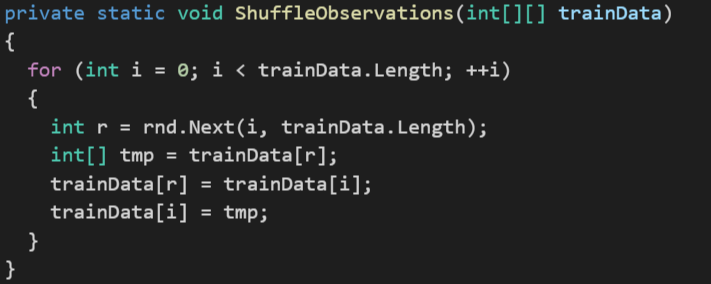


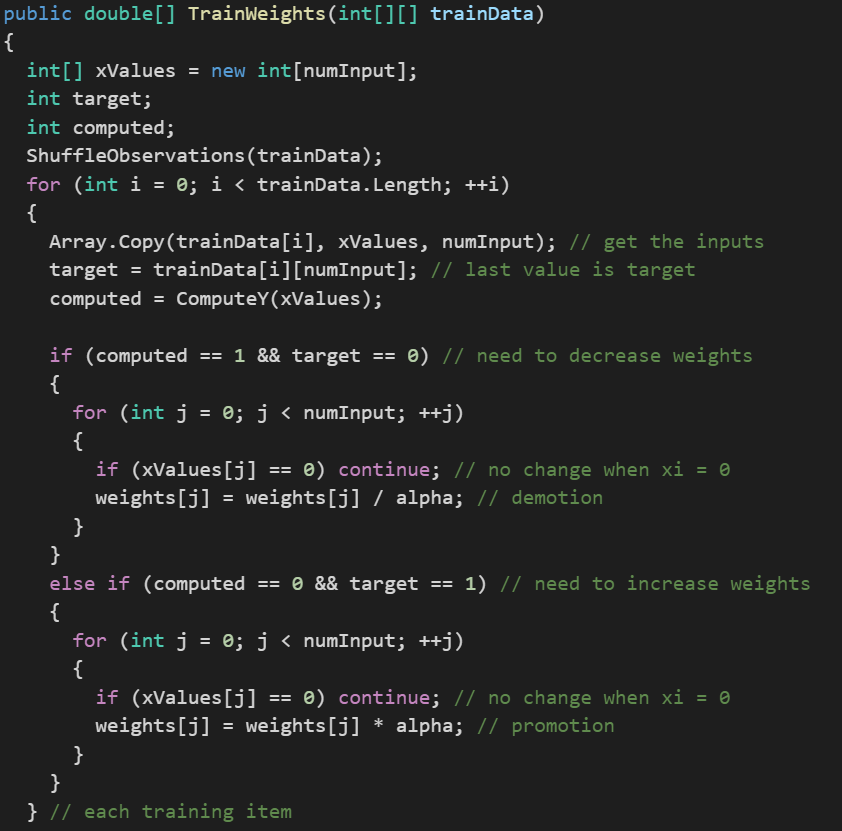
Once we have trainData and testData, we can now make the model by implementing Winnow Algorithm on the train data. We are going to initialize the winnow with the number of attributes/features, in our case they are 87 and then we call the **TrainWeights**, which is going to compute the weights of the feature. In the **Winnow class,** we have **assigned initial weights as 1** and **threshold as total features divide by 2** and **alpha value as 2**. Aplha is the value by which we are going to multiply or divide the weights.





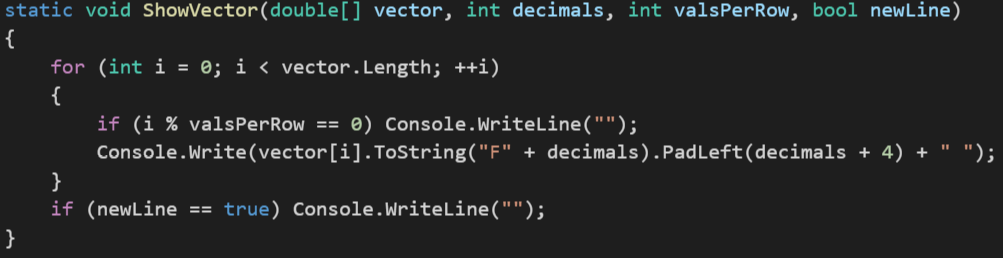
In the **TrainWeights function,** we are going to use the initial weights and then update the weights based on the conditions. If the current model predicts the output correctly, don’t change anything. If it predicts true but should predict false, it is over-shooting, so weights that were used in the prediction (i.e. the weights attached to active features) are reduced i.e., divide the weights of every feature by alpha. Conversely, if the prediction is false but the correct result should be true, the active features are not used enough to reach the threshold, so they should be bumped up i.e., multiply the weights of every feature by alpha. While training the model, we shuffle the observation as for computing the weights winnow use the train data. We are using the Fisher-Yates shuffle algorithm.



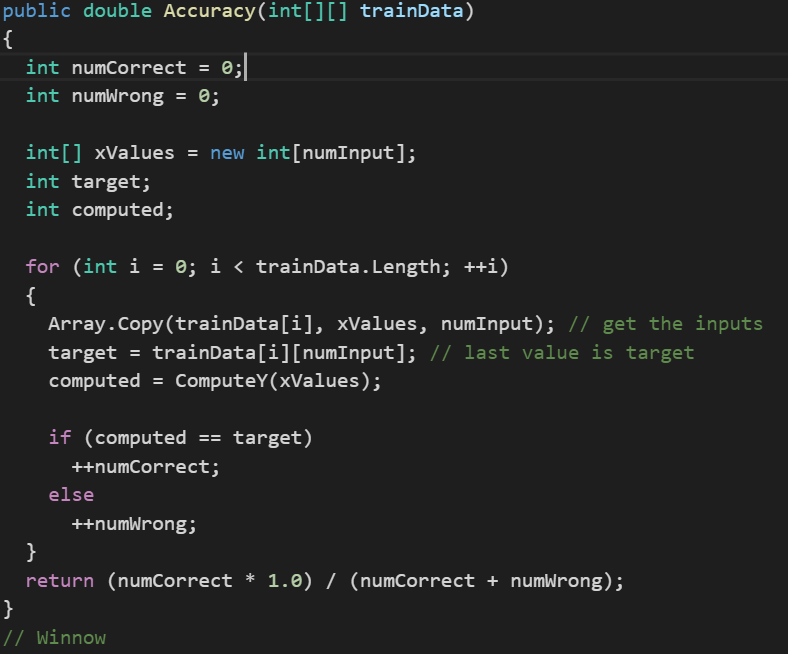


We have another **showVector function,** this function is used to display the final weights of the features, which is

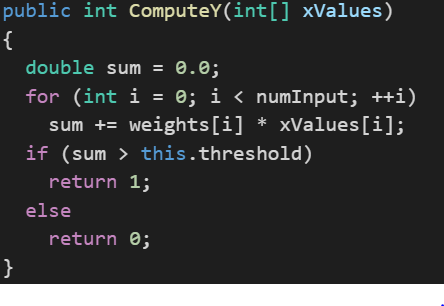
assigned to them by them by Winnow, based on the update function of winnow, it take number of features display in the row and whether you want everything to be in one line or more



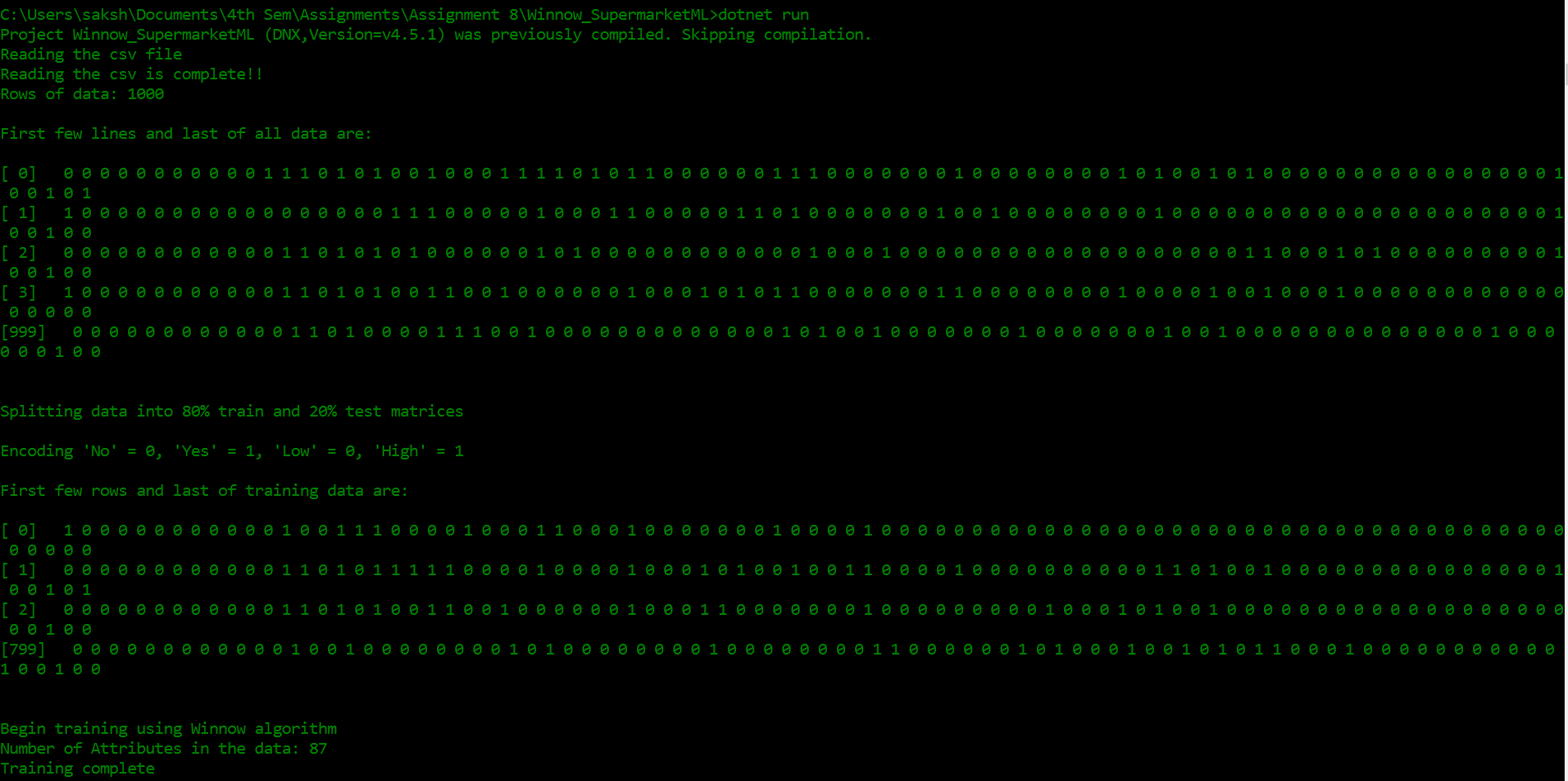
We can find the accuracy as well, after computing the weights of the features, this function computes the accuracy of the dataset, based on the weights which we have computed, what it does is it applies the algorithm to the dataset and try to predict the severity of the claim and then it compares computed value with the target value, later it gives the total correct divide by total number of observation.

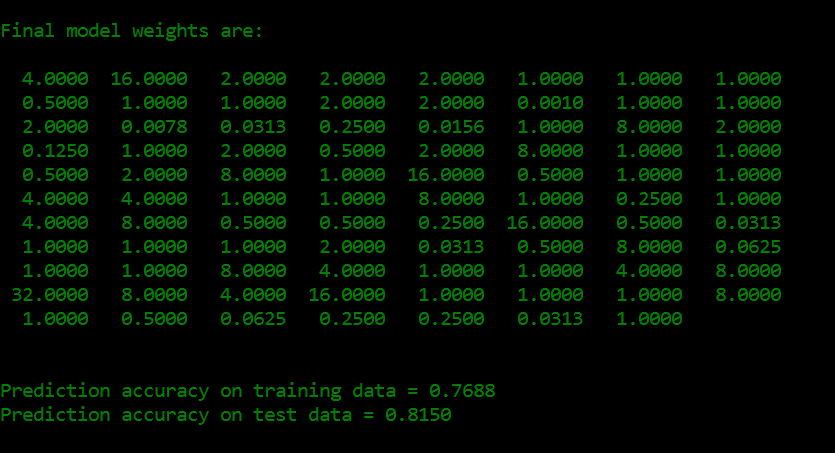


We have another **computeY function**, t his function is to predict the value of dependent variable, what it does is sum up the weights of all the feature whose value is 1 if sum is greater than threshold, it returns 1, else it return 0



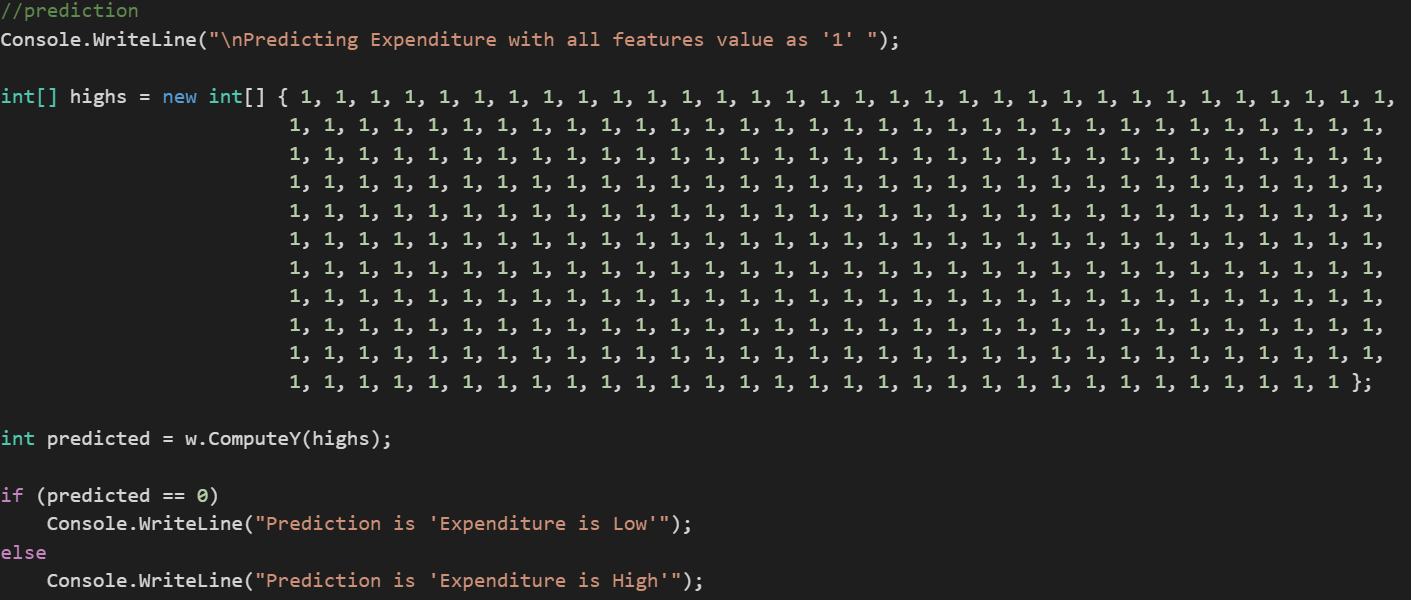
**Output**

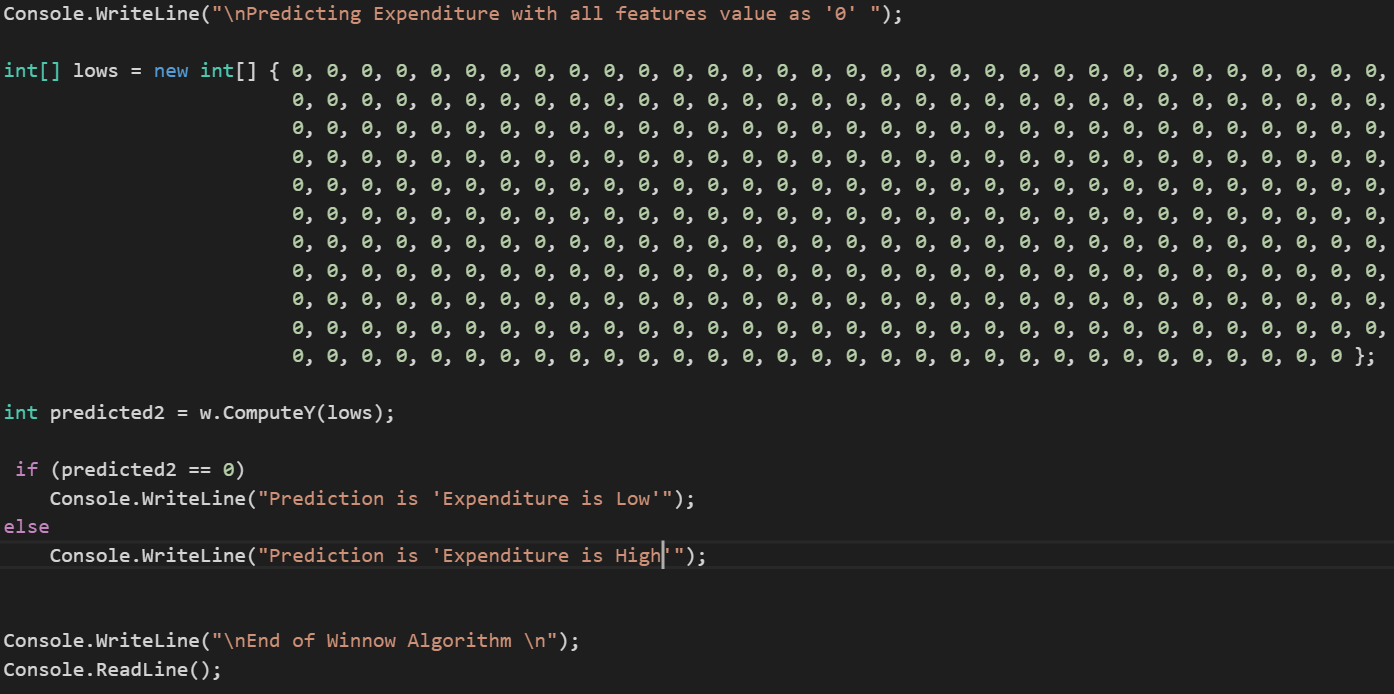


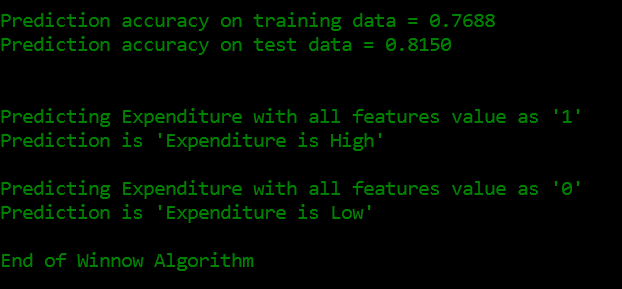


**My algorithm accuracy is 76.88% for train data and 81.50% for test data.**

After this we predicted the values of expenditure, one with all 1’s and others with all zero’s







1. **How could you improve on Winnow?**

**Solution:**

**To improve the accuracy of the Winnow Algorithm in cases** where we are not getting the appropriate result, we should shuffle the data**, we should come up a new random scrambling method** and run the TrainWeights functions i.e, **update function multiple times**, which will lead to better weights to the important feature and eventually that will lead to a better prediction. Winnow Algorithm is for linear function between dependent and independent variable in case of non-linear relationship between variables, we can introduce multiple hidden layers, this is concecpt behind Artifical Neural Network.

1. What’s the dataset size after which the winnow algorithm becomes unwidely?

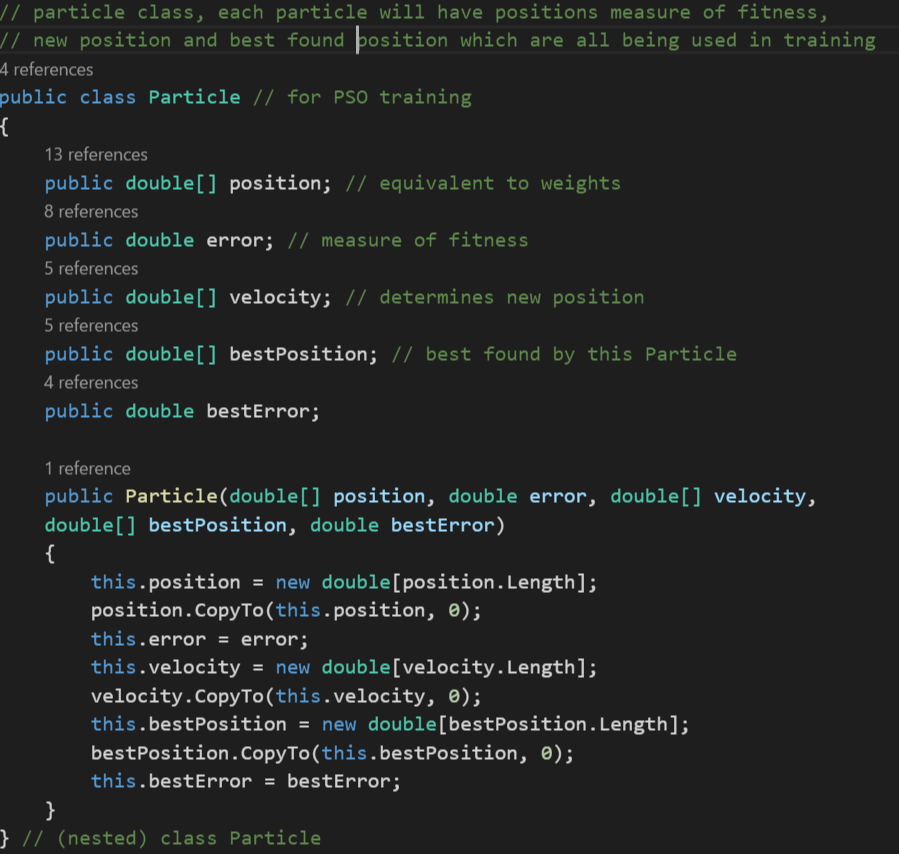
**Solution:**

The dataset size for which winnow algorithm behaves unwidely is when datasets are small. It works with a very small loss in performance when a larger set is used, and in fact in some cases performance actually improves. The main performance loss is one of speed, but even this can be lessened, especially in the case of Weighted Majority, by a pruning method. Also Winnow gives better result when the number of features on which the weights depend are more. The dependent features gives better accuracy result. Winnow is much more robust in high dimension feature. Winnow perform better in cases when relationship is linear amony dependent and independent variables.

**Particle Swarm Optimization**

Particle swarm optimization (PSO) is a population based stochastic optimization technique. PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA), which we are going to implement next. The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. Compared to GA, the advantages of PSO are that PSO is easy to implement. PSO simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So, what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.



PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest.

After finding the two best values, the particle updates its velocity and positions with following

***v = v + c1 \* rand() \* (pbest[] - present) + c2 \* rand() \* (gbest[] - present)***

***present = persent + v***

v is the particle velocity, persent is the current particle (solution). pbest[] and gbest[] are defined as stated before. rand () is a random number between (0,1). c1, c2 are learning factors. usually c1 = c2 = 2. The code will stop when the maximum iterations, which is 1000 in our case or minimum error criteria is attained.

Overfitting generates non‐smooth prediction curves. Usually characterized by weights that have very large or very small values. Therefore, one way to reduce overfitting is to prevent model weights from becoming very small or large. In order to prevent the magnitude of model weight values from becoming large, the idea of regularization is to penalize weight values by adding those weight values to the calculation of the error term. If weight values are included in the total error term that’s being minimized, then smaller weight values will generate smaller error values

L1 weight regularization penalizes weight values by adding the sum of their absolute values to the error term

***L1 = sum((oi‐ti) \* (ai-ti) /n) + (alpha) \* sum(Math.Abs(wj))***

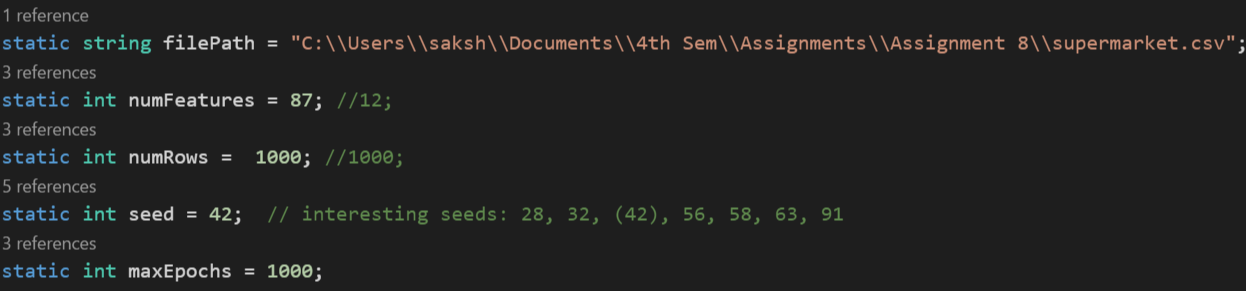
L2 weight regularization penalizes weight values by adding the sum of their squared values to the error term

***L2 = sum((oi‐ti) \* (ai-ti) /n) + (alpha) \* sum(Math.Abs(wj) \* Math.Abs(wj))***

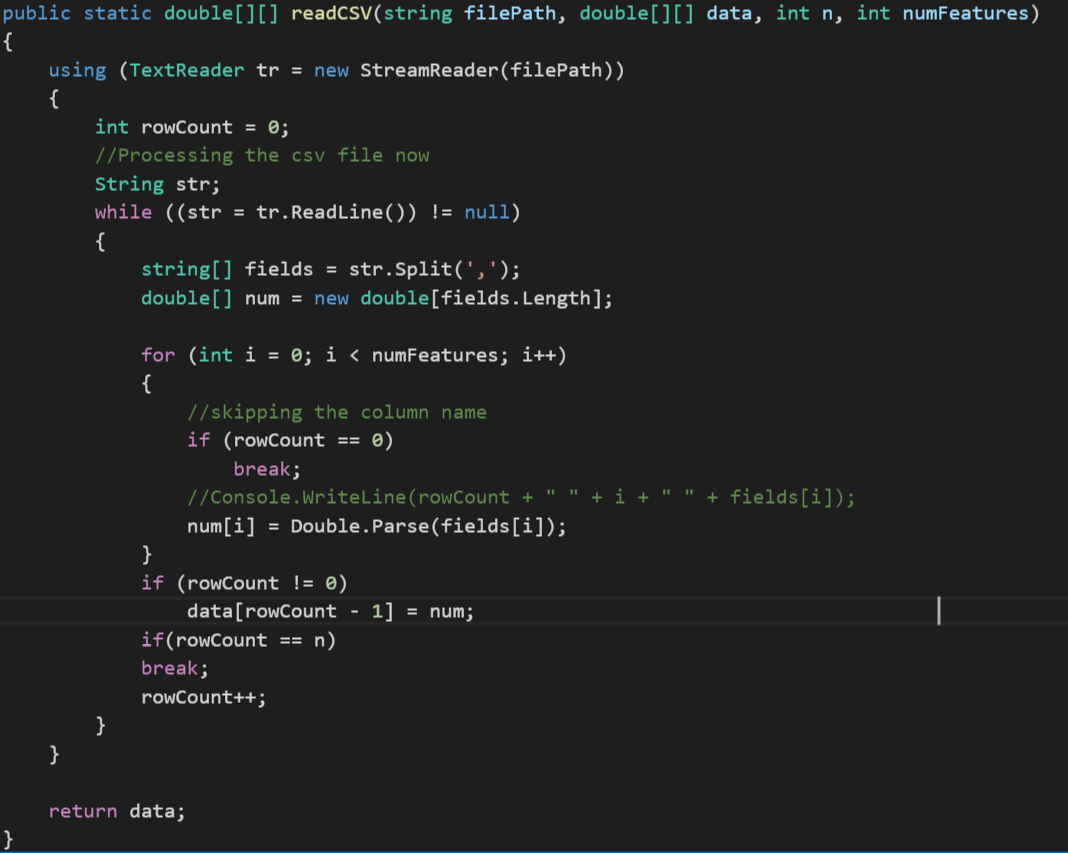
L1 regularization, in contrast to L2, can sometimes have a beneficial side effect of driving one or more weight values to 0.0, which effectively means the associated feature isn’t needed, like with the Winnowlinear model (feature selection)

**Code Explanation:**

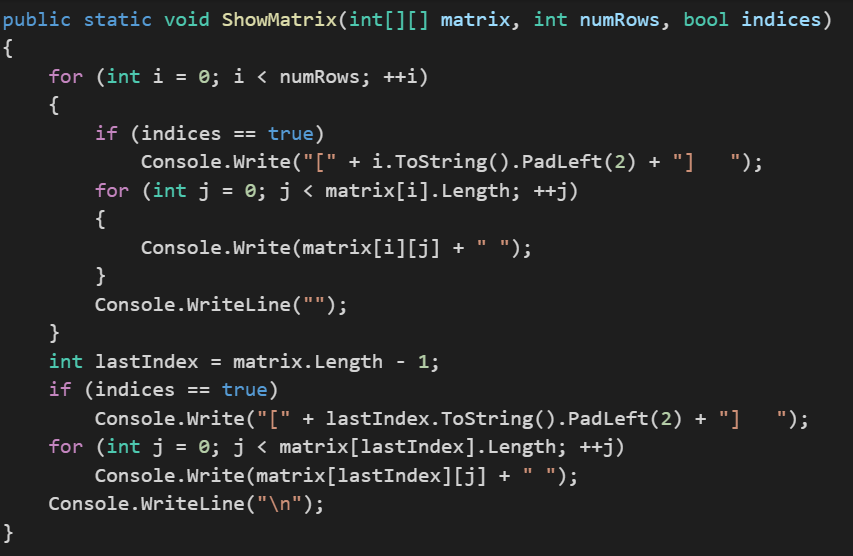
The dataset has 87 features and 1000 rows so the values are set at the top to be used throughout the code. The seed has been selected to 42



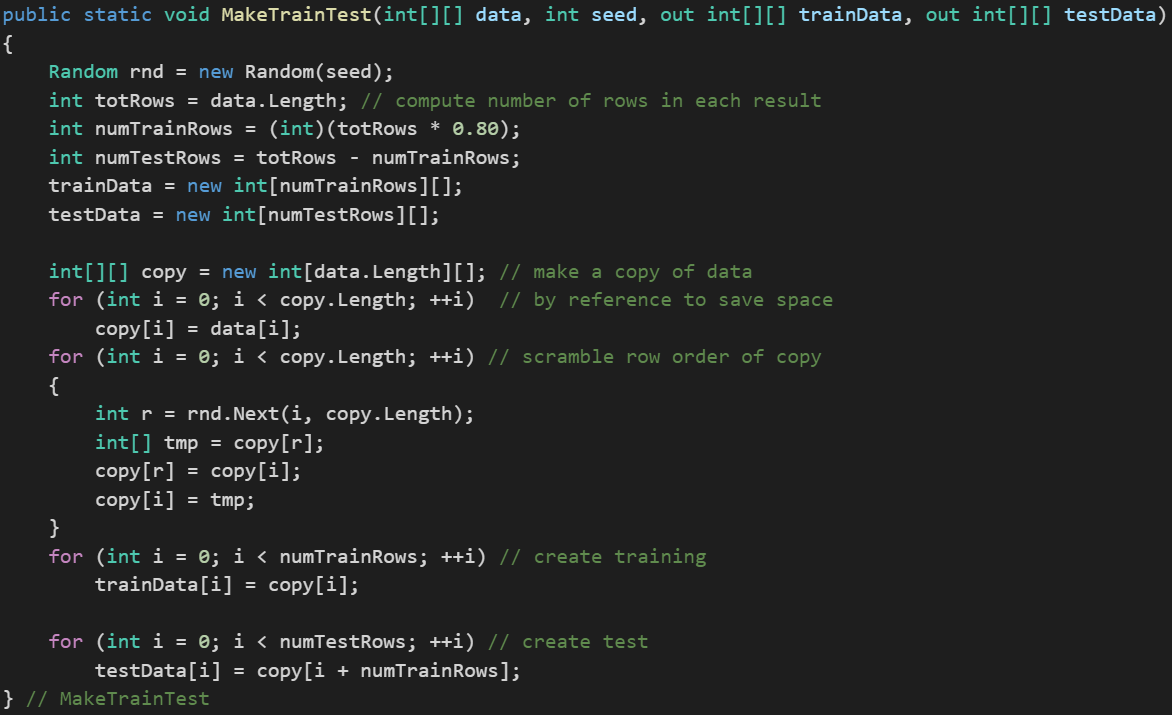
To run the algorithm, we are going to read the csv, **readCSV** function is to read the csv, and form a matrix, which is later used for training the model and later predicting the expenditure based on the observation or features given. The code to read the csv file which has been called from the main function. It splits the data with the delimiter “,”. The values of csv file get stored in a double array called data which is returned by the function and later used to train the weights and calculate the regularization.



Once the dataset is formed, we can use **showMatrix function,** this function is to display first and last few rows of the dataset, it takes the dataset which we formed, the number of rows you want to display and boolean indices, do you want to show the index number of row or not.

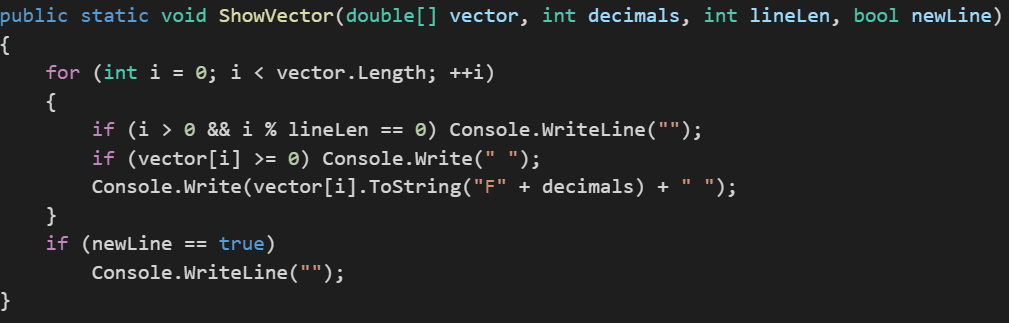


Once we have verified the dataset, we can now make the train and test from the dataset. For this we have **MakeTrainTest function,** this function make train and test data from the dataset, we split the data into 80% and 20%, 80% is to train the model based on Winnow Algorithm and rest 20% is to predict the accuracy of the algorithm. Before splitting the dataset, we set the seed value, seed value is a Random number which is used for generating the same trainData again. The function is going to output the result into **trainData** and **testData.**

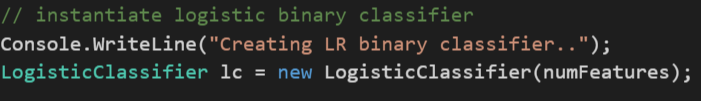


We have another **showVector function,** this function is used to display the final weights of the features, which is

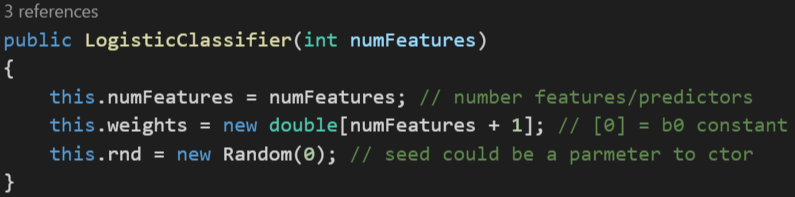
assigned to them by them by Winnow, based on the update function of winnow, it take number of features display in the row and whether you want everything to be in one line or more



We begin with Binary Classification with creating the instance of LogicalClassifier which takes the number of features as the parameter.



The weights, random number and the number of features get assign with the value in the constructor of LogicalClassifier



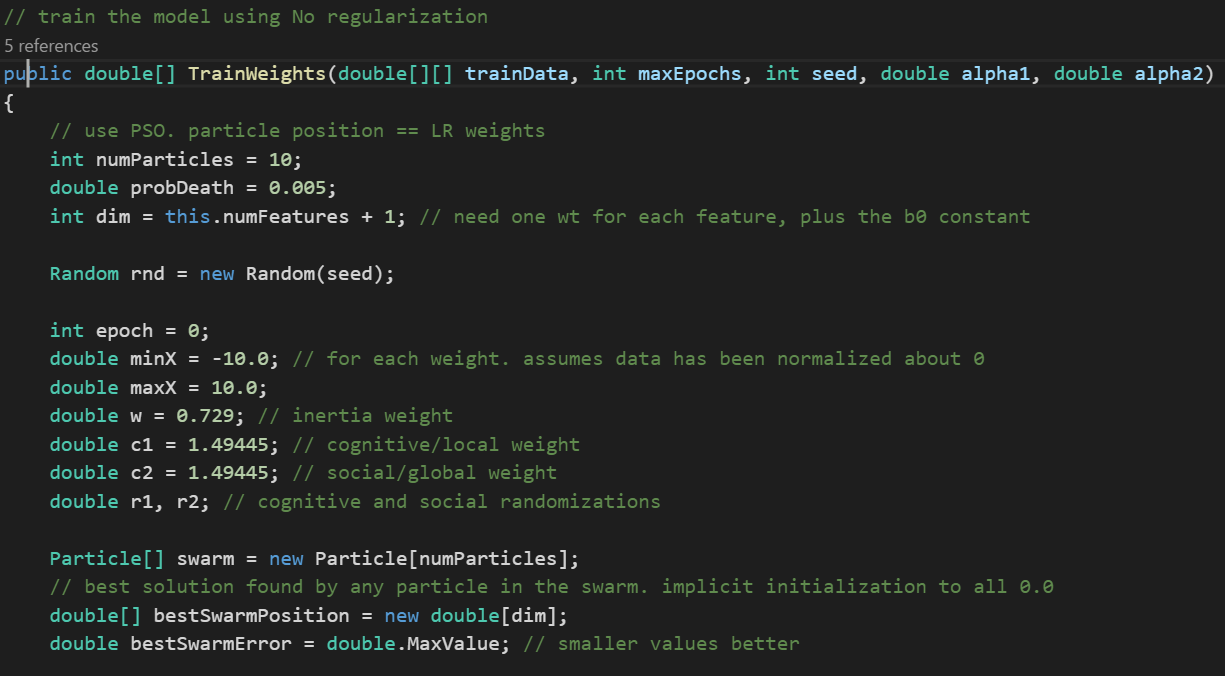
Once we have trainData and testData, we can are now going to implement three models

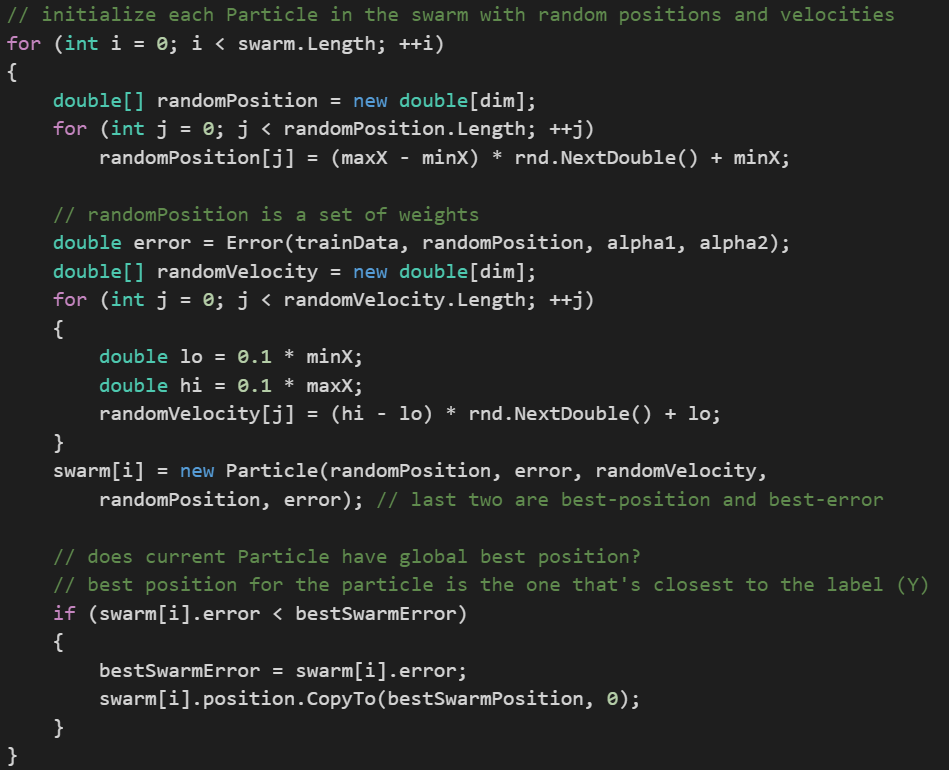
1. **No Regularization**

In no regularization, the alpha1 and alpha2 value as 0.0 and then train the dataset and predict the accuracy

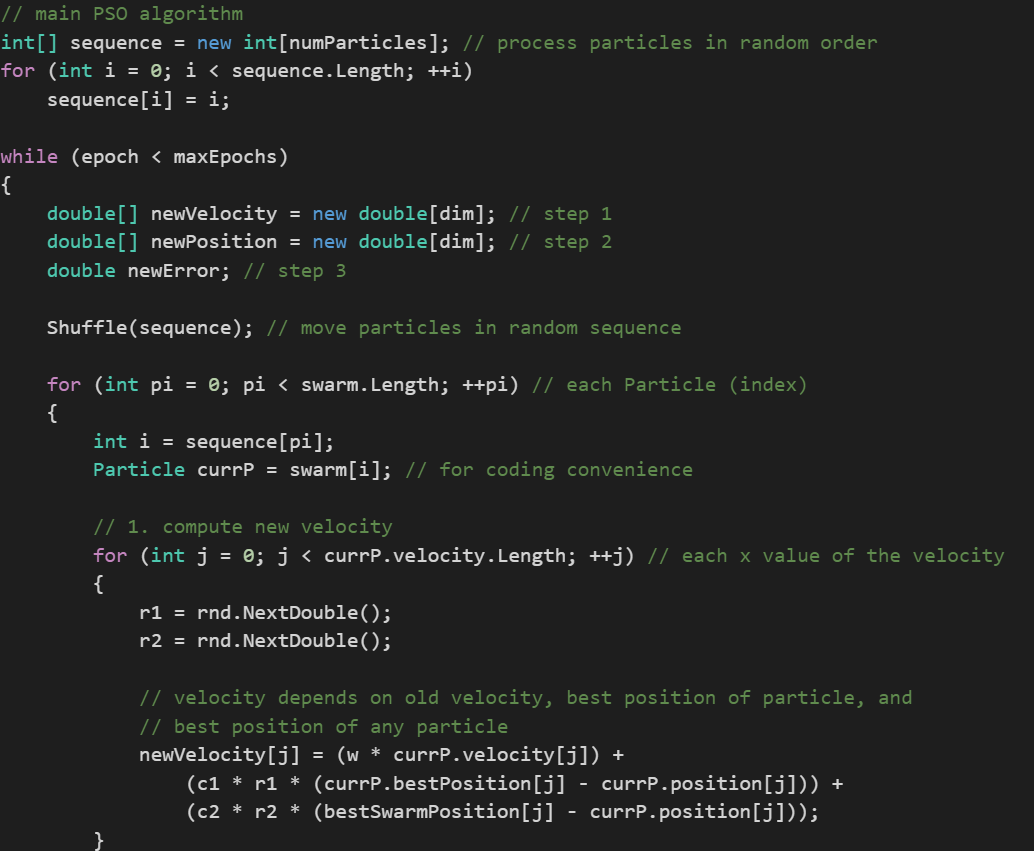


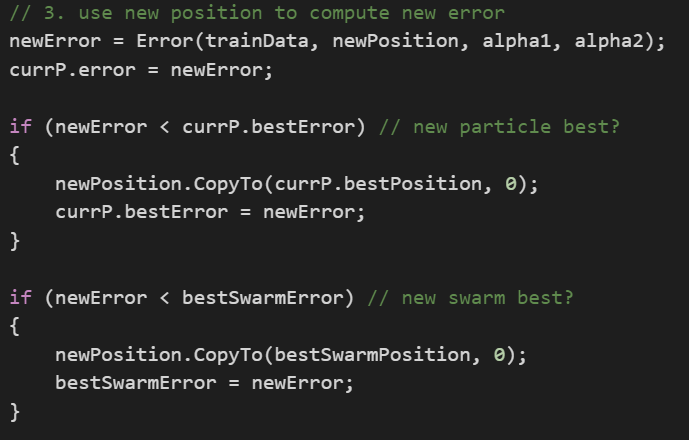
To calculate the weights, we first initialize each particle or a feature with random position and velocities. Initial value of minX and maxX is -10 and 10 respectively. The random position is set of weights





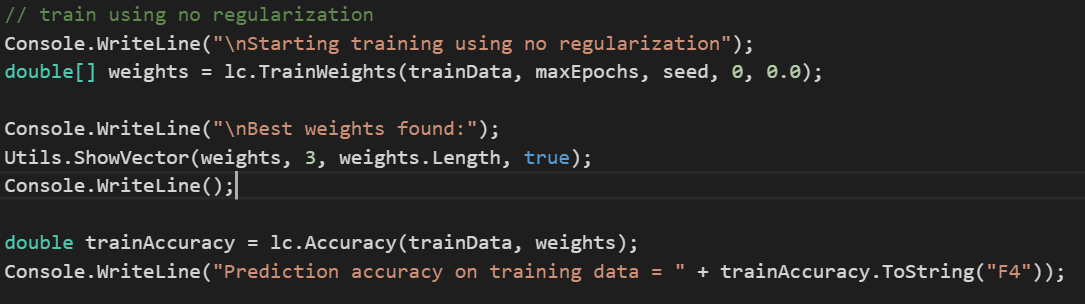
We shuffle the particles aka weights randomly. We then calculate the new velocity of each particle depending upon its best position and current position, old velocity. Using the new velocity the new position of the particle is calculated. The new position we have calculated, we use that to compute the new error. This was we calculate the weights of the particles

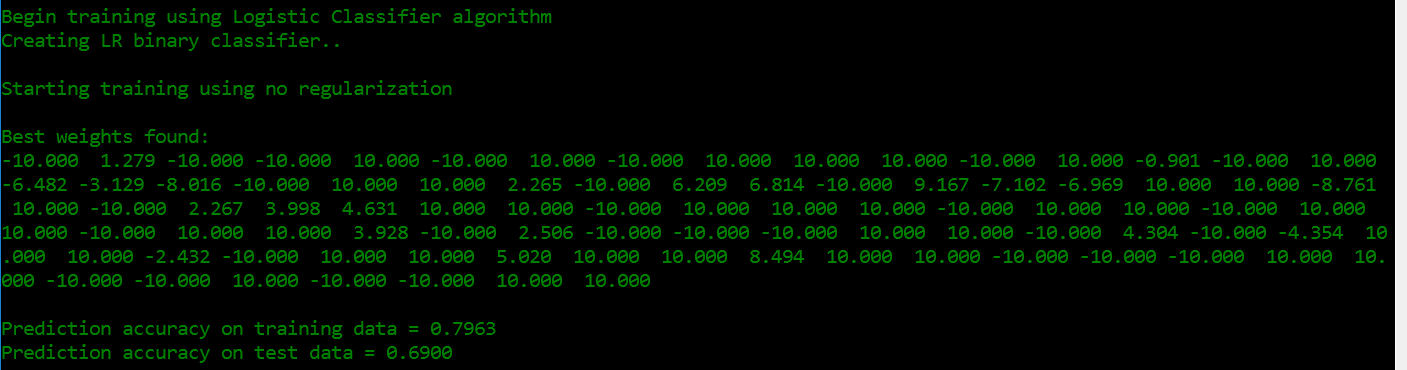






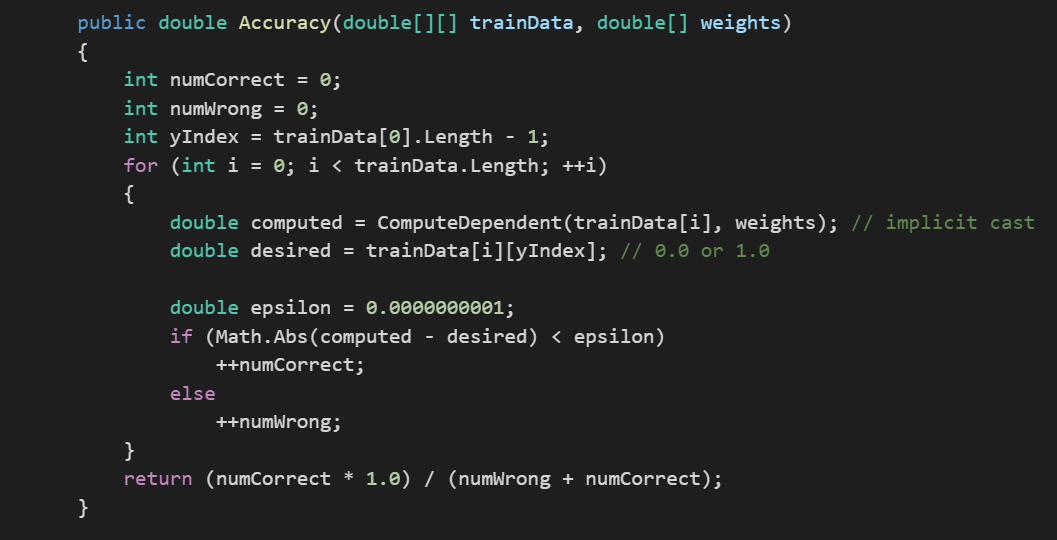
The train weights code is run for epcohs we had defined earlier. In our code we have set the epoch value as 1000 so the weights will be trained 1000 times and at the end we will get the best weights after 1000 iterations. As an output it returns the weights of the features which will be used to calculate the accuracy of the train data and test data



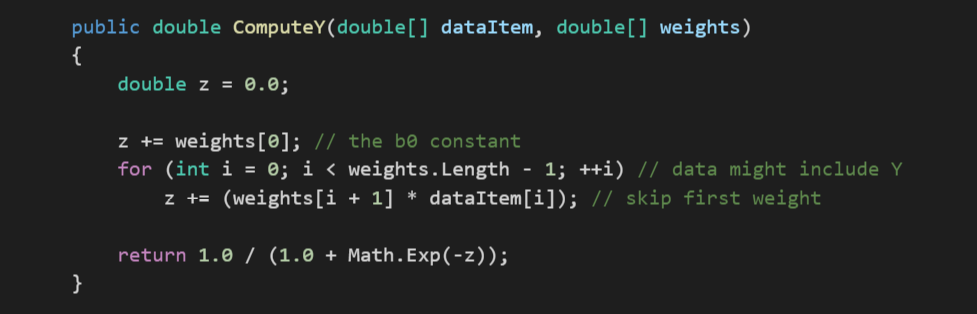


**The prediction accuracy for train data is 79.63% and for test data is 69%.** There must be a change of overfitting, but overfitting occurs only when we have small or large weights. To make sure model does not overfit the dataset, we are going to use L1 and L2 Regularization.

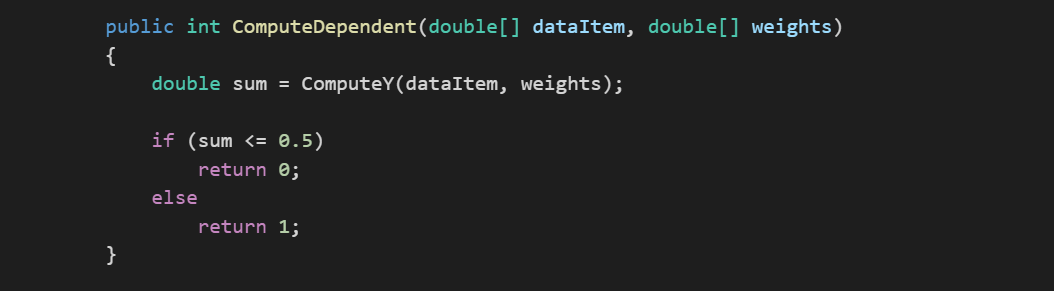
Above computed weights we can use to calculate the accuracy of the train data and the test data. Accuracy is calculated by dividing number of 1’s with the total number of values.



we sum up the weights and the value of the data and return the value (1/(1+e^-z))

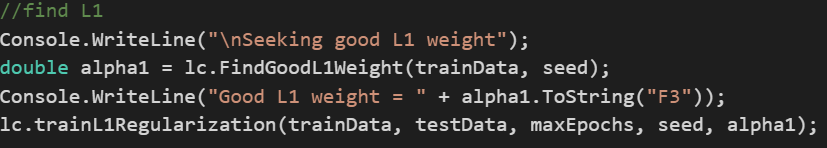


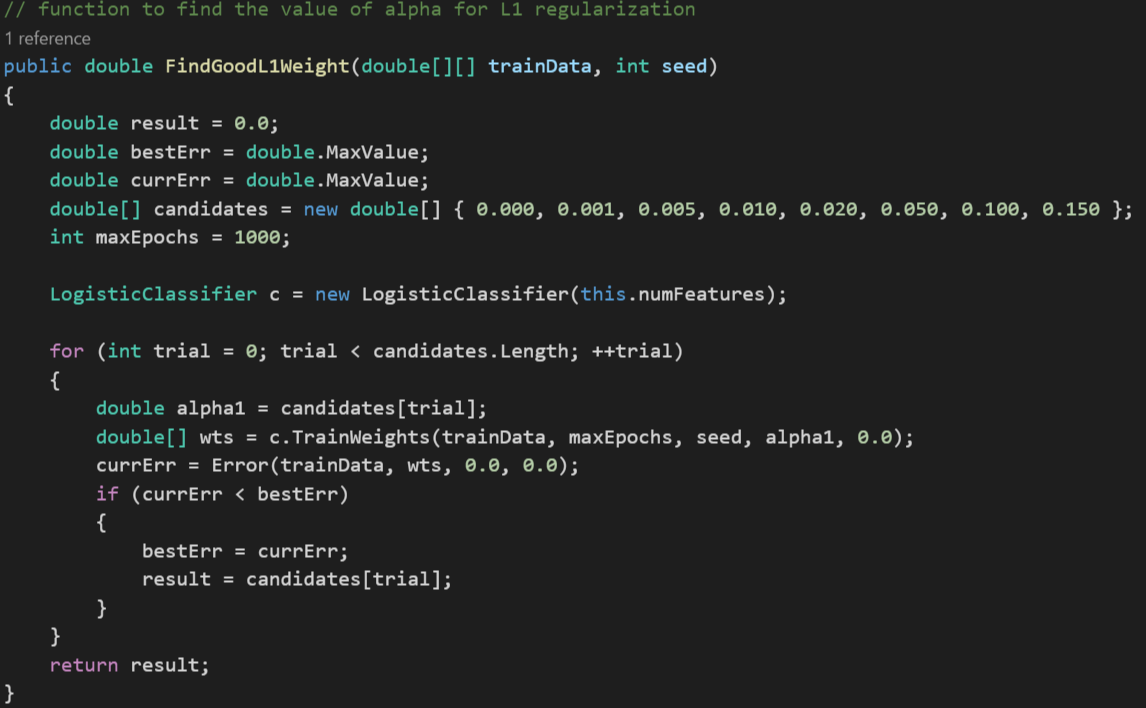
If the value returned is less than 0.5 then 0 is returned else 1 is written



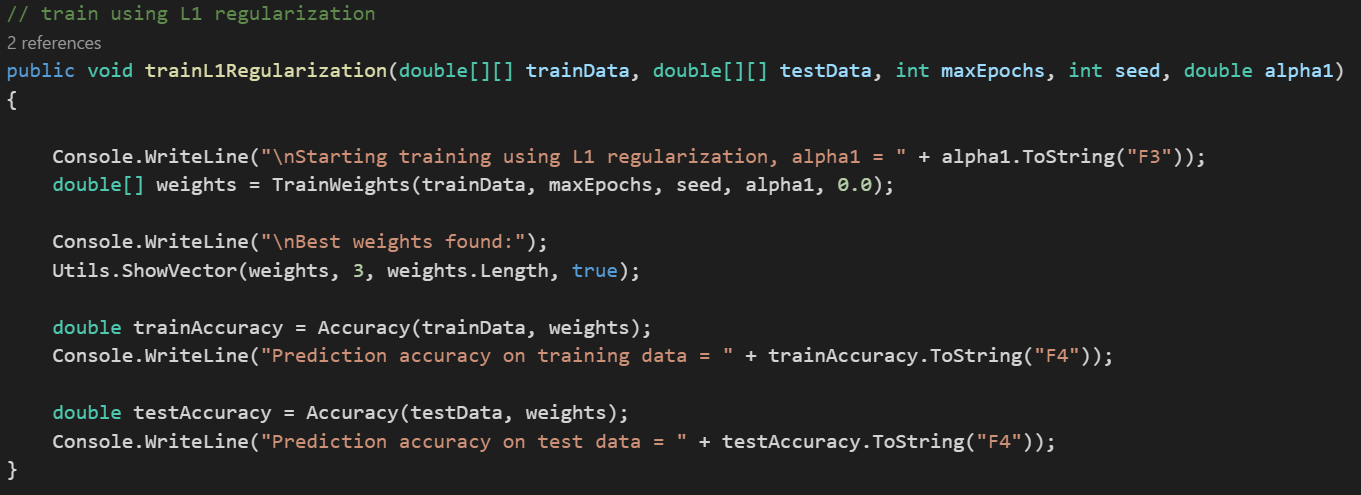
1. **L1 Regularization**

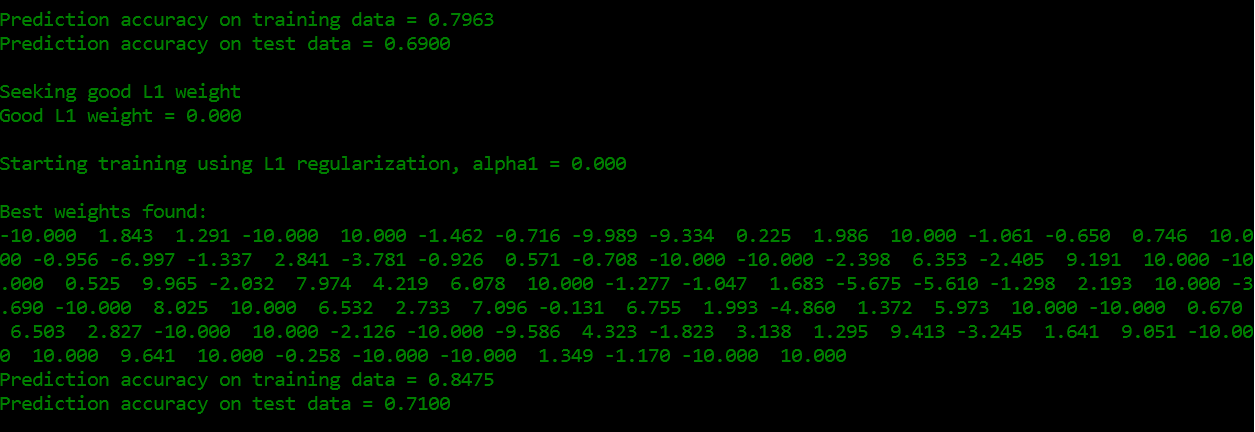
To implement L1 regularization, we are going to compute the alpha value using **FindGoodL1Weight function,** and then train the model using **trainL1Regularization** based on that alpha value and then predict the accuracy of the model





While training the model, we are going to add the absolute value of the weights to the error, so that the error values become large, if the values become large of weights.





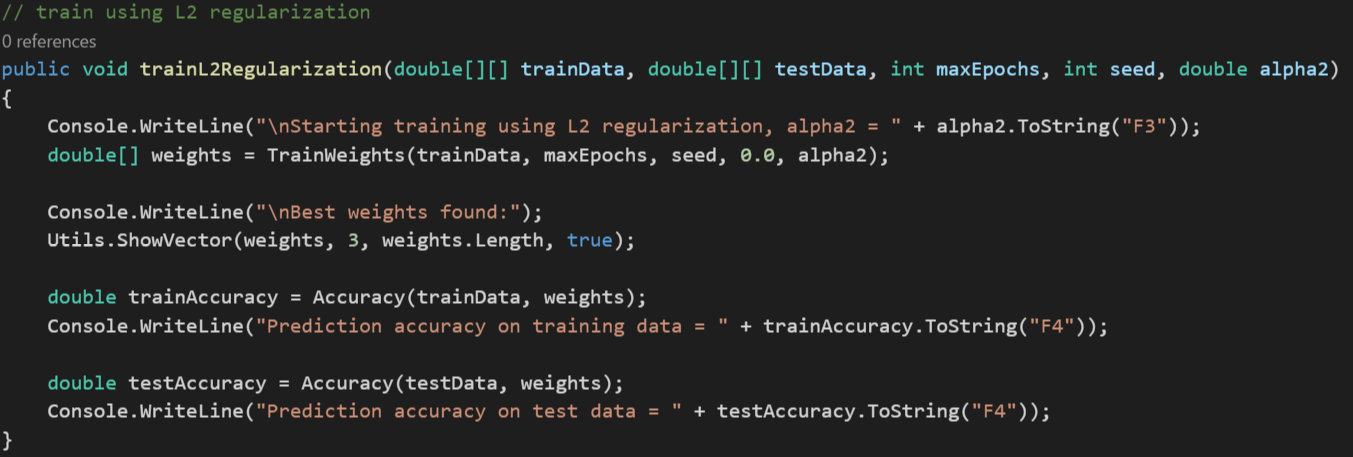
**The prediction accuracy for train data is 84.75% and for test data is 71%.**  We are also going to implement the L2 regularization.

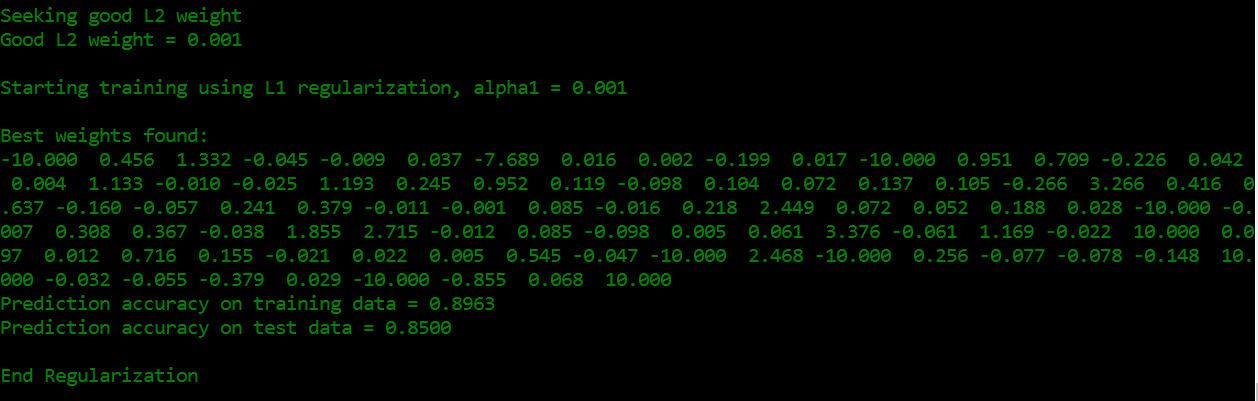
1. **L2 Regularization**

To implement L2 regularization, we are going to compute the alpha value using **FindGoodL2Weight function,** and then train the model using **trainL2Regularization** based on that alpha value and then predict the accuracy of the model



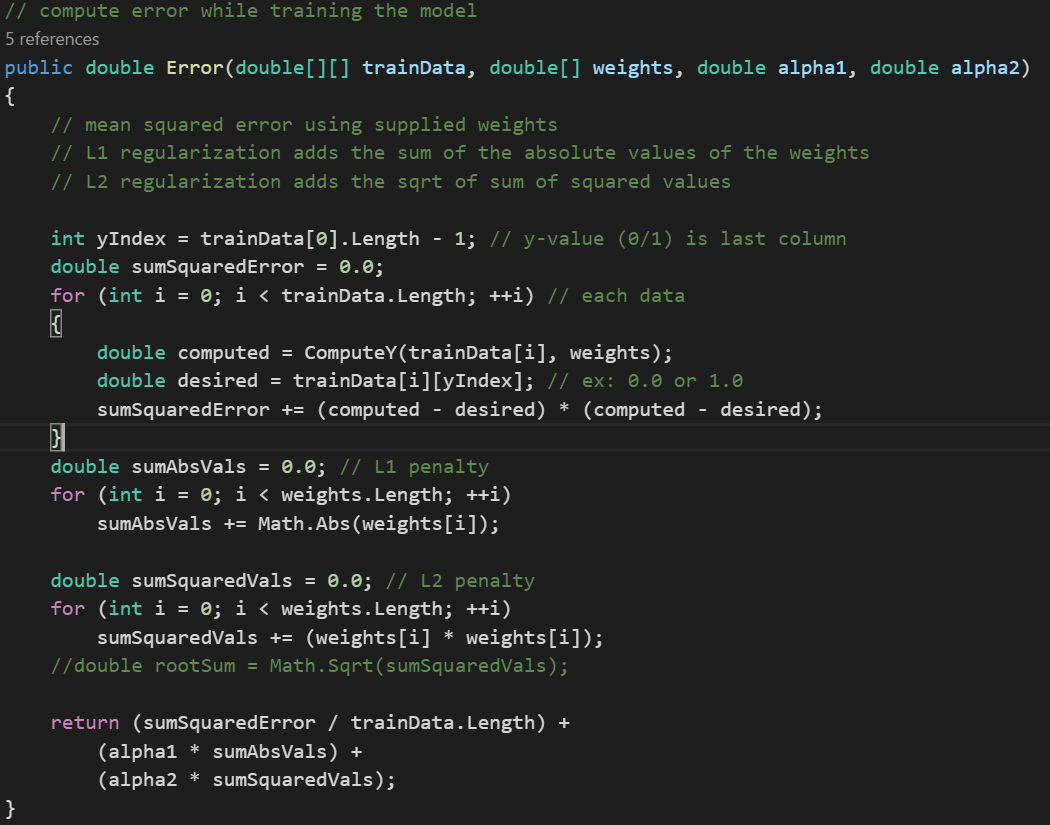
While training the model, we are going to add the square of the weights to the error, so that the error values become large, if the values become large of weights.





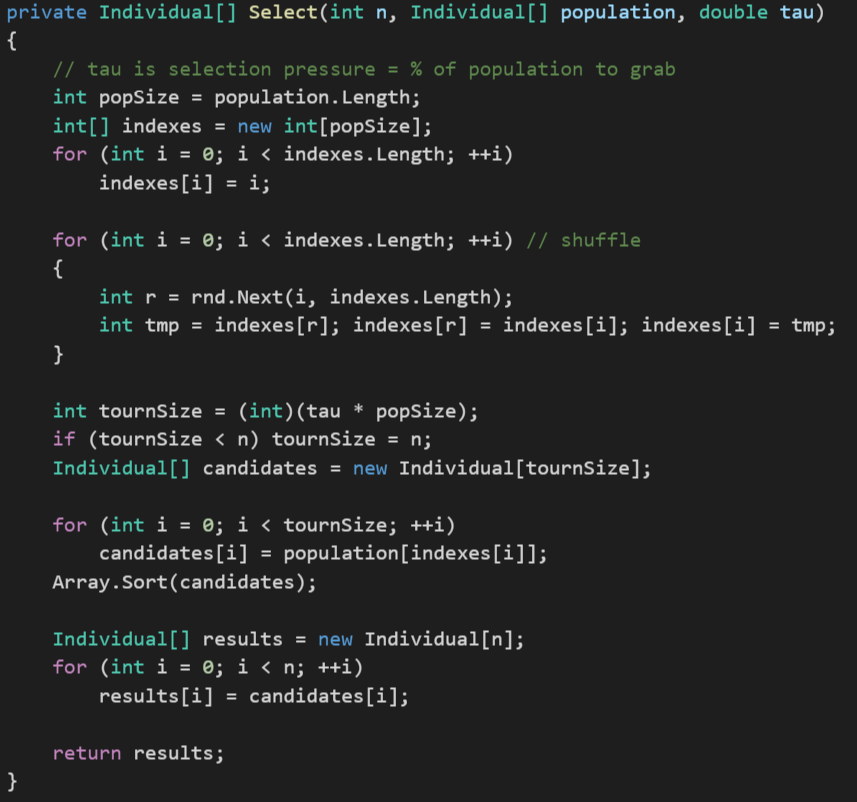
**The prediction accuracy for train data is 89.63% and for test data is 85%**

**The function to compute the error**, which is going to use while training the model. While training the model for L1 regularization, alpha2 will be zero and while training the model for L2 regularization, alpha1 will be zero

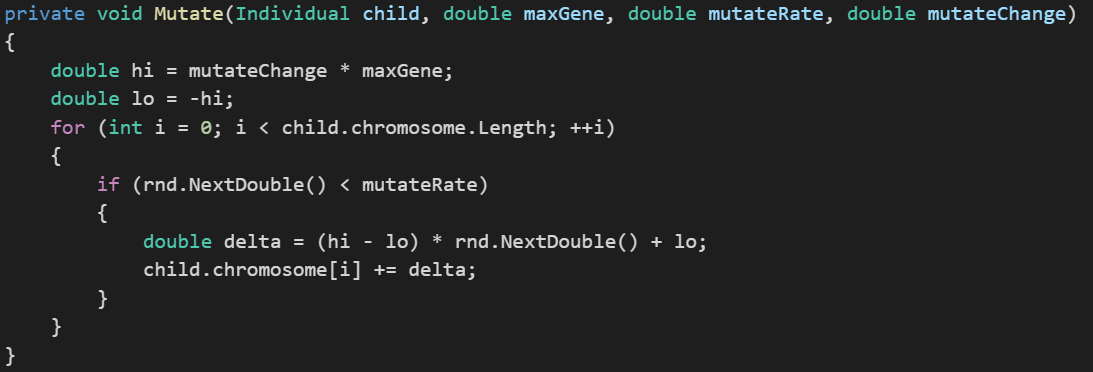


**Evolutionary Optimization**

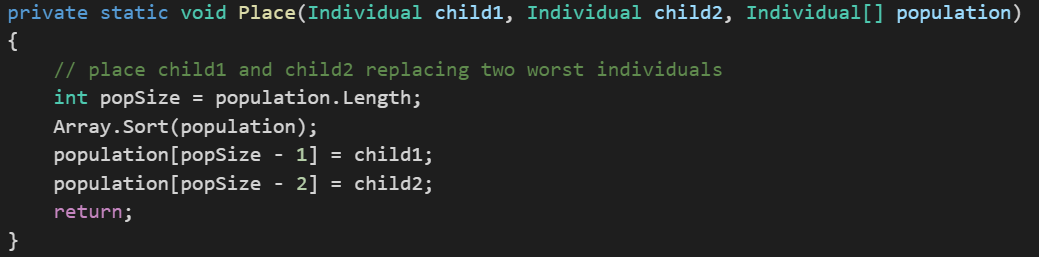
1. Select two good, but not necessarily best, parent individuals from the population



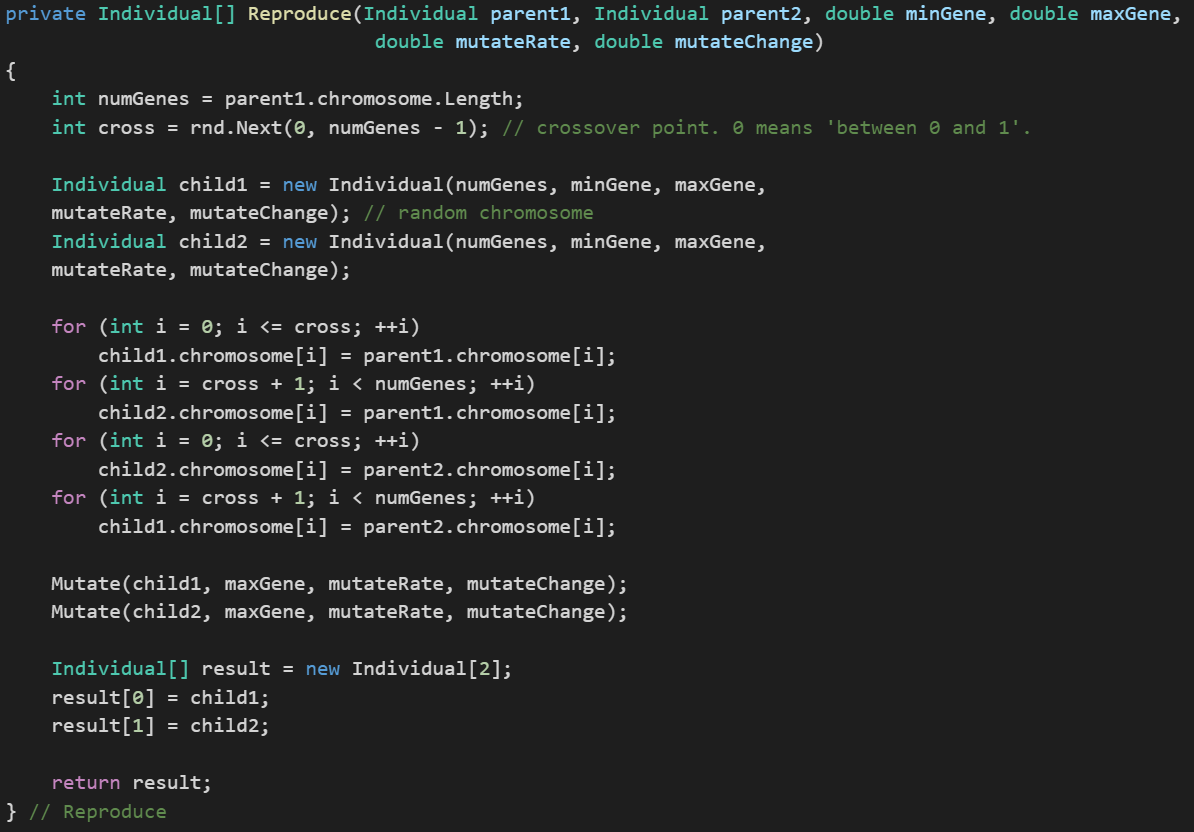
1. Crossover is used to generate two child individuals, which hopefully combine good characteristics of the parents to give even better solutions
2. Two children are randomly mutated slightly to introduce new information into the system



1. The children are placed into the population, replacing two weak individuals



Steps one through four are collectively called reproduction and are core evolutionary optimization mechanisms



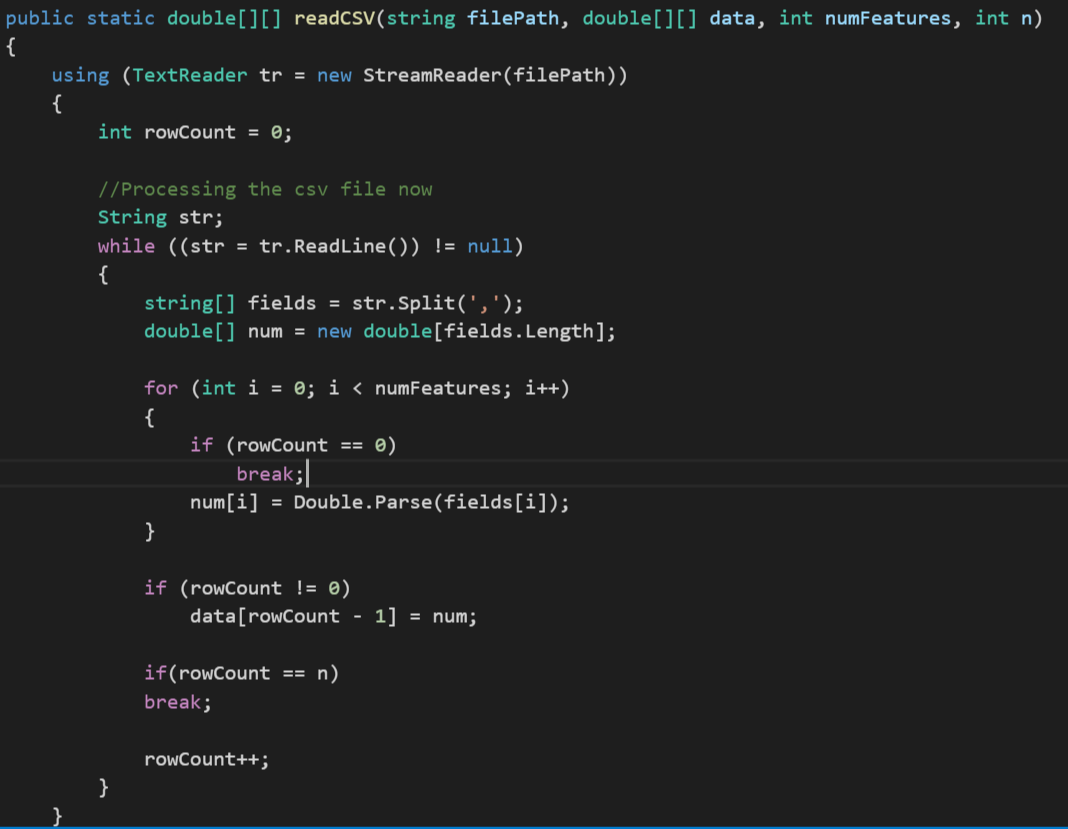
1. Immigration is optional, a random individual is generated and placed into the population, replacing a weak individual

Evolutionary optimization repeats steps one through five. Until some stopping condition, typically a maximum number of generations, is reached. Roulette Wheel Selection fitness level is used to associate a probability of selection with each individual chromosome. This could be imagined similar to a Roulette wheel in a casino:

* Usually a proportion of the wheelis assigned to each of the possible selections based on their fitness value
* While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be

**Code Explanation:**

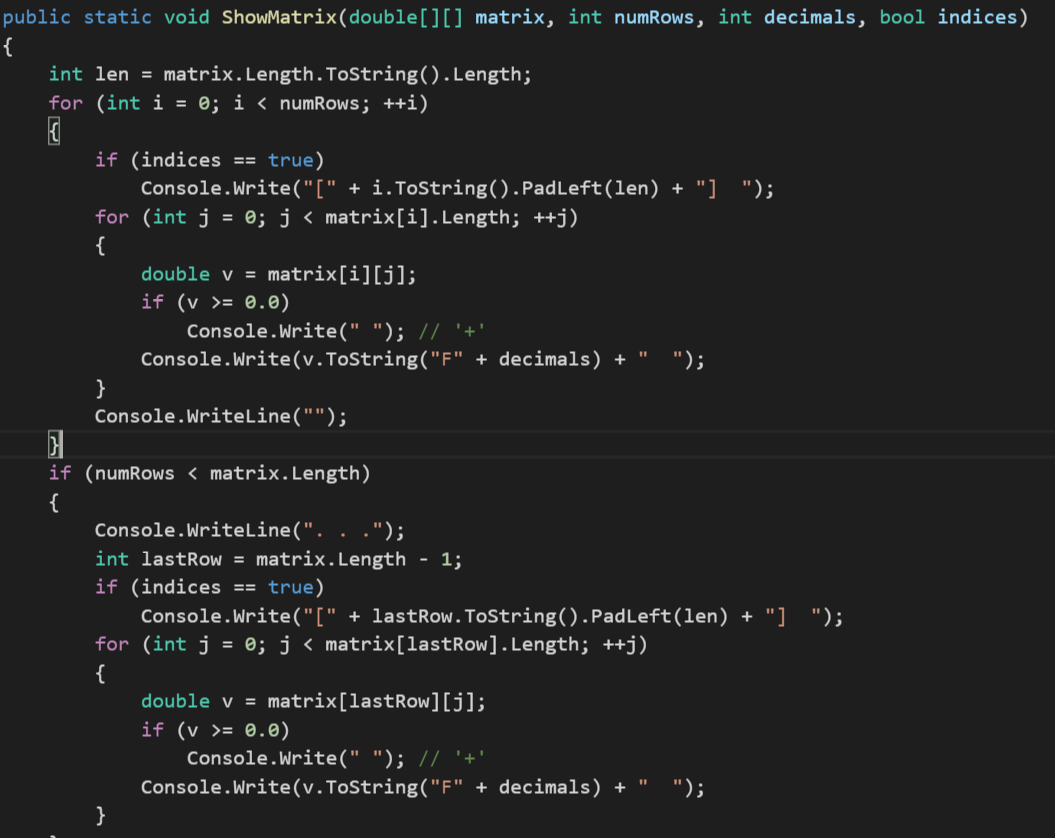
To run the algorithm, we are going to read the csv, **readCSV** function is to read the csv, and form a matrix, which is later used for training the model and later predicting the expenditure based on the observation or features given.



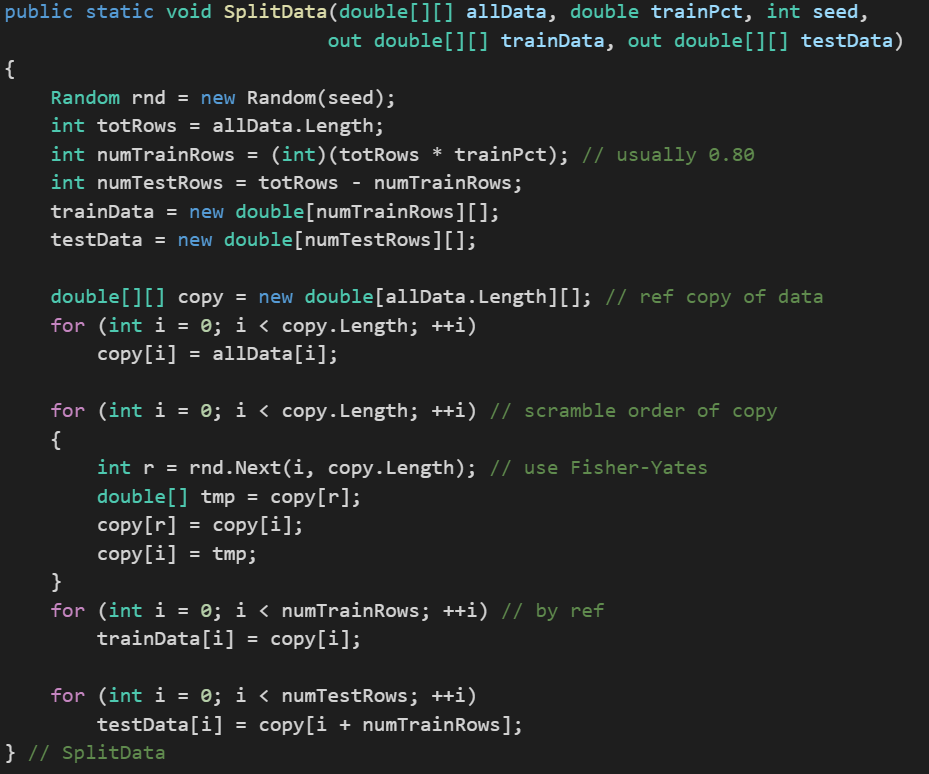
We set the training parameters required for processing the data. We set the popsize i.e., population size to number of rows. Maximum generation was set to 2000. Mutate Rate implies to ow many genes in a newly-generated child's chromosome will be mutated, set to 0.2 in our case. Mutate change implies magnitude of change of mutated genes, set to 0.01. Tau implies "selection pressure" controls the likelihood that the two best individuals in the population will be selected as parents for reproduction (Larger values of tau increase the chances that the two best individuals will be chosen), set to 0.4.



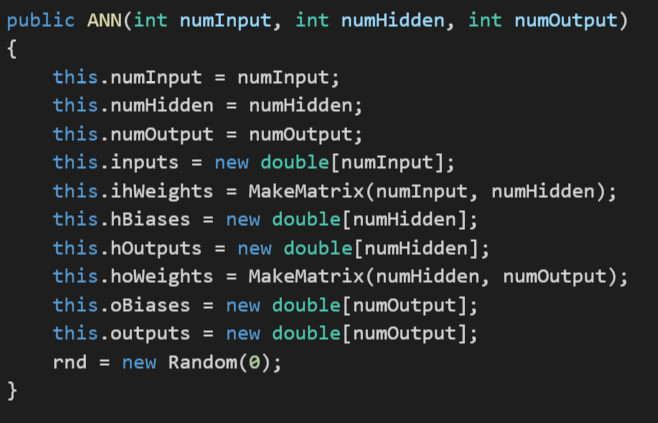
Once the dataset is formed, we can use **showMatrix function,** this function is to display first and last few rows of the dataset, it takes the dataset which we formed, the number of rows you want to display and boolean indices, do you want to show the index number of row or not.

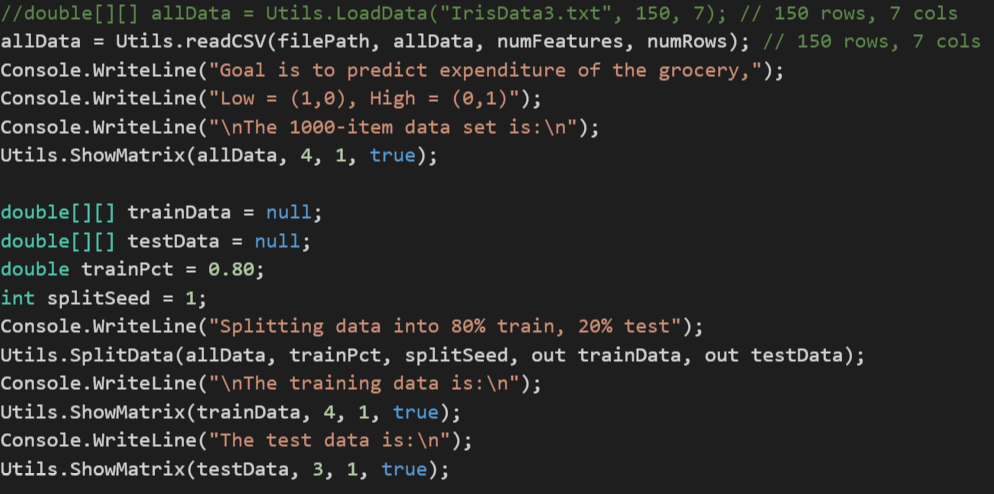


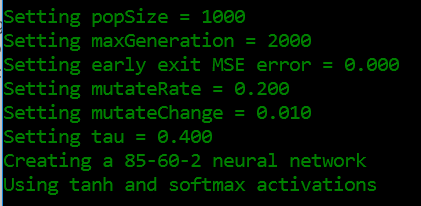
Once we have verified the dataset, we can now make the train and test from the dataset. For this we have **SplitData function,** this function make train and test data from the dataset, we split the data into 80% and 20%, 80% is to train the model based on Winnow Algorithm and rest 20% is to predict the accuracy of the algorithm. Before splitting the dataset, we set the seed value, seed value is a Random number which is used for generating the same trainData again. The function is going to output the result into **trainData** and **testData.**



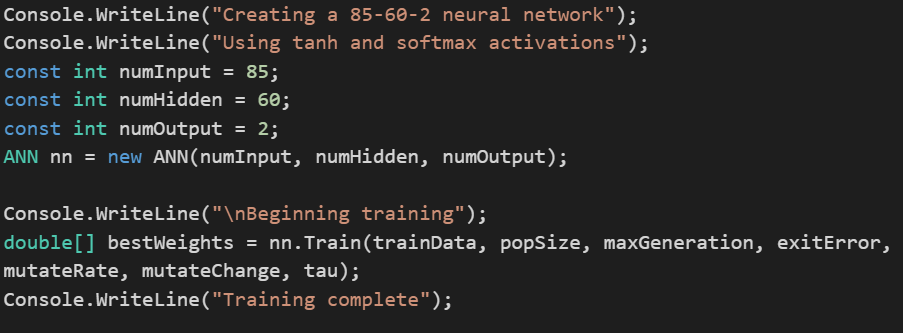
Once we have trainData and testData, we can now make the model by implementing Evolutionary Optimization on the train data. We are going to initialize the ANN with the number of input ndoes, number of hidden nodes and number of output and various other values of the particle and then we call the **TrainWeights**, which is going to compute the weights of the feature.



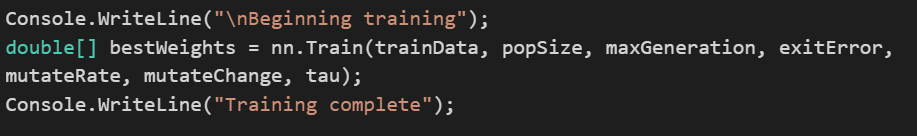


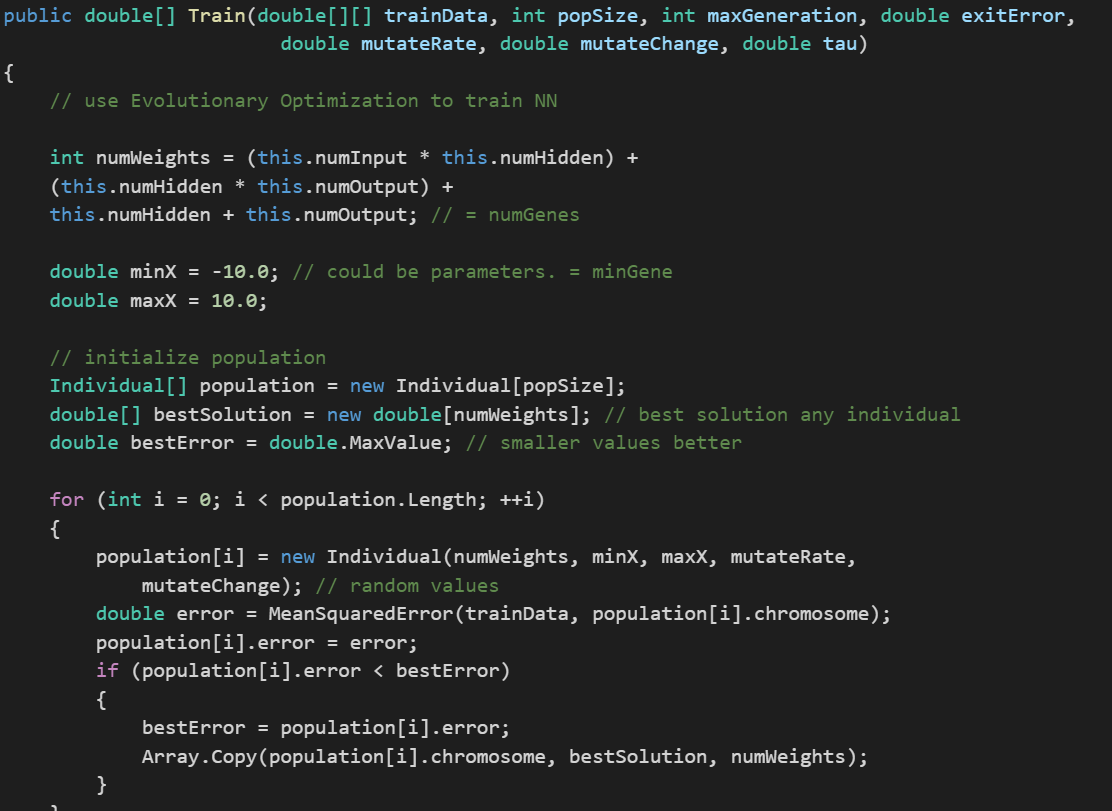


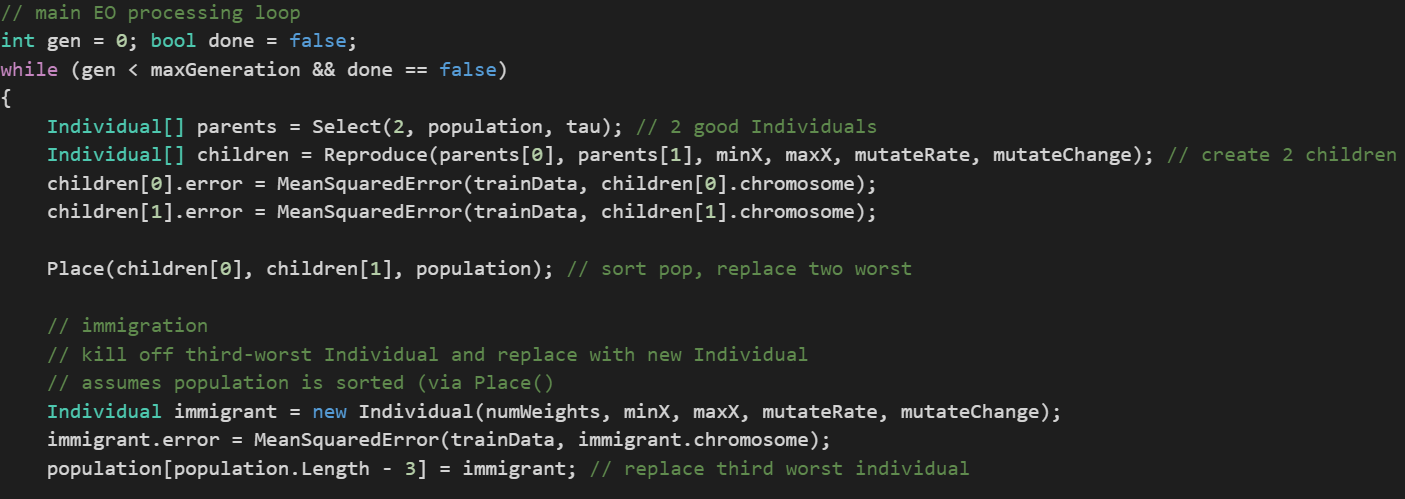
Creating the 85-60-2 neural network model. We set the neural network with Input layers as 86, Hidden layers as 60, and Output nodes as 2.

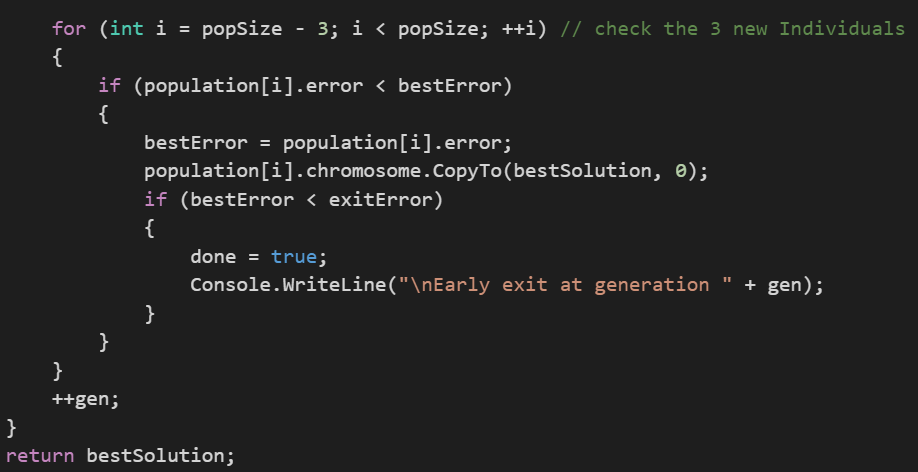


Begin the training of the model by sending the set parameters to the Train function of ANN



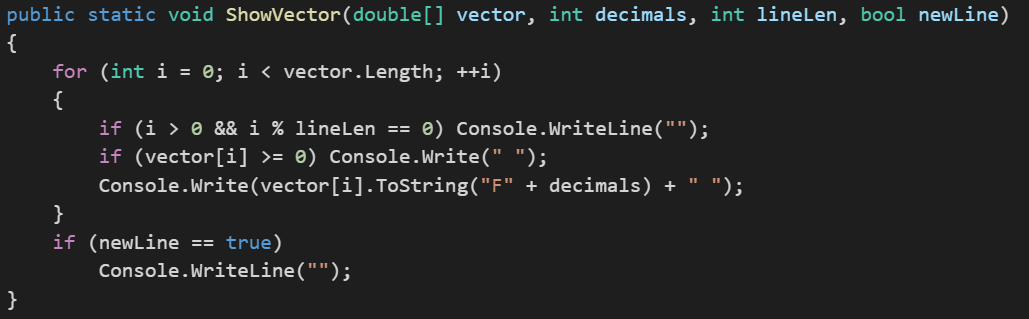




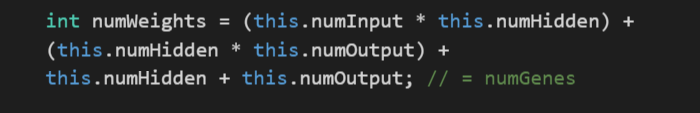


We have another **showVector function,** this function is used to display the final weights of the features, which is

assigned to them by them by Winnow, based on the update function of winnow, it take number of features display in the row and whether you want everything to be in one line or more



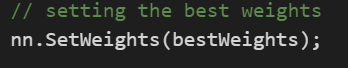
The total number of generated weights for the neural network would be the number of nodes in the input layer \* number of hidden layer + number of hidden layer \* number of output layer + number of hidden layer + number of output layer.

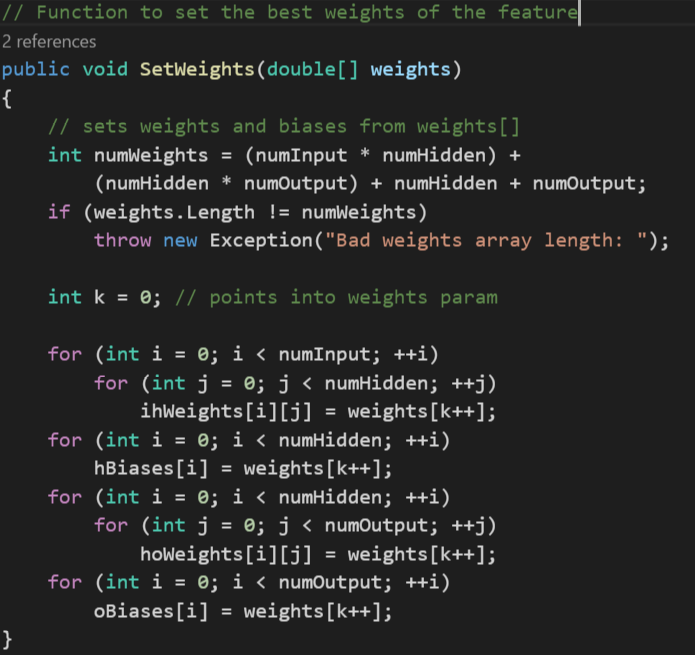


We create the instances of Individual which is equal to popsize. After initializing each individual instance with set values we calculate the mean squared error which is square of computed value – actual value. We return the sum of square error by length of training data which gives the meanSquaredError.

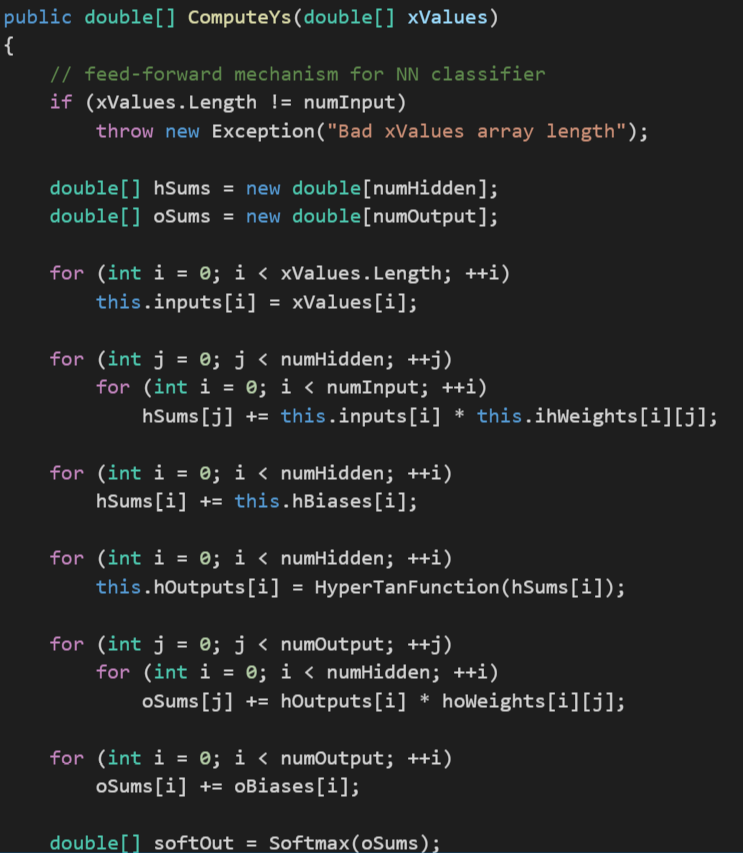


Function to set the best weights of the feature using **SetWeights** function.

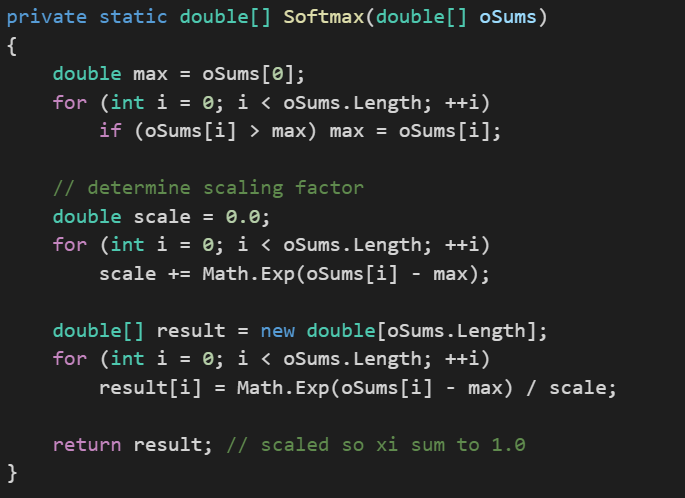




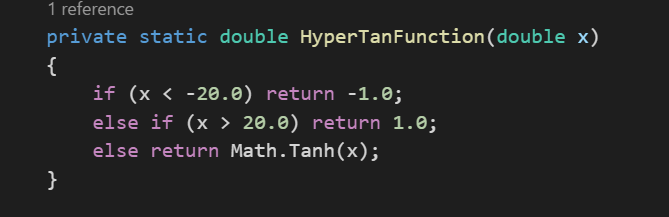
Function to predict the value of the class based on the observation



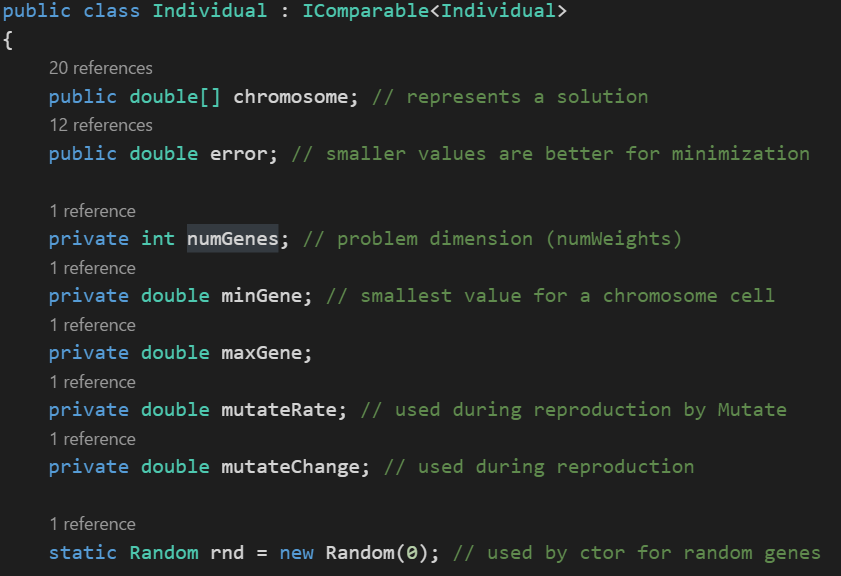
To calculate the Softmax Regression we use the following function

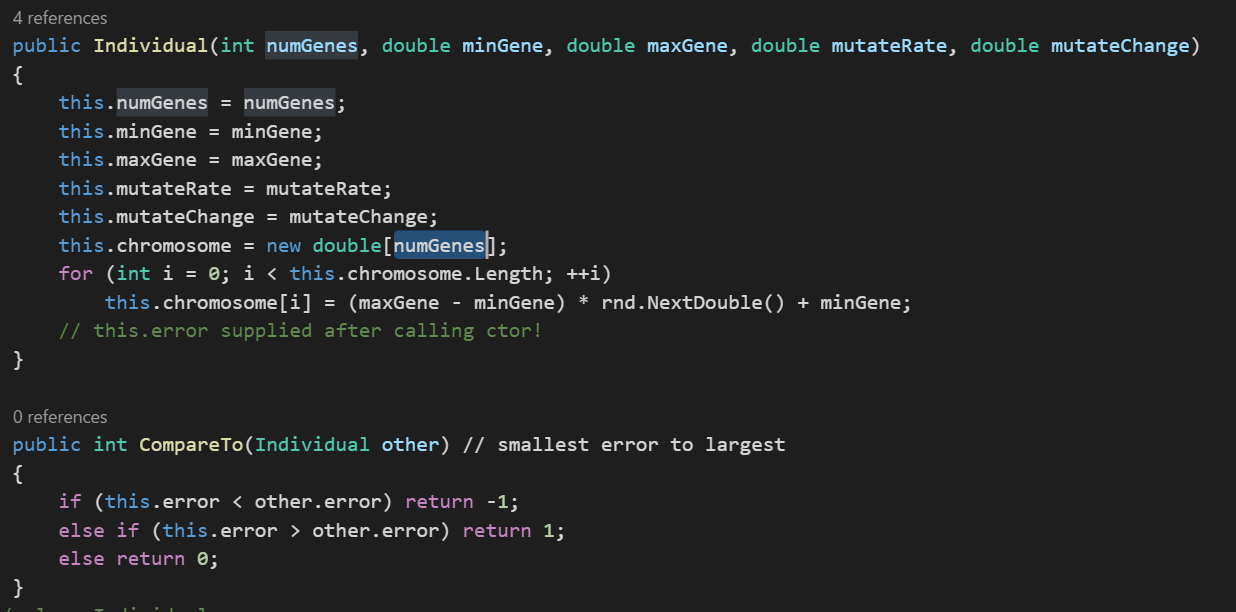


We multiply the weights \* the input values to see whether it is less than -20 if yes than return -1 or if it is greater than 20 then return 1 or if lies between -20 and 20 return Math.Tanh(x).

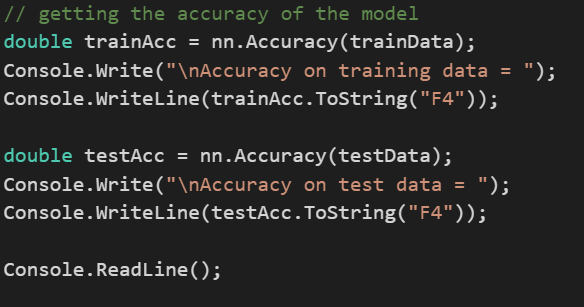


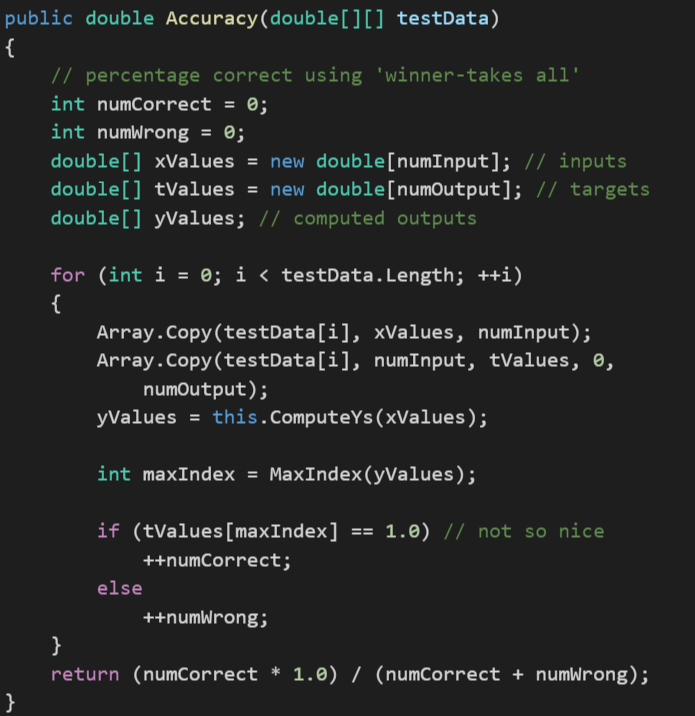
**Individual Class**

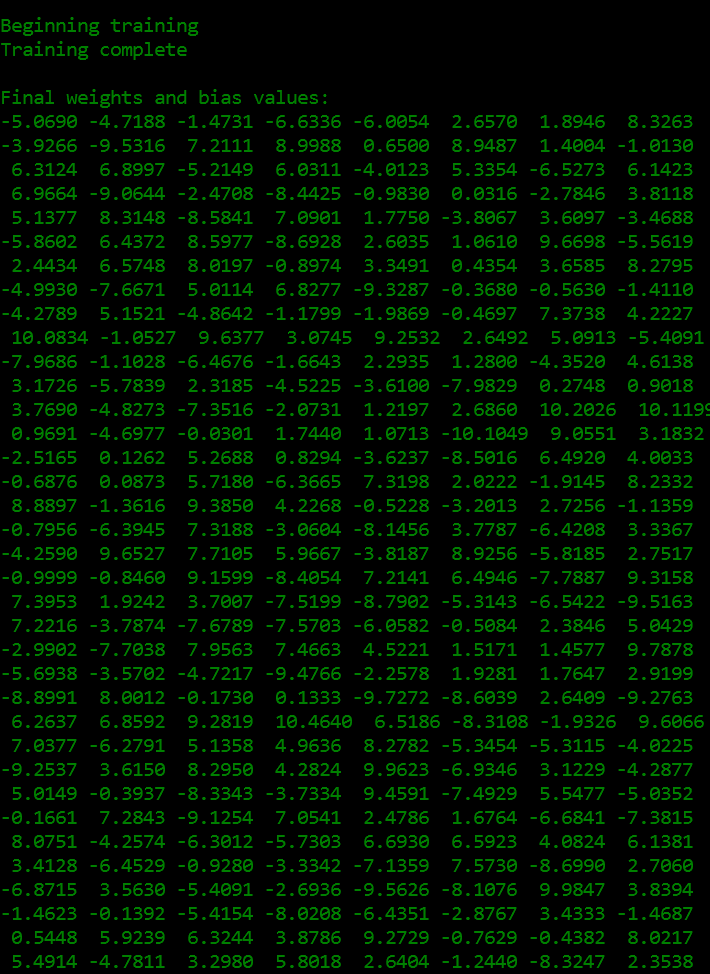


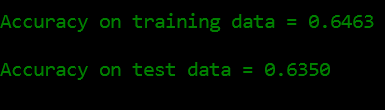


Get the accuracy of the train and test model









**The prediction accuracy of train data is 64.63% and test data is 63.50%**

**Conclusion**

Implementing the Winnow Algorithm for one iteration yield an accuracy of **76.88%** on train data. Since the non-linearity of the logistic curve dramatically increases the regression’s modeling power, our accuracy increases to **79.63%,** but to make sure we haven’t overfitted the model we used two regularization method L1 and L2. L1 gave us accuracy of **84.75%** which implies that our model improved and not overfitted applying L2 drastically increased the accuracy to **89.63%**. Evolutionary Optimistic algorithm yielded an accuracy of **64.5%,** which is less than all the three models. Logistic Binary Classification gave us the highest accuracy out of three models.