EPA ToxCast Challenge

LELPredictor

Scott Norin (RigelFive) – Hudson Ohio USA
rigelfive@gmail.com
14 - 28 May 2014

I. Problem Statement

The EPA created a tool to predict the toxicity of new man-made chemicals and passed a proof-of-concept phase in 2009. This tool utilized chemical toxicity data from over 30 years which comprised over $2 billion in animal testing. One of the parameters to be calculated from this test includes the Lowest Effect Level (LEL), which is the lowest dosage of the chemical that shows adverse effects in animals.

The goal of the TopCoder / EPA ToxCast Challenge is to build an algorithm using the data provided to predict a chemical's systemic -log(LEL) value. This prediction tool is to provide significant value through saving an extensive amount of time, money and quality of life for animals in performing these tests. The data provided for the challenge includes in vitro assays, chemical properties and chemical structural descriptors.

II. Requirements

A set of atomized requirements were generated to assist in the formulation of the LELPredictor algorithm. The essential requirements for the tool were to:

1) read input data provided,
2) perform numerical computations to determine the correlation from a specific set of intrinsic parameters with the LEL value,
3) output the predicted LEL values into an array to be scored by the TopCoder website.

III. Design Decisions

Three methodologies were attempted to maximize the score for the ToxCast challenge. The focus of this tool was to utilize intrinsic parameters such as molecular weight rather than attempting to correlate extrinsic parameters such as AC50 or -log(LEL). The three approaches attempted include :

1) Utilize the freeware Python-based code RDKit to determine the intrinsic parameters of TPSA, max Q, min Q, log P and molecular weight.
2) Utilize values provided in the data set including a count of the chemical's atoms, bonds, chains, groups, rings and molecular weight.
3) Utilize a single intrinsic parameter associated with the chemical's size which is a function of the molecular weight.

Through processing of the data files, the RDKit tool was unable to process all of the SMILES format chemical structures that were provided in this competition. While, the RDKit tool was able to calculate intrinsic parameters for some of the chemicals, individual tailoring of the chemical input was not able to be completed in time to achieve the competition deadline.

The topological polar surface area (TPSA) is a value that helps to assess the ability of a molecule to permeate through cells based on the surface area . Max Q and min Q are values that quantify the electronic charging typically seen with a molecule. The value of log P provides an indirect assessment of a molecule to be in solution versus act as a condensed solid. Each of these values are typically used with informatics methodologies to determine a molecules characteristic behavior.

The final algorithm generated by approach #3 utilized the input value of molecular weight and had better correlation statistics than what was determined in approach #2 that used inputs of atoms, bonds, chains, groups, and rings that was provided in the data set.

While the approach used is over simplified, it is difficult to derive extrinsic properties of adjacent chemicals by any method and have a high confidence that would offset the necessary EPA testing.

IV. Software Architecture

The main methods used to generate the -log(LEL) values are generated using classes shown in the UML2 block definition diagram in Figure 1. The LELPredictor tool utilizes code that is grouped into three categories:

1) data structures to handle the chemical data,
2) classes to read data provided for the TopCoder EPA ToxCast competition,
3) a neural network classes that are used to minimize the error of a unique curve fit through a loosely correlated set of data.

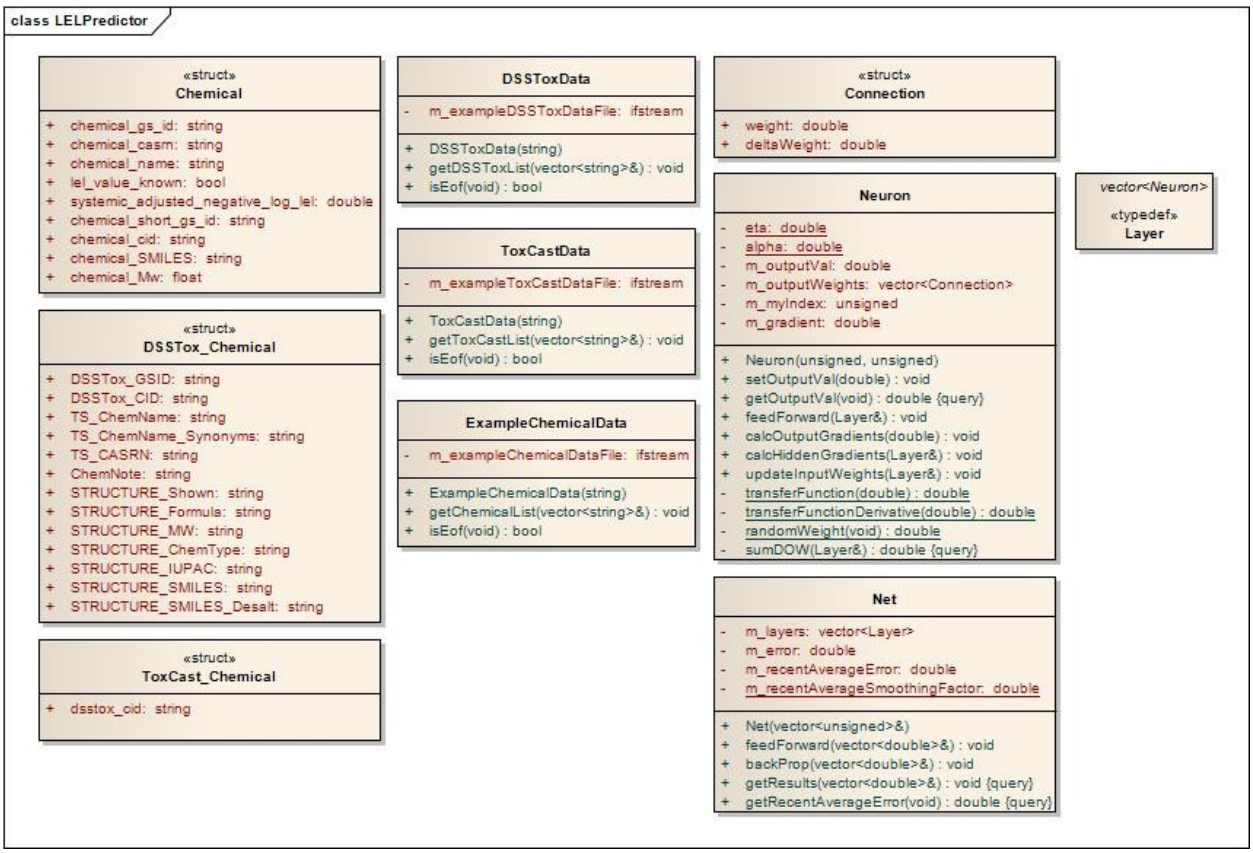


Figure 1 – Block Definition Diagram for the RigelFive LELPredictor

The algorithm that was finally developed utilized a single parameter input which was the value of molecular weight. The known -log(LEL) values from a set of chemicals were used with the molecular weight parameter to train a small multi-layer perceptron (MLP) neural network. The trained neural network was used with an adjacent set of chemicals where the -log(LEL) value is unknown. The neural network code used was based off of the publicly available tutorials by David Miller at the following link: http://www.millermattson.com/dave/?p=54

The topology of the neural network used to estimate the -log(LEL) value is shown in Figure 2. The neural network utilized a parabolic filter to normalize the input values and a parabolic amplifier function to generate the output values. The parabolic amplifier was defined using the provided chemical data in the ToxCast data files (those with LEL values known). The equations for the parabolic filters and amplifier are provided in Appendix C.

The correlation of molecular weight to the estimated -log(LEL) value is shown in Figure 3.

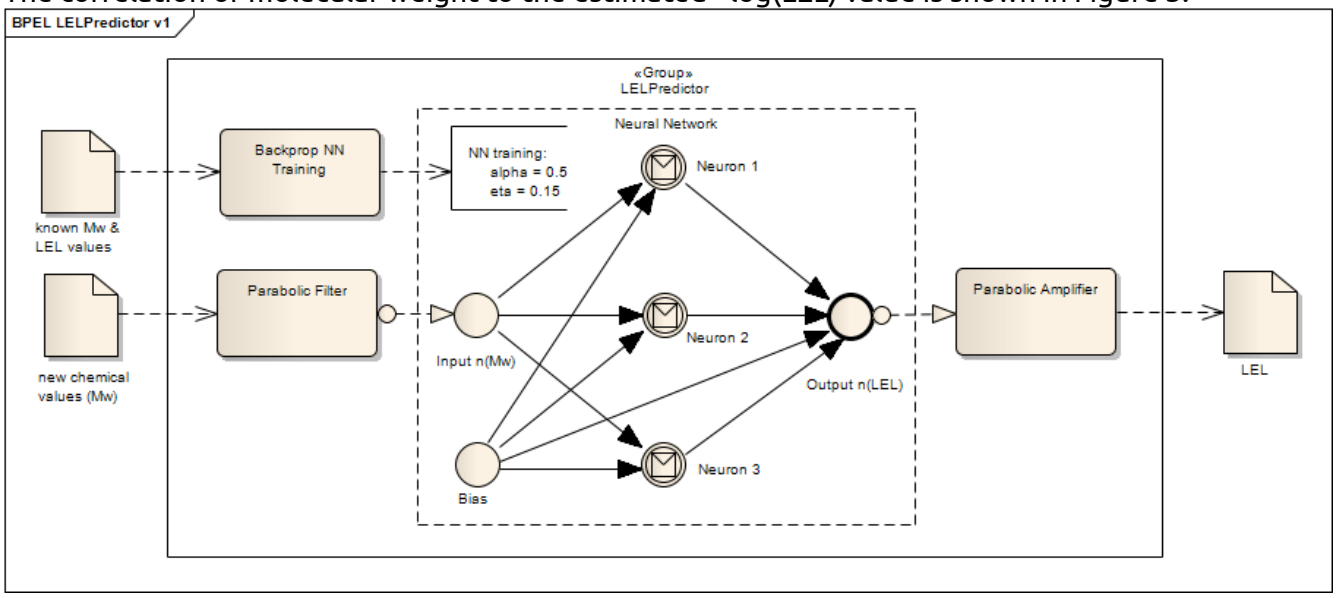


Figure 2 – High Level Architecture Used for the LELPredictor Tool

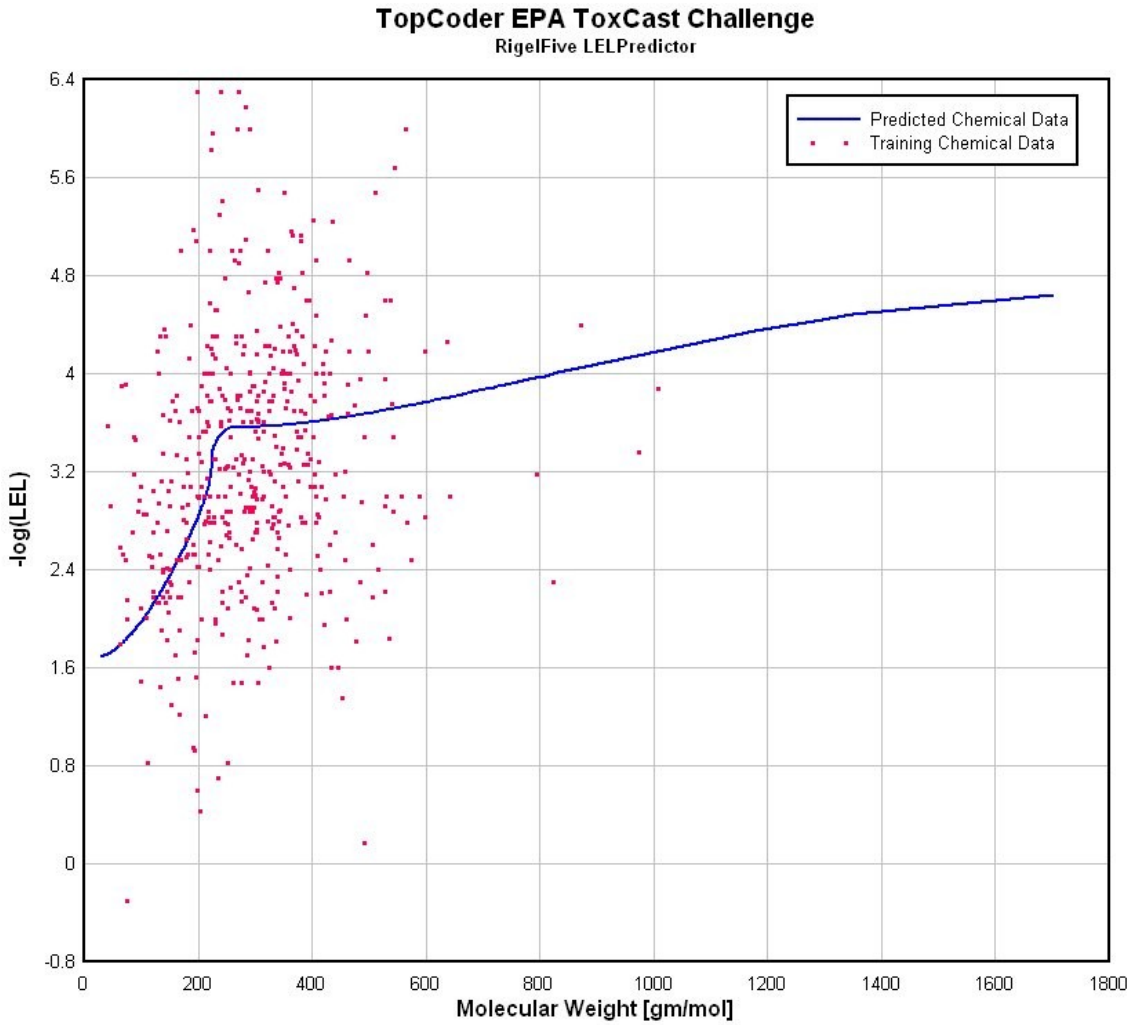


Figure 3 – Correlation Determined using the RigelFive LELPredictor

Conclusions

An estimation of the -log(LEL) value was fit to known data that has a low level of correlation to a single parameter. This low correlation is attributed to the -log(LEL) value being an extrinsic parameter than cannot be cleanly derived using intrinsic parameters . A curve fit through the -log(LEL) data was generated by minimizing the error using a neural network scheme that utilized a single parameter input of molecular weight.

If additional time were provided, the approach using input parameters calculated from the informatics code RDKit would be pursued further. The input parameters of TPSA, max Q, min Q and log P should be derived using improved SMILES chemical structures that are compatible with RDKit.

The provisional score using this method generated 869,183 points. The final score generated 693,773 points and finished in 24th place for this competition.

Appendix A - LELPredictor v1 C++ code (final submission)

```
 1 /* ================================================================================================================= *
 2  * EPA ToxCast Challenge / TopCoder
 3  * PredictLEL v1 [Competition Sensitive]
 4  * RigelFive - Hudson Ohio USA
 5  * 12 May 2014 - 14 May 2014
 6  * ================================================================================================================= *
 7  * version history:
 8  * v0: Utilize Mw, atoms, bonds, chains, groups and rings as inputs.
 9  * v1: Utilize Mw as the only input.
10  *
11  * ================================================================================================================= *
12  * Atomized Software Specification:
13  * Customer Requirements:
14  *  1. Predict the LEL value for untested compounds using previous tests of other chemicals.
15  *
16  * Functional Requirements:
17  *  1. Estimate the LEL value based on information provided from untested compounds.
18  *
19  * Interface Requirements:
20  *  1. Output a double[] with the negative log value of the LEL estimated.
21  *
22  * Performance Requirements:
23  *  1. No time or memory requirements as no computations are required in the submitted code
24  *
25  * Design Requirements:
26  *  1. Utilize C++.
27  *  2. Model a NN to calculate the LEL value using a pseudo Joback-like method where
28  *     the main input parameters are: Mw, atoms, bonds, chains, groups and rings.
29  *  3. Determine generalized extrinsic parameters can be estimated from specific intrinsic parameters.
30  *  4. Do not use external databases related to the EPA or other agencies to determine the LEL values.
31  *
32  * Quality Requirements:
33  *  1. Maximize overall score of the algorithm (objective: 1,000,000 points).
34  *
35  * Verification Requirements:
36  *  1. Output 1854 values in a double[] array.
37  *
38  * ================================================================================================================= *
39  */
40
41 #define _LELPredictor_ 0; // output the version number
42
43 #include <cstdlib>
44 #include <cassert>              // remove
45 #include <cmath>                // remove
46 #include <fstream>              // remove
47 #include <sstream>              // remove
```

```
47 #include <sstream>              // remove
48 #include <algorithm>            // remove
49 #include <cstdio>
50 #include <ctime>                // remove
51 #include <vector>
52 #include <iostream>
53 #include <boost/tokenizer.hpp>  // remove
54
55 using namespace std;
56
57 struct Chemical {
58     string chemical_gs_id;
59     string chemical_casrn;
60     string chemical_name;
61     bool lel_value_known;
62     double systemic_adjusted_negative_log_lel;
63     string chemical_short_gs_id;   // truncate the "DSSTox_GSID_" and keep the id number in a string
64     string chemical_cid;     // matches the id in the DSSTox_Chemical and ToxCast_Chemical data, found in the DSSTox_Chemical_List
65     string chemical_SMILES;
66     float chemical_Mw;
67 };
68 struct DSSTox_Chemical {
69      string DSSTox_GSID;
70      string DSSTox_CID;
71      string TS_ChemName;
72      string TS_ChemName_Synonyms;
73      string TS_CASRN;
74      string ChemNote;
75      string STRUCTURE_Shown;
76      string STRUCTURE_Formula;
77      string STRUCTURE_MW;
78      string STRUCTURE_ChemType;
79      string STRUCTURE_IUPAC;
80      string STRUCTURE_SMILES;
81      string STRUCTURE_SMILES_Desalt;
82 };
83 struct ToxCast_Chemical {
84     string dsstox_cid; //DSSTox_CID
85 };
86
87 struct Connection {
88     double weight;
89     double deltaWeight;
90 };
91
92 class Neuron;
93 typedef vector<Neuron> Layer;
```

```
 94 class Neuron {
 95 public:
 96     Neuron(unsigned numOutputs, unsigned myIndex);
 97     void setOutputVal(double val) { m_outputVal = val; }
 98     double getOutputVal(void) const { return m_outputVal; }
 99     void feedForward(const Layer &prevLayer);
100     void calcOutputGradients(double targetVal);
101     void calcHiddenGradients(const Layer &nextLayer);
102     void updateInputWeights(Layer &prevLayer);
103
104 private:
105     static double eta;
106     static double alpha;
107     static double transferFunction(double x);
108     static double transferFunctionDerivative(double x);
109     static double randomWeight(void) { return rand() / double(RAND_MAX); }
110     double sumDOW(const Layer &nextLayer) const;
111     double m_outputVal;
112     vector<Connection> m_outputWeights;
113     unsigned m_myIndex;
114     double m_gradient;
115 };
116 double Neuron::eta = 0.15;
117 double Neuron::alpha = 0.50;
118 void Neuron::updateInputWeights(Layer &prevLayer) {
119     for (unsigned n = 0; n < prevLayer.size(); ++n) {
120         Neuron &neuron = prevLayer[n];
121         double oldDeltaWeight = neuron.m_outputWeights[m_myIndex].deltaWeight;
122         double newDeltaWeight =
123                 eta
124                 * neuron.getOutputVal()
125                 * m_gradient
126                 + alpha
127                 * oldDeltaWeight;
128         neuron.m_outputWeights[m_myIndex].deltaWeight = newDeltaWeight;
129         neuron.m_outputWeights[m_myIndex].weight += newDeltaWeight;
130     }
131 }
132 double Neuron::sumDOW(const Layer &nextLayer) const {
133     double sum = 0.0;
134
135     for (unsigned n = 0; n < nextLayer.size() - 1; ++n) {
136         sum += m_outputWeights[n].weight * nextLayer[n].m_gradient;
137     }
138     return sum;
139 }
140 void Neuron::calcHiddenGradients(const Layer &nextLayer) {
```

```
141     double dow = sumDOW(nextLayer);
142     m_gradient = dow * Neuron::transferFunctionDerivative(m_outputVal);
143 }
144 void Neuron::calcOutputGradients(double targetVal) {
145     double delta = targetVal - m_outputVal;
146     m_gradient = delta * Neuron::transferFunctionDerivative(m_outputVal);
147 }
148 double Neuron::transferFunction(double x) {
149     return tanh(x);
150 }
151 double Neuron::transferFunctionDerivative(double x) {
152     return 1.0 - x * x;
153 }
154 void Neuron::feedForward(const Layer &prevLayer) {
155     double sum = 0.0;
156 
157     for (unsigned n = 0; n < prevLayer.size(); ++n) {
158         sum += prevLayer[n].getOutputVal() *
159                 prevLayer[n].m_outputWeights[m_myIndex].weight;
160     }
161     m_outputVal = Neuron::transferFunction(sum);
162 }
163 Neuron::Neuron(unsigned numOutputs, unsigned myIndex) {
164     for (unsigned c = 0; c < numOutputs; ++c) {
165         m_outputWeights.push_back(Connection());
166         m_outputWeights.back().weight = randomWeight();
167     }
168     m_myIndex = myIndex;
169 }
170 
171 // ****************** class Net ******************
172 class Net
173 {
174 public:
175     Net(const vector<unsigned> &topology);
176     void feedForward(const vector<double> &inputVals);
177     void backProp(const vector<double> &targetVals);
178     void getResults(vector<double> &resultVals) const;
179     double getRecentAverageError(void) const { return m_recentAverageError; }
180 
181 private:
182     vector<Layer> m_layers; // m_layers[layerNum][neuronNum]
183     double m_error;
184     double m_recentAverageError;
185     static double m_recentAverageSmoothingFactor;
186 };
```

```
187
188 double Net::m_recentAverageSmoothingFactor = 100.0; // Number of training samples to average over
189
190 void Net::getResults(vector<double> &resultVals) const
191 {
192     resultVals.clear();
193
194     for (unsigned n = 0; n < m_layers.back().size() - 1; ++n) {
195         resultVals.push_back(m_layers.back()[n].getOutputVal());
196     }
197 }
198
199 void Net::backProp(const vector<double> &targetVals)
200 {
201     // Calculate overall net error (RMS of output neuron errors)
202
203     Layer &outputLayer = m_layers.back();
204     m_error = 0.0;
205
206     for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {
207         double delta = targetVals[n] - outputLayer[n].getOutputVal();
208         m_error += delta * delta;
209     }
210     m_error /= outputLayer.size() - 1; // get average error squared
211     m_error = sqrt(m_error); // RMS
212
213     // Implement a recent average measurement
214
215     m_recentAverageError =
216             (m_recentAverageError * m_recentAverageSmoothingFactor + m_error)
217             / (m_recentAverageSmoothingFactor + 1.0);
218
219     // Calculate output layer gradients
220
221     for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {
222         outputLayer[n].calcOutputGradients(targetVals[n]);
223     }
224
225     // Calculate hidden layer gradients
226
227     for (unsigned layerNum = m_layers.size() - 2; layerNum > 0; --layerNum) {
228         Layer &hiddenLayer = m_layers[layerNum];
229         Layer &nextLayer = m_layers[layerNum + 1];
230
231         for (unsigned n = 0; n < hiddenLayer.size(); ++n) {
232             hiddenLayer[n].calcHiddenGradients(nextLayer);
233         }
```

```
234     }
235
236     // For all layers from outputs to first hidden layer,
237     // update connection weights
238
239     for (unsigned layerNum = m_layers.size() - 1; layerNum > 0; --layerNum) {
240         Layer &layer = m_layers[layerNum];
241         Layer &prevLayer = m_layers[layerNum - 1];
242
243         for (unsigned n = 0; n < layer.size() - 1; ++n) {
244             layer[n].updateInputWeights(prevLayer);
245         }
246     }
247 }
248
249 void Net::feedForward(const vector<double> &inputVals)
250 {
251     assert(inputVals.size() == m_layers[0].size() - 1);
252
253     // Assign (latch) the input values into the input neurons
254     for (unsigned i = 0; i < inputVals.size(); ++i) {
255         m_layers[0][i].setOutputVal(inputVals[i]);
256     }
257
258     // forward propagate
259     for (unsigned layerNum = 1; layerNum < m_layers.size(); ++layerNum) {
260         Layer &prevLayer = m_layers[layerNum - 1];
261         for (unsigned n = 0; n < m_layers[layerNum].size() - 1; ++n) {
262             m_layers[layerNum][n].feedForward(prevLayer);
263         }
264     }
265 }
266
267 Net::Net(const vector<unsigned> &topology)
268 {
269     unsigned numLayers = topology.size();
270     for (unsigned layerNum = 0; layerNum < numLayers; ++layerNum) {
271         m_layers.push_back(Layer());
272         unsigned numOutputs = layerNum == topology.size() - 1 ? 0 : topology[layerNum + 1];
273
274         // We have a new layer, now fill it with neurons, and
275         // add a bias neuron in each layer.
276         for (unsigned neuronNum = 0; neuronNum <= topology[layerNum]; ++neuronNum) {
277             m_layers.back().push_back(Neuron(numOutputs, neuronNum));
278         }
279
```

```
280         // Force the bias node's output to 1.0 (it was the last neuron pushed in this layer):
281         m_layers.back().back().setOutputVal(1.0);
282     }
283 }
284
285 class ExampleChemicalData
286 {
287 public:
288     ExampleChemicalData(const string filename);
289     void getChemicalList(vector<string> &ChemicalList);
290     bool isEof(void) { return m_exampleChemicalDataFile.eof(); }
291 private:
292     ifstream m_exampleChemicalDataFile;
293 };
294 ExampleChemicalData::ExampleChemicalData(const string filename)
295 {
296     m_exampleChemicalDataFile.open(filename.c_str());
297 }
298 void ExampleChemicalData::getChemicalList(vector<string> &ChemicalList) {
299     string line;
300
301     getline(m_exampleChemicalDataFile, line);
302     ChemicalList.push_back(line);
303     return;
304 }
305
306 class DSSToxData
307 {
308 public:
309     DSSToxData(const string filename);
310     void getDSSToxList(vector<string> &DSSToxList);
311     bool isEof(void) { return m_exampleDSSToxDataFile.eof(); }
312 private:
313     ifstream m_exampleDSSToxDataFile;
314 };
315 DSSToxData::DSSToxData(const string filename)
316 {
317     m_exampleDSSToxDataFile.open(filename.c_str());
318 }
319 void DSSToxData::getDSSToxList(vector<string> &DSSToxList) {
320     string line;
321
322     getline(m_exampleDSSToxDataFile, line);
323     DSSToxList.push_back(line);
324     return;
325 }
326
```

```
327 class ToxCastData
328 {
329 public:
330     ToxCastData(const string filename);
331     void getToxCastList(vector<string> &ToxCastList);
332     bool isEof(void) { return m_exampleToxCastDataFile.eof(); }
333 private:
334     ifstream m_exampleToxCastDataFile;
335 };
336 ToxCastData::ToxCastData(const string filename)
337 {
338     m_exampleToxCastDataFile.open(filename.c_str());
339 }
340 void ToxCastData::getToxCastList(vector<string> &ToxCastList) {
341     string line;
342 
343     getline(m_exampleToxCastDataFile, line);
344     ToxCastList.push_back(line);
345     return;
346 }
347 
348 string chop_dsstox (string id_to_chop) {
349     // truncate the "DSSTox_GSID_" and keep the id number in a string
350 
351     stringstream ss(id_to_chop);
352     string blank1;
353     string blank2;
354     string id;
355 
356     getline(ss,blank1,'_') &&
357     getline(ss,blank2,'_') &&
358     getline(ss,id);
359 
360     return id;
361 }
362 
363 vector<string> tokenize(const string& line )
364 {
365     boost::escaped_list_separator<char> sep( '\\', ',', '"' ) ;
366     boost::tokenizer< boost::escaped_list_separator<char> > tokenizer( line, sep ) ;
367     return std::vector<std::string>( tokenizer.begin(), tokenizer.end() ) ;
368 }
369 
370 void output_c(Chemical c) {
371     cout << c.chemical_gs_id << " " <<
372             c.chemical_cid << " " <<
```

```
373             c.chemical_casrn << " " <<
374             c.chemical_name << " " <<
375             c.chemical_Mw << " " <<
376             c.chemical_SMILES  << " " <<
377             c.systemic_adjusted_negative_log_lel << endl;
378 }
379
380 void output_dc(DSSTox_Chemical dc) {
381     cout << dc.DSSTox_GSID << " " <<
382             dc.DSSTox_CID << " " <<
383             dc.TS_ChemName << " " <<
384             dc.TS_ChemName_Synonyms << " " <<
385             dc.TS_CASRN << " " <<
386             dc.ChemNote << " " <<
387             dc.STRUCTURE_Shown << " " <<
388             dc.STRUCTURE_Formula << " " <<
389             dc.STRUCTURE_MW << " " <<
390             dc.STRUCTURE_ChemType << " " <<
391             dc.STRUCTURE_IUPAC << " " <<
392             dc.STRUCTURE_SMILES << " " <<
393             dc.STRUCTURE_SMILES_Desalt << endl;
394 }
395
396 void output_tc(ToxCast_Chemical tc) {
397     cout << tc.dsstox_cid << " " <<
398             endl;
399 }
400
401 void showVectorVals(string label, vector<double> &v)
402 {
403     cout << label << " ";
404     for (unsigned i = 0; i < v.size(); ++i) {
405         cout << v[i] << " ";
406     }
407
408     cout << endl;
409 }
410
411 double sum(vector<double> v) {
412     double sum = 0.0;
413
414     int vSIZE = v.size();
415     for (int i = 0; i < vSIZE; i++) {
416         sum = sum + double(v.at(i));
417     }
418     return double(sum);
419 }
```

```
420 double average(vector<double> v) {
421     double sum = 0.0;
422 
423     int vSIZE = v.size();
424     for (int i = 0; i < vSIZE; i++) {
425         sum = sum + abs(v.at(i));
426     }
427     return double(sum/vSIZE);
428 }
429 double stddev2(vector<double> v) {
430     double s2;
431     double sum = 0.0;
432     double sum2 = 0.0;
433     double avg;
434 
435     int vSIZE = v.size();
436 
437     for (int i = 0; i < vSIZE; i++) {
438         sum = sum + abs(v.at(i));
439     }
440     avg = double(sum / vSIZE);
441 
442     for (int i = 0; i < vSIZE; i++) {
443         sum2 = sum2 + pow(((double) v.at(i) - avg),2.0);
444     }
445 
446     s2 = (1.0/((double) vSIZE-1.0)) * abs(sum2);
447 
448     return s2;
449 }
450 double min(vector<double> v) {
451     double min1 = 1e+09;
452     int vSIZE = v.size();
453 
454     for (int i = 0; i < vSIZE; i++) {
455         if (v.at(i) < min1) { min1 = v.at(i); };
456     }
457 
458     return double(min1);
459 }
460 double max(vector<double> v) {
461     double max1 = 0;
462 
463     int vSIZE = v.size();
464 
465     for (int i = 0; i < vSIZE; i++) {
```

```
466         if (v.at(i) > max1) { max1 = v.at(i); };
467     }
468
469     return double(max1);
470 }
471
472 double parabola(double a, double b, double c, double x) {
473     double y;
474
475     y = (a*pow((x-b),2.0)) + c;
476
477     return y;
478 }
479 double parabolic_filter(double x, double x_mean, double x_min, double x_max) {
480     double y, norm_x;
481     double a1, a2, b, c1, c2;
482
483     norm_x = x / x_max;
484
485     a1 = 1.0 / pow((x_mean-x_min),2.0);
486     a2 = 1.0 / pow((x_max-x_mean),2.0);
487     b = x_mean;
488     c1 = 0.0;
489     c2 = 0.0;
490
491     if (x < 0.0) {
492         y = -1.0;
493     } else if (x > x_max) {
494         y = 1.0;
495     } else {
496         if (x <= x_mean) {
497             y = -parabola(a1, b, c1, x);
498         }
499         else if (x > x_mean) {
500             y = parabola(a2, b, c2, x);
501         }
502     }
503     return y;
504 }
505 double parabolic_amplifier(double y, double x_mean, double x_min, double x_max) {
506
507     double x;
508     double a1, a2, b, c1, c2;
509
510     a1 = 1.0 / pow((x_mean-x_min),2.0);
511     a2 = 1.0 / pow((x_max-x_mean),2.0);
512     b = x_mean;
```

```
513     c1 = 0.0;
514     c2 = 0.0;
515 
516     if (y > 0) {
517         x = (1.0/(2.0*a2))*((2.0*a2*b)+(2.0*sqrt(a2*y-a2*c2)));
518     } else {
519         x = (1.0/(2.0*a1))*((2.0*a1*b)-(2.0*sqrt(-a1*y-a1*c1)));
520     }
521     return x;
522 }
523 
524 int main(int argc, char** argv) {
525 
526     string path = "/home/scott/Arctria/ARC-34 Yogi/v00/";
527     ExampleChemicalData exChemicalData("/home/scott/Arctria/ARC-34 Yogi/data/ToxCast_MM_Data/ToxRefDB_Challenge_Training.csv");
528     DSSToxData exDSSToxData("/home/scott/Arctria/ARC-34 Yogi/data/DSSTox/csv/TOX21S_v4a_8599_11Dec2013.csv");
529     ToxCastData exToxCastData("/home/scott/Arctria/ARC-34 Yogi/data/DSSTox/csv/toxprint_v2_vs_TOX21S_v4a_8599_03Dec2013.csv");
530 
531     // Initialize parameters
532     vector<double> the_big_answer;
533 
534     vector<string> tokenized_string_vector;
535 
536     vector<string> ChemicalList;
537     vector<string> DSS_ToxData_ChemicalList;
538     vector<string> ToxCast_ChemicalList;
539     vector<double> LEL_values;
540     vector<double> Mw_values;
541     vector<double> Predicted_LEL_values;
542 
543     vector<Chemical> Initial_Chemicals;
544     vector<Chemical> Training_Chemicals;
545     vector<Chemical> Predict_Chemicals;
546     vector<DSSTox_Chemical> DSS_Chemicals;
547     vector<ToxCast_Chemical> ToxCast_Chemicals;
548 
549     Chemical c;
550     DSSTox_Chemical dc;
551     ToxCast_Chemical tc;
552     Chemical pc;
553 
554     string in;
555     string blank_line;
556 
557     int val;
558     double d_val;
```

```
559
560     double Mw_min;
561     double Mw_mean;
562     double Mw_max;
563
564     double LEL_min;
565     double LEL_mean;
566     double LEL_max;
567
568     // LELPredict Program Timing Parameters for Training NN
569     bool startTraining = false;
570     bool finishTraining = false;
571     bool startCalculations = false;
572     bool finishCalculations = false;
573
574     // Program Timing Parameters
575     double training_start_time = 0.0;
576     double training_finish_time = 0.0;
577     double training_elapsed_time = 0.0;
578     double calculation_start_time = 0.0;
579     double calculation_finish_time = 0.0;
580     double calculation_elapsed_time = 0.0;
581
582     // Text analysis
583     bool quote_string = false;
584
585     for (int i = 1; i < argc; ++i) {
586         string arg = argv[i];
587         if ((arg == "-t") || (arg == "--training")) {
588             cout << "LELPredictor v00\nRigelFive - 12 May 2014" << endl;
589             cout << "================================================================================" << endl;
590
591             // Start the training
592             if (startTraining == false) {
593                 training_start_time = clock();
594                 startTraining = true;
595             }
596
597             // Read the .csv training file to build the Chemical List
598             while (!exChemicalData.isEof()) {
599                 exChemicalData.getChemicalList(ChemicalList);
600             }
601
602             // Parse the data using comma separated values from the Chemical List
603             for (int i = 1; i < ChemicalList.size()-1; i++) {
604                 in = ChemicalList.at(i);
605
```

```
606                 tokenized_string_vector.clear();
607                 boost::tokenizer<boost::escaped_list_separator<char> > tok(in);
608 
609                 for(boost::tokenizer<boost::escaped_list_separator<char> >::iterator beg=tok.begin(); beg!=tok.end();++beg){
610                     tokenized_string_vector.push_back(*beg);
611                 }
612 
613                 c.chemical_gs_id = tokenized_string_vector.at(0);
614                 c.chemical_casrn = tokenized_string_vector.at(1);
615                 c.chemical_name = tokenized_string_vector.at(2);
616                 c.chemical_short_gs_id = chop_dsstox(c.chemical_gs_id);
617                 c.chemical_cid = "-";
618                 c.chemical_SMILES = "-"; // temporary... changed to real value looked up from DSS_Chemicals
619                 c.chemical_Mw = 0.0; // temporary... changed to real value looked up from DSS_Chemicals
620 
621                 blank_line = tokenized_string_vector.at(3);
622 
623                 if (blank_line.find('?') != string::npos) {
624                     c.lel_value_known = false;
625                     c.systemic_adjusted_negative_log_lel = -1.0;
626                     Predict_Chemicals.push_back(c);
627                     Initial_Chemicals.push_back(c);
628 
629                 } else {
630                     c.lel_value_known = true;
631                     istringstream(blank_line) >> c.systemic_adjusted_negative_log_lel;
632                     LEL_values.push_back(c.systemic_adjusted_negative_log_lel);
633                     Training_Chemicals.push_back(c);
634                     Initial_Chemicals.push_back(c);
635                 }
636             }
637 
638             LEL_min = min(LEL_values);
639             LEL_mean = average(LEL_values);
640             LEL_max = max(LEL_values);
641 
642             cout << "Training Set Statistics:" << endl;
643             cout << "   Min LEL Value: " << min(LEL_values) << endl;
644             cout << "   Max LEL Value: " << LEL_max << endl;
645             cout << "   Average LEL Value: " << LEL_mean << endl;
646             cout << "   Sigma^2 of LEL Value: " << stddev2(LEL_values) << endl;
647             cout << "-------------------------------------------------------------------------------" << endl;
648 
649             // Build DSSTox_Chemical List
650             while (!exDSSToxData.isEof()) {
651                 exDSSToxData.getDSSToxList(DSS_ToxData_ChemicalList);
```

```
652             }
653
654             for (int i = 1; i < DSS_ToxData_ChemicalList.size()-1; i++) {
655                 in = DSS_ToxData_ChemicalList.at(i);
656                 tokenized_string_vector.clear();
657
658                 boost::escaped_list_separator<char> sep( '`', ',', '\"' ) ;         // apparently the character ` is never used!!!
659                 boost::tokenizer<boost::escaped_list_separator<char> > tok(in, sep);
660
661                 for(boost::tokenizer<boost::escaped_list_separator<char> >::iterator beg=tok.begin(); beg!=tok.end();++beg) {
662                     tokenized_string_vector.push_back(*beg);
663                 }
664
665                 dc.DSSTox_GSID = tokenized_string_vector.at(0);
666                 dc.DSSTox_CID = tokenized_string_vector.at(1);
667                 dc.TS_ChemName = tokenized_string_vector.at(2);
668                 dc.TS_ChemName_Synonyms = tokenized_string_vector.at(3);
669                 dc.TS_CASRN = tokenized_string_vector.at(4);
670                 dc.ChemNote = tokenized_string_vector.at(5);
671                 dc.STRUCTURE_Shown = tokenized_string_vector.at(6);
672                 dc.STRUCTURE_Formula = tokenized_string_vector.at(7);
673                 dc.STRUCTURE_MW = tokenized_string_vector.at(8);
674                 dc.STRUCTURE_ChemType = tokenized_string_vector.at(9);
675                 dc.STRUCTURE_IUPAC = tokenized_string_vector.at(10);
676                 dc.STRUCTURE_SMILES = tokenized_string_vector.at(11);
677                 dc.STRUCTURE_SMILES_Desalt = tokenized_string_vector.at(12);
678
679                 DSS_Chemicals.push_back(dc);
680
681                 istringstream(tokenized_string_vector.at(8)) >> d_val;
682                 Mw_values.push_back(d_val);
683             }
684
685             Mw_min = min(Mw_values);
686             Mw_mean = average(Mw_values);
687             Mw_max = max(Mw_values);
688
689             cout << "   Min Mw Value: " << min(Mw_values) << endl;
690             cout << "   Max Mw Value: " << Mw_max << endl;
691             cout << "   Average Mw Value: " << Mw_mean << endl;
692             cout << "   Sigma^2 of Mw Value: " << stddev2(Mw_values) << endl;
693             cout << "-------------------------------------------------------------------------------" << endl;
694
695             // Build ToxCast_Chemical List
696             while (!exToxCastData.isEof()) {
697                 exToxCastData.getToxCastList(ToxCast_ChemicalList);
698             }
```

```
699
700             for (int i = 1; i < ToxCast_ChemicalList.size()-1; i++) {
701                 in = ToxCast_ChemicalList.at(i);
702                 tokenized_string_vector.clear();
703
704                 boost::escaped_list_separator<char> sep( '\\', ',', '\"' ) ;
705                 boost::tokenizer<boost::escaped_list_separator<char> > tok(in, sep);
706
707                 for(boost::tokenizer<boost::escaped_list_separator<char> >::iterator beg=tok.begin(); beg!=tok.end();++beg) {
708                     tokenized_string_vector.push_back(*beg);
709                 }
710
711                 tc.dsstox_cid = tokenized_string_vector.at(0);
712
713                 ToxCast_Chemicals.push_back(tc);
714
715             }
716
717             // Output basic information about data in memory
718             cout << "   Total Chemicals Listed: " << ChemicalList.size()-2 << endl;     // reduced by 2 for the header line and footer line
719             cout << "   Size of Training Chemicals List: " << Training_Chemicals.size() << endl;
720             cout << "   Size of Prediction Chemicals List: " << Predict_Chemicals.size() << endl;
721             cout << "-------------------------------------------------------------------------------" << endl;
722             cout << "   Total DSS Tox Chemicals: " << DSS_Chemicals.size() << endl;
723             cout << "   Total ToxCast Chemicals: " << ToxCast_Chemicals.size() << endl;
724             cout << "-------------------------------------------------------------------------------" << endl;
725
726             // Verify that the training set is using a CAS Registry Number (pretty sure in the data guide)
727             int nocas = 0;
728             for (int i = 0; i < Training_Chemicals.size(); i++) {
729                 c = Training_Chemicals.at(i);
730                 in = c.chemical_casrn;
731                 if (in.find("NOCAS") != string::npos) {
732                     nocas++;
733                 }
734             }
735
736             cout << "   Number of non-CAS chemicals in the training set: " << nocas << endl;
737             cout << "-------------------------------------------------------------------------------" << endl;
738
739             cout << "Training Chemical SMILES Structures:" << endl;
740             cout << "-------------------------------------------------------------------------------" << endl;
741
742             int count_training_match_dsstox = 0;
743
744             for (int i = 0; i < Training_Chemicals.size(); i++) {
```

```
745                 c = Training_Chemicals.at(i);
746
747                 for (int k = 0; k < DSS_Chemicals.size(); k++) {
748                     dc = DSS_Chemicals.at(k);
749                     if (dc.DSSTox_GSID == c.chemical_short_gs_id) {
750                         c.chemical_cid = dc.DSSTox_CID;
751                         istringstream(dc.STRUCTURE_MW) >> c.chemical_Mw;
752                         c.chemical_SMILES = dc.STRUCTURE_SMILES;
753                         Training_Chemicals.at(i) = c;
754                         count_training_match_dsstox++;
755                     }
756                 }
757                 cout << c.chemical_SMILES << endl;
758             }
759
760             int count_predict_match_dsstox = 0;
761
762             cout << "Predict Chemical SMILES Structures:" << endl;
763             cout << "--------------------------------------------------------------------------------" << endl;
764
765             for (int i = 0; i < Predict_Chemicals.size(); i++) {
766                 c = Predict_Chemicals.at(i);
767
768                 for (int k = 0; k < DSS_Chemicals.size(); k++) {
769                     dc = DSS_Chemicals.at(k);
770                     if (dc.DSSTox_GSID == c.chemical_short_gs_id) {
771                         c.chemical_cid = dc.DSSTox_CID;
772                         istringstream(dc.STRUCTURE_MW) >> c.chemical_Mw;
773                         c.chemical_SMILES = dc.STRUCTURE_SMILES;
774                         Predict_Chemicals.at(i) = c;
775                         count_predict_match_dsstox++;
776                     }
777                 }
778
779                 cout << c.chemical_SMILES << endl;
780             }
781
782             vector<unsigned> topology;
783             topology.clear();
784             topology.push_back(1);  // 1 input:  Mw
785             topology.push_back(3); // 3 nodes in the hidden layer
786             topology.push_back(1);  // LEL value
787
788             cout << "Creating a <" <<
789                     topology.at(0) << " " <<
790                     topology.at(1) << " " <<
791                     topology.at(2) <<
```

```
                    topology.at(2) <<
                    "> neural network" << endl;

            Net myNet(topology);

            vector<double> inputVals, targetVals, resultVals;
            int trainingPass = 0;

            double NN_cycle_start = clock();
            double NN_cycle_stop = NN_cycle_start + (300*1000000.0);
            while (((myNet.getRecentAverageError() > 0.001) || (trainingPass < 5)) && (clock() < NN_cycle_stop)) {
                trainingPass++;
                for (int i = 0; i < Training_Chemicals.size(); i++) {
                    inputVals.clear();
                    targetVals.clear();

                    c = Training_Chemicals.at(i);

                    inputVals.push_back(parabolic_filter((double) c.chemical_Mw, Mw_mean, Mw_min, Mw_max));
                    targetVals.push_back(parabolic_filter((double) c.systemic_adjusted_negative_log_lel, LEL_mean, LEL_min, LEL_max));

                    myNet.feedForward(inputVals);
                    myNet.getResults(resultVals);

                    assert(targetVals.size() == topology.back());
                    myNet.backProp(targetVals);

                }
            }

            cout << "Neural Net Recent Average Error: " << myNet.getRecentAverageError() << endl;
            cout << "Total Training Passes:  " << trainingPass << endl;
            cout << "-------------------------------------------------------------------------------" << endl;

            // finish training calculations
            finishTraining = true;

            if (finishTraining == true) {
                training_finish_time = clock();
                training_elapsed_time = training_finish_time - training_start_time;
                cout << "training elapsed time: " << (training_elapsed_time / 1000000.0) << " sec" << endl;
                finishTraining = true;
                startCalculations = true;
            }

            // start calculations
            startCalculations = true;
```

```
838             if (startCalculations == true) {
839                 calculation_start_time = clock();
840                 startCalculations = false;
841             }
842 
843             // Calculate the LEL value using the trained NN
844             for (int i = 0; i < Predict_Chemicals.size(); i++) {
845                 inputVals.clear();
846                 c = Predict_Chemicals.at(i);
847 
848                 inputVals.push_back(parabolic_filter((double)c.chemical_Mw, Mw_mean, Mw_min, Mw_max));
849 
850                 myNet.feedForward(inputVals);
851                 myNet.getResults(resultVals);
852 
853                 c.systemic_adjusted_negative_log_lel = parabolic_amplifier(resultVals.at(0), LEL_mean, LEL_min, LEL_max);
854 
855                 Predicted_LEL_values.push_back(c.systemic_adjusted_negative_log_lel);
856                 Predict_Chemicals.at(i) = c;
857             }
858 
859             cout << "-------------------------------------------------------------------------------" << endl;
860 
861             LEL_min = min(Predicted_LEL_values);
862             LEL_mean = average(Predicted_LEL_values);
863             LEL_max = max(Predicted_LEL_values);
864 
865             cout << "Calculation / Prediction Set Statistics:" << endl;
866             cout << "   Min LEL Value: " << LEL_min << endl;
867             cout << "   Max LEL Value: " << LEL_max << endl;
868             cout << "   Average LEL Value: " << LEL_mean << endl;
869             cout << "   Sigma^2 of LEL Value: " << stddev2(Predicted_LEL_values) << endl;
870             cout << "-------------------------------------------------------------------------------" << endl;
871 
872             // calculate the answers based on training
873             finishCalculations = true;
874 
875             cout << "class LELPredictor { " << endl;
876             cout << "public:" << endl << "  vector<double> predictLEL(void);" << endl;
877             cout << "};" << endl;
878 
879             cout << "vector<double> LELPredictor::predictLEL(void) {" << endl;
880             cout << "   // Output the LEL values" << endl;
881             cout << "   vector<double> lel;" << endl << endl;
882 
883             for (int i = 0; i < Initial_Chemicals.size(); i++) {
884                 c = Initial_Chemicals.at(i);
```

```
885
886                 for (int j = 0; j < Predict_Chemicals.size(); j++) {
887                     pc = Predict_Chemicals.at(j);
888
889                     if(c.chemical_gs_id == pc.chemical_gs_id) {
890                         c.systemic_adjusted_negative_log_lel = pc.systemic_adjusted_negative_log_lel;
891                         Initial_Chemicals.at(i) = c;
892                     }
893                 }
894
895                 cout << "   lel.push_back(" << c.systemic_adjusted_negative_log_lel << ");" << endl;
896             }
897             cout << "   return lel;" << endl;
898             cout << "}" << endl << endl;
899
900             cout << "--------------------------------------------------------------------------------" << endl;
901
902             if (finishCalculations == true) {
903                 calculation_finish_time = clock();
904                 calculation_elapsed_time = calculation_finish_time - training_start_time;
905                 cout << "total elapsed time: " << (calculation_elapsed_time / 1000000.0) << " sec" << endl;
906                 cout << "================================================================================" << endl;
907                 finishCalculations = false;
908             }
909         }
910     }
911     return 0;
912 }
```

Appendix B - Output of Software Code

```
LELPredictor v01
RigelFive - 14 May 2014
================================================================================
Training Set Statistics:
   Min LEL Value: -0.301
   Max LEL Value: 6.301
   Average LEL Value: 3.2614
   Sigma^2 of LEL Value: 1.06245
--------------------------------------------------------------------------------
   Min Mw Value: 30.026
   Max Mw Value: 1701.2
   Average Mw Value: 263.587
   Sigma^2 of Mw Value: 21761.9
--------------------------------------------------------------------------------
   Total Chemicals Listed: 1854
   Size of Training Chemicals List: 483
   Size of Prediction Chemicals List: 1371
--------------------------------------------------------------------------------
   Total DSS Tox Chemicals: 1854
   Total ToxCast Chemicals: 8599
--------------------------------------------------------------------------------
   Number of non-CAS chemicals in the training set: 15
--------------------------------------------------------------------------------
Creating a <1 3 1> neural network
Neural Net Recent Average Error: 0.117189
Total Training Passes:  216418
--------------------------------------------------------------------------------
training elapsed time: 302.588 sec
--------------------------------------------------------------------------------
Calculation / Prediction Set Statistics:
   Min LEL Value: 1.68579
   Max LEL Value: 4.64588
   Average LEL Value: 2.95098
   Sigma^2 of LEL Value: 0.430856
--------------------------------------------------------------------------------
class LELPredictor {
public:
  vector<double> predictLEL(void);
};
vector<double> LELPredictor::predictLEL(void) {
   // Output the LEL values
   vector<double> lel;

   lel.push_back(3.56194);
   lel.push_back(3.574);
   lel.push_back(2.28469);
   lel.push_back(5.301);
   lel.push_back(-0.301);
```

```
 49    lel.push_back(6.301);
 50    lel.push_back(3.56297);
 51    lel.push_back(3.55568);
 52    lel.push_back(2.44423);
 53    lel.push_back(3.602);
 54    lel.push_back(3.5627);
 55    lel.push_back(2.273);
 56    lel.push_back(2.21577);
 57    lel.push_back(3);
 58    lel.push_back(3.56916);
 59    lel.push_back(2);
 60    lel.push_back(3.824);
 61    lel.push_back(3.59659);
 62    lel.push_back(1.602);
 63    lel.push_back(2.08122);
 64    lel.push_back(2.13152);
 65    lel.push_back(3.25);
 66    lel.push_back(1.903);
 67    lel.push_back(2);
 68    lel.push_back(3.5417);
 69    lel.push_back(3.56133);
 70    lel.push_back(2.67512);
 71    lel.push_back(2.76656);
 72    lel.push_back(3.17356);
 73    lel.push_back(2.76702);
 74    lel.push_back(3.57716);
 75    lel.push_back(3.56425);
 76    lel.push_back(3.74004);
 77    lel.push_back(2.50096);
 78    lel.push_back(3.562);
 79    lel.push_back(2.23254);
 80    lel.push_back(3.56867);
 81    lel.push_back(3.903);
 82    lel.push_back(3.61651);
 83    lel.push_back(1.347);
 84    lel.push_back(2.47336);
 85    lel.push_back(2.477);
 86    lel.push_back(1.83919);
 87    lel.push_back(3.01182);
 88    lel.push_back(3.079);
 89    lel.push_back(2.03908);
 90    lel.push_back(1.88895);
 91    lel.push_back(5);
 92    lel.push_back(2.59563);
 93    lel.push_back(3.61567);
 94    lel.push_back(3.1783);
 95    lel.push_back(2.505);
 96    lel.push_back(3.60339);
```

```
 97    lel.push_back(2.38407);
 98    lel.push_back(3.63617);
 99    lel.push_back(2.80698);
100    lel.push_back(2.35725);
101    lel.push_back(2.45508);
102    lel.push_back(2.56603);
103    lel.push_back(3.57634);
104    lel.push_back(2.38407);
105    lel.push_back(2.41892);
106    lel.push_back(3.56133);
107    lel.push_back(2.15757);
108    lel.push_back(3.64968);
109    lel.push_back(3.66737);
110    lel.push_back(2.90279);
111    lel.push_back(2.6646);
112    lel.push_back(2.84365);
113    lel.push_back(1.72501);
114    lel.push_back(3.12004);
115    lel.push_back(3.38969);
116    lel.push_back(1.77564);
117    lel.push_back(2.824);
118    lel.push_back(1.79933);
119    lel.push_back(2.854);
120    lel.push_back(2.12438);
121    lel.push_back(2.08152);
122    lel.push_back(3.56216);
123    lel.push_back(3.56477);
124    lel.push_back(2.523);
125    lel.push_back(3.57404);
126    lel.push_back(3.60876);
127    lel.push_back(3.87129);
128    lel.push_back(2.2168);
129    lel.push_back(1.98754);
130    lel.push_back(2.72584);
131    lel.push_back(2.23254);
132    lel.push_back(3.60235);
133    lel.push_back(2.24013);
134    lel.push_back(2.18519);
135    lel.push_back(3.58238);
136    lel.push_back(2.24048);
137    lel.push_back(2.35745);
138    lel.push_back(2.56684);
139    lel.push_back(0.426);
140    lel.push_back(2.40178);
141    lel.push_back(2.426);
142    lel.push_back(3.61123);
143    lel.push_back(3.56579);
144    lel.push_back(2.02633);
```

```
145    lel.push_back(2.18518);
146    lel.push_back(2.68441);
147    lel.push_back(2.699);
148    lel.push_back(2.40141);
149    lel.push_back(3.61902);
150    lel.push_back(2.31441);
151    lel.push_back(3.61187);
152    lel.push_back(3.55317);
153    lel.push_back(5.079);
154    lel.push_back(3.59341);
155    lel.push_back(0.949);
156    lel.push_back(2.28147);
157    lel.push_back(2.597);
158    lel.push_back(2.011);
159    lel.push_back(3.56275);
160    lel.push_back(2.66417);
161    lel.push_back(2.57369);
162    lel.push_back(3.26);
163    lel.push_back(3.903);
164    lel.push_back(2.41926);
165    lel.push_back(3.86018);
166    lel.push_back(2.13966);
167    lel.push_back(3.43033);
168    lel.push_back(3.63483);
169    lel.push_back(2.38407);
170    lel.push_back(3.57102);
171    lel.push_back(3.63615);
172    lel.push_back(1.9162);
173    lel.push_back(3.60566);
174    lel.push_back(3.57099);
175    lel.push_back(2.574);
176    lel.push_back(3.389);
177    lel.push_back(3.69575);
178    lel.push_back(2.98186);
179    lel.push_back(2.28822);
180    lel.push_back(3.176);
181    lel.push_back(3.58673);
182    lel.push_back(3.699);
183    lel.push_back(4.301);
184    lel.push_back(3.56201);
185    lel.push_back(3.17507);
186    lel.push_back(3.00924);
187    lel.push_back(3.56148);
188    lel.push_back(3.079);
189    lel.push_back(3.62097);
190    lel.push_back(1.83442);
191    lel.push_back(3.255);
192    lel.push_back(2.879);
```

```
193    lel.push_back(2.74572);
194    lel.push_back(2.34836);
195    lel.push_back(4.523);
196    lel.push_back(3.079);
197    lel.push_back(2.01325);
198    lel.push_back(3.54424);
199    lel.push_back(2.64458);
200    lel.push_back(1.96264);
201    lel.push_back(3.54462);
202    lel.push_back(3.59127);
203    lel.push_back(4.301);
204    lel.push_back(2.38407);
205    lel.push_back(3.921);
206    lel.push_back(2.4644);
207    lel.push_back(2.83318);
208    lel.push_back(3.347);
209    lel.push_back(4.222);
210    lel.push_back(3.01574);
211    lel.push_back(2.45544);
212    lel.push_back(2.90196);
213    lel.push_back(2.59563);
214    lel.push_back(2.34912);
215    lel.push_back(1.962);
216    lel.push_back(3.721);
217    lel.push_back(3.62926);
218    lel.push_back(2.99566);
219    lel.push_back(1.204);
220    lel.push_back(2.43649);
221    lel.push_back(4.745);
222    lel.push_back(2.83223);
223    lel.push_back(3.42454);
224    lel.push_back(2.44751);
225    lel.push_back(3.02188);
226    lel.push_back(3.98287);
227    lel.push_back(2.8013);
228    lel.push_back(2.34912);
229    lel.push_back(1.86535);
230    lel.push_back(3.678);
231    lel.push_back(5.079);
232    lel.push_back(3.61277);
233    lel.push_back(2.24844);
234    lel.push_back(3.00924);
235    lel.push_back(1.97498);
236    lel.push_back(2.17759);
237    lel.push_back(3.56188);
238    lel.push_back(3.59066);
239    lel.push_back(2.29822);
240    lel.push_back(3.37469);
```

```
241    lel.push_back(4.301);
242    lel.push_back(2.33993);
243    lel.push_back(3.56252);
244    lel.push_back(3.18774);
245    lel.push_back(3.56351);
246    lel.push_back(3.59257);
247    lel.push_back(3.62293);
248    lel.push_back(2.08091);
249    lel.push_back(1.83899);
250    lel.push_back(2.87926);
251    lel.push_back(2.64292);
252    lel.push_back(2.45544);
253    lel.push_back(4.347);
254    lel.push_back(3.469);
255    lel.push_back(3.02442);
256    lel.push_back(3.204);
257    lel.push_back(2.08065);
258    lel.push_back(1.88357);
259    lel.push_back(3.824);
260    lel.push_back(3.57228);
261    lel.push_back(3.5675);
262    lel.push_back(2.87977);
263    lel.push_back(2.49104);
264    lel.push_back(2.24809);
265    lel.push_back(2.23254);
266    lel.push_back(1.98752);
267    lel.push_back(4);
268    lel.push_back(3.56211);
269    lel.push_back(3.5855);
270    lel.push_back(3.58851);
271    lel.push_back(2.92733);
272    lel.push_back(2.60472);
273    lel.push_back(3.57722);
274    lel.push_back(1.699);
275    lel.push_back(2.62435);
276    lel.push_back(3.56594);
277    lel.push_back(3.54462);
278    lel.push_back(3.82528);
279    lel.push_back(2.36616);
280    lel.push_back(3.5158);
281    lel.push_back(4.602);
282    lel.push_back(3.53884);
283    lel.push_back(3.58841);
284    lel.push_back(2.53789);
285    lel.push_back(2.12454);
286    lel.push_back(4.602);
287    lel.push_back(3.56035);
288    lel.push_back(2.47296);
```

```
289    lel.push_back(3.56479);
290    lel.push_back(2.51006);
291    lel.push_back(3.55157);
292    lel.push_back(3.57103);
293    lel.push_back(2.58533);
294    lel.push_back(2.91494);
295    lel.push_back(3.47964);
296    lel.push_back(1.815);
297    lel.push_back(1.88357);
298    lel.push_back(2.52843);
299    lel.push_back(2.08348);
300    lel.push_back(2.68745);
301    lel.push_back(3.56875);
302    lel.push_back(2.301);
303    lel.push_back(2.56566);
304    lel.push_back(3.62);
305    lel.push_back(3.4817);
306    lel.push_back(2.29824);
307    lel.push_back(3.62226);
308    lel.push_back(1.99406);
309    lel.push_back(3.6416);
310    lel.push_back(1.875);
311    lel.push_back(2.83221);
312    lel.push_back(2.301);
313    lel.push_back(3.491);
314    lel.push_back(3);
315    lel.push_back(2.40141);
316    lel.push_back(2.23254);
317    lel.push_back(2.15464);
318    lel.push_back(3.48085);
319    lel.push_back(3.57954);
320    lel.push_back(3.56443);
321    lel.push_back(2.08122);
322    lel.push_back(2.36133);
323    lel.push_back(2.90299);
324    lel.push_back(2.38);
325    lel.push_back(2.50041);
326    lel.push_back(2.23219);
327    lel.push_back(3.079);
328    lel.push_back(1.98136);
329    lel.push_back(3.481);
330    lel.push_back(3.69065);
331    lel.push_back(3.56193);
332    lel.push_back(3.57568);
333    lel.push_back(2.60473);
334    lel.push_back(2.13917);
335    lel.push_back(2.15757);
336    lel.push_back(3);
```

```
337    lel.push_back(3.57604);
338    lel.push_back(3.58937);
339    lel.push_back(2.2011);
340    lel.push_back(2.23288);
341    lel.push_back(2.50966);
342    lel.push_back(2.52843);
343    lel.push_back(3.824);
344    lel.push_back(2.28466);
345    lel.push_back(2.7661);
346    lel.push_back(1.602);
347    lel.push_back(3.37148);
348    lel.push_back(2.08852);
349    lel.push_back(3.56228);
350    lel.push_back(4.921);
351    lel.push_back(1.77569);
352    lel.push_back(1.77197);
353    lel.push_back(1.699);
354    lel.push_back(3.79293);
355    lel.push_back(3.56856);
356    lel.push_back(2.514);
357    lel.push_back(2.74617);
358    lel.push_back(2.47297);
359    lel.push_back(2.18518);
360    lel.push_back(2.53851);
361    lel.push_back(2.12454);
362    lel.push_back(3.56394);
363    lel.push_back(2.06027);
364    lel.push_back(3.875);
365    lel.push_back(4.778);
366    lel.push_back(2.347);
367    lel.push_back(4.176);
368    lel.push_back(3.56162);
369    lel.push_back(2.79989);
370    lel.push_back(2.076);
371    lel.push_back(4.477);
372    lel.push_back(3.66323);
373    lel.push_back(1.76815);
374    lel.push_back(2.738);
375    lel.push_back(3.56149);
376    lel.push_back(2.28217);
377    lel.push_back(2.33151);
378    lel.push_back(3.523);
379    lel.push_back(2.473);
380    lel.push_back(3.903);
381    lel.push_back(3.59335);
382    lel.push_back(3.477);
383    lel.push_back(2.98184);
384    lel.push_back(3.523);
```

```
385    lel.push_back(2.783);
386    lel.push_back(3.56158);
387    lel.push_back(3.56442);
388    lel.push_back(2.222);
389    lel.push_back(3.046);
390    lel.push_back(3.7573);
391    lel.push_back(3.56658);
392    lel.push_back(3.57016);
393    lel.push_back(2.125);
394    lel.push_back(3.6372);
395    lel.push_back(1.85538);
396    lel.push_back(1.91049);
397    lel.push_back(2.69226);
398    lel.push_back(3.68612);
399    lel.push_back(3.125);
400    lel.push_back(2.95431);
401    lel.push_back(2.29787);
402    lel.push_back(3.58548);
403    lel.push_back(3.56826);
404    lel.push_back(3.959);
405    lel.push_back(3.60666);
406    lel.push_back(2.176);
407    lel.push_back(3.22249);
408    lel.push_back(3.58841);
409    lel.push_back(2.44539);
410    lel.push_back(3.00989);
411    lel.push_back(2.08076);
412    lel.push_back(3.602);
413    lel.push_back(3.40395);
414    lel.push_back(2.176);
415    lel.push_back(3.56846);
416    lel.push_back(2.03994);
417    lel.push_back(3.56155);
418    lel.push_back(2.34912);
419    lel.push_back(3.56921);
420    lel.push_back(3.56278);
421    lel.push_back(2.98305);
422    lel.push_back(3.699);
423    lel.push_back(3.45858);
424    lel.push_back(3.59873);
425    lel.push_back(2.32169);
426    lel.push_back(2.05318);
427    lel.push_back(3.59449);
428    lel.push_back(2.80888);
429    lel.push_back(2.523);
430    lel.push_back(2.70515);
431    lel.push_back(2.12422);
432    lel.push_back(4.921);
```

```
433    lel.push_back(2.176);
434    lel.push_back(2.13184);
435    lel.push_back(3.5777);
436    lel.push_back(2.76658);
437    lel.push_back(3.56381);
438    lel.push_back(3.678);
439    lel.push_back(2.41926);
440    lel.push_back(2.56603);
441    lel.push_back(2.477);
442    lel.push_back(2.24048);
443    lel.push_back(3.72589);
444    lel.push_back(2.31514);
445    lel.push_back(4.44444);
446    lel.push_back(3.59761);
447    lel.push_back(3.61567);
448    lel.push_back(3.58745);
449    lel.push_back(2.409);
450    lel.push_back(3.125);
451    lel.push_back(2.20866);
452    lel.push_back(2.80981);
453    lel.push_back(2.45504);
454    lel.push_back(3.67527);
455    lel.push_back(2.40482);
456    lel.push_back(2.65372);
457    lel.push_back(2.28466);
458    lel.push_back(2.47364);
459    lel.push_back(2.54775);
460    lel.push_back(2.16998);
461    lel.push_back(2.78774);
462    lel.push_back(3.50527);
463    lel.push_back(4.155);
464    lel.push_back(3.57949);
465    lel.push_back(3.56682);
466    lel.push_back(4.602);
467    lel.push_back(2.17031);
468    lel.push_back(3.47809);
469    lel.push_back(2.74435);
470    lel.push_back(2.85521);
471    lel.push_back(2.4109);
472    lel.push_back(2.34912);
473    lel.push_back(3.56876);
474    lel.push_back(3.92153);
475    lel.push_back(3.58686);
476    lel.push_back(2.85475);
477    lel.push_back(2.204);
478    lel.push_back(2.12457);
479    lel.push_back(1.9162);
480    lel.push_back(2.95431);
```

```
481    lel.push_back(2.04239);
482    lel.push_back(2.727);
483    lel.push_back(2.33189);
484    lel.push_back(2.53207);
485    lel.push_back(3.56635);
486    lel.push_back(2.83366);
487    lel.push_back(2.824);
488    lel.push_back(1.89441);
489    lel.push_back(3.272);
490    lel.push_back(2.78822);
491    lel.push_back(3.78067);
492    lel.push_back(1.91572);
493    lel.push_back(2.06696);
494    lel.push_back(2.62351);
495    lel.push_back(3.56158);
496    lel.push_back(3.56445);
497    lel.push_back(3.65455);
498    lel.push_back(3.56186);
499    lel.push_back(2.64329);
500    lel.push_back(2.22466);
501    lel.push_back(3.56875);
502    lel.push_back(2.2245);
503    lel.push_back(2.45466);
504    lel.push_back(3.5787);
505    lel.push_back(3.56153);
506    lel.push_back(3.284);
507    lel.push_back(3.57565);
508    lel.push_back(3.78906);
509    lel.push_back(2.38407);
510    lel.push_back(2.12454);
511    lel.push_back(4.699);
512    lel.push_back(3.68105);
513    lel.push_back(3.42791);
514    lel.push_back(3.56048);
515    lel.push_back(3.5988);
516    lel.push_back(3.201);
517    lel.push_back(3.255);
518    lel.push_back(3.56147);
519    lel.push_back(3.56135);
520    lel.push_back(2.60472);
521    lel.push_back(3.17351);
522    lel.push_back(3.56777);
523    lel.push_back(3.56929);
524    lel.push_back(3.56166);
525    lel.push_back(3.6582);
526    lel.push_back(2.38708);
527    lel.push_back(3.1783);
528    lel.push_back(3.3731);
```

```
529    lel.push_back(4.255);
530    lel.push_back(3.57166);
531    lel.push_back(3.59755);
532    lel.push_back(3.56429);
533    lel.push_back(2.47297);
534    lel.push_back(3.63905);
535    lel.push_back(2.82146);
536    lel.push_back(3.5776);
537    lel.push_back(3.01117);
538    lel.push_back(0.602);
539    lel.push_back(3.5164);
540    lel.push_back(2.66417);
541    lel.push_back(2.38407);
542    lel.push_back(3.34964);
543    lel.push_back(3.56987);
544    lel.push_back(3.59811);
545    lel.push_back(3.56233);
546    lel.push_back(2.54775);
547    lel.push_back(3.55895);
548    lel.push_back(2.17741);
549    lel.push_back(3.60011);
550    lel.push_back(0.924);
551    lel.push_back(4);
552    lel.push_back(2.01981);
553    lel.push_back(3.56206);
554    lel.push_back(3.56625);
555    lel.push_back(2);
556    lel.push_back(3.824);
557    lel.push_back(2.74437);
558    lel.push_back(3.54451);
559    lel.push_back(3.52456);
560    lel.push_back(4.222);
561    lel.push_back(4);
562    lel.push_back(2.2808);
563    lel.push_back(4.149);
564    lel.push_back(2.602);
565    lel.push_back(3.48174);
566    lel.push_back(2.23219);
567    lel.push_back(3.56688);
568    lel.push_back(2.875);
569    lel.push_back(3.58623);
570    lel.push_back(2.52284);
571    lel.push_back(1.89972);
572    lel.push_back(5.959);
573    lel.push_back(3.57224);
574    lel.push_back(3.57489);
575    lel.push_back(3.55528);
576    lel.push_back(3.63129);
```

```
577    lel.push_back(3.632);
578    lel.push_back(2.12454);
579    lel.push_back(4.301);
580    lel.push_back(2.778);
581    lel.push_back(2.38407);
582    lel.push_back(3.60571);
583    lel.push_back(3.89375);
584    lel.push_back(2.778);
585    lel.push_back(3.477);
586    lel.push_back(2.125);
587    lel.push_back(2.477);
588    lel.push_back(2.002);
589    lel.push_back(2.90301);
590    lel.push_back(3.68464);
591    lel.push_back(3);
592    lel.push_back(3.0245);
593    lel.push_back(1.88376);
594    lel.push_back(1.81202);
595    lel.push_back(2.00007);
596    lel.push_back(3.84979);
597    lel.push_back(2.06739);
598    lel.push_back(3.56253);
599    lel.push_back(2.25516);
600    lel.push_back(2.979);
601    lel.push_back(2.01981);
602    lel.push_back(3.66);
603    lel.push_back(2.18055);
604    lel.push_back(3.6561);
605    lel.push_back(3.076);
606    lel.push_back(3.5811);
607    lel.push_back(3.824);
608    lel.push_back(3.63904);
609    lel.push_back(2.40178);
610    lel.push_back(2.00046);
611    lel.push_back(3.49399);
612    lel.push_back(3.87496);
613    lel.push_back(3.301);
614    lel.push_back(1.301);
615    lel.push_back(2.08154);
616    lel.push_back(3.75111);
617    lel.push_back(2.08091);
618    lel.push_back(2.21645);
619    lel.push_back(1.80779);
620    lel.push_back(3.699);
621    lel.push_back(4);
622    lel.push_back(3.56828);
623    lel.push_back(3);
624    lel.push_back(3.60098);
```

```
625    lel.push_back(2.63488);
626    lel.push_back(2.29858);
627    lel.push_back(3.56684);
628    lel.push_back(2.74617);
629    lel.push_back(2.41817);
630    lel.push_back(0.171);
631    lel.push_back(3.56474);
632    lel.push_back(2.64312);
633    lel.push_back(3.90792);
634    lel.push_back(4.201);
635    lel.push_back(3.56008);
636    lel.push_back(3.699);
637    lel.push_back(3.62097);
638    lel.push_back(2.24844);
639    lel.push_back(2.69798);
640    lel.push_back(2.83226);
641    lel.push_back(3.64784);
642    lel.push_back(3.79829);
643    lel.push_back(3.57788);
644    lel.push_back(4.018);
645    lel.push_back(3.721);
646    lel.push_back(1.8533);
647    lel.push_back(1.955);
648    lel.push_back(2.658);
649    lel.push_back(3.78109);
650    lel.push_back(3.62227);
651    lel.push_back(4.155);
652    lel.push_back(4);
653    lel.push_back(2.921);
654    lel.push_back(2.10979);
655    lel.push_back(2.398);
656    lel.push_back(1.842);
657    lel.push_back(3.00924);
658    lel.push_back(2.23256);
659    lel.push_back(3.8184);
660    lel.push_back(3.58432);
661    lel.push_back(3.58206);
662    lel.push_back(4.301);
663    lel.push_back(3.48114);
664    lel.push_back(3.40333);
665    lel.push_back(2.54775);
666    lel.push_back(4.155);
667    lel.push_back(2.875);
668    lel.push_back(3.72178);
669    lel.push_back(3.56131);
670    lel.push_back(3.234);
671    lel.push_back(3.57226);
672    lel.push_back(2.47336);
```

```
673    lel.push_back(3.032);
674    lel.push_back(4.347);
675    lel.push_back(3.59163);
676    lel.push_back(4);
677    lel.push_back(1.98725);
678    lel.push_back(3.6033);
679    lel.push_back(3.37151);
680    lel.push_back(3.268);
681    lel.push_back(2.28132);
682    lel.push_back(2.68355);
683    lel.push_back(3.56777);
684    lel.push_back(2.54775);
685    lel.push_back(3.6466);
686    lel.push_back(2.08061);
687    lel.push_back(3.851);
688    lel.push_back(2.66417);
689    lel.push_back(3.56411);
690    lel.push_back(3.56267);
691    lel.push_back(2.12454);
692    lel.push_back(3.65554);
693    lel.push_back(3.59706);
694    lel.push_back(2.08122);
695    lel.push_back(1.77568);
696    lel.push_back(3.56147);
697    lel.push_back(2.26451);
698    lel.push_back(4.824);
699    lel.push_back(2.409);
700    lel.push_back(2.176);
701    lel.push_back(2.16998);
702    lel.push_back(2.29822);
703    lel.push_back(2.2168);
704    lel.push_back(2.45314);
705    lel.push_back(3.56233);
706    lel.push_back(3.61848);
707    lel.push_back(2.602);
708    lel.push_back(4.046);
709    lel.push_back(3.56543);
710    lel.push_back(2.523);
711    lel.push_back(2.68484);
712    lel.push_back(0.824);
713    lel.push_back(5);
714    lel.push_back(2.357);
715    lel.push_back(3.699);
716    lel.push_back(2.65332);
717    lel.push_back(3.5988);
718    lel.push_back(3.66084);
719    lel.push_back(1.9629);
720    lel.push_back(2.43689);
```

```
721    lel.push_back(4.176);
722    lel.push_back(3.58628);
723    lel.push_back(2.03949);
724    lel.push_back(1.99351);
725    lel.push_back(2.01335);
726    lel.push_back(2.23288);
727    lel.push_back(2.81028);
728    lel.push_back(2.46362);
729    lel.push_back(3.61048);
730    lel.push_back(3.347);
731    lel.push_back(1.91073);
732    lel.push_back(2.10236);
733    lel.push_back(3.778);
734    lel.push_back(2.574);
735    lel.push_back(3.59385);
736    lel.push_back(3.561);
737    lel.push_back(3.93096);
738    lel.push_back(2.98065);
739    lel.push_back(3.5003);
740    lel.push_back(3.56263);
741    lel.push_back(2.78867);
742    lel.push_back(3.60752);
743    lel.push_back(1.89441);
744    lel.push_back(3.51175);
745    lel.push_back(3.71478);
746    lel.push_back(3.56216);
747    lel.push_back(3.56158);
748    lel.push_back(2.74305);
749    lel.push_back(3.56298);
750    lel.push_back(2.81);
751    lel.push_back(2.68484);
752    lel.push_back(3.66063);
753    lel.push_back(3.58979);
754    lel.push_back(3.62097);
755    lel.push_back(3.56777);
756    lel.push_back(2.62374);
757    lel.push_back(2.41319);
758    lel.push_back(3.079);
759    lel.push_back(3.70698);
760    lel.push_back(3.07542);
761    lel.push_back(3.127);
762    lel.push_back(4.301);
763    lel.push_back(3.56378);
764    lel.push_back(3.64845);
765    lel.push_back(2.03737);
766    lel.push_back(2.00031);
767    lel.push_back(3.54123);
768    lel.push_back(2.8999);
```

```
769    lel.push_back(2.32278);
770    lel.push_back(2.33224);
771    lel.push_back(2.35725);
772    lel.push_back(3.653);
773    lel.push_back(2.301);
774    lel.push_back(3.176);
775    lel.push_back(2.29323);
776    lel.push_back(3.58413);
777    lel.push_back(3.58047);
778    lel.push_back(2.125);
779    lel.push_back(2.02631);
780    lel.push_back(1.80361);
781    lel.push_back(2.85521);
782    lel.push_back(3.56159);
783    lel.push_back(2.903);
784    lel.push_back(3.569);
785    lel.push_back(2.2168);
786    lel.push_back(2.10979);
787    lel.push_back(2.778);
788    lel.push_back(1.98752);
789    lel.push_back(4.363);
790    lel.push_back(3.337);
791    lel.push_back(3.481);
792    lel.push_back(2.74527);
793    lel.push_back(3.4221);
794    lel.push_back(2.201);
795    lel.push_back(2.15399);
796    lel.push_back(3.56683);
797    lel.push_back(3.64967);
798    lel.push_back(2.778);
799    lel.push_back(3.602);
800    lel.push_back(3.61546);
801    lel.push_back(2.8107);
802    lel.push_back(2.91319);
803    lel.push_back(2.12454);
804    lel.push_back(2.23222);
805    lel.push_back(2.34067);
806    lel.push_back(2.28181);
807    lel.push_back(3.58198);
808    lel.push_back(2.54816);
809    lel.push_back(2.02587);
810    lel.push_back(3.65686);
811    lel.push_back(2.78867);
812    lel.push_back(3.56135);
813    lel.push_back(3.64634);
814    lel.push_back(2.17591);
815    lel.push_back(3.73363);
816    lel.push_back(3.56274);
```

```
817    lel.push_back(3.72298);
818    lel.push_back(5.255);
819    lel.push_back(3.58736);
820    lel.push_back(2.54694);
821    lel.push_back(3.56262);
822    lel.push_back(0.824);
823    lel.push_back(3.63347);
824    lel.push_back(1.824);
825    lel.push_back(3.58426);
826    lel.push_back(2.15434);
827    lel.push_back(2.15464);
828    lel.push_back(2.70445);
829    lel.push_back(3.57729);
830    lel.push_back(1.98779);
831    lel.push_back(3.5787);
832    lel.push_back(3.909);
833    lel.push_back(2.51045);
834    lel.push_back(3.697);
835    lel.push_back(4.778);
836    lel.push_back(3.61095);
837    lel.push_back(3.52481);
838    lel.push_back(2.43153);
839    lel.push_back(2.31514);
840    lel.push_back(2.959);
841    lel.push_back(3.70384);
842    lel.push_back(4.00609);
843    lel.push_back(3.57338);
844    lel.push_back(3.58444);
845    lel.push_back(1.77553);
846    lel.push_back(1.77945);
847    lel.push_back(5.097);
848    lel.push_back(3.58648);
849    lel.push_back(3.49423);
850    lel.push_back(5.824);
851    lel.push_back(3.778);
852    lel.push_back(3.51903);
853    lel.push_back(4.523);
854    lel.push_back(2.92703);
855    lel.push_back(2.60475);
856    lel.push_back(3.57208);
857    lel.push_back(3.61564);
858    lel.push_back(2);
859    lel.push_back(2.38407);
860    lel.push_back(2.00031);
861    lel.push_back(2.20089);
862    lel.push_back(3.56073);
863    lel.push_back(2.1766);
864    lel.push_back(3.64787);
```

```
865    lel.push_back(4.48132);
866    lel.push_back(3.68869);
867    lel.push_back(2.799);
868    lel.push_back(2.18518);
869    lel.push_back(3.538);
870    lel.push_back(2.46556);
871    lel.push_back(3);
872    lel.push_back(3.574);
873    lel.push_back(3.61781);
874    lel.push_back(1.91073);
875    lel.push_back(2.40178);
876    lel.push_back(3.53901);
877    lel.push_back(6);
878    lel.push_back(2.824);
879    lel.push_back(3.60217);
880    lel.push_back(5);
881    lel.push_back(2.66399);
882    lel.push_back(3.222);
883    lel.push_back(2.398);
884    lel.push_back(2.02631);
885    lel.push_back(3.71101);
886    lel.push_back(1.903);
887    lel.push_back(3.57569);
888    lel.push_back(4.66);
889    lel.push_back(2.24013);
890    lel.push_back(2.80977);
891    lel.push_back(2.11731);
892    lel.push_back(3.173);
893    lel.push_back(3.56921);
894    lel.push_back(3.79296);
895    lel.push_back(3.176);
896    lel.push_back(3.56344);
897    lel.push_back(3.51491);
898    lel.push_back(2.53848);
899    lel.push_back(3);
900    lel.push_back(3.56147);
901    lel.push_back(2.602);
902    lel.push_back(3.56158);
903    lel.push_back(2.824);
904    lel.push_back(3.56274);
905    lel.push_back(3.52447);
906    lel.push_back(2.9813);
907    lel.push_back(2.76655);
908    lel.push_back(3.721);
909    lel.push_back(2.51364);
910    lel.push_back(1.89441);
911    lel.push_back(1.83008);
912    lel.push_back(2.23288);
```

```
913    lel.push_back(2);
914    lel.push_back(2.24844);
915    lel.push_back(2.31441);
916    lel.push_back(2.7457);
917    lel.push_back(3.53687);
918    lel.push_back(3.56297);
919    lel.push_back(2.35725);
920    lel.push_back(2.47297);
921    lel.push_back(2.34029);
922    lel.push_back(4.176);
923    lel.push_back(3.45022);
924    lel.push_back(2.90354);
925    lel.push_back(3.004);
926    lel.push_back(2.66417);
927    lel.push_back(3.84653);
928    lel.push_back(2.38371);
929    lel.push_back(3.5905);
930    lel.push_back(2.60515);
931    lel.push_back(2.648);
932    lel.push_back(3.778);
933    lel.push_back(3.56252);
934    lel.push_back(2.60545);
935    lel.push_back(2.77);
936    lel.push_back(2.78867);
937    lel.push_back(2.72517);
938    lel.push_back(2.875);
939    lel.push_back(3.59712);
940    lel.push_back(2.699);
941    lel.push_back(3.56201);
942    lel.push_back(3.943);
943    lel.push_back(1.79949);
944    lel.push_back(3.67595);
945    lel.push_back(2.85517);
946    lel.push_back(2.02633);
947    lel.push_back(3.84346);
948    lel.push_back(2.38371);
949    lel.push_back(2.09555);
950    lel.push_back(3.357);
951    lel.push_back(3.56704);
952    lel.push_back(2.21645);
953    lel.push_back(2.22466);
954    lel.push_back(3.56254);
955    lel.push_back(3.67722);
956    lel.push_back(2.875);
957    lel.push_back(2.36614);
958    lel.push_back(3.57573);
959    lel.push_back(2.699);
960    lel.push_back(1.96905);
```

```
 961    lel.push_back(2.62);
 962    lel.push_back(3.55891);
 963    lel.push_back(2.875);
 964    lel.push_back(3.40333);
 965    lel.push_back(3.58151);
 966    lel.push_back(2.26486);
 967    lel.push_back(3.22637);
 968    lel.push_back(3.57819);
 969    lel.push_back(2.99721);
 970    lel.push_back(3.57007);
 971    lel.push_back(1.90507);
 972    lel.push_back(2.86791);
 973    lel.push_back(4.079);
 974    lel.push_back(3.5164);
 975    lel.push_back(2.40178);
 976    lel.push_back(1.98752);
 977    lel.push_back(3.58289);
 978    lel.push_back(1.951);
 979    lel.push_back(3.60566);
 980    lel.push_back(4.011);
 981    lel.push_back(3.54711);
 982    lel.push_back(3.551);
 983    lel.push_back(3.456);
 984    lel.push_back(3.64656);
 985    lel.push_back(2.90354);
 986    lel.push_back(2.398);
 987    lel.push_back(1.83475);
 988    lel.push_back(2.36577);
 989    lel.push_back(3.54441);
 990    lel.push_back(3.71766);
 991    lel.push_back(3.927);
 992    lel.push_back(2.90299);
 993    lel.push_back(2.06739);
 994    lel.push_back(3.5843);
 995    lel.push_back(3.55528);
 996    lel.push_back(3);
 997    lel.push_back(1.85317);
 998    lel.push_back(3.54936);
 999    lel.push_back(2.34067);
1000    lel.push_back(3.204);
1001    lel.push_back(3.57634);
1002    lel.push_back(3.56637);
1003    lel.push_back(4.745);
1004    lel.push_back(2.933);
1005    lel.push_back(2.699);
1006    lel.push_back(3.53701);
1007    lel.push_back(4.194);
1008    lel.push_back(2.079);
```

```
1009    lel.push_back(3.56986);
1010    lel.push_back(3.37229);
1011    lel.push_back(2.48239);
1012    lel.push_back(1.84395);
1013    lel.push_back(2.33186);
1014    lel.push_back(2.52922);
1015    lel.push_back(3.58075);
1016    lel.push_back(1.75041);
1017    lel.push_back(3.5686);
1018    lel.push_back(1.98779);
1019    lel.push_back(3.57066);
1020    lel.push_back(3.778);
1021    lel.push_back(2.778);
1022    lel.push_back(3.58282);
1023    lel.push_back(4.046);
1024    lel.push_back(3.54462);
1025    lel.push_back(2.85693);
1026    lel.push_back(2.87773);
1027    lel.push_back(3.58683);
1028    lel.push_back(3.11903);
1029    lel.push_back(2.18551);
1030    lel.push_back(2.87823);
1031    lel.push_back(1.9629);
1032    lel.push_back(3.62661);
1033    lel.push_back(3.62469);
1034    lel.push_back(2.482);
1035    lel.push_back(5.155);
1036    lel.push_back(2.26522);
1037    lel.push_back(3.699);
1038    lel.push_back(2.301);
1039    lel.push_back(3.95339);
1040    lel.push_back(3.61048);
1041    lel.push_back(2.873);
1042    lel.push_back(3.40509);
1043    lel.push_back(3.778);
1044    lel.push_back(2.47297);
1045    lel.push_back(4);
1046    lel.push_back(3.56156);
1047    lel.push_back(3.56217);
1048    lel.push_back(1.90507);
1049    lel.push_back(2.08122);
1050    lel.push_back(3.62887);
1051    lel.push_back(3.56155);
1052    lel.push_back(2.78867);
1053    lel.push_back(3.67372);
1054    lel.push_back(5);
1055    lel.push_back(3.04232);
1056    lel.push_back(0.699);
```

```
1057    lel.push_back(2.06709);
1058    lel.push_back(2.40178);
1059    lel.push_back(3.523);
1060    lel.push_back(3.56477);
1061    lel.push_back(3.56122);
1062    lel.push_back(2.12454);
1063    lel.push_back(4.176);
1064    lel.push_back(3.57287);
1065    lel.push_back(3.54937);
1066    lel.push_back(2.398);
1067    lel.push_back(2.67493);
1068    lel.push_back(3.903);
1069    lel.push_back(3.57637);
1070    lel.push_back(3.35068);
1071    lel.push_back(2.31478);
1072    lel.push_back(3.37238);
1073    lel.push_back(1.92746);
1074    lel.push_back(2.51883);
1075    lel.push_back(2.40178);
1076    lel.push_back(2.8);
1077    lel.push_back(3.56324);
1078    lel.push_back(3.60565);
1079    lel.push_back(3.6595);
1080    lel.push_back(3.49501);
1081    lel.push_back(3.61963);
1082    lel.push_back(2.477);
1083    lel.push_back(3.66806);
1084    lel.push_back(3.57944);
1085    lel.push_back(2.66553);
1086    lel.push_back(3.60227);
1087    lel.push_back(4.05776);
1088    lel.push_back(2.759);
1089    lel.push_back(3.875);
1090    lel.push_back(3.49315);
1091    lel.push_back(1.9162);
1092    lel.push_back(2.778);
1093    lel.push_back(1.824);
1094    lel.push_back(2.22466);
1095    lel.push_back(3.61323);
1096    lel.push_back(3.71766);
1097    lel.push_back(2.49064);
1098    lel.push_back(2.66379);
1099    lel.push_back(3.67207);
1100    lel.push_back(2.155);
1101    lel.push_back(2.6646);
1102    lel.push_back(2.398);
1103    lel.push_back(3.76076);
1104    lel.push_back(2.66417);
```

```
1105    lel.push_back(2.301);
1106    lel.push_back(3.301);
1107    lel.push_back(1.77181);
1108    lel.push_back(2.81058);
1109    lel.push_back(3.57012);
1110    lel.push_back(3.77252);
1111    lel.push_back(3.61247);
1112    lel.push_back(3.1108);
1113    lel.push_back(2.40178);
1114    lel.push_back(3.621);
1115    lel.push_back(3.56145);
1116    lel.push_back(3.56516);
1117    lel.push_back(2.602);
1118    lel.push_back(2.80904);
1119    lel.push_back(2.38351);
1120    lel.push_back(2.699);
1121    lel.push_back(2.99629);
1122    lel.push_back(3.079);
1123    lel.push_back(2.18551);
1124    lel.push_back(3.58128);
1125    lel.push_back(2.648);
1126    lel.push_back(2.1852);
1127    lel.push_back(2.02631);
1128    lel.push_back(5.176);
1129    lel.push_back(3.59974);
1130    lel.push_back(1.8344);
1131    lel.push_back(3.56148);
1132    lel.push_back(5.495);
1133    lel.push_back(2.90354);
1134    lel.push_back(3.477);
1135    lel.push_back(1.98781);
1136    lel.push_back(3.90394);
1137    lel.push_back(3.125);
1138    lel.push_back(3.62758);
1139    lel.push_back(3.56477);
1140    lel.push_back(3.56254);
1141    lel.push_back(3.824);
1142    lel.push_back(2.12422);
1143    lel.push_back(2.64329);
1144    lel.push_back(3.57205);
1145    lel.push_back(3.61306);
1146    lel.push_back(3.56187);
1147    lel.push_back(3.60224);
1148    lel.push_back(2.70231);
1149    lel.push_back(2.09);
1150    lel.push_back(2.903);
1151    lel.push_back(3.00924);
1152    lel.push_back(2.31514);
```

```
1153    lel.push_back(2.28181);
1154    lel.push_back(2.76618);
1155    lel.push_back(2.86666);
1156    lel.push_back(3.56477);
1157    lel.push_back(2.49104);
1158    lel.push_back(2.125);
1159    lel.push_back(1.81);
1160    lel.push_back(3.56479);
1161    lel.push_back(3.53265);
1162    lel.push_back(3.49348);
1163    lel.push_back(3.56275);
1164    lel.push_back(3.56298);
1165    lel.push_back(2.222);
1166    lel.push_back(2.778);
1167    lel.push_back(1.488);
1168    lel.push_back(3);
1169    lel.push_back(2.43786);
1170    lel.push_back(2.12738);
1171    lel.push_back(3.617);
1172    lel.push_back(3.32039);
1173    lel.push_back(3.523);
1174    lel.push_back(3.09656);
1175    lel.push_back(3.56147);
1176    lel.push_back(3.68916);
1177    lel.push_back(3.56226);
1178    lel.push_back(2.824);
1179    lel.push_back(3.62421);
1180    lel.push_back(2.79943);
1181    lel.push_back(3.68864);
1182    lel.push_back(2.903);
1183    lel.push_back(4);
1184    lel.push_back(2.50888);
1185    lel.push_back(3.58887);
1186    lel.push_back(3.56149);
1187    lel.push_back(3.576);
1188    lel.push_back(1.89443);
1189    lel.push_back(4.301);
1190    lel.push_back(1.89418);
1191    lel.push_back(2.56584);
1192    lel.push_back(2.34067);
1193    lel.push_back(2.47297);
1194    lel.push_back(3.62221);
1195    lel.push_back(2.7661);
1196    lel.push_back(3.61243);
1197    lel.push_back(3.301);
1198    lel.push_back(1.83899);
1199    lel.push_back(3.58535);
1200    lel.push_back(4.222);
```

```
1201    lel.push_back(3.40568);
1202    lel.push_back(3.64561);
1203    lel.push_back(2.84491);
1204    lel.push_back(3.57013);
1205    lel.push_back(2.68377);
1206    lel.push_back(3.56275);
1207    lel.push_back(2.70033);
1208    lel.push_back(3.56847);
1209    lel.push_back(3.824);
1210    lel.push_back(2.254);
1211    lel.push_back(3.46556);
1212    lel.push_back(1.91073);
1213    lel.push_back(2.11011);
1214    lel.push_back(2.11731);
1215    lel.push_back(3.75708);
1216    lel.push_back(2.45003);
1217    lel.push_back(2.23288);
1218    lel.push_back(3.67368);
1219    lel.push_back(2.00684);
1220    lel.push_back(3.783);
1221    lel.push_back(1.447);
1222    lel.push_back(1.97523);
1223    lel.push_back(2.41964);
1224    lel.push_back(3);
1225    lel.push_back(2.20055);
1226    lel.push_back(2.85869);
1227    lel.push_back(3.778);
1228    lel.push_back(2.778);
1229    lel.push_back(2.23288);
1230    lel.push_back(3.59546);
1231    lel.push_back(2.68441);
1232    lel.push_back(2.28113);
1233    lel.push_back(2.11011);
1234    lel.push_back(2.23288);
1235    lel.push_back(2.2009);
1236    lel.push_back(1.88357);
1237    lel.push_back(2.29822);
1238    lel.push_back(4.824);
1239    lel.push_back(2.92703);
1240    lel.push_back(2.079);
1241    lel.push_back(3);
1242    lel.push_back(2.7465);
1243    lel.push_back(2.54775);
1244    lel.push_back(3.44895);
1245    lel.push_back(3.721);
1246    lel.push_back(2.47297);
1247    lel.push_back(3.5635);
1248    lel.push_back(3.40915);
```

```
1249    lel.push_back(2.176);
1250    lel.push_back(3.854);
1251    lel.push_back(3.204);
1252    lel.push_back(2.40106);
1253    lel.push_back(2.176);
1254    lel.push_back(3.56149);
1255    lel.push_back(3.60325);
1256    lel.push_back(3.921);
1257    lel.push_back(3.56679);
1258    lel.push_back(2.23219);
1259    lel.push_back(4.903);
1260    lel.push_back(2.699);
1261    lel.push_back(2.05908);
1262    lel.push_back(2.64375);
1263    lel.push_back(3.5687);
1264    lel.push_back(2.23288);
1265    lel.push_back(3.61198);
1266    lel.push_back(4.778);
1267    lel.push_back(3.56931);
1268    lel.push_back(1.95657);
1269    lel.push_back(3.56149);
1270    lel.push_back(3.4814);
1271    lel.push_back(2.01306);
1272    lel.push_back(3.60681);
1273    lel.push_back(3.079);
1274    lel.push_back(3.954);
1275    lel.push_back(3.49476);
1276    lel.push_back(3.56348);
1277    lel.push_back(2.10978);
1278    lel.push_back(2.4564);
1279    lel.push_back(1.92733);
1280    lel.push_back(3.46732);
1281    lel.push_back(3.341);
1282    lel.push_back(3.022);
1283    lel.push_back(2.704);
1284    lel.push_back(2.29787);
1285    lel.push_back(3.56875);
1286    lel.push_back(3.678);
1287    lel.push_back(5.409);
1288    lel.push_back(3.6595);
1289    lel.push_back(2.10949);
1290    lel.push_back(2.60475);
1291    lel.push_back(1.77551);
1292    lel.push_back(2.778);
1293    lel.push_back(6);
1294    lel.push_back(2.125);
1295    lel.push_back(3.58835);
1296    lel.push_back(2.80854);
```

```
1297    lel.push_back(3.56638);
1298    lel.push_back(2.62477);
1299    lel.push_back(2.17725);
1300    lel.push_back(2.301);
1301    lel.push_back(1.68579);
1302    lel.push_back(3.55321);
1303    lel.push_back(5.477);
1304    lel.push_back(2.24812);
1305    lel.push_back(3.58202);
1306    lel.push_back(3.602);
1307    lel.push_back(3.8882);
1308    lel.push_back(2.60515);
1309    lel.push_back(3.91851);
1310    lel.push_back(2.86691);
1311    lel.push_back(2.602);
1312    lel.push_back(2.1456);
1313    lel.push_back(3.60894);
1314    lel.push_back(2.45314);
1315    lel.push_back(2.27652);
1316    lel.push_back(2.76611);
1317    lel.push_back(4.301);
1318    lel.push_back(3.73603);
1319    lel.push_back(2.08061);
1320    lel.push_back(3.71876);
1321    lel.push_back(3.64058);
1322    lel.push_back(3.57288);
1323    lel.push_back(2.01981);
1324    lel.push_back(2.72359);
1325    lel.push_back(4.32909);
1326    lel.push_back(1.98769);
1327    lel.push_back(3.477);
1328    lel.push_back(4.824);
1329    lel.push_back(2.49974);
1330    lel.push_back(3.58975);
1331    lel.push_back(3.56636);
1332    lel.push_back(2.83318);
1333    lel.push_back(2.16965);
1334    lel.push_back(2.77);
1335    lel.push_back(2.75658);
1336    lel.push_back(2.979);
1337    lel.push_back(3.56284);
1338    lel.push_back(1.81219);
1339    lel.push_back(3.57014);
1340    lel.push_back(3.84757);
1341    lel.push_back(2.20022);
1342    lel.push_back(2.392);
1343    lel.push_back(3.63489);
1344    lel.push_back(3.854);
```

```
1345    lel.push_back(2.90248);
1346    lel.push_back(4.176);
1347    lel.push_back(1.477);
1348    lel.push_back(2.34912);
1349    lel.push_back(3.56819);
1350    lel.push_back(3.699);
1351    lel.push_back(2.68441);
1352    lel.push_back(2.33187);
1353    lel.push_back(2.225);
1354    lel.push_back(4.602);
1355    lel.push_back(2.33187);
1356    lel.push_back(3.61963);
1357    lel.push_back(2.45544);
1358    lel.push_back(2.24809);
1359    lel.push_back(4);
1360    lel.push_back(2.85569);
1361    lel.push_back(2.509);
1362    lel.push_back(3);
1363    lel.push_back(2.52922);
1364    lel.push_back(3.38);
1365    lel.push_back(3.56149);
1366    lel.push_back(2.778);
1367    lel.push_back(3.56826);
1368    lel.push_back(2.11699);
1369    lel.push_back(2.46206);
1370    lel.push_back(3.21904);
1371    lel.push_back(2.34634);
1372    lel.push_back(4.204);
1373    lel.push_back(3.61841);
1374    lel.push_back(2.34067);
1375    lel.push_back(2.11011);
1376    lel.push_back(2.00031);
1377    lel.push_back(1.91596);
1378    lel.push_back(2.24016);
1379    lel.push_back(3.58115);
1380    lel.push_back(3.55168);
1381    lel.push_back(3.58837);
1382    lel.push_back(3.07326);
1383    lel.push_back(2.66379);
1384    lel.push_back(2.18518);
1385    lel.push_back(3.235);
1386    lel.push_back(3.57223);
1387    lel.push_back(3.1783);
1388    lel.push_back(2.06739);
1389    lel.push_back(3.56192);
1390    lel.push_back(3.55665);
1391    lel.push_back(3.176);
1392    lel.push_back(3.78169);
```

```
1393    lel.push_back(2.16998);
1394    lel.push_back(2.824);
1395    lel.push_back(3.477);
1396    lel.push_back(3.61369);
1397    lel.push_back(2.41929);
1398    lel.push_back(3.54446);
1399    lel.push_back(2.721);
1400    lel.push_back(4.25);
1401    lel.push_back(2.34912);
1402    lel.push_back(3.658);
1403    lel.push_back(2.71958);
1404    lel.push_back(3.37069);
1405    lel.push_back(3.57791);
1406    lel.push_back(2.36577);
1407    lel.push_back(3.537);
1408    lel.push_back(3.58937);
1409    lel.push_back(1.95567);
1410    lel.push_back(3.62916);
1411    lel.push_back(1.86535);
1412    lel.push_back(3.59102);
1413    lel.push_back(2.29858);
1414    lel.push_back(2.29731);
1415    lel.push_back(3.71501);
1416    lel.push_back(2.52882);
1417    lel.push_back(3.56225);
1418    lel.push_back(2.681);
1419    lel.push_back(3.176);
1420    lel.push_back(1.82989);
1421    lel.push_back(2.145);
1422    lel.push_back(2.8952);
1423    lel.push_back(3.78934);
1424    lel.push_back(3.17667);
1425    lel.push_back(1.9162);
1426    lel.push_back(3.54899);
1427    lel.push_back(2.31479);
1428    lel.push_back(3.42791);
1429    lel.push_back(2.658);
1430    lel.push_back(2.80884);
1431    lel.push_back(2.88335);
1432    lel.push_back(2.92648);
1433    lel.push_back(2.17262);
1434    lel.push_back(3.5846);
1435    lel.push_back(2.21576);
1436    lel.push_back(3.341);
1437    lel.push_back(2.481);
1438    lel.push_back(2.80977);
1439    lel.push_back(2.03964);
1440    lel.push_back(2.61432);
```

```
1441    lel.push_back(3.56273);
1442    lel.push_back(3.5736);
1443    lel.push_back(3.56269);
1444    lel.push_back(3.18375);
1445    lel.push_back(3.56536);
1446    lel.push_back(2.83223);
1447    lel.push_back(3.58284);
1448    lel.push_back(2.49974);
1449    lel.push_back(6.176);
1450    lel.push_back(3.00989);
1451    lel.push_back(2.08123);
1452    lel.push_back(2.06727);
1453    lel.push_back(2.38407);
1454    lel.push_back(3.046);
1455    lel.push_back(2);
1456    lel.push_back(2.41926);
1457    lel.push_back(4);
1458    lel.push_back(3.079);
1459    lel.push_back(3.60689);
1460    lel.push_back(3.57026);
1461    lel.push_back(2.63615);
1462    lel.push_back(3.96596);
1463    lel.push_back(2.61207);
1464    lel.push_back(3);
1465    lel.push_back(3.07704);
1466    lel.push_back(3.72571);
1467    lel.push_back(2.58446);
1468    lel.push_back(2.00048);
1469    lel.push_back(4.125);
1470    lel.push_back(3.57291);
1471    lel.push_back(2.13917);
1472    lel.push_back(2.523);
1473    lel.push_back(2.398);
1474    lel.push_back(1.98752);
1475    lel.push_back(2.50928);
1476    lel.push_back(3.7444);
1477    lel.push_back(2.176);
1478    lel.push_back(3.59729);
1479    lel.push_back(2.23254);
1480    lel.push_back(3.58459);
1481    lel.push_back(3.59925);
1482    lel.push_back(3.77443);
1483    lel.push_back(3.477);
1484    lel.push_back(3.57676);
1485    lel.push_back(3.61628);
1486    lel.push_back(2.34875);
1487    lel.push_back(2.71436);
1488    lel.push_back(2.43649);
```

```
1489    lel.push_back(2.24794);
1490    lel.push_back(2.44479);
1491    lel.push_back(4.64588);
1492    lel.push_back(2.43688);
1493    lel.push_back(2.11033);
1494    lel.push_back(2.09493);
1495    lel.push_back(2.921);
1496    lel.push_back(2.937);
1497    lel.push_back(2.35725);
1498    lel.push_back(2.34067);
1499    lel.push_back(1.98108);
1500    lel.push_back(2.86791);
1501    lel.push_back(6.301);
1502    lel.push_back(3.31883);
1503    lel.push_back(2.92544);
1504    lel.push_back(2.40216);
1505    lel.push_back(4.079);
1506    lel.push_back(2.34912);
1507    lel.push_back(3.6689);
1508    lel.push_back(3.57044);
1509    lel.push_back(2.527);
1510    lel.push_back(3.56414);
1511    lel.push_back(3.64892);
1512    lel.push_back(3.58537);
1513    lel.push_back(2.4014);
1514    lel.push_back(2.875);
1515    lel.push_back(3.63042);
1516    lel.push_back(3.56324);
1517    lel.push_back(2.47296);
1518    lel.push_back(3.56232);
1519    lel.push_back(4.125);
1520    lel.push_back(3.4143);
1521    lel.push_back(3.62549);
1522    lel.push_back(3.0212);
1523    lel.push_back(2.02235);
1524    lel.push_back(1.96917);
1525    lel.push_back(3.56201);
1526    lel.push_back(2.1852);
1527    lel.push_back(3.56638);
1528    lel.push_back(2.64841);
1529    lel.push_back(2.08823);
1530    lel.push_back(3.46707);
1531    lel.push_back(2.28146);
1532    lel.push_back(3.301);
1533    lel.push_back(5.244);
1534    lel.push_back(2.38407);
1535    lel.push_back(3.602);
1536    lel.push_back(3.5625);
```

```
1537    lel.push_back(2.23288);
1538    lel.push_back(1.89972);
1539    lel.push_back(3.5162);
1540    lel.push_back(3.34972);
1541    lel.push_back(3.602);
1542    lel.push_back(3.56915);
1543    lel.push_back(3.60834);
1544    lel.push_back(3.5755);
1545    lel.push_back(3.58746);
1546    lel.push_back(3.61009);
1547    lel.push_back(1.602);
1548    lel.push_back(2.504);
1549    lel.push_back(1.824);
1550    lel.push_back(3.53218);
1551    lel.push_back(2.50888);
1552    lel.push_back(3.56135);
1553    lel.push_back(2.13184);
1554    lel.push_back(2.426);
1555    lel.push_back(3.58085);
1556    lel.push_back(3.301);
1557    lel.push_back(3.55654);
1558    lel.push_back(2.28786);
1559    lel.push_back(2.68484);
1560    lel.push_back(2.789);
1561    lel.push_back(3.56247);
1562    lel.push_back(3.55878);
1563    lel.push_back(3.57124);
1564    lel.push_back(2.824);
1565    lel.push_back(2.046);
1566    lel.push_back(1.96917);
1567    lel.push_back(4);
1568    lel.push_back(5.477);
1569    lel.push_back(3.56499);
1570    lel.push_back(1.93325);
1571    lel.push_back(3.609);
1572    lel.push_back(2.04647);
1573    lel.push_back(3.12004);
1574    lel.push_back(3.56813);
1575    lel.push_back(2.86415);
1576    lel.push_back(3.59095);
1577    lel.push_back(3.55891);
1578    lel.push_back(3.60386);
1579    lel.push_back(4);
1580    lel.push_back(2.20884);
1581    lel.push_back(3.449);
1582    lel.push_back(3.56158);
1583    lel.push_back(3.07628);
1584    lel.push_back(1.477);
```

```
1585    lel.push_back(3.63328);
1586    lel.push_back(2.41964);
1587    lel.push_back(3.60927);
1588    lel.push_back(2.4109);
1589    lel.push_back(2.32532);
1590    lel.push_back(2.83131);
1591    lel.push_back(2.26492);
1592    lel.push_back(2.74617);
1593    lel.push_back(3.59287);
1594    lel.push_back(3.56805);
1595    lel.push_back(2.10979);
1596    lel.push_back(3.5635);
1597    lel.push_back(3.56201);
1598    lel.push_back(2.34836);
1599    lel.push_back(2.78867);
1600    lel.push_back(3.57495);
1601    lel.push_back(1.89972);
1602    lel.push_back(2.15415);
1603    lel.push_back(4.273);
1604    lel.push_back(3.71069);
1605    lel.push_back(3.56155);
1606    lel.push_back(2.426);
1607    lel.push_back(1.796);
1608    lel.push_back(2.17723);
1609    lel.push_back(3.778);
1610    lel.push_back(2.24048);
1611    lel.push_back(1.9744);
1612    lel.push_back(3.125);
1613    lel.push_back(3.79746);
1614    lel.push_back(3.53188);
1615    lel.push_back(3.56515);
1616    lel.push_back(2.35762);
1617    lel.push_back(3.745);
1618    lel.push_back(3.248);
1619    lel.push_back(3.10508);
1620    lel.push_back(2.20866);
1621    lel.push_back(2.62397);
1622    lel.push_back(2.40178);
1623    lel.push_back(2.36614);
1624    lel.push_back(3.284);
1625    lel.push_back(2.982);
1626    lel.push_back(4);
1627    lel.push_back(1.82083);
1628    lel.push_back(3.5467);
1629    lel.push_back(2.87951);
1630    lel.push_back(6);
1631    lel.push_back(2.90354);
1632    lel.push_back(3.86472);
```

```
1633    lel.push_back(4.398);
1634    lel.push_back(4.38);
1635    lel.push_back(2.62393);
1636    lel.push_back(1.523);
1637    lel.push_back(2.64329);
1638    lel.push_back(3.56516);
1639    lel.push_back(3.58969);
1640    lel.push_back(3.449);
1641    lel.push_back(4.176);
1642    lel.push_back(3.53877);
1643    lel.push_back(6.301);
1644    lel.push_back(3.56168);
1645    lel.push_back(3.255);
1646    lel.push_back(2.824);
1647    lel.push_back(1.477);
1648    lel.push_back(2.87);
1649    lel.push_back(3.44895);
1650    lel.push_back(3.7402);
1651    lel.push_back(2.17558);
1652    lel.push_back(3.70163);
1653    lel.push_back(3.56148);
1654    lel.push_back(3.57239);
1655    lel.push_back(1.97523);
1656    lel.push_back(3.875);
1657    lel.push_back(2.2009);
1658    lel.push_back(3.56446);
1659    lel.push_back(3.56776);
1660    lel.push_back(3.60212);
1661    lel.push_back(3.255);
1662    lel.push_back(2.854);
1663    lel.push_back(3.75);
1664    lel.push_back(3.56774);
1665    lel.push_back(4.222);
1666    lel.push_back(3.5618);
1667    lel.push_back(2.13871);
1668    lel.push_back(2.73541);
1669    lel.push_back(3.56158);
1670    lel.push_back(4.176);
1671    lel.push_back(2.34912);
1672    lel.push_back(3.477);
1673    lel.push_back(3.56158);
1674    lel.push_back(3.59223);
1675    lel.push_back(2.55629);
1676    lel.push_back(3.64507);
1677    lel.push_back(3.56364);
1678    lel.push_back(2.903);
1679    lel.push_back(5.678);
1680    lel.push_back(5.125);
```

```
1681    lel.push_back(3.7322);
1682    lel.push_back(2.482);
1683    lel.push_back(3.91);
1684    lel.push_back(2.29858);
1685    lel.push_back(3.56153);
1686    lel.push_back(4.222);
1687    lel.push_back(3.56444);
1688    lel.push_back(3.54927);
1689    lel.push_back(1.9053);
1690    lel.push_back(2.33187);
1691    lel.push_back(3.55602);
1692    lel.push_back(2.24013);
1693    lel.push_back(2.24844);
1694    lel.push_back(3.64959);
1695    lel.push_back(3.58841);
1696    lel.push_back(3.63112);
1697    lel.push_back(3.59863);
1698    lel.push_back(2.28181);
1699    lel.push_back(2.76702);
1700    lel.push_back(2.31478);
1701    lel.push_back(3.58504);
1702    lel.push_back(3.65605);
1703    lel.push_back(2.23254);
1704    lel.push_back(3.477);
1705    lel.push_back(2.921);
1706    lel.push_back(2.585);
1707    lel.push_back(2.799);
1708    lel.push_back(3.61617);
1709    lel.push_back(5.125);
1710    lel.push_back(3.064);
1711    lel.push_back(2.92703);
1712    lel.push_back(2.02633);
1713    lel.push_back(3.56216);
1714    lel.push_back(4.921);
1715    lel.push_back(2.74572);
1716    lel.push_back(3.54462);
1717    lel.push_back(3.59332);
1718    lel.push_back(3.61755);
1719    lel.push_back(2.09556);
1720    lel.push_back(3.86612);
1721    lel.push_back(3.57436);
1722    lel.push_back(2.41926);
1723    lel.push_back(2.50925);
1724    lel.push_back(2.31441);
1725    lel.push_back(3.61436);
1726    lel.push_back(2.222);
1727    lel.push_back(2.523);
1728    lel.push_back(2.12425);
```

```
1729    lel.push_back(2.97292);
1730    lel.push_back(1.505);
1731    lel.push_back(2.49204);
1732    lel.push_back(4.409);
1733    lel.push_back(1.727);
1734    lel.push_back(1.84364);
1735    lel.push_back(2.51404);
1736    lel.push_back(2.41852);
1737    lel.push_back(3.783);
1738    lel.push_back(3.824);
1739    lel.push_back(2.8013);
1740    lel.push_back(3.56988);
1741    lel.push_back(3.62674);
1742    lel.push_back(3.699);
1743    lel.push_back(2.222);
1744    lel.push_back(3.301);
1745    lel.push_back(1.88376);
1746    lel.push_back(2.11011);
1747    lel.push_back(2.68441);
1748    lel.push_back(3.57106);
1749    lel.push_back(3.56232);
1750    lel.push_back(2.215);
1751    lel.push_back(2.854);
1752    lel.push_back(2.52843);
1753    lel.push_back(2.079);
1754    lel.push_back(1.88895);
1755    lel.push_back(2.971);
1756    lel.push_back(4.222);
1757    lel.push_back(3);
1758    lel.push_back(1.767);
1759    lel.push_back(2.98065);
1760    lel.push_back(3.231);
1761    lel.push_back(2.03949);
1762    lel.push_back(3.143);
1763    lel.push_back(3.523);
1764    lel.push_back(2.12454);
1765    lel.push_back(2.84487);
1766    lel.push_back(3.191);
1767    lel.push_back(2.68549);
1768    lel.push_back(3.55522);
1769    lel.push_back(3.56543);
1770    lel.push_back(2.50968);
1771    lel.push_back(2.15757);
1772    lel.push_back(3.57138);
1773    lel.push_back(3.58553);
1774    lel.push_back(3.58937);
1775    lel.push_back(2.49104);
1776    lel.push_back(3.62679);
```

```
1777    lel.push_back(2.6274);
1778    lel.push_back(3.301);
1779    lel.push_back(2.34912);
1780    lel.push_back(2.45505);
1781    lel.push_back(3.6854);
1782    lel.push_back(2.7896);
1783    lel.push_back(1.84383);
1784    lel.push_back(3.61054);
1785    lel.push_back(3.68312);
1786    lel.push_back(4.398);
1787    lel.push_back(4.477);
1788    lel.push_back(2.476);
1789    lel.push_back(2.24013);
1790    lel.push_back(3.83769);
1791    lel.push_back(2.60515);
1792    lel.push_back(3.57226);
1793    lel.push_back(3.71751);
1794    lel.push_back(3.17441);
1795    lel.push_back(2.34875);
1796    lel.push_back(3.56592);
1797    lel.push_back(2.28966);
1798    lel.push_back(3.62346);
1799    lel.push_back(3.57228);
1800    lel.push_back(5);
1801    lel.push_back(2.98125);
1802    lel.push_back(3.56176);
1803    lel.push_back(3.477);
1804    lel.push_back(4.574);
1805    lel.push_back(2.17262);
1806    lel.push_back(2.699);
1807    lel.push_back(3.84865);
1808    lel.push_back(3.56729);
1809    lel.push_back(4.34861);
1810    lel.push_back(3.6053);
1811    lel.push_back(2.09555);
1812    lel.push_back(2.20057);
1813    lel.push_back(2.02633);
1814    lel.push_back(3.17519);
1815    lel.push_back(2.25612);
1816    lel.push_back(4.161);
1817    lel.push_back(2.54775);
1818    lel.push_back(2.33186);
1819    lel.push_back(2.36652);
1820    lel.push_back(3.56235);
1821    lel.push_back(3.398);
1822    lel.push_back(2.26401);
1823    lel.push_back(2.432);
1824    lel.push_back(3.55654);
```

```
1825    lel.push_back(2.875);
1826    lel.push_back(2.64292);
1827    lel.push_back(3.57949);
1828    lel.push_back(2.28965);
1829    lel.push_back(2.34912);
1830    lel.push_back(3.56323);
1831    lel.push_back(3.51175);
1832    lel.push_back(2.10238);
1833    lel.push_back(3.076);
1834    lel.push_back(3.7763);
1835    lel.push_back(3.56727);
1836    lel.push_back(2.67493);
1837    lel.push_back(1.222);
1838    lel.push_back(3.824);
1839    lel.push_back(2.34799);
1840    lel.push_back(2.16998);
1841    lel.push_back(3.60192);
1842    lel.push_back(3.58256);
1843    lel.push_back(2.34875);
1844    lel.push_back(2.38407);
1845    lel.push_back(2.921);
1846    lel.push_back(2.49064);
1847    lel.push_back(3.80905);
1848    lel.push_back(2.49145);
1849    lel.push_back(3);
1850    lel.push_back(2.23218);
1851    lel.push_back(2.16965);
1852    lel.push_back(2.20836);
1853    lel.push_back(3.85935);
1854    lel.push_back(1.84395);
1855    lel.push_back(2.13152);
1856    lel.push_back(3.38517);
1857    lel.push_back(1.85361);
1858    lel.push_back(3.56512);
1859    lel.push_back(3.56849);
1860    lel.push_back(2.63615);
1861    lel.push_back(2.47258);
1862    lel.push_back(3.5412);
1863    lel.push_back(3.222);
1864    lel.push_back(3.57105);
1865    lel.push_back(3.61373);
1866    lel.push_back(3.48295);
1867    lel.push_back(2.523);
1868    lel.push_back(3.56297);
1869    lel.push_back(3.333);
1870    lel.push_back(3.56298);
1871    lel.push_back(3.57418);
1872    lel.push_back(2.41964);
```

```
1873    lel.push_back(2.215);
1874    lel.push_back(2.12457);
1875    lel.push_back(2.954);
1876    lel.push_back(3.56953);
1877    lel.push_back(2.35725);
1878    lel.push_back(2.41926);
1879    lel.push_back(3.5325);
1880    lel.push_back(3.40333);
1881    lel.push_back(3.778);
1882    lel.push_back(2.02633);
1883    lel.push_back(3.63478);
1884    lel.push_back(2.64458);
1885    lel.push_back(3.60217);
1886    lel.push_back(2.77);
1887    lel.push_back(2.64458);
1888    lel.push_back(3.14609);
1889    lel.push_back(3.57239);
1890    lel.push_back(3.56154);
1891    lel.push_back(2.38407);
1892    lel.push_back(2.21646);
1893    lel.push_back(4);
1894    lel.push_back(3.17796);
1895    lel.push_back(3.46707);
1896    lel.push_back(2.78867);
1897    lel.push_back(2.12454);
1898    return lel;
1899 }
1900
1901 -------------------------------------------------------------------------
1902 total elapsed time: 303.009 sec
1903 =========================================================================
```

Appendix C - LELPredictor Filter/Amplifer Calculations

LELPredictor Neural Network Filter / Amplifer Equations

Parabolic Filter for Molecular Weight

Max/Min/Mean Values

$x_{max} := 1701.2 \qquad x_{min} := 30.026 \qquad x_n := 263.583$

$$x := x_{min}, x_{min} + \frac{x_{max} - x_{min}}{1000} .. x_{max}$$

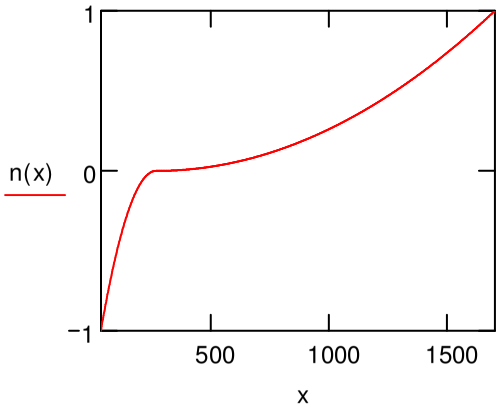
$b := x_n \qquad c1 := 0 \qquad c2 := 0$

$$a1 := \frac{1}{(x_n - x_{min})^2} \qquad a2 := \frac{1}{(x_{max} - b)^2 - (x_n - b)^2}$$

Normalized Parabolic Equations

$$n(x) := \begin{cases} -\left[a1 \cdot (x - b)^2 + c1\right] & \text{if } x < x_n \\ a2 \cdot (x - b)^2 + c2 & \text{otherwise} \end{cases}$$

Normalized Values for Parabolic Filter of Molecular Weight



Parabolic Filter for LEL value

Max/Min/Mean Values

$x_{max} := 6.301$ $x_{min} := -0.301$ $x_n := 3.2614$

$$x := x_{min}, x_{min} + \frac{x_{max} - x_{min}}{1000} .. x_{max}$$

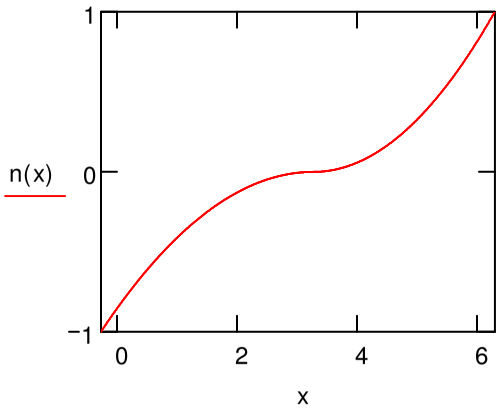
$b := x_n$ $c1 := 0$ $c2 := 0$

$$a1 := \frac{1}{(x_n - x_{min})^2} \qquad a2 := \frac{1}{(x_{max} - b)^2 - (x_n - b)^2}$$

Normalized Parabolic Equations

$$n(x) := \begin{cases} -\left[a1 \cdot (x - b)^2 + c1\right] & \text{if } x < x_n \\ a2 \cdot (x - b)^2 + c2 & \text{otherwise} \end{cases}$$

Normalized Values for Parabolic Filter of the log (-LEL) value



Parabolic Amplifier for LEL value

Restored Values from Normalized Inputs

$$f(y) := \begin{cases} \frac{1}{2 \cdot a2} \cdot \left[2 \cdot a2 \cdot b + 2 \cdot (a2 \cdot y - a2 \cdot c2)^{\frac{1}{2}} \right] & \text{if } y > 0 \\ \frac{1}{2 \cdot a1} \cdot \left[2 \cdot a1 \cdot b - 2 \cdot [(-a1) \cdot y - a1 \cdot c1]^{\frac{1}{2}} \right] & \text{otherwise} \end{cases}$$

$y := -1, -0.999 .. 1$

Restored Values for the Parabolic Amplifier
to calculate the log (-LEL) value
from the NN normalized output (-1 -> 1)