ENGR 301
Engineering Applications of Data Science

Laboratory 5
Topic: Biomedical Engineering; Subtopic: Cancer Prediction

Introduction: Biochips are an area of technology which can be used for a variety of applications, such as detecting the presence of certain compounds in a biological environment. One common biochip is a DNA microarray. Here, thousands of strands of different DNA segments are put on a surface; if a gene is active within a cell, then it will bind to the complementary strand on the surface (Figure 1). Using this technology, it is possible to analyze the expression of thousands of genes at the same time, allowing for screening patients for genetic disorders or cancer. Depending on the genes expressed, one can determine whether cells are healthy or cancerous. Moreover, the type of cancer can also be determined from finding what genes are expressed.

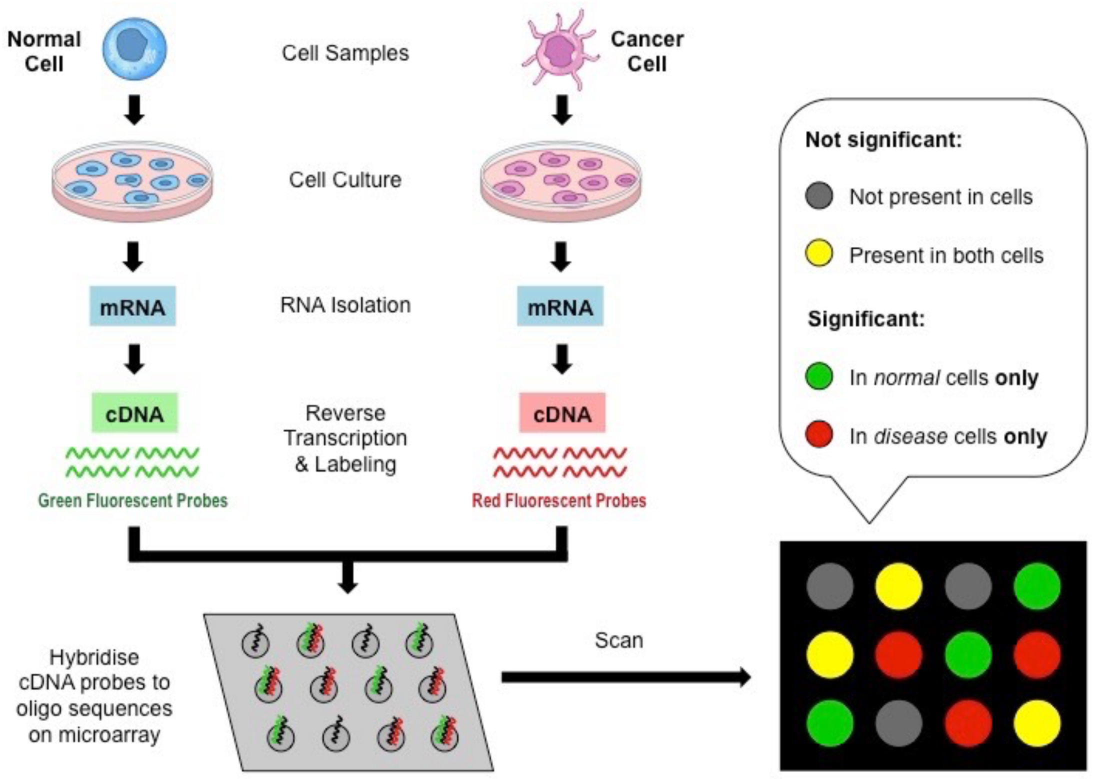

Figure 1. Schematic workflow of a DNA microarray. (Image curtesy microbenotes.com/dna-microarray/)

Discipline Specific Information: For this laboratory, you will receive data from DNA microarrays performed on 72 patients. Each patient had one of two forms of leukemia: either acute myeloid leukemia (AML) or acute lymphocytic leukemia (ALL). The DNA microarrays performed tested for the presence and expression of 7,129 different genes (these are the features). Your task for the lab is to predict what form of cancer (output) is present in a patient based on what genes are expressed (i.e., a classification problem).

This data comes from a study in Science: T. R. Golub *et al.*, Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, Science 286 531 (1999)

Tasks

Part 1 – Exploratory Data Analysis and Baseline Models

1. Import the appropriate packages (numpy, pandas, matplotlib, seaborn) into a Jupyter notebook. The data for this part is provided on Canvas in a CSV file called:

gene_expression_patients.csv

2. What is the distribution of patients with ALL and AML? Is this output data balanced or unbalanced? Why?

3. Find the maximum, minimum, and mean of the first few rows, by, for example, calling "df.max(axis=1)". If the maximum, minimum, and mean for a given patient are very far apart, it is likely that the data are not on the same scale.

Is that the case here? Is this a problem? Why or why not?

4. To run the classification models, we need to encode the output as numbers (0 or 1). One way to do that in Sklearn is by using LabelEncoder. Import LabelEncoder from the "sklearn.preprocessing" package.

This work similar to a model; create a new object called "encoder" and set it equal to a new instance of "LabelEncoder()".

Next, pass the "cancer type" column from your data frame into the "encoder.fit_transform()" function, and set that equal to the "cancer type" column so it overwrites it.

Now each cancer type is encoded as either a 0 or 1.

5a. First, let's build baseline classification models. Set "x" to include all of the features except patient number and cancer type and set "y" to the cancer type.

5b. Split your data into a training set and test set, with the test set containing 30% of the data.

5c. First, train a logistic regression model and print the confusion matrix and accuracy.

5d. Next, do the same for a Naïve Bayes model. Remember, the process is the exact same except we are importing and using the "GaussianNB" model from the "sklearn.naive_bayes" package.

5e. What is the confusion matrix and accuracy telling you?

Part 2 – Scaling and Feature Reduction

With over 7,000 features as our input and only 79 measurements, it is likely that we are vastly overfitting our models, even though the accuracy is very good. In fact, the accuracy is likely *too* *good*. Therefore, even though our test set has great accuracy, it is likely that this model will not generalize to large numbers of new patients.

To combat this, we will implement feature scaling and principal component analysis.

1. First, let's scale the data. Import "StandardScaler" from the "sklearn.preprocessing" package. Set a variable "scaler" equal to an instance of the object "StandardScaler()".

Now create a new x array that contains scaled data by calling "fit.transform" on the "scaler" object and passing it the old x data.

What is the mean and standard deviation of your new scaled data?

2. Now let's do feature reduction using principal component analysis. Import "PCA" from the "sklearn.decomposition" package.

Remember that we can create a new variable "pca" and set it equal to "PCA()" and pass it "n_components = k", where k is the number of principal components we want to use. We want to determine the lowest number of features that will explain approximately 90% of the variance.

To do that, increase the value of n_components by 5 or 10 at a time and monitor the total sum of the explained variance (pca.explained_variance_ratio_.cumsum()).

(On the other hand, you could write a while loop in Python in the cell and loop over all values of n_components until the total explained variance is more than 0.9. This isn't necessary, but you can try it if you are feeling brave).

How many principal components do you need to explain 90% of the variance?

Once you determine that, create a new x array and set it equal to "pca.fit_transform()" calling your previously scaled x data.

We have now reduced our 7,000 features to less than the number of measurements.

3. Feed this data into a new logistic regression model and a new Naïve Bayes model. Don't forget to create a training and test set using your new scaled and reduced data, not the old data from Part 1. Use 30% of the data as the test set.

Determine the classification matrices and accuracies. How did they change from the models in Part 1? Do you think these models are more generalizable to new patients? Why or why not?

Part 3 – Class Balancing

In this part, we will examine the effect of balancing the cancer classes using the Synthetic Minority Oversampling Technique (SMOTE).

1. Import “SMOTE” from the “imblearn.over_sampling” package. If you are having trouble installing imblearn, please let me know.

As with PCA, create a new “smote” variable and assign it the “SMOTE()” object.

Create new x and y arrays by setting it equal to “smote.fit_resample()” passing the scaled PCA x data and the old y data.

What is the distribution of the outputs in the new y array?

2. Feed this data into a new logistic regression model and a new Naïve Bayes model. Don’t forget to create a training and test set using your new scaled and reduced data, not the old data from Part 1. Use 30% of the data as the test set.

Determine the classification matrices and accuracies. How did they change from the models in Parts 1 and 2? Do you think these models are more generalizable to new patients? Why or why not?