Research_Update

December 12, 2016

0.1 Before we start

This **notebook, presentation, functions and data** can be retrieved from GitHub:

```
git clone https://github.com/Kleurenprinter/prompred.git
```

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

```
<b>CODE SHOW </b>: turned ON
```

```
<b>CODE SHOW </b>: turned OFF
```

```
In [2]: import matplotlib.pyplot as plt
        from prompred import *
        import numpy as np
        import math
        import sklearn
        import warnings
        import pandas as pd
        from IPython.core.display import HTML
        warnings.filterwarnings('ignore')

        %matplotlib inline
```

1 The application of machine learning techniques in promoter engineering of prokaryotic microorganisms

Jim Clauwaert
Research update
December 2016

2 Machine Learning

3 Promoter Engineering

- Predict the strength of a promoter sequence used in E.coli using machine learning techniques.

- Make predictions of the promoter strength in function of the sigma factor used by the RNA polymerase.

4 Data

```
In [3]: from wand.image import Image
        Image(filename='pdf/Mutalik2013.pdf')
```

```
Out[3]:
```

Precise and reliable gene expression via standard transcription and translation initiation elements

Vivek K Mutalik[1–3], Joao C Guimaraes[1,3,4], Guillaume Cambray[1,3], Colin Lam[1,3], Marc Juul Christoffersen[1,3], Quynh-Anh Mai[1,3], Andrew B Tran[1,3], Morgan Paull[1], Jay D Keasling[1–3,5,6], Adam P Arkin[1–3,8] & Drew Endy[1,7,8]

An inability to reliably predict quantitative behaviors for novel combinations of genetic elements limits the rational engineering of biological systems. We developed an expression cassette architecture for genetic elements controlling transcription and translation initiation in *Escherichia coli*: transcription elements encode a common mRNA start, and translation elements use an overlapping genetic motif found in many natural systems. We engineered libraries of constitutive and repressor-regulated promoters along with translation initiation elements following these definitions. We measured activity distributions for each library and selected elements that collectively resulted in expression across a 1,000-fold observed dynamic range. We studied all combinations of curated elements, demonstrating that arbitrary genes are reliably expressed to within twofold relative target expression windows with ~93% reliability. We expect the genetic element definitions validated here can be collectively expanded to create collections of public-domain standard biological parts that support reliable forward engineering of gene expression at genome scales.

One main goal of synthetic biology is to make the engineering of biology easier[1,2]. DNA synthesis and assembly has progressed to the point where entire metabolic pathways, chromosomes and genomes can now be synthesized and transplanted[3–5]. However, our capacity to rationally design increasingly complicated genetic systems as enabled by improvements in DNA construction methods has not kept pace[2,6]. One of the greatest claimed barriers to efficient and scalable genetic design is the lack of standard parts that can be reused reliably in novel combinations[6,7]. Many examples instead highlight, even within well-studied organisms such as *E. coli*, how seemingly simple genetic functions behave differently in different settings[8,9]. For example, a prokaryotic ribosome-binding site (RBS) element that initiates translation for one coding sequence might not function at all with another coding sequence[10]. If the genetic elements that encode control of central cellular processes such as transcription and translation cannot be reliably reused, then there is little chance that higher-order objects encoded from such basic elements will be reliable in larger-scale systems[6,11].

Standard biological parts could, in theory, enable hierarchical abstraction of biological functions[1,2,12,13]. The behavior of integrated genetic systems could then be represented via simpler models of individual elements and ultimately mapped to underlying genetic sequences whose encoded functions are dependent on a limited number of measurable or calculable intrinsic variables. Such abstraction of function seems necessary to manage biological complexity and to allow the engineering of increasingly sophisticated genetic systems[6,12,14].

We engineered ~500 transcription and translation initiation elements that are compatible within a standardized genetic context, or expression operating unit (EOU), that enables predictable forward engineering of gene expression over a wide dynamic range. We characterized representative parts for each type by testing more than 1,200 part-part combinations to establish and validate functional composition rules while quantifying scores for part activity. From this data we also estimated the 'quality' of each part, a second-order statistic that represents the extent to which the activity of a part varies across changes in context[15]. Our results demonstrate how, when combined with standardized transcription control elements, a more physically complex design for the control of translation initiation creates simply modeled parts enabling reliable forward engineering of gene expression.

RESULTS

Prioritizing part composition puzzles

In related work, we systematically assembled and tested all combinations of frequently used prokaryotic transcription and translation control elements to quantify average part activities and also variation in activities as parts are reused in novel combinations[15]. Here we focus on developing rules for a genetic layout architecture underlying gene expression cassettes that eliminate

[1]BIOFAB International Open Facility Advancing Biotechnology, Emeryville, California, USA. [2]Lawrence Berkeley National Laboratory, Physical Biosciences Division, Berkeley, California, USA. [3]Department of Bioengineering, University of California, Berkeley, Berkeley, California, USA. [4]Department of Informatics, Computer Science and Technology Center, University of Minho, Campus de Gualtar, Braga, Portugal. [5]Department of Chemical & Biomolecular Engineering, University of California, Berkeley, Berkeley, California, USA. [6]Joint BioEnergy Institute, Emeryville, California, USA. [7]Department of Bioengineering, Stanford University, Stanford, California, USA. [8]These authors contributed equally to this work. Correspondence should be addressed to D.E. (endy@stanford.edu) or A.P.A. (aparkin@lbl.gov).

RECEIVED 30 AUGUST 2012; ACCEPTED 14 FEBRUARY 2013; PUBLISHED ONLINE 10 MARCH 2013; DOI:10.1038/NMETH.2404

4.0.1 Mutalik Database

254 promoters - 137 *randomized* promoters: randomization within 35- and 10-box - 117 *modulated* promoters: promoters built from a set of fragments
sequence length: $[35, 49]$ - varying spacer lengths - up-element
Insulated promoters

```
In [4]: %%javascript
        IPython.OutputArea.auto_scroll_threshold = 90000;
```

```
<IPython.core.display.Javascript object>
```

5 Feature creation and selection

5.1 nucleotides with respictive positions

```
In [5]: dfDatasetOrder = pd.read_csv("data/mut_rand_mod_lib.csv")
        dfDataset = dfDatasetOrder.reindex(np.random.permutation(dfDatasetOrder.inc
        dfDataset.iloc[:5, 0:2]
```

```
Out[5]:             ID                                          sequence
        63     apFAB95  AAAAAATTTATTTGCTTTCGCATCTTTTTGTACCTATAATGTGTGGAT
        217   apFAB305             TTGACAATTAATCATCCGGCTCGTAGGGTTTGTGGA
        11     apFAB41   AAAAAGAGTATTGACTTCAGGAAAATTTTTCTGTATAATAGATTCAT
        136   apFAB189              TTTTTCCTTAATCATCGGCTCGTATAATGTGTGGA
        236   apFAB326             TTCCACCTTAATCATCCGGCTCGTATAATGTGTGGA
```

5.2 Creation of two reference regions

5.3 Creation of dummy variables for every position

```
In [6]: #sequence range
        seqRange = [-47,1]

        #regions of interest (wrt 35- and 10-box)
        ROI =  [[-12,14],[-8,12]]

        #
        labels, positionBox, spacer = regionSelect(dfDataset, ROI, seqRange)
        print(dfDataset['sequence'][0],"\n",positionBox.values[0])
```

```
AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT
 [1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1
 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0
 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0]
```

6 Data Preprocessing

6.1 Sequence annotation

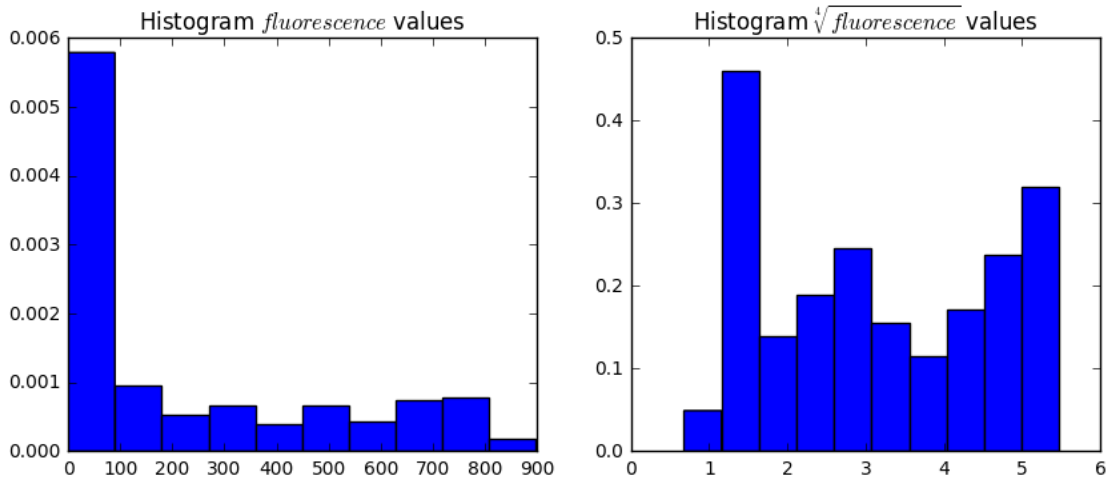
- add reference regions to used sequences

```
In [7]: dfDataset.iloc[:5,[1,2,4,5]].style.set_properties(**{"font-size":"12px"})

Out[7]: <pandas.formats.style.Styler at 0x7f02a6e19cf8>
```

6.2 Labels

- create homogenous distribution to keep estimation confidence interval over full range at a minimum

```
In [8]: yData = dfDataset['mean_score']
        yRooted = [math.sqrt(math.sqrt(u)) for u in yData]
        plt.figure(num=None, figsize=(10, 4), dpi=80, facecolor='w', edgecolor='k')
        plt.subplot(121)
        plt.hist(yData,10,normed=1)
        plt.title('Histogram $fluorescence$ values')
        plt.subplot(122)
        plt.hist(yRooted,10,normed=1)
        plt.title('Histogram $\sqrt[4]{fluorescence}$ values')

Out[8]: <matplotlib.text.Text at 0x7f029db3a4a8>
```



7 Code Implementation

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

```
<b>CODE SHOW </b>: turned ON

<b>CODE SHOW </b>: turned OFF
```

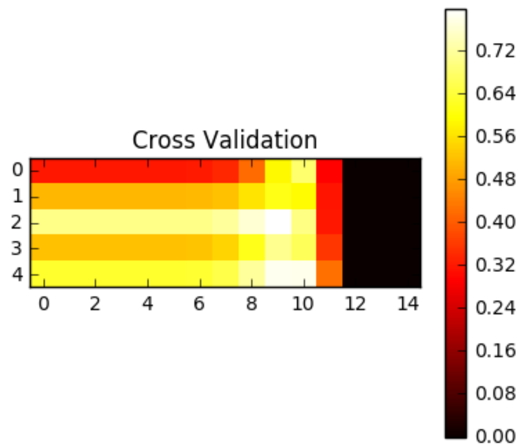
8 Supported models

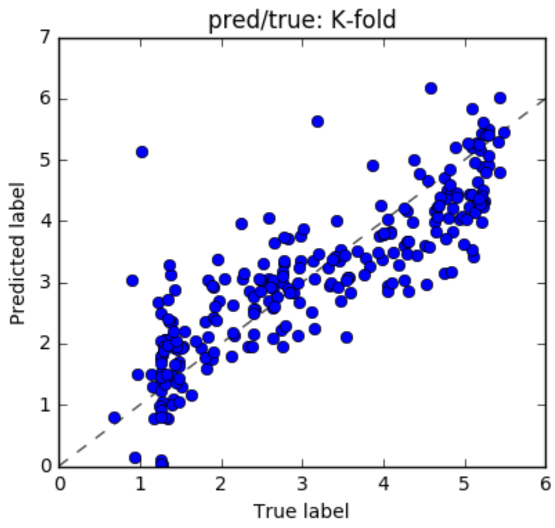
- **Ordinary Least Squares**: `OLS` Parameters: coef0
- **Ridge Regressen**: `ridge` Parameters: alpha, coef0
- **Lasso Regression**: `lasso` Parameters: alpha, coef0
- **Random Forests (Classification + Regression)** : `forestReg, forestClass` Parameters: max_depth, max_features, min_samples
- **Support vectors (Classification + Regression)** : `SVR`, `SVC` Parameters: alpha, gamma, coef0 Kernels: poly, RBF, sigmoid, . . .
- **Ridge regression kernels**: `ridge` Parameters: alpha, gamma, coef0 Kernels: poly, RBF, sigmoid, . . .

9 K-fold cross validation

10 Nested K-fold cross validation

```
In [9]: # Model specification
        parModel = {"regType":'ridge', "poly":3, "kernel":"poly", "coef0":1}
        # To be evaluated parameter(s)
        parLabel = ['alpha']
        parRange = [15]
        # Define kfold validation parameters
        testSize = 0.2
        k = 5
        kInner = 5
        #
        X = positionBox.values
        y = yRooted
        # Run function
        scoresParCV, optimalParCV = KfoldCV(X,y,k,parModel,parLabel[0],parRange[0])
```

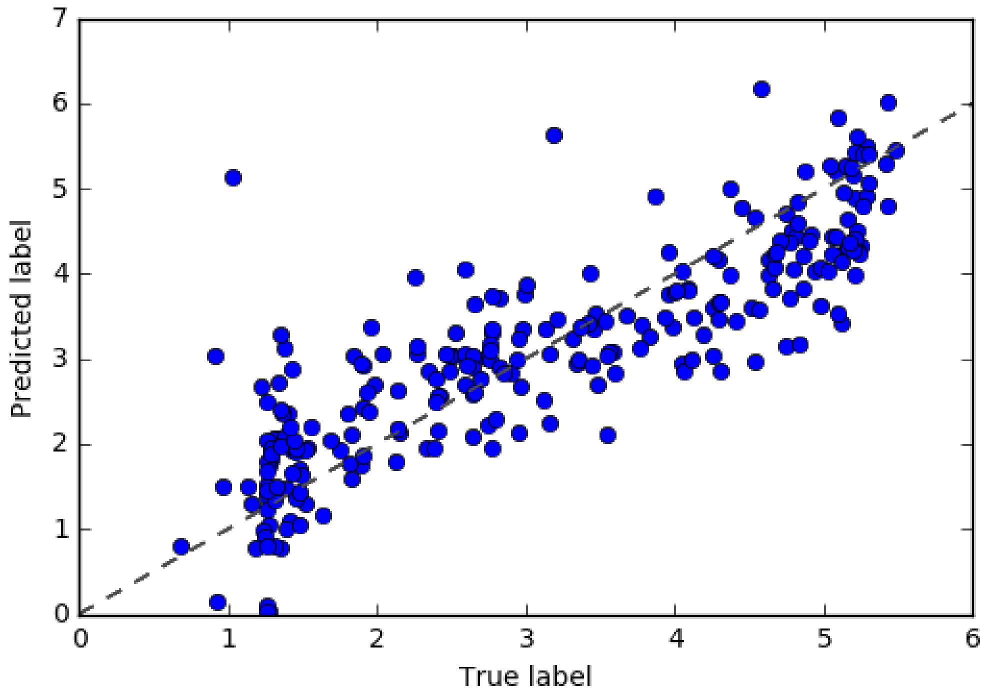




```
In [10]: meanScores = np.mean(np.ndarray.max(scoresParCV,axis=1))
         print("K FOLD CV \n---------- \n\n Maximum Score: ",np.max(scoresParCV), '
```

```
K FOLD CV
----------

 Maximum Score:  0.799712244586
 Mean optimal score:  0.71828884522
 sd optimal scores:  0.1535514781534831
 Optimal parEval:
 [ 100.   10.   10.   10.   10.]
 parEval Scores:
 [[ 0.31989756  0.31989758  0.31989776  0.31989961  0.31991811  0.32010276
   0.32192307  0.33795852  0.42008236  0.58898159  0.68370929  0.29640506
  -1.65266186 -4.44410483 -5.28106803]
 [ 0.51080289  0.51080291  0.51080306  0.51080455  0.51081945  0.51096811
   0.51241678  0.52401693  0.56517912  0.61233096  0.59169948  0.31607843
  -1.12683883 -3.4371721  -4.15089014]
 [ 0.70372468  0.7037247   0.70372491  0.70372698  0.70374768  0.70395398
   0.70595244  0.72118814  0.76496797  0.79971224  0.70280537  0.3204901
  -0.97382186 -3.33005372 -4.08065618]
 [ 0.522472    0.52247202  0.5224723   0.52247508  0.52250286  0.52277968
   0.5254527   0.54579531  0.61776066  0.71057502  0.6684023   0.35829674
  -1.14365655 -3.53183989 -4.2556942 ]
 [ 0.63368022  0.63368024  0.63368047  0.63368276  0.63370564  0.63393391
   0.63616824  0.65459056  0.72010534  0.78511671  0.78160052  0.42718334
  -1.6473034  -4.83777626 -5.79585222]]
```

```
In [11]: scoresParNCV, optimalParNCV, scoresNCV = nestedKfoldCV(X,y,k,kInner,parMoc
```



```
In [12]: print("NESTED K FOLD CV \n------------------ \n\n Maximum Score: ",np.max(s

NESTED K FOLD CV
------------------

 Maximum Score:  0.799712244586
 Mean optimal score:  0.71828884522
 sd optimal scores:  0.1535514781534831
 Optimal parEval:
 [[ 100.  100.  100.  100.  100.]
 [  10.   10.   10.   10.   10.]
 [  10.   10.   10.   10.   10.]
 [  10.   10.   10.   10.   10.]
 [  10.   10.   10.   10.   10.]]
 parEval Scores:
 [ 0.68370929  0.61233096  0.79971224  0.71057502  0.78511671]
```

11 Validation

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

```
<b>CODE SHOW </b>: turned ON

<b>CODE SHOW </b>: turned OFF
```

12 Something about rank

** Spearman's rank correlation coefficient**

13 Anderson promoter library

- 19 well-recognized promoters
- recovered by Chris Anderson
- sequence range: $ [-35,1]$
- single nucleotide variations

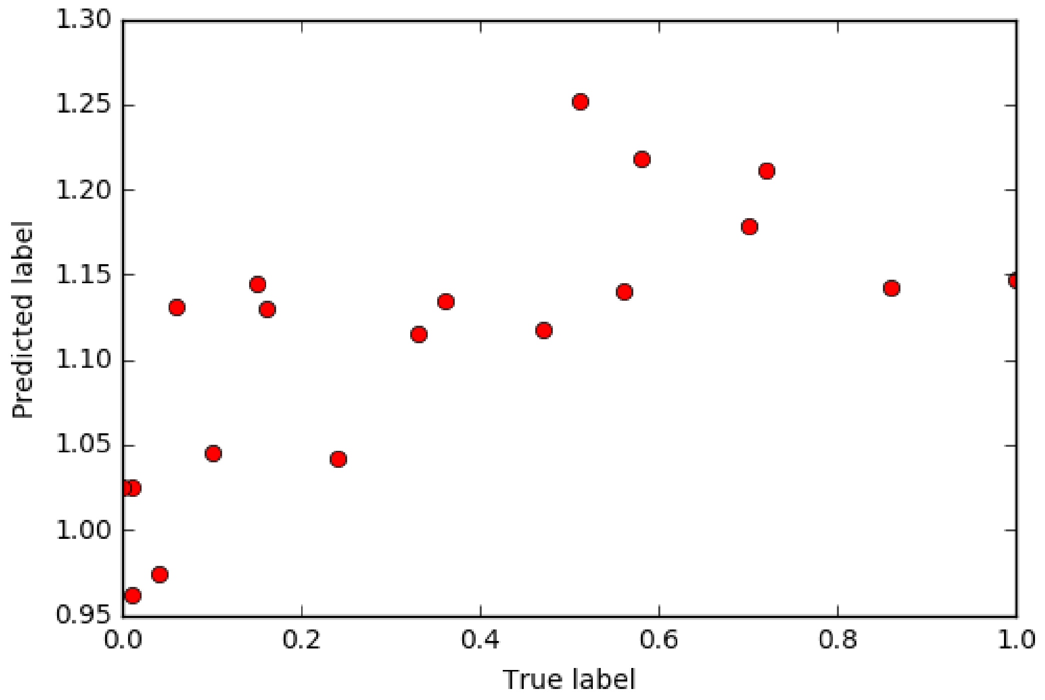
```
In [13]: dfDatasetTest = pd.read_csv("data/anderson_lib.csv")
         dfDatasetTest['sequence'] = dfDatasetTest['sequence'].str.upper()
         dfDatasetTest.iloc[:5,:3]
```

```
Out[13]:            ID                             sequence  mean_score
         0  BBa_J23100  TTGACGGCTAGCTCAGTCCTAGGTACAGTGCTAGC        1.00
         1  BBa_J23101  TTTACAGCTAGCTCAGTCCTAGGTATTATGCTAGC        0.70
         2  BBa_J23102  TTGACAGCTAGCTCAGTCCTAGGTACTGTGCTAGC        0.86
         3  BBa_J23103  CTGATAGCTAGCTCAGTCCTAGGGATTATGCTAGC        0.01
         4  BBa_J23104  TTGACAGCTAGCTCAGTCCTAGGTATTGTGCTAGC        0.72
```

```
In [14]: labelsTest, positionBoxTest, spacerTest = regionSelect(dfDatasetTest, ROI,
         Xtest = positionBoxTest.values

         parInput = {"regType":'ridge', "poly":3, "kernel":'poly', "gamma":0.1, "al
         reg = selectRegression(**parInput)
         reg.fit(X,y)
         rankPredict = reg.predict(Xtest)

         #print(np.transpose(np.vstack((dfDatasetTest['sequence'].values,dfDataset
         print(stats.spearmanr(dfDatasetTest['mean_score'].values,rankPredict))
         plt.plot(dfDatasetTest['mean_score'].values,rankPredict, 'ro')
         plt.xlabel('True label')
         plt.ylabel('Predicted label')
```

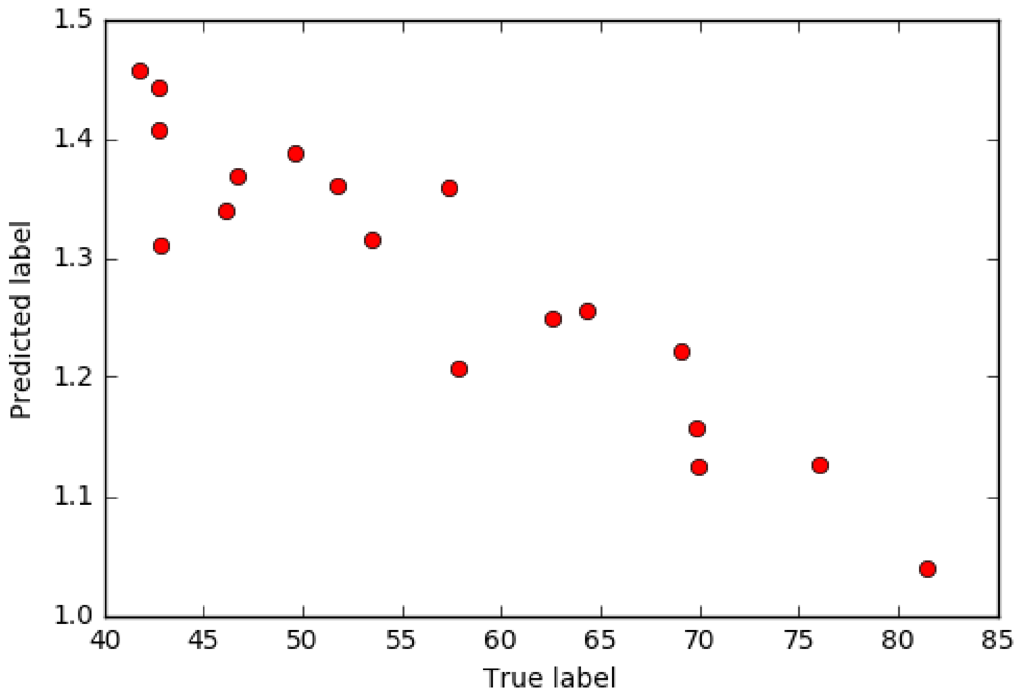
```
SpearmanrResult(correlation=0.80728709394205445, pvalue=2.9393274121457126e-05)
```

```
Out[14]: <matplotlib.text.Text at 0x7f028dfa0da0>
```



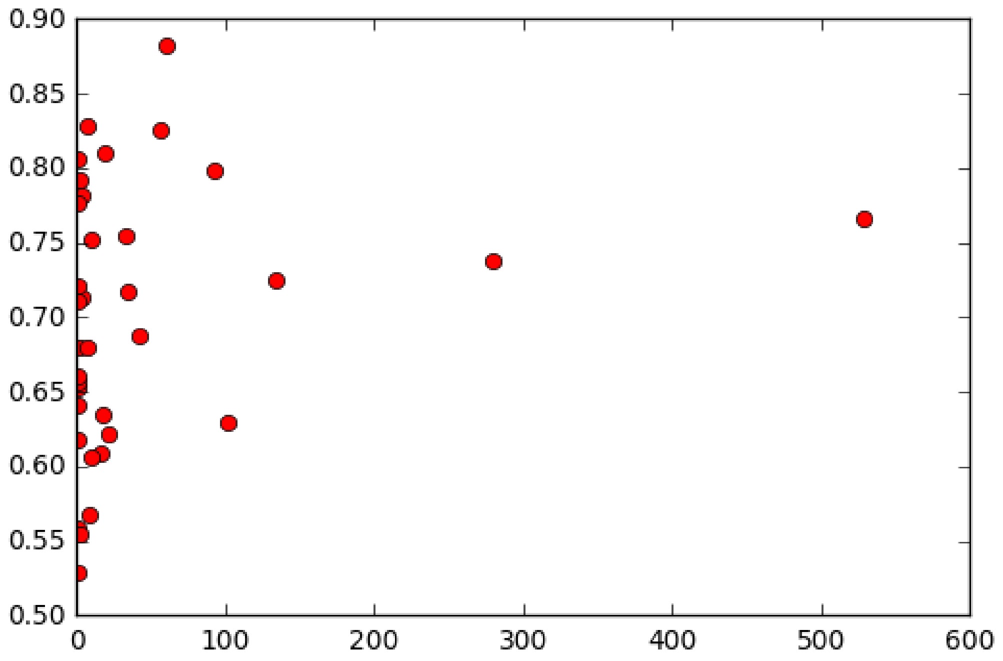
- 18 promoters
- sequence range: $ [-40,-1]$
- single nucleotide variations focused in and around TATA- and TTGACA-box

```
In [15]: dfDatasetTest = pd.read_csv("data/brewster_lib.csv")
         dfDatasetTest.iloc[:5,:3].style.set_properties(**{"font-size":"12px"})

Out[15]: <pandas.formats.style.Styler at 0x7f029c0baf28>

In [16]: labelsTest, positionBoxTest, spacerTest = regionSelect(dfDatasetTest, ROI,
         Xtest = positionBoxTest.values

         rankPredict = reg.predict(Xtest)
         #print(np.transpose(np.vstack((dfDatasetTest['sequence'].values,dfDataset
         print(stats.spearmanr(dfDatasetTest['mean_score'].values,rankPredict))
         plt.plot(dfDatasetTest['mean_score'].values,rankPredict, 'ro')
         plt.xlabel('True label')
         plt.ylabel('Predicted label')

SpearmanrResult(correlation=-0.91124871001031982, pvalue=1.4607836787089575e-07)


Out[16]: <matplotlib.text.Text at 0x7f028df8f320>
```



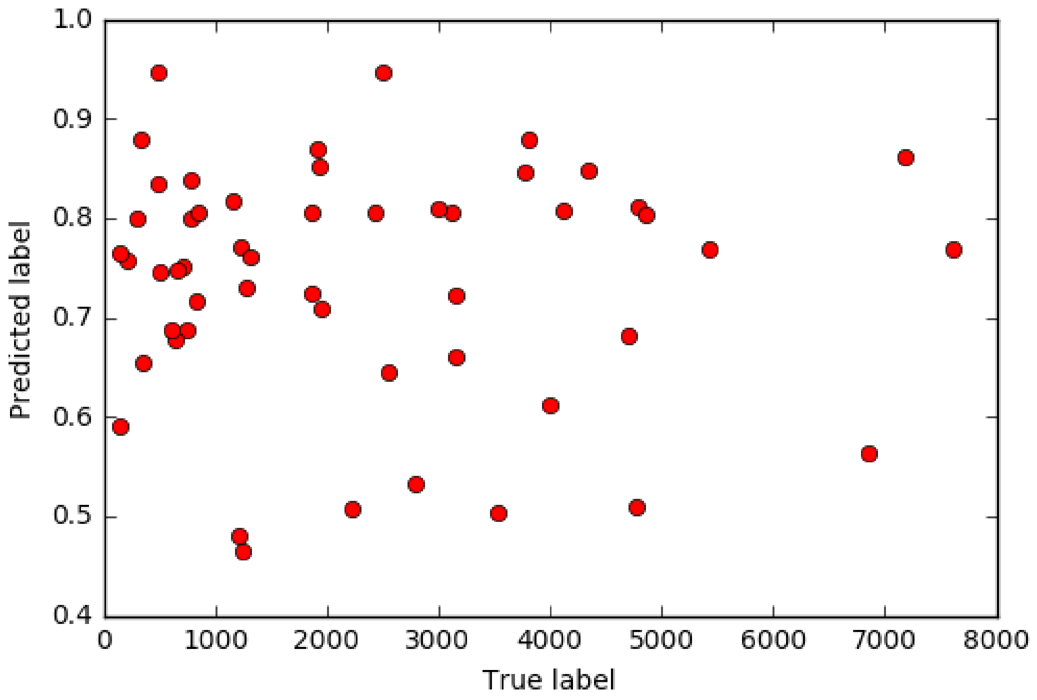
- 36 promoters
- sequence range: $ [-52,+7]$
- high level of variation over whole sequence

```
In [65]: dfDatasetTest = pd.read_csv("data/hammer_lib.csv")
         dfDatasetTest.iloc[:5,:3].style.set_properties(**{"font-size":"10px"})

Out[65]: <pandas.formats.style.Styler at 0x7fde8c521940>

In [42]: labelsTest, positionBoxTest, spacerTest = regionSelect(dfDatasetTest, ROI,
         Xtest = positionBoxTest.values
         rankPredict = reg.predict(Xtest)

         print(stats.spearmanr(dfDatasetTest['mean_score'].values,rankPredict))
         plt.plot(dfDatasetTest['mean_score'].values,rankPredict, 'ro')

SpearmanrResult(correlation=0.34255922865570765, pvalue=0.040839010065422274)


Out[42]: [<matplotlib.lines.Line2D at 0x7fde8c791e48>]
```



- 52 promoters
- sequence range: $ [-53,+4]$
- high level of variation over whole sequence

```
In [46]: dfDatasetTest = pd.read_csv("data/inbio_lib.csv")
         dfDatasetTest.iloc[:5,:3].style.set_properties(**{"font-size":"9px"})

Out[46]: <pandas.formats.style.Styler at 0x7fde8c7239b0>

In [44]: labelsTest, positionBoxTest, spacerTest = regionSelect(dfDatasetTest, ROI,
         Xtest = positionBoxTest.values
         rankPredict = reg.predict(Xtest)

         print(stats.spearmanr(dfDatasetTest['mean_score'].values,rankPredict))
         plt.plot(dfDatasetTest['mean_score'].values,rankPredict, 'ro')
         plt.xlabel('True label')
         plt.ylabel('Predicted label')

SpearmanrResult(correlation=0.024503212316951642, pvalue=0.86310286460841112)


Out[44]: <matplotlib.text.Text at 0x7fde8c778c50>
```



14 Remarks

- Model is only strong in predicting changes within and in close range of the -35 and -10 regions
- Training data only encompasses the $[-47, 1]$ region
- Simple features are used, giving a simple yet robust model
- Model has been optimized (used loss function) to minimize difference between predicted and true label
- Model cannot be further optimized with multiple dataset

15 Future work and focus

15.1 Ranked based optimization methods

- rank dependent loss function
- optimization over different datasets.

```
In [49]: import pandas as pd
         dfDatasetTest = pd.read_csv("data/pairedDBanalysis.csv")
         dfDatasetTest.iloc[:5]

Out[49]:    Unnamed: 0     ID_1     ID_2  \
         0           0  apFAB29  apFAB30
```

```
1           1  apFAB29  apFAB31
2           2  apFAB29  apFAB32
3           3  apFAB29  apFAB33
4           4  apFAB29  apFAB34

                                         sequence_1  \
0  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT
1  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT
2  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT
3  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT
4  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGATACTTACAGCCAT

                                         sequence_2  score_1  score_2  rar
0  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGTATAATGTGTGGAT   565.95   731.46    -
1  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGTATAATAGATTCAT   565.95   781.83    -
2  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGCATAATTATTTCAT   565.95   645.52    -
3  AAAAAGAGTATTGACTTAAAGTCTAACCTATAGGTAGATTTAACGTAT   565.95   340.48
4   AAAAAGAGTATTGACTATTAATCATCCGGCTCGATACTTACAGCCAT   565.95    14.63

   35boxstart_1  35boxstart_2  10boxstart_1  10boxstart_2
0            10            10            34            34
1            10            10            34            34
2            10            10            34            34
3            10            10            34            34
4            10            10            34            33
```

15.2 Neural Networks

- protein DNA interactions
- deep neural networks and convolution

```
In [ ]:
```