1. Identify the problem

This image set is of a Transfluor assay where an orphan GPCR is stably integrated into the b-arrestin GFP expressing U2OS cell line. After one hour incubation with a compound the cells were fixed with (formaldehyde).

1.1 Objective

Since the labels in the data are discrete, the predication falls into two categories, (i.e. Postive cell or Negative Cell). In machine learning this is a classification problem.

> Thus, the goal is to classify whether cell is positive or negative and predict the accuracy of the model with different kernel.

1.2 Identify data source

We used image set BBBC016v1 (https://data.broadinstitute.org/bbbc/BBBC016/) provided by Ilya Ravkin, available from the Broad Bioimage Benchmark Collection [Ljosa et al., Nature Methods, 2012]. We used a part of this dataset taking account the wells O06, O07, O16 and O22.
Features were generated by CellProfiler and classes were annotated manually. The dataset contains **40 samples of positives and negatives cells**.

- The first two columns in the dataset contain the *labels* (Positives, Negatives), and the *dose* put for each well.
- The columns 4 - 5 contain the well position on the plate, the unique ID of the image and the number of object respectively.
- The columns 6 - 155 contain *features* that have been computed from images of the cell nuclei and cell cytoplasm which can be used to build a model to predict the phenotype of the cells.

1.3 Load libraries

In [1]:
```
# a) Importing libraries.
import numpy as np

# b) Replace the occurences of ... to import the pandas library with th
        as pd
```

Out[1]:
```
Ellipsis
```

1.4 Load dataset

In [ ]:
```
# c) Importing the dataset.

# Replace the occurences of ... to indicate a string path to your file
# Example A local file could be: "C://localhost/path/to/table.csv".
file = ...

# Replace the occurences of ... to load the dataset.csv file (path assi
# using the Pandas read_csv function.
dataset =
```

1.5 Inspecting the data

The first step is to visually inspect the dataset. There are multiple ways to achieve this:

- The easiest being to request the first few records using the data.head() method. By default, “data.head()” returns the first 5 rows.
- Alternatively, one can also use “data.tail()” to return the five rows of the data.
- For both head and tail methods, there is an option to specify the number of rows by including the required number in between the parentheses when calling either method.

In [ ]:
```
# d) print the dataset.
# After reading the notes above try to display the ten rows of the data
```

You can check the number of cases, as well as the number of fields, using the shape method.

In [ ]:
```
# e) Replace the occurence ... by the shape method to get the number of
```

In the result displayed, you should be have 40 records with 155 columns.
The “info()” method provides a concise summary of the data; from the output, it provides the type of data in each column, the number of non-null values in each column, and how much memory the data frame is using.

In [ ]:
```
# f) Replace the occurence ... by the "info(verbose = True)" to get the
```

From the results above Label, Dose and Well are *categorical variables* and rest are floating or integer values.

2. Pre-processing the dataset

Data preprocessing is a crucial step for any data analysis problem. It is often a very good idea to prepare your data in such way to best expose the structure of the problem to the machine learning algorithms that you intend to use. This involves a number of activities such as:

- Assigning numerical values to categorical data.
- Handling missing values.
- Divide data into attributes and labels sets.
- Divide data into traininig and test sets.

2.1 Objective

> The goal here is encoding the class Label in an array y and get attributes in an array X. Then split the data into a *training set* and *testing set*.

2.2 split features and labels into new sets and encoding the labels into integers

In [ ]:
```
# a) Use the method drop(columns=['feature_1', "feature_2", ...])
# to drop unnecessary features 'Label', 'Dose', 'Well', 'ImageNumber',
# and affect the output to X variable
... = ...

# b) For select a feature use data['feature']. Replace the occurence ..
y = ...

#transform the class labels from their original string representation (
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y = le.fit_transform(y)

# c) Replace the occurence ... to display 5 rows of the X dataset with
print(     y)
```

After encoding the Label in an array y, the phenotype cell are now represented as class 1(i.e positive cell) and as class 0 (i.e negative cell), respectively.

2.3 Split data into training and test sets

The simplest method to evaluate the performance of a machine learning algorithm is to use different training and testing datasets. Here

- Split the available data into a training set and a testing set. (70% training, 30% test)

In [ ]:
```
# a) Use the train_test_split method from the model.selection of scikit
# Import the train_test_split method
# This method takes three parameters train_test_split(param1, param2, p
# The first parameter will be the X dataset, the second parameter will
# and the third parameter will be the size of the testing set ie (70% =
# Fill in the occurences ...

from sklearn.model_selection import ...

X_train, X_test, y_train, y_test = train_test_split(..., ..., test_size

# b) Fill in the occurences ... to display the number of rows
# and numbers of columns for the two variables (X_train and X_test)

```

3. Predictive model using Support Vector Machine (SVM)

Kernelized support vector machines are powerful models and perform well on a variety of datasets.

1. SVMs allow for *complex decision boundaries*, even if the data has only a few features.
2. They work well on *low-dimensional* and *high-dimensional* data (i.e., few and many features), but don't scale very well with the number of samples.
3. SVMs requires careful *preprocessing of the data* and *tuning* of the parameters. This is why, these days, most people instead use tree-based models such as *random forests* or *gradient boosting* (which require little or no preprocessing) in many applications.
4. SVM models are *hard to inspect*; it can be difficult to understand why a particular prediction was made, and it might be tricky to explain the model to a non-expert.

3.1 Objective

> The goal is to fit a linear model to the data using *SVC library* from the svm of scikit-learn.

In [ ]:
```python
# Follow the instructions and fill in the occurences ...

# a) Create an SVM classifier and train it on 70% of the data set.
# use the support vector classifier class, which is written as SVC in t
# This class takes one parameter, which is the kernel type 'linear'.
# We will see non-linear kernels in the next section.

from sklearn.svm import ...
svclassifier = ...(kernel= ... )

# b) The fit method of SVC class is called to train the algorithm on th
# which is passed as a parameter to the fit method.

   (          )
```

From the above result you will see the important parameters in kernel SVMs:

- Regularization parameter C.
- The choice of the kernel (linear, radial basis function(RBF) or polynomial).
- Kernel-specific parameters.

3.2 Making predictions

To make predictions, the predict method of the SVC class is used.

In [ ]:
```python
# a) Fill in the occurences to make prediction on the testing set (X_te
y_pred =    (    )
```

4. Model Accuracy

4.1 Confusion matrix

Confusion matrix measure is the most commonly used metric for classification tasks. Scikit-Learn's metrics library contains the confusion_matrix method, which can be readily used to find out the values for these important metrics.
the confusion matrix that essentially is a *two-dimensional table* where the classifier model is on one axis (vertical), and ground truth is on the other (horizontal) axis, as shown below. Either of these axes can take two values (as depicted)

Model says "+"	Model says "-"	Ground truth
`True positive`	`False negative`	Actual: "+"
`False positive`	`True negative`	Actual: "-"

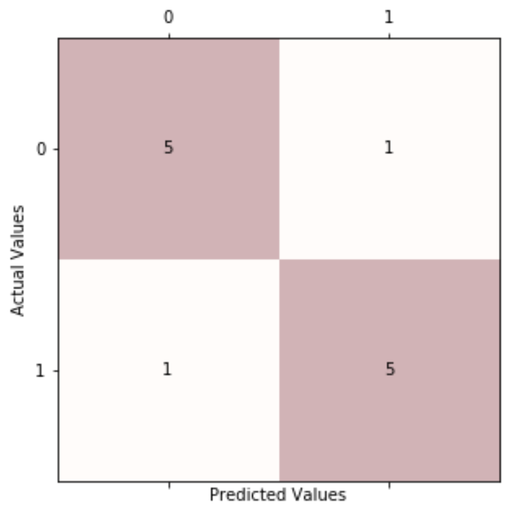
> The goal is to create a confusion matrix in order to know the accuracy of the model.

In [34]:
```
# a) fill in the occurences to import confusion_matrix method from skle

from ... import ...
cm = confusion_matrix(y_test, y_pred)

import matplotlib.pyplot as plt
from IPython.display import Image, display

fig, ax = plt.subplots(figsize=(5, 5))
ax.matshow(cm, cmap=plt.cm.Reds, alpha=0.3)
for i in range(cm.shape[0]):
     for j in range(cm.shape[1]):
         ax.text(x=j, y=i,
                s=cm[i, j],
                va='center', ha='center')
plt.xlabel('Predicted Values', )
plt.ylabel('Actual Values')
plt.show()
```



Observation

There are two possible predicted classes: "1" (i.e positive cell) and "0" (i.e negative cell).

```
a) How many positives cells are true ?
b) How many negatives cells are true ?
c) How many positives cells are false ?
d) How many negatives cells are false ?
```

4.2 Rates as computed from the confusion matrix

a) **Accuracy**: Overall, how often is the classifier correct? Calculate the accuracy of the model in percentage
$Accuracy = (\frac{TP+TN}{TP+TN+FP+FN}) * 100$
$Accuracy = ...\%$

b) **Misclassification Rate**: Overall, how often is it wrong? Calculate the "error rate" of the model in percentage $ErrorRate = (\frac{FP+FN}{TP+TN+FP+FN}) * 100$
$ErrorRate = ...\%$

5 Comparison between different kernel for no linear classification

We will implement polynomial, Gaussian, and sigmoid kernels to see which one works better for our problem.

5.1 Polynomial kernel

In [42]:
```python
# a) Replace the 'linear' kernel parameter from the SVC class by 'poly'
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

Out[42]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [ ]:
```python
# b) Make predictions on the testing set
```

In [ ]:
```python
# c) create a confusion matrix to evaluate the accuracy
```

d) calculate the accuracy and the error rate of the polynomial model.
Accuracy = ...%
Eror rate = ... %

e) Now let's repeat the same steps for *Gaussian* (kernel = 'rbf') and *sigmoid* kernels (kernel = 'sigmoid').
Which kernel work better for our problem ? ....

Observation

From the result above If we compare the performance of the different types of kernels we can clearly see that the sigmoid kernel performs the worst.

6. Make prediction on new dataset (unlabel)

the goal is to predict unlabel dataset with the best classifier run above

In [65]:
```python
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

Out[65]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [ ]:
```python
# b) importing the new dataset (file: unlabel_dataset.csv)
new_data =
```

In [ ]:
```python
# c) print the 8 rows of new_data
```

In [ ]:
```
# d) drop unrelevant features like 'Dose', 'Well', 'ImageNumber', 'Obje
new_data =
```

In [ ]:
```
# e) prediction on the new_data
new_pred =
```