Project 4

Team 14
Josh Aney - josh.aney@icloud.com
Owen Cool - ohcool57@gmail.com
Nicolas Perl - nicolas.leon.perl@gmail.com
15 November 2024

Requirements

For this project, it is required to download six datasets from the UCI Machine learning repository. Once the data sets are downloaded it will be required to do some data preprocessing. For almost all our datasets, no attribute values are missing, and for those that are missing attribute values, relatively few instances are missing values. Thus, to ensure all instances in the dataset have complete values, we will **remove all instances with missing attribute values**. The next phase of preprocessing will be for the categorical features, which will be dealt with by applying **one-hot-encoding**. For numerical variables, **z-score normalization** will be applied to ensure all values are between 0 and 1. It transforms the features, so they have the properties of a normal distribution with a mean of 0 and a standard deviation of 1.

It is also required to implement **three different population-based** approaches to train a feedforward neural network. Focusing only on training the weights of a pre-defined, untrained neural network, the weights are separately trained via 4 different approaches:

- **Backpropagation**

  The backpropagation algorithm trains the network by iteratively adjusting the weights to minimize the error between predicted and actual outputs, using gradient descent. The error is computed through a forward pass, which involves propagating the input data through each layer, where the individual nodes compute weighted sums, add biases and apply activation functions. The backward pass is used to update the individual weights based on the error gradient.

- **Genetic algorithm with real-valued chromosomes**

  The genetic algorithm trains the neural net by creating an initial population of individuals (real valued in our case). Each individual in a population is evaluated by applying a **fitness function**, which assigns a score based on how close the individual is to the optimal solution. Individuals are then selected for **reproduction** based on their fitness using **tournaments**. In **cross-over**, selected individuals combine their chromosomes to form **offspring**. Cross-over swaps parts of chromosomes between pairs to mix genetic information, simulating biological reproduction. This process creates new solutions that might inherit the best features of both parents. To maintain diversity in the population and prevent premature convergence, **mutations** are introduced in some individuals. The offspring from crossover and mutation replace some of the current population. This process is iterated until it meets a required fitness, reaches a maximum number of generations, or a stagnation point where no significant improvement occurs.

- **Differential evolution**

  Differential Evolution starts by creating a random initial population of vector-represented individuals within certain bounds. Every individual in the population is evaluated using a **fitness function.** Mutation proceeds from generating a donor vector from a set of trial vectors: Select 3 random, distinct vectors of an individual and calculate the weighted difference of 2 and add to the third by using

  $$v_j = x_j^1 + \beta\,(x_j^2 - x_j^3) \qquad \text{where} \quad \beta \in [0,2]$$

  Then, generate an offspring vector as a combination of the donor vector and a member of the population. Elements are selected using a "binomial" crossover where the selection probability is α. The offspring vector is only selected if it is an improvement over the original vector. This process is also iterated until a certain termination criterion is met.

- **Particle swarm optimization**.

  PSO starts by randomly initializing a swarm of particles, each with their own position (potential solution) and velocity (how it moves through the search space). It continues to evaluate each particle's fitness and **update** its personal and the swarm's global best. Using these values and a few random factors, each particle's velocity and then its position is updated. **Repeat** this process until a stopping criterion is met.

We will then formulate a **hypothesis** focusing on convergence rate and final performance of each of the different training algorithms for the neural networks.

We use **ten-fold cross-validation** to train on nine of the partitions and test on the remaining partition. After rotating 10 times through the partitions and every partition was used exactly once as a test set, an average of the performance is computed. We will compare these results using the metrics of **precision, recall, accuracy, and mean squared error**. Comparing these results, we will be able to **evaluate our hypothesis**.
Implementing the individual training algorithms involves the tuning of numerous hyperparameters, such as the learning rate, the batch size (since we use Minibatches), population size, crossover rate, mutation rate, scaling factor, binominal crossover probability, inertia, cognitive update rate and social update rate. **Hyperparameter tuning** is done by extracting 10% of the data (random, but stratified). Then the remaining 90% are split into ten folds of 9%. For each of the experiments, nine of the folds are combined, holding out the tenth as our test set. Tuning is done by training 10 times, once on each of the nine fold sets while tuning with the initial 10%. The hyperparameters are chosen by selecting the best of the averaged results. The **generalization ability** is tested 10 times, one for each unique held-out fold.

System Architecture

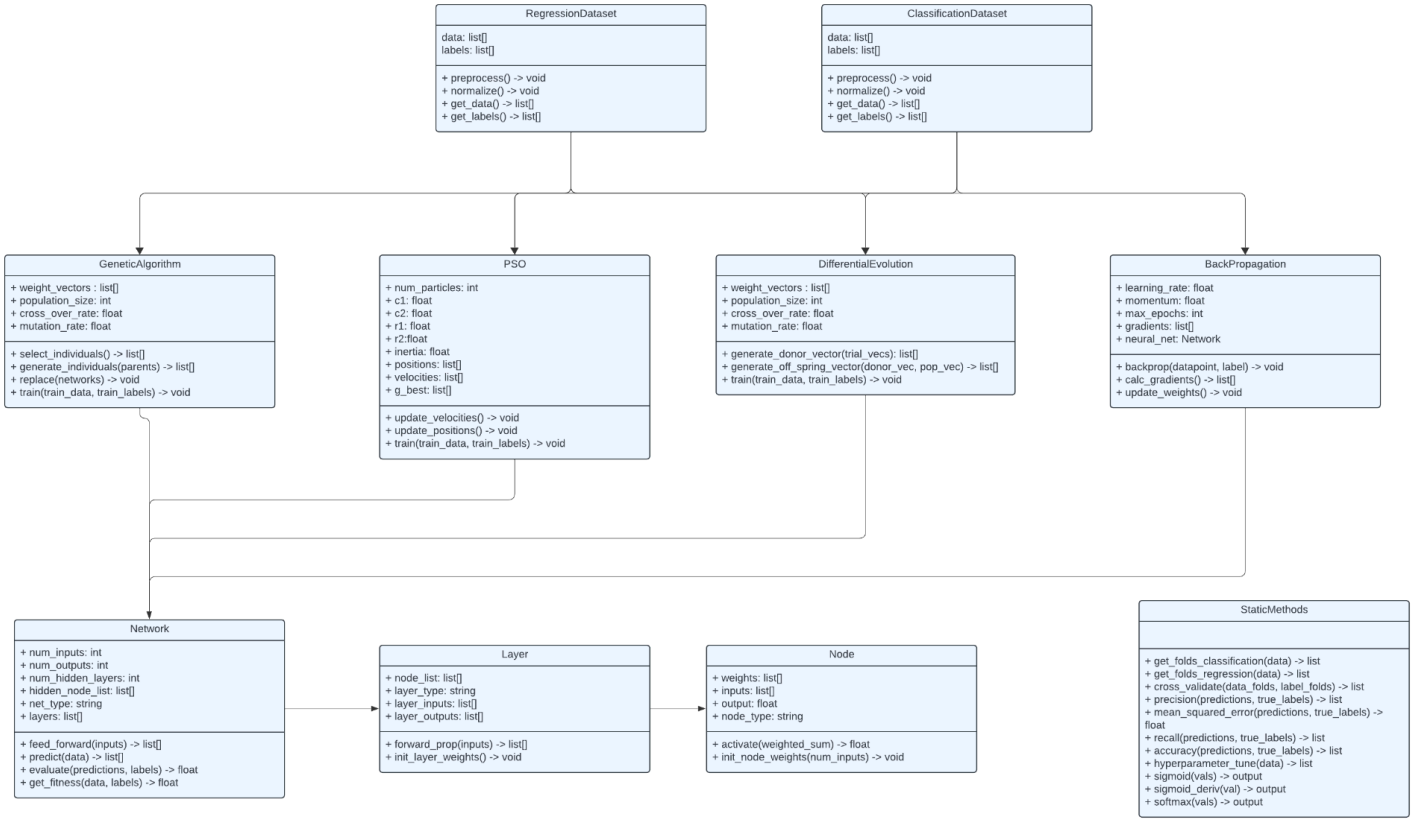

Figure 1. UML Class Diagram All of the Neural Network Training Algorithms

GeneticAlgorithm Class

- This Class contains all the necessary parts for the genetic algorithm. This class is meant to train a neural network class to make predictions on the dataset. This class will be a place that takes in one of the dataset classes.

  The attributes for this class are:

  1. weight_vectors: This is a list of weight vectors that represent all the individuals in the population.
  2. population_size: This is an int that represents how many individuals are in the population.
  3. cross_over_rate: This is a float that represents the crossover rate of the algorithm.
  4. mutation_rate: This is a float that represents the mutation rate of the algorithm.

  The functions for this class are:

  1. select_individuals(): This function is meant to select the best individuals out of the population for mutation. It then returns the most fit individuals.
  2. generate_individuals(): This function is meant to generate new individuals based on the selected individuals from the previous function. Once it has these individuals, it will then return them.
  3. replace(): This function is meant to go through the population and replace a certain number of individuals in the older population with the new individuals that were generated. This function does not return anything.

4. train(): This function takes in a set of data points and data labels. This function is meant to call all the other functions to set up the weights correctly in the neural network.

PSO Class

- This class contains all the necessities to complete the particle swarm optimization algorithm. It will then update all the weights of the neural network to the optimal set found by PSO. This class will be a place that takes in one of the dataset classes.

  The attributes of this class are:
  1. num_particles: This is an int that represents the number particles in the algorithm.
  2. c1: This is a float that represents one of the cognitive coefficients.
  3. c2: This is a float that represents another one of the cognitive coefficients.
  4. r1: This is a float that represents one of the social coefficients.
  5. r2: This is a float that represents another one of the social coefficients.
  6. inertia: This is a float that represents the inertia value that is used in the training process of the algorithm.
  7. positions: This is a list of lists that represents the positions of every particle in the algorithm.
  8. velocities: This is a list of vectors that represents the velocity vector for each particle in the algorithm.
  9. g_best: This is a list that represents the best particle position that the algorithm has found so far.

  The functions of this class are:
  1. update_velocities(): This function will update all the velocities of the particles based on the specified update rule and the inertia.
  2. update_positions(): This function will update all the positions of all the particles based on their velocity vector.
  3. train(): This function will take in a set of training data and training labels. It will then call the other functions to find the most optimal set of weights for the neural network and set up the neural network.

DifferentialEvolution Class

- This class contains all the necessities to complete the Differential Evolution algorithm. It will then update all the weights of the neural network to the optimal set found by differential evolution. This class will be a place that takes in one of the dataset classes.

  The attributes for this class are:
  1. weight_vectors: This is a list of vectors that are the set of vectors used to generate a donor vector.
  2. population_size: This is an int that represents the size of the population for differential evolution.
  3. cross_over_rate: This is a float that represents the crossover rate of the algorithm.
  4. mutation_rate: This is a float that represents the mutation rate of the algorithm.

  The functions for this class are:
  1. generate_donor_vector(): This takes in a set of trial vectors and it will then generate a donor vector based off of the trial vectors.
  2. generate_off_spring_vector(): This takes in a donor vector and a population vector. It will then generate an offspring vector and return it.

3. train(): This takes in a set of training data and a set of training labels. It will then start the training loop and train until it finds an optimal set of weights for the neural network. It will then set the neural network.

BackPropagation Class

- This class contains all the necessities to complete the backpropagation algorithm. It will then update all the weights of the neural network to the optimal set found by backpropagation. This class will be a place that takes in one of the dataset classes.

  The attributes for this class are:
  1. learning_rate: This is a float that represents the learning rate of the algorithm.
  2. momentum: This is a float that represents the momentum of the algorithm.
  3. max_epochs: This is an int that represents the maximum number of epochs the algorithm will complete before halting.
  4. gradients: This is a list that contains the gradients for all the weights in the neural network.
  5. neural_net: This is a Network class that contains all the things that need to be updated in the backpropagation algorithm.

  The functions for this class are:
  1. backpropagation(): This takes in a data point and a label. It will then update all the weights in the network accordingly based on their gradients.
  2. calc_gradients(): This function will calculate the gradients of all the weights in the network based on the derivatives of the activation functions.
  3. update_weights(): This function will update all the weights in the network object. This function will be used in the backpropagation function.

Network Class

- This class contains instances of Layer classes which contain all the nodes. This class is meant to hold all the big picture methods that call helper methods in smaller classes.

  The attributes for this class are:
  1. num_inputs: This is an int which will represent the number of input nodes the network will take.
  2. num_outputs: This is an int which will represent the number of output nodes the network will have.
  3. num_hidden_layers: This is an int which will represent the number of hidden layers in the network.
  4. hidden_node_list: This is a list of ints which represents the number of nodes in each hidden layer. The length of this list should be the same as num_hidden_layers.
  5. layers: This is a list of Layer classes that will contain each specific layer that the network contains.

  The functions for this class are:
  1. feed_forward(): This function takes in a list of inputs from the input layer of the network and it will call the feed forward method in the layer class. It will then update the nodes required in each layer.
  2. predict(): This function will take in a list of data points and it will predict each datapoint and return a list of predictions.
  3. get_fitness(): This function will take a list of data points and true labels. It will then calculate metrics for the model and return them.

Layer Class

- This class represents a layer of a network. It contains an arbitrary number of nodes and it is responsible for calling all the helper methods in the Node class to complete the overall goal of the network.

  The attributes for this class are:
  1. node_list: This is a list of Node classes that represents all the nodes in a given layer.
  2. layer_type: This is a string that represents if the layer is an input layer, hidden layer, or output layer.
  3. layer_inputs: This is a list of all the inputs for a layer from the previous layer.
  4. layer_outputs: This is a list of all the outputs for a layer going to the next layer.

  The functions for this class are:
  1. forward_prop(): This function takes in a list of inputs and applies the weights, biases, and activation.
  2. init_layer_weights(): This function calls init_node_weights for all the nodes to set up the layer correctly.

Node Class

- This class represents each of the specific nodes in a layer. These nodes contain weights and biases as well as other book keeping variables.

  The attributes of this class are:
  1. weights: This is a list of all the weight connections from the previous layer
  2. inputs: This is a list of all the inputs from the nodes at the previous layer.
  3. output: This is a float that represents the output of the node after all the weights and biases have been applied.
  4. node_type: This is a string that represents the type of node that the specific node is (input, output, hidden, etc.).

  The functions for this class are:
  1. activate(): This function takes in a weighted sum and applies the activation function to it. It then returns this value.
  2. init_node_weights(): This function takes in a number of inputs then it will randomly set up all the weights for those edges.

RegressionDataset Class

- This class is where all the data from a regression dataset will be held. It will do all the necessary things to preprocess the data before it gets handed to the network.

  The attributes for this class are:
  1. data: This is a list that will contain all the data after preprocessing without any labels
  2. labels: This is a list that will contain all the labels for the data.

  The functions for this class are:
  1. preprocess(): This function will preprocess all the data. This could be removing data points with missing values or one-hot-coding.

2. normalize(): This function will normalize all the data as specified above in the requirements.

ClassificationDataset Class

- This class is where all the data from a classification dataset will be held. It will do all the necessary things to preprocess the data before it gets handed to the network.

  The attributes for this class are:
  1. data: This is a list that will contain all the data after preprocessing without any labels
  2. labels: This is a list that will contain all the labels for the data.

  The functions for this class are:
  1. preprocess(): This function will preprocess all the data. This could be removing data points with missing values or one-hot-coding.
  2. normalize(): This function will normalize all the data as specified above in the requirements.

StaticMethods

- This contains all of the helper functions that may be needed to calculate things that are not directly related to the main algorithm or datasets.

  1. sigmoid(): This function takes in a vector and it will return a value that gets returned from the sigmoid function.
  2. sigmoid_derivative(): This function will take in a value and return the derivative of the sigmoid function at that value.
  3. softmax(): This function takes in a vector and it will run the softmax function on the vector. It will then return the new output vector.
  4. recall(): This function takes in the predicted labels and the true labels and returns the recall value.
  5. precision(): This function takes in the predicted labels and the true labels and returns the precision value.
  6. accuracy(): This function takes in the predicted labels and the true labels and returns the raw accuracy value.
  7. mean_squared_error(): This function takes in a list of predicted values and a list of true values. It then returns the calculated mean squared error.
  8. get_folds_classification(): This function takes in a set of classification data and returns a list of ten specific folds. The process of stratifying across folds happens in this function.
  9. get_folds_regression(): This function takes in a set of regression data and it will sort the data based on the response value in the data. It will then break the data into groups. Then it will draw the first item from each group for the first dataset and the second for the second dataset.
  10. cross_validate(): This function takes in a set of folds and it will perform 10-fold cross-validation. This function will return a list of metrics to evaluate the model.
  11. hyperparameter_tune(): This function will take in a set of data and tune all the hyperparameters in the model (k, sigma, etc.). It will then return the parameters that performed best on average.

System Flow

To run the system and make sure that all the requirements are fulfilled, start by creating all of the datasets that are described in the requirements. This is where the flow of the data will start. Next you will need to set up all the training algorithms. These algorithms will take a set of hyperparameters that were found from hyperparameter tuning. Hyperparameter tuning will be completed beforehand through a grid search. A hold out set of the original data will be used to test which set of hyperparameters will perform best for each algorithm. Once the set of optimal hyperparameters are found, create a class for the genetic algorithm, particle swarm optimization, and differential evolution with each of their specified hyperparameters passed in through the constructor.

Next, these classes will each take a subset of the original data to train a neural network. This subset will have the hold out fold from tuning with excluded. We will use a variation of 10-fold cross validation when evaluating the algorithms so we will need to pass in a certain set of data to each of these algorithms that also excludes the testing set found by cross validation. To do this, we will pass in our new subset of data excluding the hold out fold into the get folds function. This will get a stratified set of data folds and label folds. Once we have these folds we will pass in nine folds into the training functions in the algorithm classes. All of the algorithm classes are set up to allow this. These training functions will then find an optimal set of weights to use in the neural network.

Finally, these weights can be passed into the neural network and the neural network will be functional. To predict values from the neural network, you will need to pass a testing set of data into the predict function. This testing set will be the last fold that was not used in the training function. The network will then feed each datapoint through the neural network one at a time. The network will then give its predicted output in the final layer of the network. For classification problems, you may need to set the max of the output to one and the rest to zero so it matches the output that the get labels function returns.

Once you have the predictions, you will be able to pass the predictions and the true labels into the correct metric function. These metric functions will then give you metrics to evaluate each of the specific training algorithms.

To sum everything up, you will first create the dataset classes. Next, you will pass a certain set of data to hyperparameter tuning which will give you a set of hyperparameters. Then, you will create the training algorithm class and pass in the correct hyperparameters into the constructor. Then, you will pass in the remaining data without the tuning hold out fold into the training function in the algorithm class. Finally, you will pass the weights found in the training function into a neural network. The neural network will be functional once the weights are set. You can then predict labels for the specific dataset. By following this flow of data, you will complete all the requirements that are specified above.

Test Strategy

To ensure that we meet project requirements, we will first validate that our data has been effectively preprocessed. This means that our datasets contain only **complete instances** and our categorical values have been **encoded to binary values**. Because there are few incomplete instances to be removed, we will validate that they have been removed properly by simply outputting a message with the instances removed and comparing it to the instances we can see need to be removed in the data file. To validate that categorical values are properly encoded to binary values, we will test if our binary encoding is working as intended on some sample values and ensure that our algorithm is getting reasonable results (improperly encoded values would drastically alter the results of our experiment).

To test that our **normalization function** is working as intended, we will validate that it is applied in all situations stated in the design document, and test that it produces the correct values on some sample values that could reasonably be encountered. Further, we can test that all numerical values are between 0 and 1 after preprocessing is complete and normalization has been applied. These test cases together should be exhaustive enough to ensure that we are using normalization as intended.

To test that our **activation function, feed-forward neural network, and backpropagation** work as intended, we will run some sample iterations by using sample instances from our various datasets on their respective neural networks, outputting the weights of each node connection, the values in each weighted sum for each node in the hidden layers and output layer, the weighted sums from those values, and the result of the activation function given that sum. Then, we will print out the errors, the gradient, and the changes made to the weights of the connections between nodes. This way, we will be able to validate that all values are calculated properly and that adjustments to the weights are properly made based on those values. We will run these tests for an exhaustive selection of samples, including both regression and classification problems, so that we can fully validate that the algorithm works.

To test that our **genetic algorithm** works as intended, we will output the performance of each model in the tournament and the selected winners to ensure that selection is carried out correctly. Further, we will output the parent genetics, the crossover points, and the genetics of their offspring to validate that reproduction functions as intended. Lastly, we will display the mutations that occur to ensure that they are carried out correctly.

To test that our **differential evolution algorithm** works as intended, We will output the vectors selected, the intermediate calculations, and the fitness function outputs for each vector. This way will be able to ensure that all equations used in the algorithm are implemented correctly and produce the same output that we expect from manual calculation.

To test that our **particle swarm optimization algorithm** works as intended, we will first ensure that all global, neighborhood, or local bests are updated correctly, and that personal bests for each particle are updated correctly. Then, we will ensure that the velocity update function updates velocity correctly, and that position updates according to the particle's velocity.

To test that our algorithms stop when intended, we will output the values being considered for the stopping criterion at each iteration until the stopping criterion is met, to ensure that the criterion is actually met.

To test that our **grid search hyperparameter tuning** is working as intended, we will print values of the hyperparameters as we run the algorithm, checking to see that it changes values, moving through the whole grid. Further, we will ensure that the hyperparameter values that enable the best performance are

selected by printing the performance of the model after training for each possible set of hyperparameter values.

Upon obtaining our results, we will check to make sure that **precision, accuracy, recall, and F1 score** have been calculated correctly by printing the intermediate values used to calculate these metrics and carrying out the calculations manually.

We will check that **10-fold cross-validation** works as intended by ensuring that we get different results on different trials of the experiment and that we get reasonable results, that we run 10 trials of tuning using different folds for each run, and that the average performances are calculated properly.

Task Assignments and Schedule

Throughout the course of the project, each team member is expected to complete their work that is assigned in a timely manner. On the parts of the project that require team members to meet in person, it is expected that the team members will communicate to make a time work where they can meet and complete the task at hand. This communication can be done directly with the other team members since there are only three members in the team. All of the code that is written in this project will be uploaded to a team repository so that it can be accessed at any time if the other team members need it. It is expected that team members will keep their code up to date in the repository if they have made any changes.

Task Descriptions

1. Create Design Document - Follow the Creating a Design Document PDF and on D2L to create a design document specific to project 4.
2. Create Hypothesis - Using the descriptions of the datasets and of the neural network and evolutionary algorithms, create a hypothesis about how the different datasets will perform according to the chosen metrics.
3. Update dataset classes' methods - Using classes from project 3 as a template, implement any new methods needed for the Neural Networks according to the description of its methods in the Project 3 design document. Specifically, change the normalization methods if necessary for the new algorithms. After this task, these classes should be ready for use.
4. Implement the genetic algorithm – Create any necessary classes to implement the genetic algorithm, and implement the algorithm itself according to the description in the project 3 design document.
5. Implement the differential evolution algorithm – Create any necessary classes to implement the genetic algorithm, and implement the algorithm itself according to the description in the project 3 design document.
6. Implement the particle swarm optimization algorithm – Create any necessary classes to implement the genetic algorithm, and implement the algorithm itself according to the description in the project 3 design document.
7. Update StaticMethods - Using the StaticMethods from project 3, implement any new methods needed for methods needed for the Neural Network according to the description of its methods in the Project 3 design document. After this task, these methods should be ready for use.
8. Final Code Check - For this task, the team members will need to meet in person to make sure all the code runs as expected so that the data that gets collected later in the project is reliable. Further, team members will check that all code is well-commented and readable
9. Collect Results and Create Figures - For this task, one team member will need to create methods for generating figures from the results of our experiment and run the code to generate those figures.

10. Create a Rough Draft of the Paper - Each team member will be responsible for different sections of the paper, using course materials, the hypothesis created in the beginning of the experiment, and the results generated at the end of the experiment to complete the paper's different sections.
11. Complete the Paper - Team members will review one another's work on the paper and ensure that it meets all standards laid out on the project description on D2L, addresses the hypothesis, and accurately interprets the results.
12. Plan Video - This task requires the general layout of the video to be complete and any "dummy methods" to be implemented. This allows the actual recording of the video to go much smoother and prevents last minute changes while trying to record.
13. Record Video - For this task, a team member will record the video. Another team member will edit the video if necessary. This process will also be used as a final check for anything missing in the project. After this task is complete, the paper, video, and code will be submitted

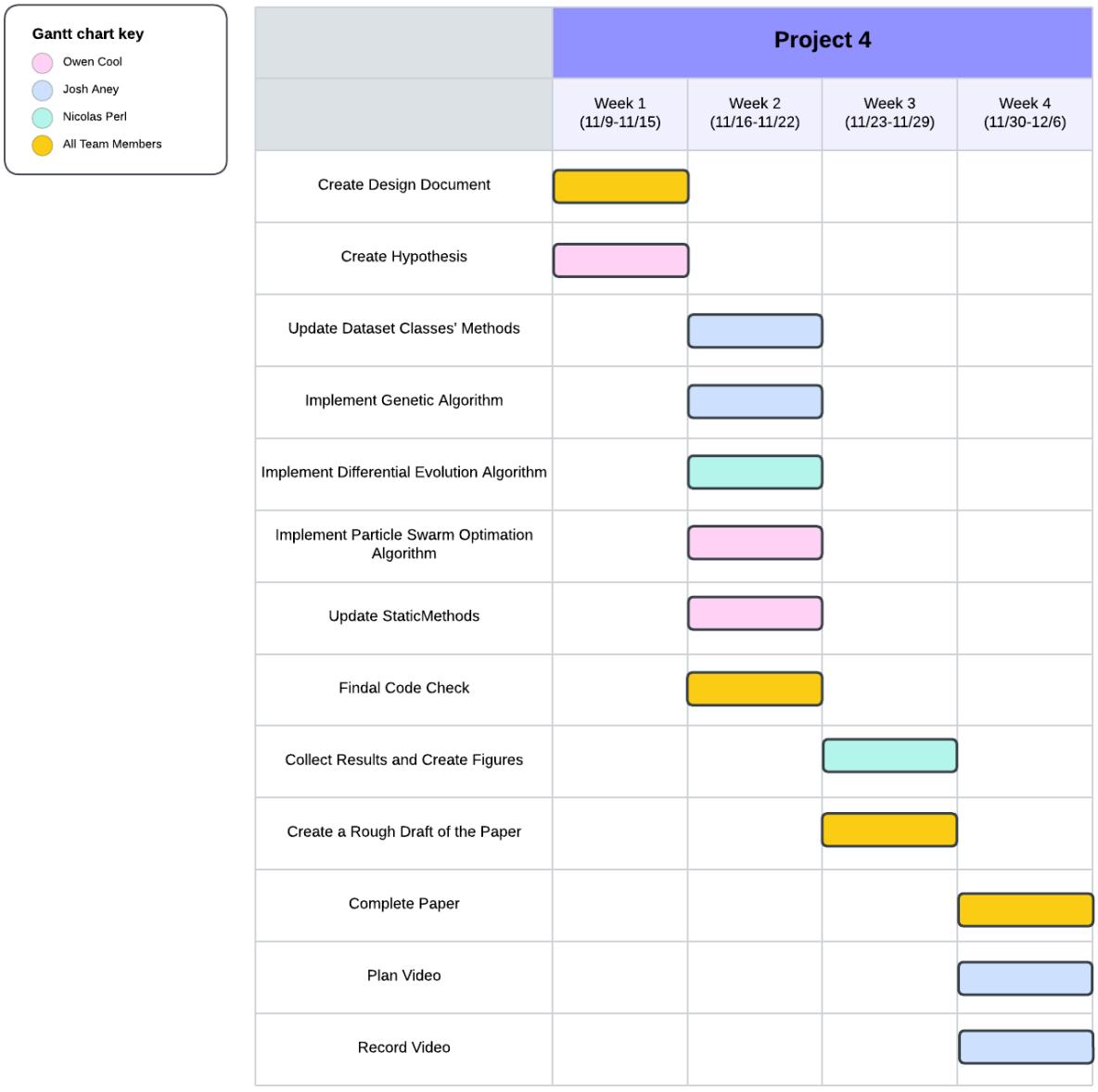

Figure 2. Gantt chart for schedule of tasks on Population Based Algorithm project.