Appendix: The Details of EV Algorithm

Here I will use an evolutionary algorithm to search for regressor genes with least prediction errors. Here are the choices in the specification of the Mini-Project:

Search space: The population is 50 individuals with 2^13 = 8,192 possible combination.

Representation of the search space: For the representation I use a simple 13-bit binary encoding mapping integers (phenotypes) to bit-strings (genotypes).

Objective function:

The fitness function is defined as AIC. I use the first 35 data points as

```
def calculate_fitness(pop_array):
    size = 50
    error = 0
    fitness = []
    for i in range(size): # 50 is the size of population
        index = [0]
        error = 0
        for j in range(13): # 14 is the size of gene length
            if(pop_array[i][j] == 1):
              index.append(j)

        for i in range(20):
            a = af.iloc[i:(34+i),index]
            b = df['Return'][i:(34+i)]

            model = sm.OLS(b, a).fit()
            AIC = model.aic
            error = error + AIC
        fitness.append(error)
    return np.array(fitness)
```

Variation operators:

Mutation operators Mutation is executed by generating a random number (from a uniform distribution over the range [0, 1]) in each bit position, and comparing it to a fixed threshold, the mutation rate, in my project the mutation rate is around 0.7. If the random number is below that rate, the value of the gene in the corresponding position is flipped.

To modify one single individual, in the code below I have randomly find a position in genes, and bit-flip 0 and 1.

```
def mutation(fittest):
    mutationPoint = randint(0, 12) # 13 is the length of genes
    if (fittest.genes[mutationPoint] == 0):
        fittest.genes[mutationPoint] = 1
```

```
    else:
        fittest.genes[mutationPoint] = 0

size = 50
for i in range(size):
    if (randint(0,7)%7 < 5): # mutate under the probability 5/7
        mutate(pop_array[i])
```

Recombination (crossover) operators which involve two parents. Similar as the mutation operators, I have first find a random point as a crossover point and then exchange the genes of two parents, the code is as below:

```
def recombination(individual1, individual2): # individual1&2 are parents
    crossOverPoint = randint(0,12) # 13 is the length of genes
    for i in range(crossOverPoint):
        temp = individual1[i]
        individual1[i] = individual2[i]
        individual2[i] = temp
    return individual1, individual2

for i in range(size):
    rand_index = randint(0,size -1)
    pop_array[rand_index], pop_array[i] = recombination(pop_array[i], pop_array[
rand_index])
```

Selection operator: A truncation selection, where the probability p_i that an individual i in population P is chosen to be a parent. here the best fitness individual is the one with the minimum fitness.

```
def select(prop_fitness):
   index_best = np.where(prop_fitness == prop_fitness.min())
   # I will get the last 20% out of the population
   index_least = np.where(prop_fitness > np.quantile(prop_fitness, 0.8))
   return index_best[0][0], index_least[0]
```

Termination criterion: I am running the algorithm for a fixed length of 20 iterations.

Evolutionary Algorithm Process

```
pop = []
size = 50
for j in range(size):
    genes = []
    for i in range(13): # totally there is 13 varables
        genes.append(randint(0,1))
    pop.append(genes)

pop_array = np.array(pop)

generation = 0
best_fit_arr = []
least_fit_arr = []
average_pop_arr = []
whole_pop_fitness = []
```

```
while True:
   print("generation:")
   print(generation)
   generation = generation + 1

   best_fit, least_fit, average_pop = getFit(pop_array)

   best_fit_arr.append(best_fit)
   least_fit_arr.append(least_fit)
   average_pop_arr.append(average_pop)

   ''' recombination step '''
   ''' is the recombination too much? '''

    for i in range(size):
       rand_index = randint(0,size -1)
       pop_array[rand_index], pop_array[i] = recombination(pop_array[i],
pop_array[rand_index])
      print("after recombination:")
      print(sum(sum(pop_array)))

   '''mutation step '''
   for i in range(size):
      if (randint(0,7)%7 < 5): # mutate under some probability
         mutate(pop_array[i])
   print("after mutation:")
   print(sum(sum(pop_array)))

   '''selection step '''
   a, b = select(calculate_fitness(pop_array))
   pop_array[b] = pop_array[a]
   print("after selection:")
   print(sum(sum(pop_array)))

   if (generation >100): break
```