Machine Learning Implementation in live-cell tracking

Bo Gu

Dec.12th 2014

Abstract

While quantitative biology has gradually become the major trend of biology, researchers have put their eyes on analysis tools that can give them more quantitative read-out of the biological system. Cell biologists are researchers that have started to use quantitative image analysis tools at the earliest time, and by using these tools, information such as the dynamics of certain molecules of interest can be quantified rather easily. Among all the approaches, live-cell cell tracking has become the most informative but also the most challenging one. To date, most of the cell-tracking scripts are semi-automatic of which parameters have to be heavily tuned as well as observational supervision. To improve this method and make it more automatic is thus a meaningful problem. In this project, we are hoping to implement machine learning into the tracking dataset and look for potential improvement of the original tracking method.

1 Introduction

As more and more sub-fields of biology is going into quantitative realm, biologists have started to pay serious attention to tools that can help to generate quantitative data. Cell biologists are one group that firstly started and still lead this trend. Thanks to the fast development of state-of-the art microscope, cell biology has undergone a radical transition from being mostly observational to real-time, high-throughput and quantitative. Using fluorescently-tagged proteins and cellular organelles, researchers can perform all sorts of experiments on live cells to understand the basic principle governing cell biology. Analyzing data from these experiments requires image processing to quantify the spatio- and temporal-dynamics of specific labeled proteins at the single-cell resolution termed 'tracking'. By tracking the cell at this resolution, information such as variability among the population of cells, which was never possible by traditional biochemical measurement, can be reflected.

However, it turned out that the following analysis part of the image is rather tricky and not close to the capability to precisely analyze the large amount of data. Particularly, live-cell tracking has become one of the biggest issues in quantitative image analysis. To date, the general process of live-cell tracking involves 3 major steps: 1. Image segmentation, 2) Cell feature extraction and 3) Image sequence assignment. Of the 3 basic steps for live-cell tracking, the 3rd step, which is to assign each single-cell in one frame to its correspondent cell in the previous frame, is the most challenging one. While there are sophisticated open-source software and packages that can help biologists process individual frames of

image such as segmentation, there are not even good way to track cells in lab, let alone the publicly available ones. In practice, researchers usually track cells by one of the five ways below: including the centroid method, the Gaussian fit method, the correlation method, the sum-absolute difference method, and the interpolation method. Among them, the most widely used method is simply by finding the nearest assignment between consecutive frame for each single cell. This strategy works for at most 50% of the situations but collapses in the situations where cells that are plated too dense or cells that are moving too fast. In recent years, researchers have come up with different new ideas about how to improve the tracking algorithm to make them work better. Both the appearance features which assumes that the shape of a certain cell changes slightly between consecutive frames and the method to maximize the smoothness of trajectory and the velocity of the cells have been raised. Clearly, the take-home message here is that more features have to be incorporated to track cells more precisely. Here, we are thinking about implementing machine learning algorithm to 1) determine which feature of the cells weights more when it comes to cell tracking. 2) whether it's possible to come up with generalization of tracking method by implementing machine learning. 3) hopefully, to generalize a good algorithm for live-cell tracking and integrate directly into the image analysis pipeline in our lab.

2 Method

2.1 Dataset

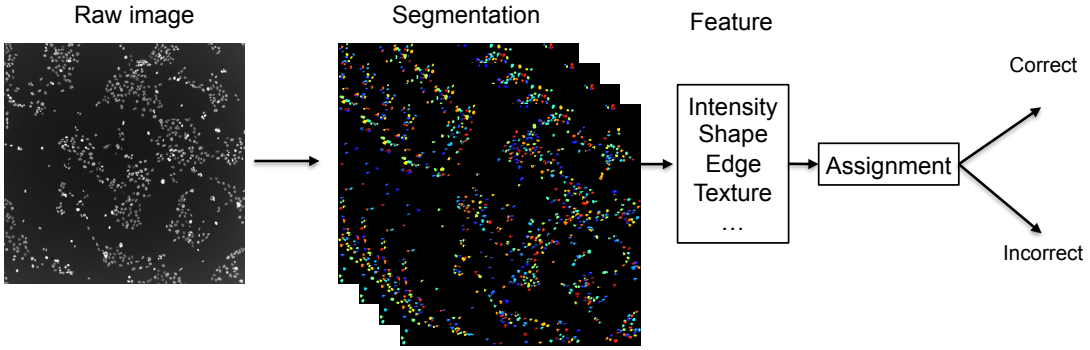
Thanks to researchers from the Meyer lab at Stanford University, we got the live-cell tracking movies consisting of hundreds of consecutive sequences of frames of images taken by confocal microscopy. Each frame of image is a greyscale image of 2160x2160 pixels. The tracking images are for different types of cells, one of which is a human mammary gland epithelial cells (MCF10A), the other is a Human Umbilical Vein Endothelial Cells (HUVEC). For each type of cells, different channels of microscope recording the localization and dynamic changes of distinct tagged proteins are gathered. The features for each single cell at each frame are extracted and recorded by an open-source software package and further customization of the scripts. Features are then recorded for each cell in the frame, for all frames, thus generating three dimensional matrices M Rmxnxk with each layer k represent the kth frame of the image sequence, the ith row representing the ith cells identified within one single frame and the jth column for the jth feature for all the cells in one frame.

With the customized script for cell tracking, individual cells between consecutive frames are assigned to its putative correspondent cells. This is done by mostly finding the nearest cells or largest overlap between each two consecutive frames. The tracking event for each single cell between consecutive frames are later on classified as either correctly tracked (assigned label '1') or mis-tracked (assigned label '0'). Conveniently, since all the features extracted are quantified by numbers, we take the difference of each feature between each consecutive frames for individual cells as their training features. The tracking problem then turns into a supervised learning problem where we can implement various learning algorithms on.

2.2 Feature extraction

2.2.1 Segmentation and feature extraction for individual frame

A typical workflow of live-cell image analysis involves the processes shown in figure 1. The first step is to preprocess the raw image into quantifiable digital data. This includes at least segment specific cellular organelle and extract essential features of each individual cell in a single frame as many as possible. Traditionally, different labs come up with their own customized script to do the image segmentation and feature extraction, which include common features such as Centroid Position, MeanIntensity, StdIntensity, MeanIntensityEdge and Perimeter. However, more complicated features that are typical in the field of digital image processing is not accessible by most of the scripts written in house. Even matlab scripts written separately are not fully functional or available for some of these features. Considering the concern and recent progress in live-cell image analysis, we feel it might be useful to incorporate some of these features into our later tracking. Conveniently, a recently developed open-source software package by researchers at MIT (Cellprofiler) incorporated the capability to extract most of the features of digital image processing. Cellprofiler simultaneously measures the size, shape, intensity and texture (including even the most powerful Gabor Features and Haralick Features) of a variety of cell types at the single-cell resolution in a high throughput manner. This greatly ease the process of image segmentation and feature extraction and make this whole process modular so that each parameters can be tweaked interactively with a GUI. In order to check the real capacity of the software, we have done thorough challenges to the segmentation power and feature extraction power and careful check of the source code (mostly written in python and matlab), and concluded that for the segmentation and feature extraction of individual frame, the software can do at least no worse job than most of the currently available scripts written in the Meyer Lab. Therefore, we decided to shift our whole working platform for image processing of individual frames to this pipeline. For each single cell at individual frame, we can extract up to 112 meaningful features and all the features are quantitative representation of specific aspect of the property of single cell.



(a)

Figure 1: Typical pipeline of live-cell quantitative image analysis. Raw data are retrieved by confocal microscopy and sequences of collected images were later segmented using customized script. Cell tracking were done by assignment of individual cells to its correspondent parent cells in previous images.

2.2.2 Live-cell tracking

After the segmentation and feature extraction, each cell is represented by a vector in R112 for each frame, the next step is to connect consecutive frame together so as to assign each single cell to the previous/next frame. To this end, we implement the tracking script from the Meyer lab instead of the ones in the tracking pipeline from Cellprofiler due to the poor performance the tracking by the software per se. After the assignment done by primarily finding the nearest cell in the next/previous frame within a certain range of searching area measured in pixels, we represent each cell by a matrix where each row vector represents the features extracted from an individual frame.

This then gives us a good starting point for building our training set. As always, the assignment done by the scripts has a certain error rate, where cells are mis-assigned even if they fit the initial conditions. By manually check the error rate of several frames and pick the data of lower error rate, we can further conduct manual correction of the dataset or simply just filter out the cells that are mis-assigned. The further filtered cells are then used as our positive dataset. Initially, to minimize the contribution of position factors in differentiating between the positive and negative training sets, we limit our negative training set to the cells that are assigned by the script (meaning they fit the condition of nearest object so that the distance factor is eliminated) but are filtered out. However, due to the fact that there will just be too little negative training set compared with positive training set and the intuition that we can't simply eliminate the contribution of distance in differentiating between positive and negative set, we relax our criteria in finding the negative training set to as much as 10 fold of the distance limit than that of the positive set (within 15 pixels). With that, the factors that contribute to negative training set is a mixture of both the distance and other factors.

After we set up our criteria for choosing positive and negative training set, we do further transformation of the dataset for easier accessibility by machine learning algorithm. Given X_k^i as the feature vector we get for the ith tracked/mis-tracked cell at the kth frame, we think that the $\Delta X = X_j^i - X_j^i - 1$ between each consecutive frames might be a good estimation for the input feature we want. We try a different set of features which are the square of the original features. The current approach eliminate the sign ambiguity.

2.2.3 Dimensionality reduction of feature space

As mentioned before, the feature vector for each cell contains 112 entries. This high dimensionality of input feature space might potentially cause over-fitting to the learning algorithm. Aside increase the training set, dimensionality reduction might also be an option to address the potential issue. To do this, we firstly implement PCA to reduce the dimensionality of our feature vectors. Not only will this pre-processing be helpful to reduce the dimensionality of the feature, but it will also be essential for implementing later on the Naive Bayes learning algorithm since the Naive Bayes assumption requires that features x are conditionally independent given y. Based on the top eigenvalue of the cross-correlation matrix we got, we decided to choose the top 20 linear combination of feature which takes up 90% of the sum of eigenvalues and reduce the input feature dimensionality from 112 down to 20.

3 Model training

We chose the classification approach cause we can simply apply different methods and compare the effect of them. We use the delta v between consecutive frame as the feature space and the positively tracked feature as positive example and the others as negative. In this sense, we should have way more negative examples than positive examples. To get the data, we do pairwise comparison between the features extracted in each consecutive frame and calculate the pairwise delta feature. Then, by applying our original tracking code, we'll identify the positively tracked pairs and its correspondent feature space.

To train our dataset, we choose to use Discriminant analysis, as well as SVM models and compared their relative efficiency and precision.

3.1 Logistic regression

We perform logistic regression on training set of different sizes by stochastic gradient ascent in MATLAB (The MathWorks, Natick, MA, USA).

3.2 Support vector machine with linear kernel

We train a support vector machine (SVM) using the LIBSVM libraray (ref). We select linear kernel (the default kernel) to get the optimal linear boundary hopefully at even higher dimensional space for input features. However, we are not sure yet whether the training-set is linearly seperable at higher dimension, so implementing other kernels has also been considered.

3.2.1 Naive Bayes

Since the features in our training set are all continuous, we have to conduct Gaussian Naive Bayes which assumes that the probability distribution of some variable x given a class, $p(x = \nu|c)$ can be computed by plugging v into the equation for a Normal distribution parmeterized by μ_c and σ_c^2 and the probability is simply given by

$$p(x = \nu|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(\nu-\mu_c)^2}{2\sigma_c^2}}$$

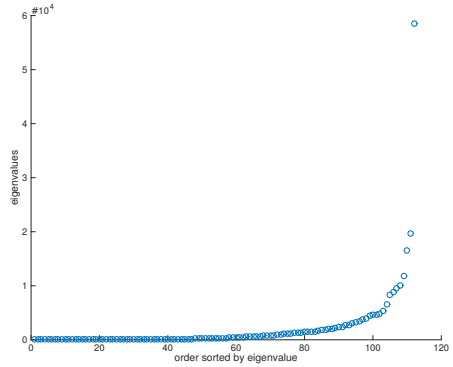
To perform Gaussian Naive Bayes, we use fitNaiveBayes function in MATLAB to create a Naive Bayes classifier and later prediction. By default, the function models the predictor distribution within each class using a Gaussian distribution as described above.

3.3 Gaussian discriminant analysis

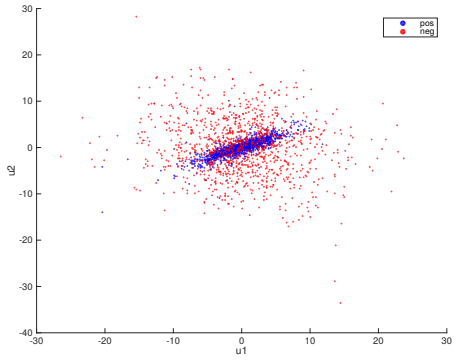
To train a Gaussian discriminant classifier, we use the fitcdiscr function in MATLAB. Later on, we use predict function to get the output label by minimizing the expected classification cost

$$y = arg \min_{y=1,\cdots,K} \sum_{k=1}^{K} P(k|x)C(y|k)$$

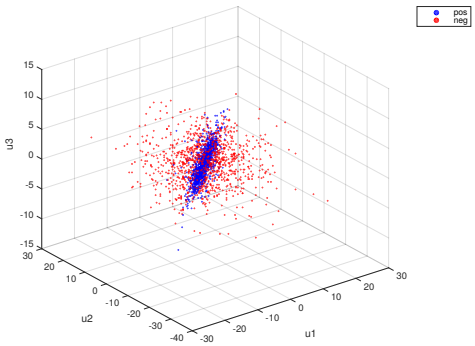
Where is the predicted classification.



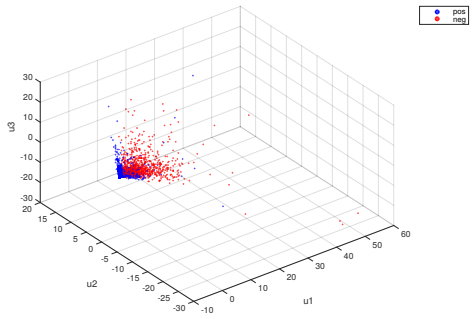
(a) sorted eigenvalue



(b) input feature x projected onto top 2 principal components



(c) input feature x projected onto top 3 principal components



(d) input feature x^2 projected onto top 3 principal components

Figure 2: Visualization of input features at lower dimension. Blue and red dots represent the positive and negative training sets respectively

4 Evaluation of learning algorithm

For each learning algorithm we perform hold-out cross validation, training on 70% of the data and testing on the rest 30% of the data. We generate training error and testing error and the learning curve for each method for comparison.

5 Result

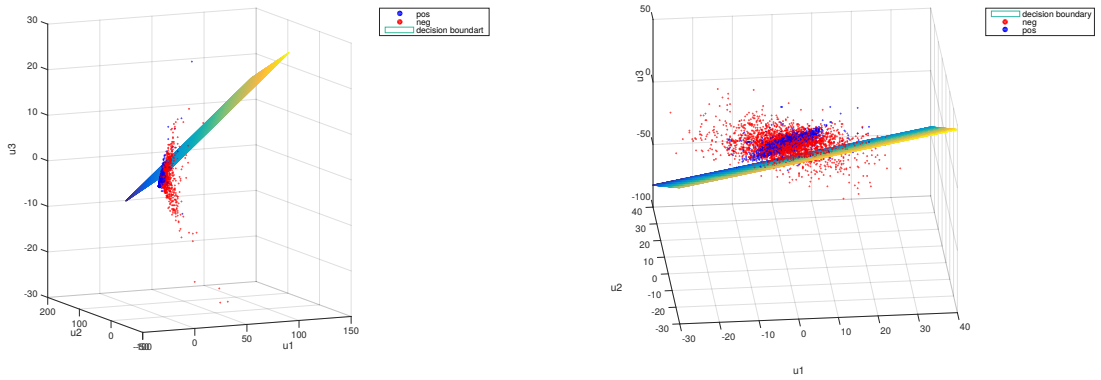
5.1 Dimension reduction of input feature

As mentioned above, for each single cell, our input vector is in 112 dimension. However, since all these features are extracted automatically by relatively independent programs developed by different group, although they are integrated into the same platform, it is not known whether all these features are independent. To check the mutual independence of the input features, we first did pairwise cross correlation of the input features. As expected, some features appear highly correlative, suggesting the possibility of dimension reduction. In addition, some pairs of features shows strong negative correlation and such

correlation turned into positive correlation when we convert the input feature X into their square feature X^2. We calculate the eigenvalues λs of the cross correlation matrix $\frac{1}{m}X^TX$, which is shown in figure 2(a) and sorted by their weight. It is interesting to see that the top 3 eigenvalue already consists about 70% of the sum of the eigenvalues. Thus, we decided to choose the top 3 or 2 eigenvalues as the principal component for visualization of the input feature. Further inspection of the eigenvalue tells us that other than the position information, texture features also have a high weight in constituting the principal components. As shown in figure 2(b) and 2(c), input features are then projected onto these three principal components for visualization. Also shown in figure 2(d) is the projection of original data X top 2 principal component. From the figure, we can tell that these dataset are linearly inseperable when projected onto the arbitrarily chosen 3 principal component. This indicates that we need to either increase the feature or to incorporate polynomial of these three components in order to use linear classifier.

5.2 Logistic regression

To check the efficiency of linear classifier, in this case the basic logistic regression without regularization, we perform logistic regression on the new feature matrix converted by $(\mu_1 + \mu_2 + \mu_3)x$. As shown in figure 3. The linear decision boundary created by logistic regression really doesn't do a good job in separating the two sets of data. This lead us to think maybe using logistic regression with the polynomial of these projected features might help and we are in the process of check that. However, do to the time limit, we don't have enough time to finish this part but it is definitely worth trying.



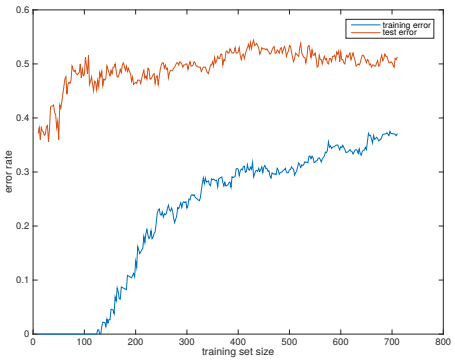
(a) decision boundary of x^2 input (b) decision boundary of x input

Figure 3: Decision boundary visualization of PCA. Blue and red dots represent positive and negative training sets respectively

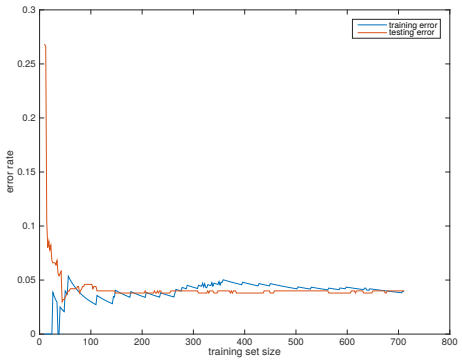
5.3 Comparisons of performances of different classifiers

To test other more complicated classifier, we choose classifiers including Gaussian Naive Bayes (NB), Support Vector machine with linear kernel (SVM) and Gaussian discriminant analysis (GDA). Shown in figure 4 are the learning curves of individual classifiers with all the input features. Interestingly, three approaches showed quite different behavior on the same training set. As expected, for all three method, training error increases with

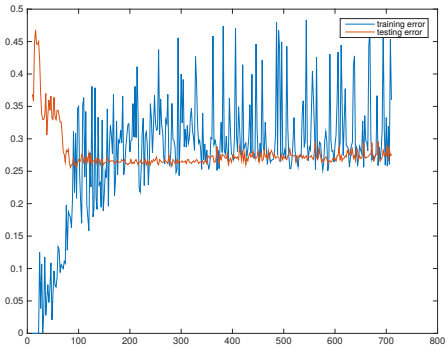
the increase of training set. When we take the testing error into consideration, GDA performs the worst and didn't do well in learning. NB performs the best and learns very well even with small number of training set. Also, the training error and testing error converge on a very small number, indicating a very successful classification capability. SVM however, shows a significant reduction of the testing error along with the increase of training set, but the change of training error showed strange pattern, where high frequent spikes are observed on the learning curve. This could either indicate the quality of the input set or simply because of the learning process is limited by the maximum iteration number in the svm code used; In most of the cases, the optimization reaches the iteration limit of the svm. Therefore, we are further considering changing from linear kernel to radial basis kernel and select appropriate parameters by performing a grid search in log-space to maximize area under the (receiver operator characteristic) curve obtained by crossvalidation. In conclusion, among all three classifiers, Gaussian NB performs the best and we are going to further optimize this approach for later use.



(a) learning curve of GDA



(b) learning curve of Gaussian Naive Bayes



(c) learning curve of SVM with linear kernel

Figure 4: Performance comparison between different learning algorithms. Blue and orange represents training and testing error respectively.

6 Future direction

In conclusion, in this project, we were managed to get new insight into the feature structure for live-cell tracking and this project provided us chances to developing or optimizing our original script for cell tracking. In addition, due to the good performance of Gaussian Naive Bayes classifier, we are having a good start of building a machine learning based approach for cell tracking.

However, the further we dig into the project, the more we feel a lot more can be done. We are at least thinking about including more dataset from different cell types with different morphology and moving speed and mix them together to train a more general model. In addition, we are also thinking about training using different set of features as well as the principal components by finding good classifier for these components.

Acknowledgement We thank the people in the Meyer lab for providing dataset and knowledgebase about live-cell tracking and also Professor Andrew Ng and the teaching staff for their guidance and hard working.

References

[1] **Use of Texture Classification To Quantify Stem Cell Differentiation During Time Lapse Imaging** Sunil Pai, Nathan Loewke,Thomas M. Baer. 2013 cs229 final project

[2] **Good Cell, Bad Cell: Classification of Segmented Images for Suitable Quantification and Analysis** Derek Macklin, Haisam Islam, Jonathan Lu. 2012 cs229 final project

[3] **CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biology 7:R100. PMID: 17076895** Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, Guertin DA, Chang JH, Lindquist RA, Moffat J, Golland P, Sabatini DM (2006)

[4] **"High-throughput, single-cell NFκb dynamics," Current Opinion in Genetics and Development,** T. K. Lee and M. W. Covert, vol. 20, no. 6, pp. 677 – 683, 2010, genetics of system biology

[5] **"Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition."** Liu, Chengjun, and Harry Wechsler. Image processing, IEEE Transactions on 11.4 (2002): 467-476.