Semi-supervised learning for phenotypic profiling of high-content screens (DRAFT)

Roger Bermudez-Chacon
Supervisor: Peter Horvath

ETH Zurich

August 28, 2013

Abstract

Semi-supervised machine learning techniques are particularly useful in experiments where data annotation and classification is time- and resource-consuming or error-prone. In biological experiments this is often the case. Here, we apply a graph-based machine learning method to classify cells in different stages of infection with the Semliki Forest Virus (SFV), which features have been extracted from image analysis of fluorescence microscopy results, obtained in turn from a genome-wide high-content screening experiment. The aim of this project is to investigate whether and to which extent intelligent control experiment design combined with semi-supervised learning can reach the accuracy of a human annotator and/or in certain cases substitute it.

Introduction

Recent advancements in high-throughput microscopy and data analysis made possible to perform large scale biological experiments and automatically evaluate them. For the detection of sub-cellular changes caused by different perturbations in the cell (RNAi or drugs), often supervised machine learning (SML) is used. Reliable training of an SML method, however, requires significant effort from a field expert.

As an alternative, semi-supervised machine learning (SSL) methods make use of information intrinsically found in the entire data, both annotated and unannotated, thus allowing to make use of a larger amount of information by exploiting, alongside with the annotated data, the relative distribution of unannotated data on the feature space[1]. Furthermore, common sources of error in manually-annotated data (subjective data interpretation by the annotator, instrument capabilities and calibration, noise, ...) can affect the quality of the annotations, and in some cases SSL techniques can account for and correct such misannotations[2].

The semi-supervised learning paradigm, under a few assumptions[1], has proven valuable in exploring and classifying biological data in fields as diverse as drug-protein interactions[3], gene expression[4], and medical diagnosis[5].

Materials and Methods

High-content screening

A human genome-wide siRNA library was used to produce human cell cultures with knocked-out genes, stored in a collection of 55 384-well plates spanning the entire genome. These cell cultures were exposed to a genetically engineered fluorescent SFV strand as a transduction vehicle for green fluorescent protein (GFP) genes, and the corresponding GFP production on all of the cultures was tracked over time. 3 repetitions of this experiment were carried out.

The protein expression was stopped at t after culture infection with SFV, for all siRNA-mediated phenotypes, and microscopic pictures of the sample were obtained under a light microscope. Wild-type cultures were also exposed to SFV, and protein expression was stopped at 4, 5, 6, and 7 hours after exposure and

t=?

[1] Smoothness, Cluster, and Manifold assumptions, see [1] p. 4-6

analyzed as negative control experiments, in the exact same manner as for the silenced samples. A control sample set for non-exposure to the virus was also collected and analyzed in the same way as described above.

Image acquisition and analysis

For every sample at each infection stage, 9 tiled images were captured via a light microscope, by composing the green fluorescent signal of the produced protein, and a blue-colored image of the nuclei.

Image segmentation of all microscopic pictures to identify individual cells, and the subsequent feature extraction per identified cell, was done with CellProfiler[6]. A total of 93 features were retrieved and used in this experiment, corresponding to color intensity, area, shape, and texture descriptors. (For the complete list of features extracted, see Appendix A)

Unannotated data

The above process was performed automatically on the 380.160 images (55 plates $\times$ 384 wells $\times$ 9 sites $\times$ 2 channels) retrieved from the siRNA-mediated phenotypes. All 93 features[2] were extracted as floating point values, and stored in text files, one line per cell. Unlabeled information for around 100.000.000 cells was collected by this process.

Annotated data

From the genome-wide information, a small subset of the data was manually annotated by an expert on SFV infection, by visually identifying cell phenotypes directly from the segmented microscopic images and cross-checking with the time annotation on the respective source plate, and classifying them into the different stages of infection. This manual point-and-click process yielded 3098 annotated cells.[3]

software reference for this?

Control data

As control cultures, 5 plates treated with a control siRNA that has no effect on the expressed phenotype of the cells were used. Control plates not exposed to SFV, as well as infected plates analyzed at 4, 5, 6, and 7 hours after infection, were used to retrieve control information during the time course. The exact same image analysis and feature extraction procedure described for the unannotated cultures was performed on these plates.

Semi-supervised learning implementation

A graph-based label propagation (label spreading[7]) approach was followed. In this kind of approach, an undirected graph is built using the data points (cells) as vertices, and edges are created for all pairs of vertices that satisify a neighboring condition, with weights proportional to some measurement of association between the pair of vertices. This degree of association is often assumed as related to the distance between the data points in the n-dimensional feature space, in a linear, exponential, or Gaussian fashion, among others, and can be either limited in space (k-nearest neighbors, cutoff distance, ...), or consist of a complete graph that considers all possible pairwise relationships.

In the original formulation, a vector of initial labels $\hat{Y}^{(0)}$ is created by assigning the actual labels to the vertices corresponding to annotated data, and neutral labels to the unannotated data; then, in an iterative fashion, described in equation 1, all vertices *learn* the labels from their neighbors at a rate α (and preserve their initial labeling at a rate $1-\alpha$). This process is repeated until convergence to a $\hat{Y}^{(\infty)}$.

$$\hat{Y}^{(t+1)} \leftarrow \alpha \mathcal{L} \hat{Y}^{(t)} + (1-\alpha)\hat{Y}^{(0)} \tag{1}$$

The *laplacian* term controlling the particular strength of label spreading between any two data points is given by

[2] In total, 95 features were extracted from the images. Spatial coordinates, however, were regarded as of little, if any value for data analysis in this work.

[3] A small number of imaging artifacts were also identified manually. However, accounting for such information was out of the scope of this work.

$$\mathcal{L} \leftarrow \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}} \tag{2}$$

with $\mathbf{W}$ the weight or affinity (distance-related) matrix between nodes in the graph (and $\mathbf{W}_{ii} = 0$), and $\mathbf{D}$ the diagonal degree matrix $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$.

In the present implementation, the availability of *experimentally supervised data* from the experimental control samples was exploited. For each of these samples, the experimental conditions were known (in particular, the time after exposure to the virus), and a specific phenotype (label) could therefore be expected. To make use of this information, initial labels were assigned not only to the manually-annotated data points, but also to these experimentally supervised data points, by using the labels corresponding to their expected infection phase.

The n data points are classified in four types: labeled data, experimentally-supervised data with high labeling confidence (data points taken from the uninfected control samples), experimentally-supervised data with standard labeling confidence (data points taken from the infected control samples), and unlabeled data, with cardinalities n_l, n_h, n_s, n_u respectively. Different learning rates per each type (α_l, α_h, α_s, and α_u, respectively) were used to construct a diagonal $n \times n$ matrix $\boldsymbol{\mathcal{A}}$, such that $\boldsymbol{\mathcal{A}}_{ii} \in \{\alpha_l, \alpha_h, \alpha_s, \alpha_u\}$, according to the class of the data point i. The actual values are found by hyperparameter optimization (detailed in next sections).

The variant of the label spreading algorithm presented here, uses then the slightly modified (*experimentally supervised data*-aware) iterative formula:

$$\hat{Y}^{(t+1)} \leftarrow \mathcal{L}\boldsymbol{\mathcal{A}}\hat{Y}^{(t)} + (\mathbf{I} - \boldsymbol{\mathcal{A}})\hat{Y}^{(0)} \tag{3}$$

Development and runtime environment

To read and analyze the data, a script was coded in Python 2.7.4. Extensive use of the open source libraries `numpy` and `scipy` were used for matrix and numerical manipulation, as well as `matplotlib` for data visualization and graphical user interface.

Many options for this script are customizable via command line parameters. Appendix B includes a description of all the possible parameters and a quick user guide.

Feature selection and normalization

The values for all the features from the annotated cells were analyzed with Weka[8]. The InfoGain attribute evaluator was used to determine the information gain IG for each dimension (feature) with respect to each class or label, and the top 7 features were chosen as the set of selected dimensions D to represent the data ad-hoc for further analysis.

Due to the heterogeneity on the range of values between the selected features, spanning several orders of magnitude, normalization of the data was required. A standard z-score normalization was applied on all the dimensions. This normalization, however, does not account for the fact that some data dimensions are more important than others in terms of information gain. The relative information gain score among the group of selected features was used to construct a scaling factor or weight w_d for feature normalization, so that distances over less important dimensions are less penalized than those over more important ones.

$$w_d = \frac{IG_d}{\max_{d' \in D} IG_{d'}} \tag{4}$$

Afterwards, the values over all z-score normalized dimensions were rescaled to their respective w_d.

The selected features, and their respective information gain score and weight, were:

Data pre-processing

Text files in both `arff` and `txt` formats containing feature information for labeled, experimentally supervised ("soft-labeled"), and unlabeled data were read into a feature matrix $\mathbf{M}$ (n data points $\times m$ features).

The `arff` files containing *labeled data* were read and loaded into the feature matrix, by filtering the read fields to include only the features obtained in the feature selection phase. The possible classes or labels are loaded from the `arff` file by parsing the line starting with `@attribute class` (this prefix can be overriden via the `--label-line-prefix` parameter in the command line). As a parameter to the program, a list of ignored labels can be also passed with the option `--ignored-labels`. Data points annotated with any of

ID	Description	IG_d score	Weight w_d
4	Standard deviation of green intensities (whole cell)	1.3384	1.0000
3	Mean of green intensities (whole cell)	1.0987	0.8209
2	Standard deviation of green intensities (nuclei)	1.0309	0.7702
1	Mean of green intensities (nuclei)	0.9315	0.6960
92	Gabor X (texture scale 5, whole cell)	0.9133	0.6824
53	Inverse difference moment (texture scale 3, whole cell)	0.9064	0.6772
54	Sum average (texture scale 5, whole cell)	0.8906	0.6654

Table 1: Selected features, information gain score, and relative weight

these label identifiers will be left out of the feature matrix. No further sampling was performed over the labeled data at this point, i.e. all remaining (non-ignored) data points were kept.

The `txt` files containing *soft-labeled data* were read a similar way, except there was no need for parsing any formatting of the files. Each line in these `txt` files corresponds to a cell, and contains the values for the features, space-separated, in the same ordering as the labeled files. To assign actual soft labels, the relative file location in the file system was used as follows: the user indicates a root directory with all the soft-labeled data, and the files are expected in differents directories within, which are internally mapped (via a python dictionary) to the actual labels.

The `txt` files containing *unlabeled data* were read exactly as described above for the soft-labeled data. A default neutral label was assigned to all entries read from this files.

Due to the large amount of information, and for ease of testing, sampling parameters over the labeled, soft-labeled and unlabeled data were implemented. The command-line parameter `--num-labeled` controls how many labeled data points to use as training data. The parameter `--num-samples N` controls how many data points to use from both soft-labeled and unlabeled data together ($N/2$ each). An additional flag parameter `--class-sampling` indicates that the script must sample the soft-labeled data uniformly over classes, to avoid sampling bias due to large differences between the number of data points on each class.

As an outcome of this pre-processing step, the feature matrix **M** containing the values of the selected features for the labeled, soft-labeled, and unlabeled cells (after sampling, when specified) was returned, along with the initial label matrix $\hat{Y}^{(0)}$.

Graph construction

The graph was internally represented by its weight matrix **W** ($\mathbf{W}_{ij} > 0$ if there exists an edge between the vertices x_i and x_j, zero otherwise), plus a $n \times m$ label matrix **Y** (n cells, m possible labels or classes), with valid values ranging from 0 to 1.

To construct **W**, pairwise distances between all data points in **M** were calculated (the distance metric to use is customizable via command line, see Appendix B for possible values), and a neighbor weight function was applied over this distances. The following neighbor weight functions were implemented:

Exponential weight: $\mathbf{W}_{ij} = e^{-dist(p_i,p_j)}$ (complete graph)

***k*-nearest neighbors:** $\mathbf{W}_{ij} = 1$ if p_j is one of the k data points closest to p_i, 0 otherwise. (graph max degree k)

In the label matrix **Y**, each column represents a possible class label. A value of $\mathbf{Y}_{i,j} = 1$ represents complete confidence that the i-th cell in the data set belongs to the j-th phenotypic class of cells. Likewise, a value of 0 indicates absolute confidence that a cell does *not* correspond to a class, and uniform values of $1/m$ indicate complete uncertainty about class membership. To encode the initial labels in **Y**, the rows corresponding to *labeled data* were one-hot encoded (a value of 1 for the column corresponding to their initial label, 0 elsewhere), whereas for the *unlabeled data* a value of $1/m$ was used on all label columns.

For the *experimentally-supervised data*, the learning rate α was also used to initialize the labels. A low learning rate means that there is strong belief that the initial labeling is correct, and the extreme values of 0 and 1 suggest that the data points should behave as fully-annotated and fully-unannotated data, respectively, as described above. According to this criterion, the labeling of the experimentally-supervised data is given by

$$\mathbf{Y}_{i,k} = \begin{cases} \dfrac{1}{\alpha(m-1)+1} & \text{if } E_i = k \\ \dfrac{\alpha}{\alpha(m-1)+1} & \text{otherwise} \end{cases} \tag{5}$$

with E_i the experimentally-assigned label for the data point i.

Label propagation

The laplacian matrix $\boldsymbol{\mathcal{L}}$ was obtained from $\mathbf{W}$, as indicated in equation 2, and the $\boldsymbol{\mathcal{A}}$ matrix encoding the learning rates was derived from the mappings between the initial labels and the class-wise learning rates.

The implemented version of the label propagation algorithm was run up to a number of maximum iterations or convergence to a steady state m-dimensional labeling vector $\hat{Y}^{(\infty)}$ (with m the number of possible classes or labels. Each entry on this vector represents a data point, and contains the values spread by the neighbors' labels.

The intuitive idea of annotating each data point with the label associated to the highest value on this vector (highest combination of the original labeling and the neighbors' labeling) does not consider the prior class distribution of the data (i.e. since the values are the sum of individual contributions from the neighbors, labels with a smaller relative frequency can not compete against more frequently-occurring labels in a fair way). To account for this, *class mass normalization*[9] was also applied on the labeling vector, prior to deciding on the actual classification, by scaling each class (column) by the estimated class distribution. The actual labeling for each data point i, after normalization, is then given by $argmax_k\ \hat{Y}_{ik}$.

In a standard labeling run, the estimated $\hat{Y}^{(\infty)}$ is then returned to the user.

Hyperparameter optimization

To evaluate the performance of the script under different hyperparameter configurations, evaluation on a held-out validation set was used. The labeled data was split into a small set of training data points and a larger set of validation points (on a 1:2 ratio). Grid search was used to explore a discretization of the parameter space and identify hyperparameter configurations that yielded the best classification results in terms of accuracy (i.e. correct identification of the virtually hidden labels from the validation points).

The ranges for the discrete partitions were decided arbitrarily, but exploiting some rough intuition of their possible values to constraint the subset of the discretized hyperparameter space to explore. The hyperparameters to be optimized, and their discrete partitions, were:

should we mention this in the report?

Hyperparameter	Discrete partitions
α_l	{0.05, 0.10, 0.15, 0.20, 0.25}
α_h	{0.00, 0.10, 0.20, 0.30}
α_s	{0.30, 0.40, 0.50, 0.60, 0.70}
α_u	{0.50, 0.60, 0.70, 0.80, 0.90}
k-nn	{4, 5, 6, 7, 8, 9}

Table 2: Hyperparameters and discrete partitions for grid search

Results

Cras pretium. Nulla malesuada ipsum ut libero. Suspendisse gravida hendrerit tellus. Maecenas quis lacus. Morbi fringilla. Vestibulum odio turpis, tempor vitae, scelerisque a, dictum non, massa. Praesent erat felis, porta sit amet, condimentum sit amet, placerat et, turpis. Praesent placerat lacus a enim. Vestibulum non eros. Ut congue. Donec tristique varius tortor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nam dictum dictum urna.

Discussion

Suspendisse erat mauris, nonummy eget, pretium eget, consequat vel, justo. Pellentesque consectetuer erat sed lacus. Nullam egestas nulla ac dui. Donec cursus rhoncus ipsum. Nunc et sem eu magna egestas malesuada. Vivamus dictum massa at dolor. Morbi est nulla, faucibus ac, posuere in, interdum ut, sapien. Proin consectetuer pretium urna. Donec sit amet nibh nec purus dignissim mattis. Phasellus vehicula elit at lacus. Nulla facilisi. Cras ut arcu. Sed consectetuer. Integer tristique elit quis felis consectetuer eleifend. Cras et lectus.

Conclusions

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec scelerisque metus. Maecenas non mi ut metus porta hendrerit. Nunc semper. Cras quis wisi ut lorem posuere tristique. Nunc vestibulum scelerisque nulla. Suspendisse pharetra sollicitudin ante. Praesent at augue sit amet ante interdum porta. Nunc bibendum augue luctus diam. Etiam nec sem. Sed eros turpis, facilisis nec, vehicula vitae, aliquam sed, nulla. Curabitur justo leo, vestibulum eget, tristique ut, tempus at, nisl.

Bibliography

[1] O. Chapelle, B. Schölkopf, A. Zien, *et al.*, *Semi-supervised learning*, vol. 2. MIT press Cambridge, 2006. 1

[2] W. Du and K. Urahama, "Error-correcting semi-supervised pattern recognition with mode filter on graphs," in *Aware Computing (ISAC), 2010 2nd International Symposium on*, pp. 6–11, IEEE, 2010. 1

[3] Z. Xia, L.-Y. Wu, X. Zhou, and S. Wong, "Semi-supervised drug-protein interaction prediction from heterogeneous biological spaces," *BMC Systems Biology*, vol. 4, no. Suppl 2, pp. 1–16, 2010. 1

[4] I. Costa, R. Krause, L. Opitz, and A. Schliep, "Semi-supervised learning for the identification of syn-expressed genes from fused microarray and in situ image data," *BMC Bioinformatics*, vol. 8, no. Suppl 10, p. S3, 2007. 1

[5] E. Bair and R. Tibshirani, "Semi-supervised methods to predict patient survival from gene expression data," *PLoS biology*, vol. 2, no. 4, p. e108, 2004. 1

[6] A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, *et al.*, "Cellprofiler: image analysis software for identifying and quantifying cell phenotypes," *Genome biology*, vol. 7, no. 10, p. R100, 2006. 2

[7] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," *Advances in neural information processing systems*, vol. 16, no. 16, pp. 321–328, 2004. 2

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009. 3

[9] X. Zhu, Z. Ghahramani, J. Lafferty, *et al.*, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, vol. 3, pp. 912–919, 2003. 5

[10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2001.

Appendix A

Features analyzed

Position and intensity		Nuclear shape features	
1	nuclei location x	11	Area
2	nuclei location y	12	Eccentricity
3	green mean intensity nuclei	13	Solidity
4	green std intensity nucleii	14	Extent
5	green mean intensity cells	15	Euler number
6	green std intensity cells	16	Perimeter
7	blue mean intensity nuclei	17	Form factor
8	blue std intensity nuclei	18	Major axis length
9	blue mean intensity cells	19	Minor axis length
10	blue std intensity cells	20	Orientation

Texture features (Scale 3)

Nuclei texture (green)		Nuclei texture (blue)		Cell texture (green)	
21	Angular 2nd moment	36	Angular 2nd moment	51	Angular 2nd moment
22	Contrast	37	Contrast	52	Contrast
23	Correlation	38	Correlation	53	Correlation
24	Variance	39	Variance	54	Variance
25	Inverse difference moment	40	Inverse difference moment	55	Inverse difference moment
26	Sum average	41	Sum average	56	Sum average
27	Sum variance	42	Sum variance	57	Sum variance
28	Sum entropy	43	Sum entropy	58	Sum entropy
29	Entropy	i 44	Entropy	59	Entropy
30	Difference variance	45	Difference variance	60	Difference variance
31	Difference entropy	46	Difference entropy	61	Difference entropy
32	Info measurement 1	47	Info measurement 1	62	Info measurement 1
33	Info measurement 2	48	Info measurement 2	63	Info measurement 2
34	Gabor X	49	Gabor X	64	Gabor X
35	Gabor Y	50	Gabor Y	65	Gabor Y

Texture features (Scale 5)

Nuclei texture (green)		Cell texture (green)	
66	Angular 2nd moment	81	Angular 2nd moment
67	Contrast	82	Contrast
68	Correlation	83	Correlation
69	Variance	84	Variance
70	Inverse difference moment	85	Inverse difference moment
71	Sum average	86	Sum average
72	Sum variance	87	Sum variance
73	Sum entropy	88	Sum entropy
74	Entropy	i 89	Entropy
75	Difference variance	90	Difference variance
76	Difference entropy	91	Difference entropy
77	Info measurement 1	92	Info measurement 1
78	Info measurement 2	93	Info measurement 2
79	Gabor X	94	Gabor X
80	Gabor Y	95	Gabor Y

Appendix B

Script parameters and help

up to date?

```
$ python hcs.py -h
usage: hcs.py [-h] [-t] [-l LABELED_FILE [LABELED_FILE ...]]
              [-u UNLABELED_FILE [UNLABELED_FILE ...]] [-s SOFT_LABELED_PATH]
              [-L NUM_LABELED_POINTS] [-n NUM_SAMPLES] [-c]
              [--max-iterations MAX_ITERATIONS] [-d WIDTH]
              [-nf {exp,knn3,knn4,knn5,knn6}]
              [-dm {euclidean,cityblock,cosine,sqeuclidean,hamming,chebyshev}]
              [-f FEATURE_INDEX [FEATURE_INDEX ...]] [-q]

Label propagation

optional arguments:
  -h, --help            show this help message and exit
  -t, --test            Performs a test run.
  -l LABELED_FILE [LABELED_FILE ...], --labeled LABELED_FILE [LABELED_FILE ...]
                        Labeled files.
  -u UNLABELED_FILE [UNLABELED_FILE ...], --unlabeled UNLABELED_FILE [UNLABELED_FILE ...]
                        Unlabeled files.
  -s SOFT_LABELED_PATH, --soft-labeled SOFT_LABELED_PATH
                        Path to soft labeled files. One directory per label
                        expected.
  -L NUM_LABELED_POINTS, --num-labeled NUM_LABELED_POINTS
                        Number of labeled data points to use. Default: use all
                        available
  -n NUM_SAMPLES, --num-samples NUM_SAMPLES
                        Number of samples. Default: 3000
  -c, --class-sampling  Distributes the number of samples given by
                        [NUM_SAMPLES] uniformly over all soft classes
  --max-iterations MAX_ITERATIONS
                        Maximum number of iterations. Default: 1000
  -d WIDTH, --display-columns WIDTH
                        Max width used for matrix display on console
  -nf {exp,knn3,knn4,knn5,knn6}, --neighborhood-function {exp,knn3,knn4,knn5,knn6}
                        Neighborhood function to use. Default: exp
  -dm {...}, --distance-metric {euclidean,cityblock,cosine,sqeuclidean,hamming,chebyshev}
                        Metric for calculating pairwise distances. Default:
                        euclidean
  -f FEATURE_INDEX [FEATURE_INDEX ...], --features FEATURE_INDEX [FEATURE_INDEX ...]
                        Selected feature indices (as given by the labeled
                        data).
  -q, --quiet           Displays progress and messages.
```