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 prediction 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 panda
        as pd
```

Out[1]: Ellipsis

In [2]:
```
import numpy as np
import pandas as pd
```

1.4 Load dataset

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

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

# Replace the occurences of ... to load the dataset.cs
# using the Pandas read_csv function.
dataset =
```

In [3]:
```
file = "/home/cedric/Documents/ML_FormationBC/exercise
dataset = pd.read_csv(file)
```

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 ter

```

In [4]:
```
dataset.head(n=10)
```

Out[4]:

	Label	Dose	Well	ImageNumber	ObjectNumber	Cytoplasm_AreaShape_Are
0	positive	30uM	O06	16	1	63
1	positive	30uM	O06	16	2	59
2	positive	30uM	O06	16	3	114
3	positive	30uM	O06	16	4	85
4	positive	30uM	O06	16	5	57
5	positive	30uM	O06	16	6	86
6	positive	30uM	O06	16	7	116
7	positive	30uM	O06	16	8	129
8	positive	30uM	O06	16	9	239
9	positive	30uM	O06	16	10	95

10 rows × 155 columns

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
```

In [6]:
```
dataset.shape
```

Out[6]:
```
(40, 155)
```

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 =
```

In [17]:
```
dataset.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Columns: 155 entries, Label to Nuclei_Location_MaxIntensi
ty_Z_GFP
dtypes: float64(126), int64(26), object(3)
memory usage: 48.5+ KB
```

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', "featur
# to drop unnecessary features 'Label', 'Dose', 'Well'
# and affect the output to X variable
... = ...

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

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

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

# c) Replace the occurence ... to display 5 rows of th
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.

In [23]:
```
X = dataset.drop(columns=['Label', 'Dose', 'Well', 'Im
y = dataset['Label']

#transform the class labels from their original string
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

print(X.head(), y)
```

```
   Cytoplasm_AreaShape_Area  Cytoplasm_AreaShape_Center_X
\
0                       637                           511
1                       595                           130
2                      1148                           364
3                       850                            46
4                       572                            95

   Cytoplasm_AreaShape_Center_Y  Cytoplasm_AreaShape_Cent
er_Z  \
0                            25
1
1                             9
1
2                            45
1
3                            32
1
4                            30
1

   Cytoplasm_AreaShape_Compactness  Cytoplasm_AreaShape_E
ccentricity  \
0                         1.879897
0.577598
1                         2.651236
0.603286
2                         1.868392
0.450451
3                         2.217524
0.786959
4                         2.119470
0.851912

   Cytoplasm_AreaShape_EulerNumber  Cytoplasm_AreaShape_E
xtent  \
0                                0                    0.3
89841
1                                0                    0.3
30556
2                                0                    0.2
94813
3                                0                    0.3
```

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.se
# Import the train_test_split method
# This method takes three parameters train_test_split(
# The first parameter will be the X dataset, the secor
# and the third parameter will be the size of the test
# Fill in the occurences ...

from sklearn.model_selection import ...

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

# b) Fill in the occurences ... to display the number
# and numbers of columns for the two variables (X_trai
```

In [24]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
X_train.shape, X_test.shape
```

Out[24]:
```
((28, 150), (12, 150))
```

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 [ ]:

```
# Follow the instructions and fill in the occurences .

# a) Create an SVM classifier and train it on 70% of t
# use the support vector classifier class, which is wr
# This class takes one parameter, which is the kernel
# 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 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.

In [25]:
```
from sklearn.svm import SVC

svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

Out[25]:
```
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, shri
nking=True,
  tol=0.001, verbose=False)
```

3.2 Making predictions

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

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

In [29]:
```
y_pred = svclassifier.predict(X_test)
```

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

4. Model Accuracy

4.1 Confusion matrix

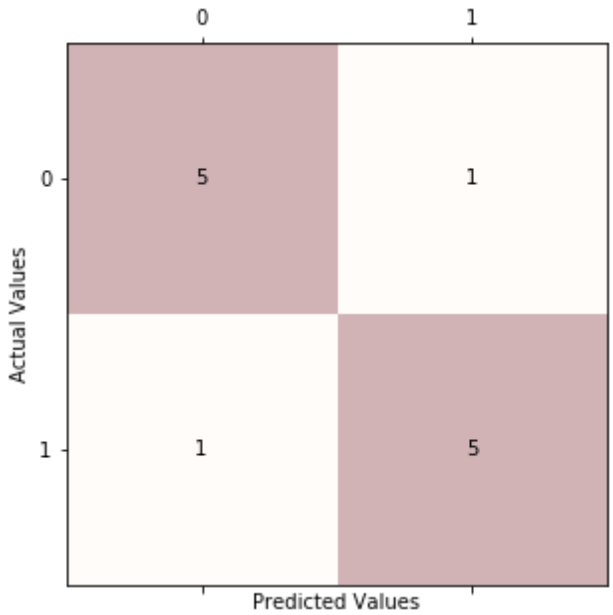
Confusion matrix measure is the most commonly used metric for classification tasks. Scikit-Learn's metrics library contains the confusion_matrix method, which can be readily used to find out the values for these important metrics. the confusion matrix that essentially is a *two-dimensional table* where 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.

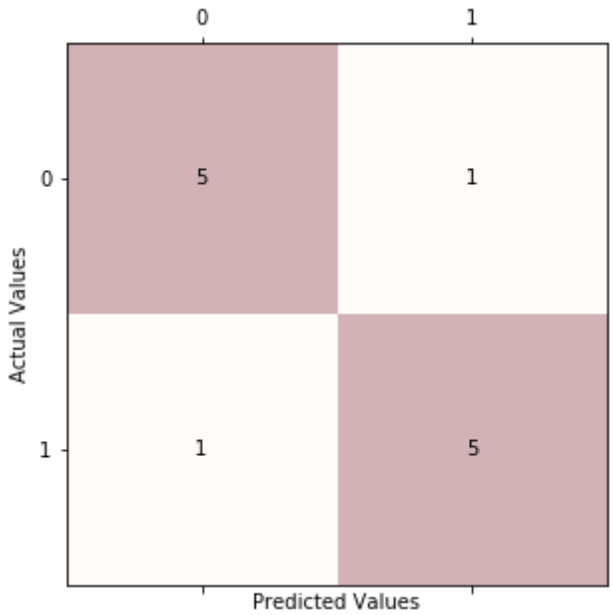
In [34]:

```
# a) fill in the occurences to import confusion_matrix

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()
```



In [32]:

```
from sklearn.metrics import confusion_matrix
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 $ErrorRate = (\frac{FP+FN}{TP+TN+FP+FN}) * 100$

$ErrorRate = ...\%$

In [40]:
```
print("Accuracy", (5+5)/12*100)
print("Error rate", (1+1)/12*100)
```

```
Accuracy 83.33333333333334
Error rate 16.666666666666664
```

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]:
```
# a) Replace the 'linear' kernel parameter from the SV
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, shri
nking=True,
  tol=0.001, verbose=False)
```

In [43]:
```
svclassifier = SVC(kernel='poly')
svclassifier.fit(X_train, y_train)
```

Out[43]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto',
kernel='poly',
  max_iter=-1, probability=False, random_state=None, shri
nking=True,
  tol=0.001, verbose=False)
```

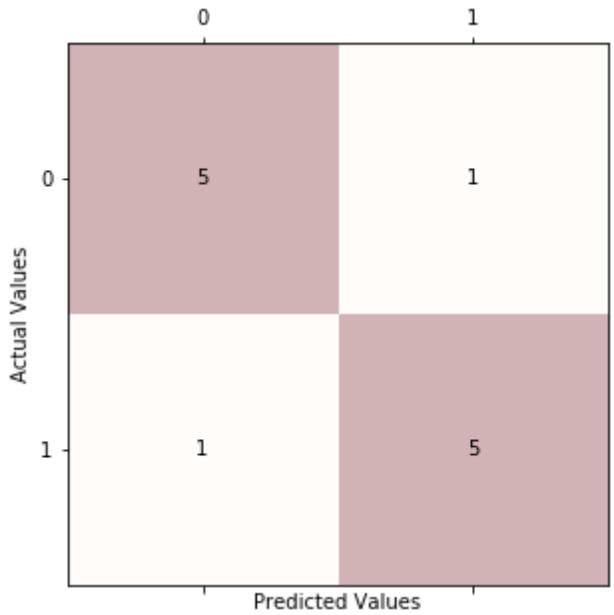
In [ ]:
```
# b) Make predictions on the testing set
```

In [45]:
```
y_pred = svclassifier.predict(X_test)
print(y_pred)
```

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

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

In [46]:
```
from sklearn.metrics import confusion_matrix
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()
```

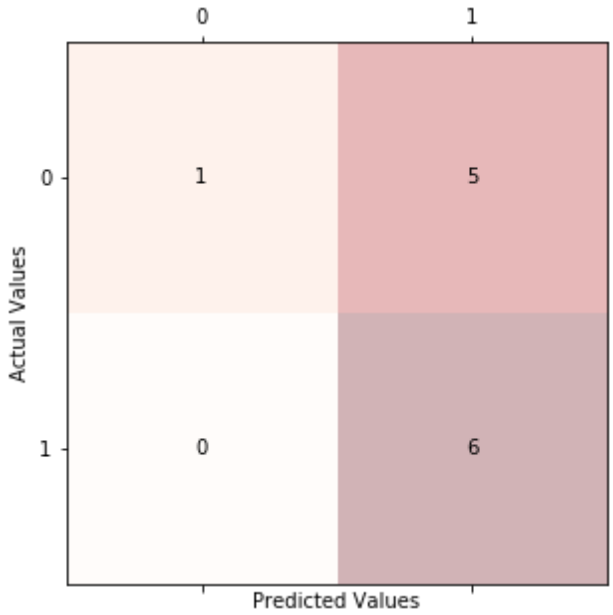


d) calculate the accuracy and the error rate of the polynomial model. Accuracy = ...%
Eror rate = ... %

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

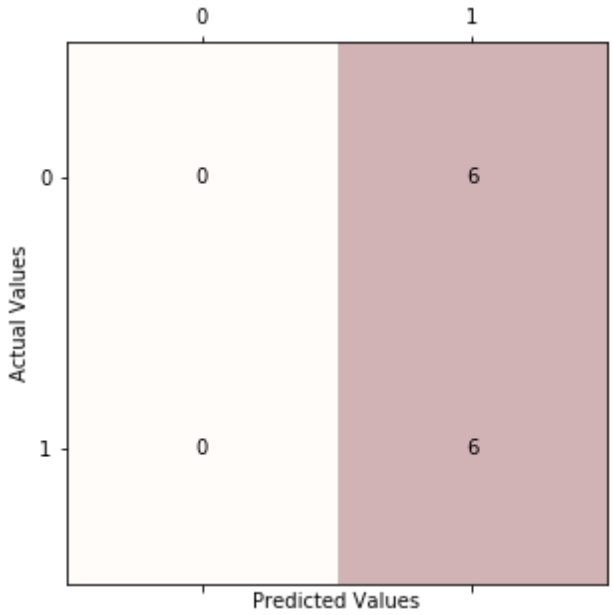
In [51]:

```
# Gauusian kernel

svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

from sklearn.metrics import confusion_matrix
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()
```



In [53]:

```
# sigmoid kernel

svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

from sklearn.metrics import confusion_matrix
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

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]:
```
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, shri
nking=True,
  tol=0.001, verbose=False)
```

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

In [66]:
```
new_data = pd.read_csv('/home/cedric/Documents/ML_Form
```

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

```

In [67]:
```
new_data.head(n=8)
```

Out[67]:

	Dose	Well	ImageNumber	ObjectNumber	Cytoplasm_AreaShape_Area	Cytopl
0	30uM	O06	16	11	2376	
1	30uM	O06	16	12	3574	
2	30uM	O06	16	13	3753	
3	30uM	O06	16	14	2036	
4	30uM	O06	16	15	1484	
5	30uM	O06	16	16	4843	
6	30uM	O06	16	17	1575	
7	30uM	O06	16	18	5457	

8 rows × 154 columns

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

In [68]:
```
new_data = new_data.drop(columns=['Dose', 'Well', 'Ima
```

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

In [69]:
```
new_pred = svclassifier.predict(new_data)
print(new_pred)
```

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