CS512- Assignment5

by

Channamallikarjun Rolli

MainMLR:

**import** time  
**from** numpy **import** \*  
**from** sklearn **import** svm  
**import** csv  
**import** mlr  
**import** FromDataFileMLR  
**import** FromFinessFileMLR  
  
**class** Fitness:  
  
 **def** CreateInitialVelocity(self, numOfPop, numOfFea):  
 **for** i **in** range(numOfPop):  
 **for** j **in** range(numOfFea):  
 self.VelocityM[i][j] = random.random()  
 *#------------------------------------------------------------------------------  
 #Initail population and Initial Local Best matrix* **def** createInitialPopulation(self, numOfPop, numOfFea):  
 population = random.random((numOfPop,numOfFea))  
 **for** i **in** range(numOfPop):  
 V = self.getAValidrow(numOfFea)  
 **for** j **in** range(numOfFea):  
 population[i][j] = V[j]  
 **return** population  
 *#------------------------------------------------------------------------------* **def** getAValidrow(self, numOfFea, eps=0.015):  
 sum = 0  
 **while** (sum < 3):  
 V = zeros(numOfFea)  
 **for** j **in** range(numOfFea):  
 r = random.uniform(0,1)  
 **if** (r < eps):  
 V[j] = 1  
 **else**:  
 V[j] = 0  
 sum = V.sum()  
 **return** V  
 *#------------------------------------------------------------------------------  
 # The following creates an output file.* **def** createAnOutputFile(self):  
 file\_name = **None** algorithm = **None** timestamp = time.strftime(**"%Y-%m-%d-%H-%M-%S"**, time.localtime())  
 **if** ( (file\_name == **None**) **and** (algorithm != **None**)):  
 file\_name = **"{}\_{}\_gen{}\_{}.csv"**.format(algorithm.\_\_class\_\_.\_\_name\_\_,  
 algorithm.model.\_\_class\_\_.\_\_name\_\_, algorithm.gen\_max,timestamp)  
 **elif** file\_name==**None**:  
 file\_name = **"{}.csv"**.format(timestamp)  
 fileOut = open(file\_name, **'w'**)  
 fileW = csv.writer(fileOut)  
  
 fileW.writerow([**'Descriptor ID'**, **'Fitness'**, **'Model'**,**'R2'**, **'Q2'**, \  
 **'R2Pred\_Validation'**, **'R2Pred\_Test'**])  
  
 **return** fileW  
 *#-------------------------------------------------------------------------------------------* **def** createANewPopulation(self, numOfPop, numOfFea, OldPopulation, fitness):  
 NewPopulation = ndarray((numOfPop, numOfFea))  
  
 self.alpha -= (0.17 / self.NofIterations)  
 p = 0.5 \* (1 + self.alpha)  
 **for** i **in** range(numOfPop):  
 **for** j **in** range(numOfFea):  
 **if** self.VelocityM[i][j] <= self.alpha:  
 NewPopulation[i][j] = OldPopulation[i][j]  
 **elif** (self.VelocityM[i][j] > self.alpha) & (self.VelocityM[i][j] <= p):  
 NewPopulation[i][j] = self.LocalBestM[i][j]  
 **elif** (self.VelocityM[i][j] > p) & (self.VelocityM[i][j] <= 1):  
 NewPopulation[i][j] = self.GlobalBestRow[j]  
 **else**:  
 NewPopulation[i][j] = OldPopulation[i][j]  
 **return** NewPopulation  
 *#-------------------------------------------------------------------------------------------* **def** FindGlobalBestRow(self):  
 IndexOfBest = 0  
 numOfPop = self.LocalBestM.shape[0]  
 **for** i **in** range(numOfPop):  
 **if** (self.LocalBestM\_Fit[i] < self.GlobalBestFitness) \  
 & (self.LocalBestM\_Fit[i] > 0):  
 self.GlobalBestFitness = self.LocalBestM\_Fit[i]  
 IndexOfBest = i  
 copyto(self.GlobalBestRow, self.LocalBestM[IndexOfBest])  
 *#-------------------------------------------------------------------------------------------* **def** UpdateNewLocalBestMatrix(self, NewPopulation, NewPopFitness):  
 numOfPop = self.LocalBestM.shape[0]  
 **for** i **in** range(numOfPop):  
 **if** self.LocalBestM\_Fit[i] > NewPopFitness[i]:  
 copyto(self.LocalBestM[i], NewPopulation[i])  
 *#-------------------------------------------------------------------------------------------* **def** UpdateVelocityMatrix(self, NewPop, c1=2, c2=2, inertiaWeight=0.9):  
 numOfPop = self.VelocityM.shape[0]  
 numOfFea = self.VelocityM.shape[1]  
 **for** i **in** range(numOfPop):  
 **for** j **in** range(numOfFea):  
 term1 = c1 \* random.random() \* (self.LocalBestM[i][j] - NewPop[i][j])  
 term2 = c2 \* random.random() \* (self.GlobalBestRow[j] - NewPop[i][j])  
 self.VelocityM[i][j]=term1+term2+(inertiaWeight\*self.VelocityM[i][j])  
 *#-------------------------------------------------------------------------------------------* **def** PerformOneMillionIteration(self, numOfPop, numOfFea, population, fitness, model, fileW,  
 TrainX, TrainY, ValidateX, ValidateY, TestX, TestY):  
 NumOfGenerations = 1  
 OldPopulation = population  
 **while** (NumOfGenerations < self.NofIterations):  
 population = self.createANewPopulation(numOfPop, numOfFea, OldPopulation, fitness)  
 fittingStatus, fitness = self.fitnessdata.validate\_model(model,fileW, population, \  
 TrainX, TrainY, ValidateX, ValidateY, TestX, TestY)  
  
 self.UpdateNewLocalBestMatrix(population, fitness)  
 self.FindGlobalBestRow()  
 self.UpdateVelocityMatrix(population)  
  
 NumOfGenerations = NumOfGenerations + 1  
 print(NumOfGenerations)  
 **return** *#------------------------------------------------------------------------------* **def** \_\_init\_\_(self, numOfPop, numOfFea):  
 self.filedata = FromDataFileMLR.DataFromFile()  
 self.fitnessdata = FromFinessFileMLR.FitnessResults()  
 self.NofIterations = 2000  
 self.alpha = 0.5  
 self.GlobalBestRow = ndarray(numOfFea)  
 self.GlobalBestFitness = 10000  
 self.VelocityM = ndarray((numOfPop, numOfFea))  
 self.LocalBestM = ndarray((numOfPop, numOfFea))  
 self.LocalBestM\_Fit = ndarray(numOfPop)  
  
*#--------------------------------------------------------------------------------------------  
#Main program***def** main():  
 *# Number of descriptor should be 385 and number of population should be 50 or more* numOfPop = 50  
 numOfFea = 385  
  
 *# create an object of Multiple Linear Regression model.  
 # The class is located in mlr file* model = mlr.MLR()  
 filedata = FromDataFileMLR.DataFromFile()  
 fitnessdata = FromFinessFileMLR.FitnessResults()  
 analyzer = Fitness(numOfPop, numOfFea)  
  
 *# create an output file. Name the object to be FileW* fileW = analyzer.createAnOutputFile()  
  
 *# we continue exhancing the model; however if after 1000 iteration no  
 # enhancement is done, we can quit* unfit = 1000  
  
 *# Final model requirements: The following is used to evaluate each model. The minimum  
 # values for R^2 of training should be 0.6, R^2 of Validation should be 0.5 and R^2 of  
 # test should be 0.5* R2req\_train = .6  
 R2req\_validate = .5  
 R2req\_test = .5  
  
 *# getAllOfTheData is in FromDataFileMLR file. The following places the data  
 # (training data, validation data, and test data) into associated matrices* TrainX, TrainY, ValidateX, ValidateY, TestX, TestY = filedata.getAllOfTheData()  
 TrainX, ValidateX, TestX = filedata.rescaleTheData(TrainX, ValidateX, TestX)  
  
 fittingStatus = unfit  
 population = analyzer.createInitialPopulation(numOfPop,numOfFea)  
 fittingStatus, fitness = fitnessdata.validate\_model(model,fileW, population, \  
 TrainX, TrainY, ValidateX, ValidateY, TestX, TestY)  
  
 analyzer.CreateInitialVelocity(numOfPop, numOfFea)  
 copyto(analyzer.LocalBestM, population) *#initializing LocalBestMatrix as the initial population* copyto(analyzer.LocalBestM\_Fit, fitness)  
 analyzer.FindGlobalBestRow()  
  
 analyzer.PerformOneMillionIteration(numOfPop, numOfFea, population, fitness, model, fileW, \  
 TrainX, TrainY, ValidateX, ValidateY, TestX, TestY)  
*#main routine ends in here  
#------------------------------------------------------------------------------*main()  
*#------------------------------------------------------------------------------*

FromDataFileMLR:

**import** time *# provides timing for benchmarks***from** numpy **import** \* *# provides complex math and array functions***from** sklearn **import** svm *# provides Support Vector Regression***import** csv  
**import** math  
**import** sys  
  
**class** DataFromFile:  
 *#------------------------------------------------------------------------------* **def** getTwoDecPoint(self, x):  
 **return** float(**"%.2f"**%x)  
 *#------------------------------------------------------------------------------* **def** placeDataIntoArray(self, fileName):  
 **with** open(fileName, mode=**'r'**) **as** csvfile:  
 datareader = csv.reader(csvfile, delimiter=**','**, quotechar=**' '**)  
 dataArray = array([row **for** row **in** datareader], dtype=float64, order=**'C'**)  
  
 **if** (min(dataArray.shape) == 1): *# flatten arrays of one row or column* **return** dataArray.flatten(order=**'C'**)  
 **else**:  
 **return** dataArray  
 *#------------------------------------------------------------------------------* **def** getAllOfTheData(self):  
 TrainX = self.placeDataIntoArray(**'Train-Data.csv'**)  
 TrainY = self.placeDataIntoArray(**'Train-pIC50.csv'**)  
 ValidateX = self.placeDataIntoArray(**'Validation-Data.csv'**)  
 ValidateY = self.placeDataIntoArray(**'Validation-pIC50.csv'**)  
 TestX = self.placeDataIntoArray(**'Test-Data.csv'**)  
 TestY = self.placeDataIntoArray(**'Test-pIC50.csv'**)  
 **return** TrainX, TrainY, ValidateX, ValidateY, TestX, TestY  
 *#------------------------------------------------------------------------------* **def** rescaleTheData(self, TrainX, ValidateX, TestX):  
  
 *# 1 degree of freedom means (ddof) N-1 unbiased estimation* TrainXVar = TrainX.var(axis = 0, ddof=1)  
 TrainXMean = TrainX.mean(axis = 0)  
  
 **for** i **in** range(0, TrainX.shape[0]):  
 TrainX[i,:] = (TrainX[i,:] - TrainXMean)/sqrt(TrainXVar)  
 **for** i **in** range(0, ValidateX.shape[0]):  
 ValidateX[i,:] = (ValidateX[i,:] - TrainXMean)/sqrt(TrainXVar)  
 **for** i **in** range(0, TestX.shape[0]):  
 TestX[i,:] = (TestX[i,:] - TrainXMean)/sqrt(TrainXVar)  
  
 **return** TrainX, ValidateX, TestX  
 *#------------------------------------------------------------------------------*

FromFitnessFileMLR:

**import** time *# provides timing for benchmarks***from** numpy **import** \* *# provides complex math and array functions  
#from sklearn import svm # provides Support Vector Regression***import** csv  
**import** math  
**import** sys  
**import** hashlib  
  
**import** FromDataFileMLR  
**import** mlr  
  
**class** FitnessResults:  
 **def** \_\_init\_\_(self):  
 self.filedata = FromDataFileMLR.DataFromFile()  
 *#------------------------------------------------------------------------------* **def** r2(self, y, yHat):  
 *#Coefficient of determination* numer = ((y - yHat)\*\*2).sum() *# Residual Sum of Squares* denom = ((y - y.mean())\*\*2).sum()  
 r2 = 1 - numer/denom  
 **return** r2  
 *#------------------------------------------------------------------------------* **def** r2Pred(self, yTrain, yTest, yHatTest):  
 numer = ((yHatTest - yTest)\*\*2).sum()  
 denom = ((yTest - yTrain.mean())\*\*2).sum()  
 r2Pred = 1 - numer/denom  
 **return** r2Pred  
 *#------------------------------------------------------------------------------* **def** cv\_predict(self, model, set\_x, set\_y):  
 *# Predict using cross validation.* yhat = empty\_like(set\_y)  
 **for** idx **in** range(0, yhat.shape[0]):  
 train\_x = delete(set\_x, idx, axis=0)  
 train\_y = delete(set\_y, idx, axis=0)  
 modelName = model.fit(train\_x, train\_y)  
 yhat[idx] = model.predict(set\_x[idx])  
 **return** yhat  
 *#------------------------------------------------------------------------------  
 #Ahmad Hadaegh: Modified on: July 16, 2013* **def** calc\_fitness(self, xi, Y, Yhat, c=2):  
 *"""  
 Calculate fitness of a prediction.  
 xi : array\_like -- Mask of features to measure fitness of. Must be of dtype bool.  
 c : float -- Adjustment parameter.  
 """* p = len(xi) *# Number of selected parameters* n = len(Y) *# Sample size* numer = ((Y - Yhat)\*\*2).sum()/n *# Mean square error* pcn = p\*(c/n)  
 **if** pcn >= 1:  
 **return** 1000  
 denom = (1 - pcn)\*\*2  
 theFitness = numer/denom  
 **return** theFitness  
 *#------------------------------------------------------------------------------  
 #Ahmad Hadaegh: Modified on: July 16, 2013* **def** InitializeTracks(self):  
 trackDesc = {}  
 trackFitness = {}  
 trackModel = {}  
 trackR2 = {}  
 trackQ2 = {}  
 trackR2PredValidation = {}  
 trackR2PredTest = {}  
  
 **return** trackDesc, trackFitness, trackModel, trackR2, trackQ2, \  
 trackR2PredValidation, trackR2PredTest  
 *#------------------------------------------------------------------------------  
 #Ahmad Hadaegh: Modified on: July 16, 2013* **def** initializeYDimension(self):  
 yTrain = {}  
 yHatTrain = {}  
 yHatCV = {}  
 yValidation = {}  
 yHatValidation = {}  
 yTest = {}  
 yHatTest = {}  
 **return** yTrain, yHatTrain, yHatCV, yValidation, yHatValidation, yTest, yHatTest  
 *#------------------------------------------------------------------------------* **def** OnlySelectTheOnesColumns(self, popI):  
 numOfFea = popI.shape[0]  
 xi = zeros(numOfFea)  
 **for** j **in** range(numOfFea):  
 xi[j] = popI[j]  
  
 xi = xi.nonzero()[0]  
 xi = xi.tolist()  
 **return** xi  
 *#------------------------------------------------------------------------------* **def** validate\_model(self, model, fileW, population, TrainX, TrainY, ValidateX, ValidateY, TestX, TestY):  
 numOfPop = population.shape[0]  
 fitness = zeros(numOfPop)  
 c = 2  
 false = 0  
 true = 1  
 predictive = false  
  
 trackDesc, trackFitness,trackModel,trackR2, trackQ2, \  
 trackR2PredValidation, trackR2PredTest = self.InitializeTracks()  
  
 yTrain, yHatTrain, yHatCV, yValidation, \  
 yHatValidation, yTest, yHatTest = self.initializeYDimension()  
  
 unfit = 1000  
 itFits = 1  
 **for** i **in** range(numOfPop):  
 xi = self.OnlySelectTheOnesColumns(population[i])  
  
 idx = hashlib.sha1(array(xi)).digest()  
  
 X\_train\_masked = TrainX.T[xi].T  
  
 X\_validation\_masked = ValidateX.T[xi].T  
 X\_test\_masked = TestX.T[xi].T  
  
 **try**:  
 model\_desc = model.fit(X\_train\_masked, TrainY)  
 **except**:  
 **return** unfit, fitness  
  
 *# Computed predicted values* Yhat\_cv = self.cv\_predict(model, X\_train\_masked, TrainY) *# Cross Validation* Yhat\_validation = model.predict(X\_validation\_masked)  
 Yhat\_test = model.predict(X\_test\_masked)  
  
 *# Compute R2 statistics (Prediction for Valiation and Test set)* q2\_loo = self.r2(TrainY, Yhat\_cv)  
 q2\_loo = self.filedata.getTwoDecPoint(q2\_loo)  
  
 r2pred\_validation = self.r2Pred(TrainY, ValidateY, Yhat\_validation)  
 r2pred\_validation = self.filedata.getTwoDecPoint(r2pred\_validation)  
  
 r2pred\_test = self.r2Pred(TrainY, TestY, Yhat\_test)  
 r2pred\_test = self.filedata.getTwoDecPoint(r2pred\_test)  
  
 Y\_fitness = append(TrainY, ValidateY)  
 Yhat\_fitness = append(Yhat\_cv, Yhat\_validation)  
  
 fitness[i] = self.calc\_fitness(xi, Y\_fitness, Yhat\_fitness, c)  
  
 **if** predictive **and** ((q2\_loo < 0.5) **or** (r2pred\_validation < 0.5) **or** (r2pred\_test < 0.5)):  
 *# if it's not worth recording, just return the fitness* print(**"ending the program because of predictive is: "**, predictive)  
 **continue** *# Compute predicted Y\_hat for training set.* Yhat\_train = model.predict(X\_train\_masked)  
 r2\_train = self.r2(TrainY, Yhat\_train)  
  
 idxLength = len(xi)  
  
 *# store stats* trackDesc[idx] = str(xi)  
  
 trackFitness[idx] = self.filedata.getTwoDecPoint(fitness[i])  
  
 trackModel[idx] = model\_desc  
  
 trackR2[idx] = self.filedata.getTwoDecPoint(r2\_train)  
 trackQ2[idx] = self.filedata.getTwoDecPoint(q2\_loo)  
 trackR2PredValidation[idx] = self.filedata.getTwoDecPoint(r2pred\_validation)  
 trackR2PredTest[idx] = self.filedata.getTwoDecPoint(r2pred\_test)  
  
 yTrain[idx] = TrainY.tolist()  
  
 yHatTrain[idx] = Yhat\_train.tolist()  
 **for** i **in** range(len(yHatTrain[idx])):  
 yHatTrain[idx][i] = self.filedata.getTwoDecPoint(yHatTrain[idx][i])  
  
 yHatCV[idx] = Yhat\_cv.tolist()  
 **for** i **in** range(len(yHatCV[idx])):  
 yHatCV[idx][i] = self.filedata.getTwoDecPoint(yHatCV[idx][i])  
  
 yValidation[idx] = ValidateY.tolist()  
  
 yHatValidation[idx] = Yhat\_validation.tolist()  
 **for** i **in** range(len(yHatValidation[idx])):  
 yHatValidation[idx][i] = self.filedata.getTwoDecPoint(yHatValidation[idx][i])  
  
 yTest[idx] = TestY.tolist()  
  
 yHatTest[idx] = Yhat\_test.tolist()  
 **for** i **in** range(len(yHatTest[idx])):  
 yHatTest[idx][i] = self.filedata.getTwoDecPoint(yHatTest[idx][i])  
  
 self.write(model,fileW, trackDesc, trackFitness, trackModel, trackR2,\  
 trackQ2,trackR2PredValidation, trackR2PredTest)  
  
 **return** itFits, fitness  
 *#---------------------------------------------------------------------------* **def** write(self, model,fileW, trackDesc, trackFitness, trackModel, trackR2,\  
 trackQ2,trackR2PredValidation, trackR2PredTest):  
  
 **for** key **in** trackFitness.keys():  
 fileW.writerow([trackDesc[key], trackFitness[key], trackModel[key], \  
 trackR2[key], trackQ2[key], trackR2PredValidation[key], trackR2PredTest[key]])  
  
 *#fileOut.close()  
 #------------------------------------------------------------------------------*

mlr:

*"""Multiple Linear Regression"""***import** numpy **as** np   
  
  
**class** MLR:  
 *"""Multiple Linear Regression"""* **def** \_\_init\_\_(self):  
 *"""Initialization"""* self.coef = **None  
   
 def** fit(self, x\_set, y\_set):  
 *"""Fit to training X and Y arrays"""  
 # Add a column of 1's for the intercept* x\_set = np.append(np.ones((x\_set.shape[0], 1)), x\_set, axis=1)  
 self.coef = np.linalg.lstsq(x\_set, y\_set)[0]  
 **return 'MLR'  
   
 def** predict(self, x\_set):  
 *"""Predict a Y from an X, object must already be fitted."""  
 # matrix multiplication of X appended with a column of 1's (for  
 # intercepts) and the coeficients* **if** len(x\_set.shape) == 1:  
 x\_set = np.reshape(x\_set, (1, x\_set.shape[0]))  
 x\_set = np.append(np.ones((x\_set.shape[0], 1)), x\_set, axis=1)  
 **return** np.dot(x\_set, self.coef)  
  
 **def** printing(self):  
 *"""Predict a Y from an X, object must already be fitted."""* print(**"How are you doing?"**)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Descriptor ID | Fitness | Model | R2 | Q2 | R2Pred\_Validation | R2Pred\_Test |
|  |  |  |  |  |  |  |
| [229, 254, 357, 362] | 0.64 | MLR | 0.06 | -0.16 | 0.11 | -0.53 |
|  |  |  |  |  |  |  |
| [7, 22, 37, 56, 114, 131] | 0.94 | MLR | 0.09 | -0.17 | -0.69 | -0.65 |
|  |  |  |  |  |  |  |
| [16, 17, 88, 155, 225, 296, 302, 360] | 1.15 | MLR | 0.19 | -0.25 | -0.79 | -1.02 |
|  |  |  |  |  |  |  |
| [52, 121, 230, 279] | 0.7 | MLR | 0.06 | -0.15 | -0.13 | -0.25 |
|  |  |  |  |  |  |  |
| [48, 116, 125, 183] | 0.66 | MLR | 0.03 | -0.2 | 0.1 | 0.01 |
|  |  |  |  |  |  |  |
| [35, 266, 323, 382] | 0.69 | MLR | 0.08 | -0.14 | -0.11 | -0.13 |
|  |  |  |  |  |  |  |
| [27, 40, 60, 120, 151, 193, 213, 331] | 1.5 | MLR | 0.32 | -0.38 | -1.76 | -0.92 |
|  |  |  |  |  |  |  |
| [16, 60, 213, 237, 294, 355] | 1.04 | MLR | 0.21 | -0.01 | -1.32 | -1.28 |
|  |  |  |  |  |  |  |
| [72, 112, 128, 171, 314, 350, 368] | 0.83 | MLR | 0.35 | 0.13 | -0.55 | -0.32 |
|  |  |  |  |  |  |  |
| [23, 171, 205, 275, 334] | 0.57 | MLR | 0.32 | 0.1 | 0.16 | 0.46 |
|  |  |  |  |  |  |  |
| [26, 67, 159, 204, 218, 307, 310, 338, 354, 366] | 3.08 | MLR | 0.13 | -3.03 | -1.25 | -1.52 |
|  |  |  |  |  |  |  |
| [99, 170, 339, 381] | 0.56 | MLR | 0.27 | 0.03 | 0.14 | 0.2 |
|  |  |  |  |  |  |  |
| [14, 68, 81, 284] | 0.72 | MLR | 0.06 | -0.12 | -0.3 | -0.13 |
|  |  |  |  |  |  |  |
| [54, 179, 236, 250] | 0.62 | MLR | 0.13 | -0.08 | 0.07 | 0.18 |
|  |  |  |  |  |  |  |
| [15, 75, 168, 222, 267, 343, 344] | 0.82 | MLR | 0.28 | -0.13 | -0.08 | 0.39 |
|  |  |  |  |  |  |  |
| [3, 289, 329, 379] | 0.72 | MLR | 0.07 | -0.12 | -0.31 | -0.38 |
|  |  |  |  |  |  |  |
| [103, 111, 114, 219, 228] | 0.77 | MLR | 0.04 | -0.29 | 0 | -1.04 |
|  |  |  |  |  |  |  |
| [153, 222, 310] | 0.64 | MLR | 0.04 | -0.12 | -0.09 | 0.01 |
|  |  |  |  |  |  |  |
| [87, 244, 279, 378] | 0.71 | MLR | 0.08 | -0.11 | -0.25 | -0.24 |
|  |  |  |  |  |  |  |
| [47, 94, 166, 244, 257, 293] | 0.72 | MLR | 0.25 | -0.26 | 0.31 | 0.16 |
|  |  |  |  |  |  |  |
| [63, 135, 137, 338] | 0.77 | MLR | 0.19 | 0 | -0.73 | -0.8 |
|  |  |  |  |  |  |  |
| [18, 45, 87, 102, 108, 302, 318] | 1.18 | MLR | 0.25 | -0.69 | -0.44 | -0.37 |
|  |  |  |  |  |  |  |
| [38, 179, 181, 193, 217, 295, 300, 353, 356] | 1.42 | MLR | 0.3 | -0.17 | -1.51 | -1.54 |
|  |  |  |  |  |  |  |
| [2, 8, 30, 142, 201, 231, 291, 300, 381] | 0.99 | MLR | 0.21 | -0.11 | -0.27 | -0.23 |
|  |  |  |  |  |  |  |
| [58, 83, 140, 197, 235, 341] | 0.81 | MLR | 0.11 | -0.17 | -0.16 | -0.34 |
|  |  |  |  |  |  |  |
| [68, 87, 228, 253, 281, 316, 323, 328, 343] | 1.95 | MLR | 0.4 | 0.09 | -3.6 | -8.51 |
|  |  |  |  |  |  |  |
| [127, 142, 175, 176, 334] | 0.61 | MLR | 0.2 | 0.01 | 0.13 | -0.3 |
|  |  |  |  |  |  |  |
| [16, 205, 213, 244, 250, 254, 287, 317] | 0.99 | MLR | 0.19 | -0.34 | -0.11 | 0.37 |
|  |  |  |  |  |  |  |
| [130, 172, 174, 321] | 0.64 | MLR | 0.05 | -0.19 | 0.15 | -0.22 |
|  |  |  |  |  |  |  |
| [87, 168, 264] | 0.57 | MLR | 0.21 | 0.04 | -0.05 | 0.19 |
|  |  |  |  |  |  |  |
| [14, 26, 43, 51, 66, 84, 141, 144] | 2.01 | MLR | 0.37 | -2.32 | -0.22 | -0.12 |
|  |  |  |  |  |  |  |
| [17, 41, 92, 180, 356, 371] | 0.97 | MLR | 0.26 | 0.04 | -1.16 | -1.54 |
|  |  |  |  |  |  |  |
| [136, 307, 311, 366] | 0.66 | MLR | 0.09 | -0.06 | -0.11 | -0.15 |
|  |  |  |  |  |  |  |
| [170, 197, 207, 287, 298] | 0.68 | MLR | 0.29 | -0.16 | 0.14 | 0.18 |
|  |  |  |  |  |  |  |
| [32, 97, 139, 190, 247, 263, 279, 337, 352] | 1.54 | MLR | 0.27 | -0.86 | -0.72 | -1.06 |
|  |  |  |  |  |  |  |
| [103, 163, 180, 235, 270, 297, 309] | 0.75 | MLR | 0.25 | -0.02 | -0.01 | 0.11 |
|  |  |  |  |  |  |  |
| [5, 78, 97, 170, 224, 315, 384] | 0.92 | MLR | 0.37 | -0.39 | -0.02 | -0.01 |
|  |  |  |  |  |  |  |
| [61, 158, 187, 282, 308] | 0.78 | MLR | 0.16 | -0.12 | -0.35 | -0.08 |
|  |  |  |  |  |  |  |
| [109, 119, 232, 242, 281, 352, 376] | 1.42 | MLR | 0.18 | -0.95 | -0.87 | -0.34 |
|  |  |  |  |  |  |  |
| [186, 194, 198, 208, 288, 307] | 1.09 | MLR | 0.24 | -0.59 | -0.55 | -0.31 |
|  |  |  |  |  |  |  |
| [91, 189, 198] | 1.26 | MLR | 0.11 | -1.85 | -0.11 | -0.18 |
|  |  |  |  |  |  |  |
| [15, 82, 196, 295] | 0.75 | MLR | 0.09 | -0.09 | -0.47 | -0.16 |
|  |  |  |  |  |  |  |
| [51, 66, 89, 225, 230, 245, 340, 353] | 1.41 | MLR | 0.12 | -0.21 | -1.76 | -1.19 |
|  |  |  |  |  |  |  |
| [43, 84, 169, 224, 243, 349] | 0.79 | MLR | 0.23 | -0.02 | -0.34 | -0.64 |
|  |  |  |  |  |  |  |
| [20, 129, 169, 205, 309] | 0.73 | MLR | 0.09 | -0.18 | -0.05 | -0.28 |
|  |  |  |  |  |  |  |
| [81, 107, 141, 161, 214, 233, 327] | 0.83 | MLR | 0.22 | -0.1 | -0.18 | 0.07 |
|  |  |  |  |  |  |  |
| [0, 50, 122, 144, 219, 249, 297, 337] | 1.02 | MLR | 0.23 | -0.22 | -0.41 | -0.65 |
|  |  |  |  |  |  |  |
| [24, 31, 83, 101, 184, 262, 338, 344] | 0.76 | MLR | 0.46 | 0.13 | -0.11 | 0.03 |
|  |  |  |  |  |  |  |
| [37, 54, 62, 297] | 0.71 | MLR | 0.1 | -0.14 | -0.2 | -0.15 |
|  |  |  |  |  |  |  |
| [30, 76, 79, 116, 164, 299, 330] | 1.15 | MLR | 0.46 | -0.81 | -0.14 | -1.4 |
|  |  |  |  |  |  |  |
| [159, 204, 218, 307, 310, 338, 354] | 0.98 | MLR | 0.18 | -0.16 | -0.62 | -0.52 |
|  |  |  |  |  |  |  |
| [87, 244, 279, 378, 381] | 0.74 | MLR | 0.09 | -0.15 | -0.13 | -0.17 |
|  |  |  |  |  |  |  |
| [99, 170, 197, 207, 298] | 0.58 | MLR | 0.3 | 0 | 0.3 | 0.29 |
|  |  |  |  |  |  |  |
| [24, 31, 83, 101, 170, 184, 338, 344] | 0.72 | MLR | 0.51 | 0.16 | -0.02 | 0.17 |
|  |  |  |  |  |  |  |
| [60, 120, 151, 193, 331, 381] | 1.19 | MLR | 0.24 | -0.53 | -1.05 | -0.95 |
|  |  |  |  |  |  |  |
| [52, 121, 230] | 0.64 | MLR | 0.05 | -0.13 | -0.11 | -0.32 |
|  |  |  |  |  |  |  |
| [103, 111, 219, 228] | 0.74 | MLR | 0.04 | -0.26 | -0.16 | -1.36 |
|  |  |  |  |  |  |  |
| [127, 142, 175, 176, 334, 339, 381] | 0.81 | MLR | 0.21 | -0.14 | -0.01 | -0.43 |
|  |  |  |  |  |  |  |
| [72, 99, 112, 171, 314, 339, 350, 368, 381] | 0.84 | MLR | 0.38 | -0.03 | 0.09 | 0.54 |
|  |  |  |  |  |  |  |
| [23, 171, 205, 275, 334, 339] | 0.67 | MLR | 0.32 | -0.01 | 0.11 | 0.47 |
|  |  |  |  |  |  |  |
| [170, 307, 381] | 0.54 | MLR | 0.28 | 0.06 | 0.04 | 0.02 |
|  |  |  |  |  |  |  |
| [99, 170, 339, 381] | 0.56 | MLR | 0.27 | 0.03 | 0.14 | 0.2 |
|  |  |  |  |  |  |  |
| [14, 68, 81, 284] | 0.72 | MLR | 0.06 | -0.12 | -0.3 | -0.13 |
|  |  |  |  |  |  |  |
| [32, 97, 190, 247, 279, 337, 339, 352, 381] | 1.87 | MLR | 0.26 | -0.71 | -2.04 | -2.22 |
|  |  |  |  |  |  |  |
| [16, 17, 155, 296, 302, 339] | 0.87 | MLR | 0.15 | -0.18 | -0.38 | 0.28 |
|  |  |  |  |  |  |  |
| [87, 170] | 0.49 | MLR | 0.26 | 0.13 | 0.05 | 0.11 |
|  |  |  |  |  |  |  |
| [35, 323] | 0.58 | MLR | 0.02 | -0.09 | -0.05 | -0.08 |
|  |  |  |  |  |  |  |
| [43, 224, 243, 339, 349] | 0.74 | MLR | 0 | -0.27 | 0.08 | 0 |
|  |  |  |  |  |  |  |
| [5, 97, 170, 381, 384] | 0.71 | MLR | 0.31 | -0.03 | -0.2 | -0.12 |
|  |  |  |  |  |  |  |
| [22, 37, 56, 99, 114] | 0.72 | MLR | 0.12 | -0.07 | -0.18 | 0.09 |
|  |  |  |  |  |  |  |
| [15, 75, 168, 222, 343, 344] | 0.84 | MLR | 0.07 | -0.25 | -0.14 | -0.02 |
|  |  |  |  |  |  |  |
| [68, 281, 316, 323, 328, 343] | 1.2 | MLR | 0.13 | -0.52 | -1.12 | -0.36 |
|  |  |  |  |  |  |  |
| [99, 222, 310, 339, 381] | 0.67 | MLR | 0.02 | -0.19 | 0.22 | 0.22 |
|  |  |  |  |  |  |  |
| [63, 137, 338] | 0.71 | MLR | 0.21 | 0.08 | -0.75 | -0.82 |
|  |  |  |  |  |  |  |
| [17, 41, 92, 180, 339, 371] | 0.94 | MLR | 0.17 | -0.05 | -0.89 | -1.07 |
|  |  |  |  |  |  |  |
| [289, 329, 379, 381] | 0.64 | MLR | 0.04 | -0.14 | 0.09 | 0.01 |
|  |  |  |  |  |  |  |
| [38, 99, 179, 181, 193, 295] | 1.11 | MLR | 0.13 | -0.62 | -0.58 | -0.33 |
|  |  |  |  |  |  |  |
| [48, 116, 125, 183, 339] | 0.79 | MLR | 0.1 | -0.11 | -0.39 | -0.16 |
|  |  |  |  |  |  |  |
| [58, 83, 197, 235] | 0.7 | MLR | 0.09 | -0.08 | -0.26 | -0.34 |
|  |  |  |  |  |  |  |
| [30, 76, 79, 99, 164, 170, 299, 330] | 1.32E+27 | MLR | 0.57 | -2.66E+27 | 0 | -0.95 |
|  |  |  |  |  |  |  |
| [130, 174, 321] | 0.59 | MLR | 0.05 | -0.14 | 0.15 | -0.2 |
|  |  |  |  |  |  |  |
| [15, 82, 196, 339] | 0.83 | MLR | 0.1 | -0.22 | -0.62 | -0.01 |
|  |  |  |  |  |  |  |
| [0, 50, 144, 219, 249, 297, 337, 339, 381] | 1.54 | MLR | 0.27 | -0.74 | -0.93 | -1.13 |
|  |  |  |  |  |  |  |
| [14, 26, 43, 51, 66, 84, 141, 144] | 2.01 | MLR | 0.37 | -2.32 | -0.22 | -0.12 |
|  |  |  |  |  |  |  |
| [107, 141, 161, 170, 214, 233, 339] | 0.79 | MLR | 0.4 | -0.05 | -0.09 | 0.12 |
|  |  |  |  |  |  |  |
| [109, 119, 232, 281, 352, 376, 381] | 1.46 | MLR | 0.15 | -0.16 | -2.34 | -1.46 |
|  |  |  |  |  |  |  |
| [94, 166, 170, 244, 257, 293, 339] | 0.96 | MLR | 0.29 | -0.62 | 0.21 | 0.21 |
|  |  |  |  |  |  |  |
| [51, 66, 89, 170, 245, 340] | 0.71 | MLR | 0.31 | 0.03 | -0.11 | -0.03 |
|  |  |  |  |  |  |  |
| [20, 129, 169, 205] | 0.66 | MLR | 0.09 | -0.14 | 0.03 | -0.15 |
|  |  |  |  |  |  |  |
| [99, 186, 194, 198, 208, 307, 381] | 1.26 | MLR | 0.2 | -1.05 | -0.13 | -0.21 |
|  |  |  |  |  |  |  |
| [236, 250] | 0.59 | MLR | 0.04 | -0.11 | -0.06 | 0.07 |
|  |  |  |  |  |  |  |
| [8, 30, 142, 201, 231, 291, 381] | 0.79 | MLR | 0.15 | -0.15 | 0.06 | -0.23 |
|  |  |  |  |  |  |  |
| [229, 357, 362] | 0.61 | MLR | 0.03 | -0.14 | 0.07 | -0.06 |
|  |  |  |  |  |  |  |
| [16, 60, 213, 237, 294] | 0.77 | MLR | 0.08 | -0.16 | -0.26 | 0.01 |
|  |  |  |  |  |  |  |
| [45, 87, 108, 170, 302] | 0.62 | MLR | 0.27 | -0.03 | 0.17 | 0.18 |
|  |  |  |  |  |  |  |
| [91, 99, 170, 198, 381] | 0.66 | MLR | 0.28 | -0.07 | 0.05 | 0.08 |
|  |  |  |  |  |  |  |
| [16, 205, 213, 244] | 0.61 | MLR | 0.16 | -0.05 | 0.1 | 0.12 |
|  |  |  |  |  |  |  |
| [158, 170, 187, 282, 308] | 0.65 | MLR | 0.28 | -0.01 | 0.01 | 0.18 |
|  |  |  |  |  |  |  |
| [163, 180, 270, 297] | 0.64 | MLR | 0.18 | 0.04 | -0.21 | -0.01 |
|  |  |  |  |  |  |  |