# cs109a_hw5_109_submit

October 24, 2018

## 1 CS109A Introduction to Data Science:

### 1.1 Homework 5: Logistic Regression, High Dimensionality and PCA

**Harvard University Fall 2018 Instructors**: Pavlos Protopapas, Kevin Rader **Submitted by**: Erin Williams, Avriel Epps

In [1]:
```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/maste
HTML(styles)
```

Out[1]:
```
<IPython.core.display.HTML object>
```

#### 1.1.1 INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas https://canvas.harvard.edu/courses/42693/pages/homework-policies-and-submission-instructions.
- Restart the kernel and run the whole notebook again before you submit.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.
- As much as possible, try and stick to the hints and functions we import at the top of the homework, as those are the ideas and tools the class supports and is aiming to teach. And if a problem specifies a particular library you're required to use that library, and possibly others from the import list.

Names of people you have worked with goes here:
Avriel Epps, Erin Williams

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

import statsmodels.api as sm
from statsmodels.api import OLS

from sklearn.decomposition import PCA
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

import math
from scipy.special import gamma

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

from IPython.display import display
```

Cancer Classification from Gene Expressions

In this problem, we will build a classification model to distinguish between two related classes of cancer, acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML), using gene expression measurements. The data set is provided in the file `data/dataset_hw5_1.csv`. Each row in this file corresponds to a tumor tissue sample from a patient with one of the two forms of Leukemia. The first column contains the cancer type, with 0 indicating the ALL class and 1 indicating the AML class. Columns 2-7130 contain expression levels of 7129 genes recorded from each tissue sample.

In the following questions, we will use linear and logistic regression to build classification models for this data set. We will also use Principal Components Analysis (PCA) to reduce its dimensions.

Question 1 [25 pts]: Data Exploration

First step is to split the observations into an approximate 50-50 train-test split. Below is some code to do this for you (we want to make sure everyone has the same splits).

**1.1** Take a peek at your training set: you should notice the severe differences in the measurements from one gene to the next (some are negative, some hover around zero, and some are well into the thousands). To account for these differences in scale and variability, normalize each predictor to vary between 0 and 1.

**1.2** Notice that the resulting training set contains more predictors than observations. Do you foresee a problem in fitting a classification model to such a data set? Explain in 3 or fewer sentences.

**1.3** Lets explore a few of the genes and see how well they discriminate between cancer classes. Create a single figure with four subplots arranged in a 2x2 grid. Consider the following four genes: `D29963_at`, `M23161_at`, `hum_alu_at`, and `AFFX-PheX-5_at`. For each gene overlay two histograms of the gene expression values on one of the subplots, one histogram for each cancer type. Does it

appear that any of these genes discriminate between the two classes well? How are you able to tell?

**1.4** Since our data has dimensions that are not easily visualizable, we want to reduce the dimensionality of the data to make it easier to visualize. Using PCA, find the top two principal components for the gene expression data. Generate a scatter plot using these principal components, highlighting the two cancer types in different colors and different markers ('x' vs 'o', for example). How well do the top two principal components discriminate between the two classes? How much of the variance within the predictor set do these two principal components explain?

**1.5** Plot the cumulative variance explained in the feature set as a function of the number of PCA-components (up to the first 50 components). Do you feel 2 components is enough, and if not, how many components would you choose to consider? Justify your choice in 3 or fewer sentences. Finally, determine how many components are needed to explain at least 90% of the variability in the feature set.

**Answers:** First step is to split the observations into an approximate 50-50 train-test split. Below is some code to do this for you (we want to make sure everyone has the same splits).

```
In [3]: np.random.seed(9002)
        df = pd.read_csv('data/dataset_hw5_1.csv')
        msk = np.random.rand(len(df)) < 0.5
        data_train = df[msk]
        data_test = df[~msk]
```

**1.1:** Take a peek at your training set: you should notice the severe differences in the measurements from one gene to the next (some are negative, some hover around zero, and some are well into the thousands). To account for these differences in scale and variability, normalize each predictor to vary between 0 and 1.

```
In [4]: # your code here
        display(data_train.head())
        #data_train.describe()
```

| | Cancer_type | AFFX-BioB-5_at | AFFX-BioB-M_at | AFFX-BioB-3_at | \ |
|---|---|---|---|---|---|
| 0 | 0 | -214 | -153 | -58 | |
| 2 | 0 | -106 | -125 | -76 | |
| 5 | 0 | -67 | -93 | 84 | |
| 9 | 0 | -476 | -213 | -18 | |
| 10 | 0 | -81 | -150 | -119 | |

| | AFFX-BioC-5_at | AFFX-BioC-3_at | AFFX-BioDn-5_at | AFFX-BioDn-3_at | \ |
|---|---|---|---|---|---|
| 0 | 88 | -295 | -558 | 199 | |
| 2 | 168 | -230 | -284 | 4 | |
| 5 | 25 | -179 | -323 | -135 | |
| 9 | 301 | -403 | -394 | -42 | |
| 10 | 78 | -152 | -340 | -36 | |

| | AFFX-CreX-5_at | AFFX-CreX-3_at | ... | U48730_at | U58516_at | \ |
|---|---|---|---|---|---|---|
| 0 | -176 | 252 | ... | 185 | 511 | |

```
2             -122             70     ...             156        649
5             -127             -2     ...              48        224
9             -144             98     ...             241       1214
10            -141             96     ...             186        573

    U73738_at  X06956_at  X16699_at  X83863_at  Z17240_at  L49218_f_at  \
0        -125        389        -37        793        329           36
2          57        504        -26        250        314           14
5          60        194        -10        291         41            8
9         127        255         50       1701       1108           61
10        -57        694        -19        636        205           17

    M71243_f_at  Z78285_f_at
0           191          -37
2            56          -25
5            -2          -80
9           525          -83
10          127          -13

[5 rows x 7130 columns]
```

```
In [5]: def scale_data(df):
            scaler = MinMaxScaler()
            df_fit = scaler.fit(data_train)
            df_transform = scaler.transform(df)
            scaled_df = pd.DataFrame(df_transform, columns = df.columns.values)
            return scaled_df
```

```
In [6]: data_train_norm = scale_data(data_train)
        data_test_norm = scale_data(data_test)
        display(data_train_norm.describe())
```

```
       Cancer_type  AFFX-BioB-5_at  AFFX-BioB-M_at  AFFX-BioB-3_at  \
count     40.00000       40.000000       40.000000       40.000000
mean       0.37500        0.640347        0.719472        0.369477
std        0.49029        0.182889        0.186767        0.237206
min        0.00000        0.000000        0.000000        0.000000
25%        0.00000        0.596530        0.631115        0.201744
50%        0.00000        0.653025        0.745597        0.323256
75%        1.00000        0.731762        0.844423        0.500000
max        1.00000        1.000000        1.000000        1.000000

       AFFX-BioC-5_at  AFFX-BioC-3_at  AFFX-BioDn-5_at  AFFX-BioDn-3_at  \
count       40.000000       40.000000        40.000000        40.000000
mean         0.512253        0.529472         0.550653         0.654253
std          0.243956        0.231075         0.214332         0.216589
min          0.000000        0.000000         0.000000         0.000000
```

```
25%          0.325824        0.386853        0.408101        0.547153
50%          0.553846        0.584052        0.542683        0.683986
75%          0.720330        0.683728        0.713850        0.753381
max          1.000000        1.000000        1.000000        1.000000

       AFFX-CreX-5_at  AFFX-CreX-3_at     ...       U48730_at  U58516_at  \
count       40.000000       40.000000     ...       40.000000  40.000000
mean         0.581803        0.535615     ...        0.385916   0.351822
std          0.228836        0.231285     ...        0.234882   0.195379
min          0.000000        0.000000     ...        0.000000   0.000000
25%          0.484127        0.342807     ...        0.247305   0.213235
50%          0.634921        0.573086     ...        0.370620   0.337596
75%          0.739796        0.725058     ...        0.498652   0.407289
max          1.000000        1.000000     ...        1.000000   1.000000

       U73738_at  X06956_at  X16699_at  X83863_at  Z17240_at  L49218_f_at  \
count  40.000000  40.000000  40.000000  40.000000  40.000000    40.000000
mean    0.656206   0.173726   0.698519   0.334762   0.283348     0.615608
std     0.217202   0.154501   0.201592   0.204001   0.171412     0.200119
min     0.000000   0.000000   0.000000   0.000000   0.000000     0.000000
25%     0.556792   0.104175   0.602778   0.210597   0.200870     0.506906
50%     0.715457   0.143865   0.737037   0.302760   0.281739     0.607735
75%     0.813232   0.195904   0.856481   0.434328   0.342391     0.708564
max     1.000000   1.000000   1.000000   1.000000   1.000000     1.000000

       M71243_f_at  Z78285_f_at
count    40.000000    40.000000
mean      0.162620     0.404570
std       0.202527     0.199781
min       0.000000     0.000000
25%       0.057054     0.243280
50%       0.085271     0.416667
75%       0.176434     0.512097
max       1.000000     1.000000

[8 rows x 7130 columns]
```

**1.2:** Notice that the results training set contains significantly more predictors than observations. Do you foresee a problem in fitting a classification model to such a data set?

**Answer**: When we have more predictors than observations, there is no unique solution using a standard linear regression model. We'll need to use advanced machine learning techniques to solve this.

**1.3:** Lets explore a few of the genes and see how well they discriminate between cancer classes. Create a single figure with four subplots arranged in a 2x2 grid. Consider the following four genes: `D29963_at`, `M23161_at`, `hum_alu_at`, and `AFFX-PheX-5_at`. For each gene overlay two histograms of the gene expression values on one of the subplots, one histogram for each cancer type. Does it appear that any of these genes discriminate between the two classes well? How are you able to

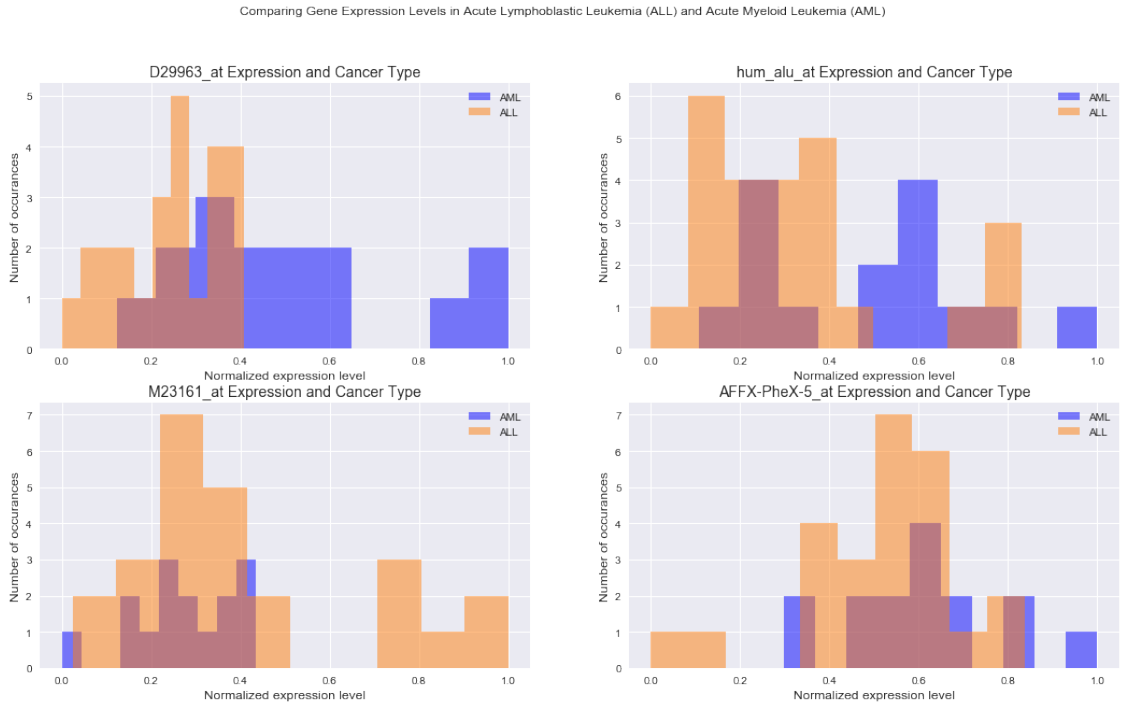
tell?

In [7]:
```
#data_train_norm['Cancer_type' == 0.0]
AML_cancer = data_train_norm[data_train_norm.Cancer_type==1.0]
ALL_cancer = data_train_norm[data_train_norm.Cancer_type==0.0]
```

In [8]:
```
# def group_by_cancer_type(df):
#     Cancer_type = [0.0, 1.0]
#     gene_list = ['D29963_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5_at']
#     for i in Cancer_type:
#         true_cancer = df[df.Cancer_type==1.0]
#         false_caner = df[df.Cancer_type==1.0]
```

In [9]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18,10))

axes[0,0].hist(AML_cancer['D29963_at'], color = 'b', alpha = 0.5, label = "AML" )
axes[0,0].hist(ALL_cancer['D29963_at'], color = 'tab:orange', alpha = 0.5, label = "ALL
axes[0,0].set_xlabel('Normalized expression level', fontsize =12)
axes[0,0].set_ylabel('Number of occurances', fontsize = 12)
axes[0,0].set_title("D29963_at Expression and Cancer Type", fontsize = 14)
axes[0,0].legend()

axes[1,0].hist(AML_cancer['M23161_at'], color = 'b', alpha = 0.5, label = "AML" )
axes[1,0].hist(ALL_cancer['M23161_at'], color = 'tab:orange', alpha = 0.5, label = "ALL
axes[1,0].set_xlabel('Normalized expression level', fontsize =12)
axes[1,0].set_ylabel('Number of occurances', fontsize = 12)
axes[1,0].set_title("M23161_at Expression and Cancer Type", fontsize = 14)
axes[1,0].legend()

axes[0,1].hist(AML_cancer['hum_alu_at'], color = 'b', alpha = 0.5, label = "AML" )
axes[0,1].hist(ALL_cancer['hum_alu_at'], color = 'tab:orange', alpha = 0.5, label = "AL
axes[0,1].set_xlabel('Normalized expression level', fontsize =12)
axes[0,1].set_ylabel('Number of occurances', fontsize = 12)
axes[0,1].set_title("hum_alu_at Expression and Cancer Type", fontsize = 14)
axes[0,1].legend()

axes[1,1].hist(AML_cancer['AFFX-PheX-5_at'], color = 'b', alpha = 0.5, label = "AML" )
axes[1,1].hist(ALL_cancer['AFFX-PheX-5_at'], color = 'tab:orange', alpha = 0.5, label =
axes[1,1].set_xlabel('Normalized expression level', fontsize =12)
axes[1,1].set_ylabel('Number of occurances', fontsize = 12)
axes[1,1].set_title("AFFX-PheX-5_at Expression and Cancer Type", fontsize = 14)
axes[1,1].legend()

fig.suptitle("Comparing Gene Expression Levels in Acute Lymphoblastic Leukemia (ALL) an
```

Out[9]:
```
Text(0.5,0.98,'Comparing Gene Expression Levels in Acute Lymphoblastic Leukemia (ALL) a
```




**Answer**: Some of the genes appear to discriminate better than others, as evidenced by the size and concentration of our histogram. For example, we see lower levels of expression of `D29963_at`, `M23161_at`, and `hum_alu_at` for ALL patients, whereas their expression in AML patients does not appear to follow a specific pattern. For the gene `AFFX-PheX-5_at`, both AML and ALL patients seem to follow a normal curve. We can clearly see that one gene alone is not a good predictor of cancer type.

**1.4:** Since our data has dimensions that are not easily visualizable, we want to reduce the dimensionality of the data to make it easier to visualize. Using PCA, find the top two principal components for the gene expression data. Generate a scatter plot using these principal components, highlighting the two cancer types in different colors. How well do the top two principal components discriminate between the two classes? How much of the variance within the data do these two principal components explain?

```
In [10]: def prepare_data(df, target_col='Cancer_type'):
             x_data = df.drop(target_col, axis=1)
             y_data = df[target_col]
             return x_data, y_data
```

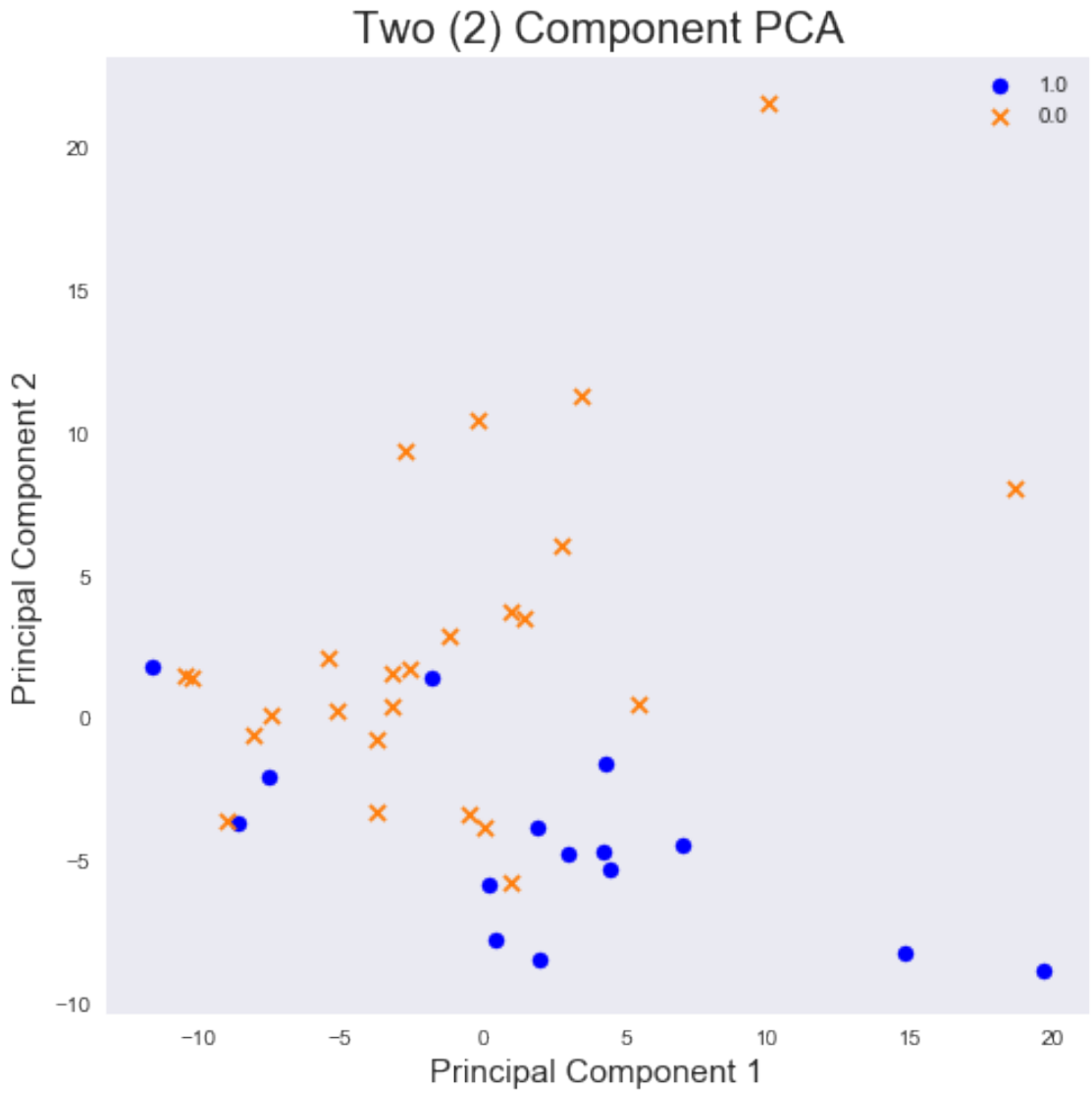
```
In [11]: x_train, y_train = prepare_data(data_train_norm)
```

```
In [12]: # your code here
         pca = PCA(n_components=2)
         x_train_pca = pca.fit_transform(x_train)
         # display(x_train_pca)
         x_train_principals = pd.DataFrame(x_train_pca, columns = ['principal component 1', 'pr
         x_train_princ_df = pd.concat([x_train_principals, y_train], axis = 1)
         x_train_princ_df.head()
```

Out[12]:

| | principal component 1 | principal component 2 | Cancer_type |
|---|---|---|---|
| 0 | 5.453069 | 0.497705 | 0.0 |
| 1 | -2.679879 | 9.352714 | 0.0 |
| 2 | -8.935284 | -3.599366 | 0.0 |
| 3 | 18.689682 | 8.075207 | 0.0 |
| 4 | -5.133550 | 0.319486 | 0.0 |

In [13]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('Two (2) Component PCA', fontsize = 20)
targets = [1.0,0.0]
colors = ['b', 'tab:orange']
markers = ['o','x']
for target, color, marker in zip(targets,colors, markers):
    indicesToKeep = x_train_princ_df['Cancer_type'] == target
    ax.scatter(x_train_princ_df.loc[indicesToKeep, 'principal component 1']
               , x_train_princ_df.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50, marker = marker)
ax.legend(targets)
ax.grid()
```

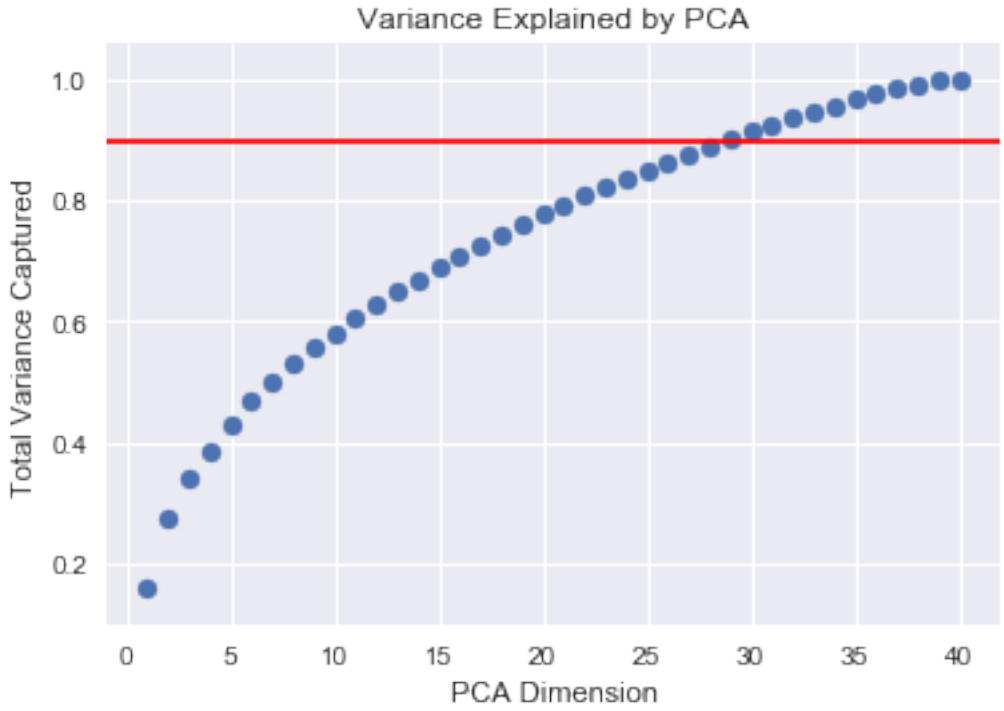



```
In [14]: np.sum(pca.explained_variance_ratio_[0:2])

Out[14]: 0.2731782945208864
```

**Answer:** Two component PCA give us some discrimination between the two classes, but there is no distinct boundary. The variance ratio is 0.27317, which tell us that these components account for about 27% of the variability in the model.

**1.5** Plot the cumulative variance explained in the feature set as a function of the number of PCA-components (up to the first 50 components). Do you feel 2 components is enough, and if not, how many components would you choose to consider? Justify your choice in 3 or fewer sentences. Finally, determine how many components are needed to explain at least 90% of the variability in the feature set.

```
In [15]: pca_transform_50 = PCA(n_components=50)
         x_train_pca_50 = pca_transform_50.fit_transform(x_train)

         plt.scatter(range(1,len(pca_transform_50.explained_variance_ratio_)+1),np.cumsum(pca_t
         plt.xlabel("PCA Dimension")
         plt.ylabel("Total Variance Captured")
         plt.title("Variance Explained by PCA");
         plt.axhline(y=0.9, color='r', linestyle = '-');
```




**Answer**: We do not believe 2 components is enough. Ideally, we'd look at 10-11 components because that's where the variance ration curve begins to taper off. To achieve 90% total variance is about 29 components.

Question 2 [25 pts]: Linear Regression vs. Logistic Regression

In class we discussed how to use both linear regression and logistic regression for classification. For this question, you will work with a single gene predictor, `D29963_at`, to explore these two methods.

**2.1** Fit a simple linear regression model to the training set using the single gene predictor `D29963_at` to predict cancer type and plot the histogram of predicted values. We could interpret the scores predicted by the regression model for a patient as an estimate of the probability that the patient has `Cancer_type`=1 (AML). Is there a problem with this interpretation?

**2.2** The fitted linear regression model can be converted to a classification model (i.e. a model that predicts one of two binary classes 0 or 1) by classifying patients with predicted score greater than 0.5 into `Cancer_type`=1, and the others into the `Cancer_type`=0. Evaluate the classification accuracy of the obtained classification model on both the training and test sets.

**2.3** Next, fit a simple logistic regression model to the training set. How do the training and test classification accuracies of this model compare with the linear regression model? If there are no substantial differences, why do you think this happens?

**2.3** Next, fit a simple logistic regression model to the training set. How do the training and test classification accuracies of this model compare with the linear regression model? If there are any substantial differences, why do you think they occur or not?

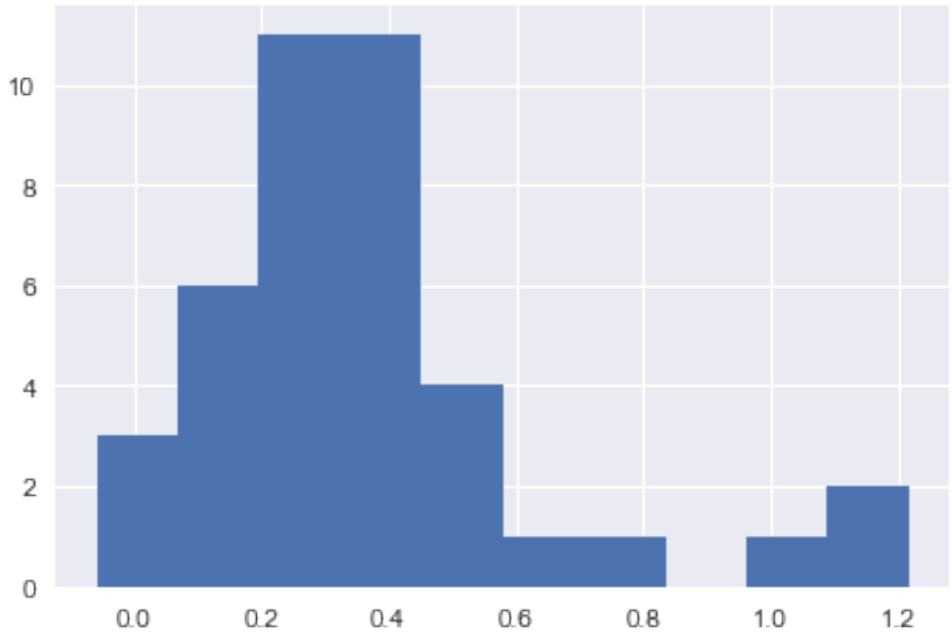
Remember, you need to set the regularization parameter for sklearn's logistic regression function to be a very large value in order to **not** regularize (use 'C=100000').

**2.4** Create a figure with 4 items displayed on the same plot: - the quantitative response from the linear regression model as a function of the gene predictor `D29963_at`. - the predicted probabilities of the logistic regression model as a function of the gene predictor `D29963_at`.
- the true binary response for the test set points for both models in the same plot. - a horizontal line at $y = 0.5$. Based on these plots, does one of the models appear better suited for binary classification than the other? Explain in 3 sentences or fewer.

**Answers:** **2.1:** Fit a simple linear regression model to the training set using the single gene predictor `D29963_at` to predict cancer type. We could interpret the scores predicted by the regression model for a patient as an estimate of the probability that the patient has `Cancer_type`=1 (AML). Is there a problem with this interpretation?

```
In [16]: # your code here
         x_train_single = np.asarray(x_train['D29963_at']).reshape(-1,1)

In [17]: from sklearn.linear_model import LinearRegression

         x_train_cst = sm.add_constant(x_train_single)

         regr = LinearRegression()
         regr_single = regr.fit(x_train_cst, y_train)
         y_pred_single = regr.predict(x_train_cst)
         print("Slope: ", regr_single.coef_, "Intercept: ",regr_single.intercept_)

         def lr(t):
             return t*regr_single.coef_ + regr_single.intercept_

         # plt.scatter(x_train_single, y_train,  color='black')
         # plt.plot(x_train_single, y_pred, color='blue', linewidth=3)

         plt.hist(y_pred_single, bins = 10);
```

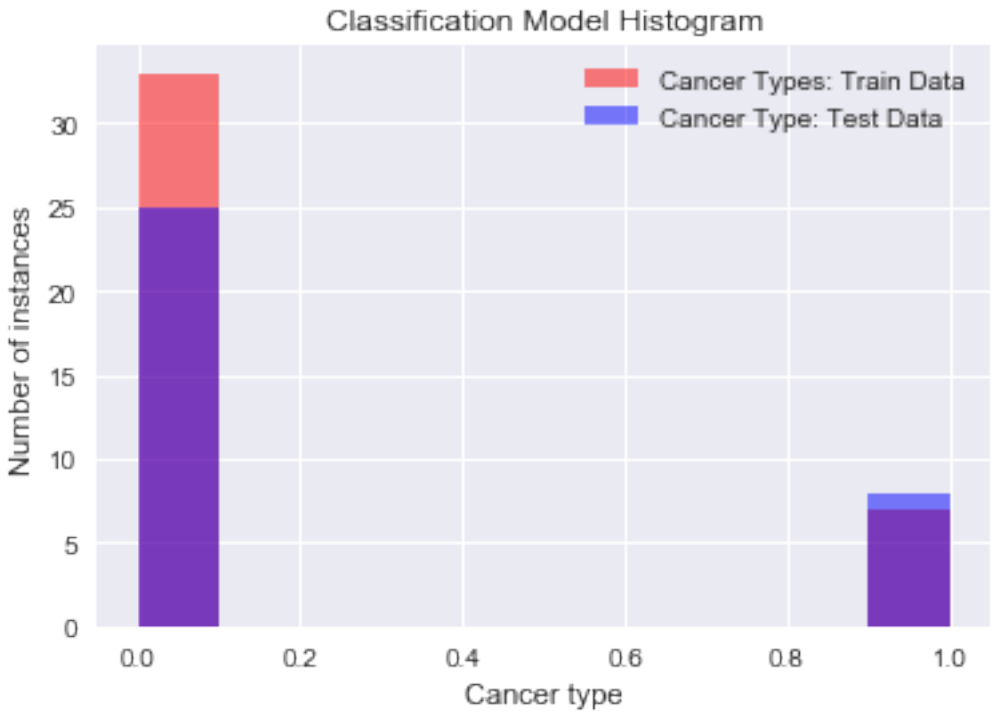
```
Slope:  [0.         1.27643062] Intercept:  -0.058746140611556474
```



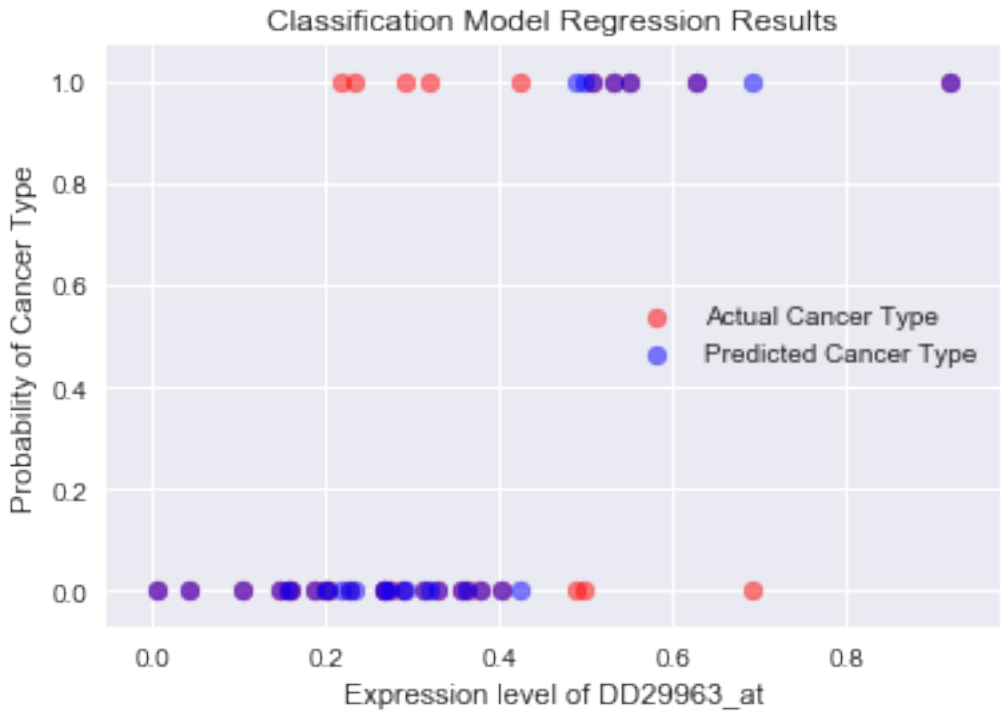

**Answer:** This is not a good model because there are only two outcome values we consider (0 and 1), and our result includes values that lie below, above, and in between those values.

**2.2:** The fitted linear regression model can be converted to a classification model (i.e. a model that predicts one of two binary classes 0 or 1) by classifying patients with predicted score greater than 0.5 into `Cancer_type=1`, and the others into the `Cancer_type=0`. Evaluate the classification accuracy of the obtained classification model on both the training and test sets.

```
In [18]: data_test_norm = scale_data(data_test)
         x_test, y_test = prepare_data(data_test_norm)
         x_test_single = np.asarray(x_test['D29963_at']).reshape(-1,1)
         x_test_cst = sm.add_constant(x_test_single)

In [19]: # your code here
         y_pred_train_recode = pd.cut(y_pred_single, [-1,0.5,2], labels=[0,1])

         y_pred = regr.predict(x_test_cst)

         y_pred_test_recode = pd.cut(y_pred,[-1,0.5,2], labels=[0,1])

         plt.hist(y_pred_train_recode, color = 'r', alpha = 0.5, label = "Cancer Types: Train
         plt.hist(y_pred_test_recode, color = 'b', alpha = 0.5, label = "Cancer Type: Test Data
         plt.xlabel("Cancer type")
         plt.ylabel("Number of instances")
         plt.title('Classification Model Histogram')
         plt.legend()

Out[19]: <matplotlib.legend.Legend at 0x275fc01cc88>
```




```
In [20]: plt.scatter(x_test_single, y_test, color = 'r', alpha = 0.5, label = "Actual Cancer Ty
         plt.scatter(x_test_single, y_pred_test_recode, color = 'b', alpha = 0.5,label = 'Predi
         plt.xlabel('Expression level of DD29963_at')
         plt.ylabel('Probability of Cancer Type')
         plt.title('Classification Model Regression Results')
         plt.legend()

Out[20]: <matplotlib.legend.Legend at 0x275fb475588>
```




```
In [21]: train_score1 = accuracy_score(y_train, y_pred_train_recode)*100
         test_score1 = accuracy_score(y_test, y_pred_test_recode)*100
         print(train_score1, test_score1)
```

```
80.0 75.75757575757575
```

**Answer:** The classification model adds limited value. Since our `y_train` and `y_test` are already values of 0 and 1, we don't transform until we get a predicted y value. This show a clear cut between the 0 and 1 values, but doesn't enhance our understanding of the expression level of `DD29963_at` affects cancer types.

**2.3:** Next, fit a simple logistic regression model to the training set. How do the training and test classification accuracies of this model compare with the linear regression model? If there are no substantial differences, why do you think this happens?

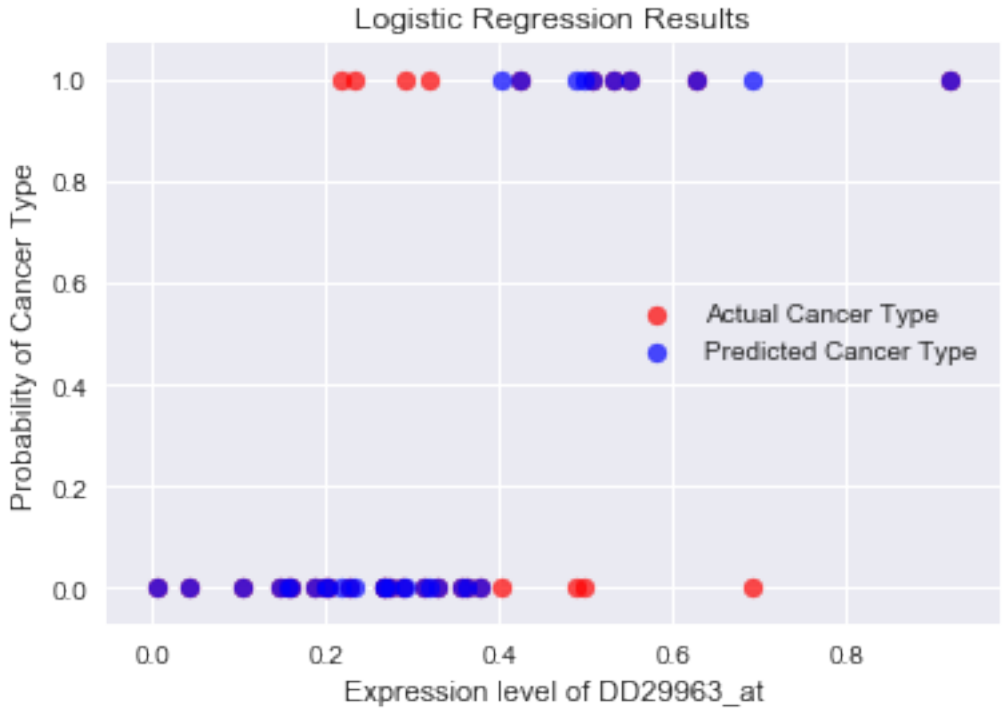
```
In [22]: # your code here
         fitted_logit = LogisticRegression(C = 1000000, solver = 'newton-cg', max_iter=250).fi
         y_pred_log_train = fitted_logit.predict(x_train_single)
         y_pred_log_test = fitted_logit.predict(x_test_single)
         print(fitted_logit.coef_)
         print(fitted_logit.intercept_)

         train_score = accuracy_score(y_train, y_pred_log_train)*100
         test_score = accuracy_score(y_test, y_pred_log_test)*100
         print(train_score, test_score)
```

```
[[10.26533564]]
[-3.99511683]
80.0 75.75757575757575
```

```
In [23]: def logreg(t):
             return t*fitted_logit.coef_ + fitted_logit.intercept_
```

```
In [24]: plt.scatter(x_test_single, y_test, color = 'r', alpha = 0.7, label = "Actual Cancer Ty
         plt.scatter(x_test_single, y_pred_log_test, color = 'b', alpha = 0.7,label = 'Predicte
         plt.title('Logistic Regression Results')
         plt.xlabel('Expression level of DD29963_at')
         plt.ylabel('Probability of Cancer Type')
         plt.legend()
         #plt.plot(x_test_single, logreg(x_test_single), color = 'g')
```

```
Out[24]: <matplotlib.legend.Legend at 0x275fc0dcda0>
```

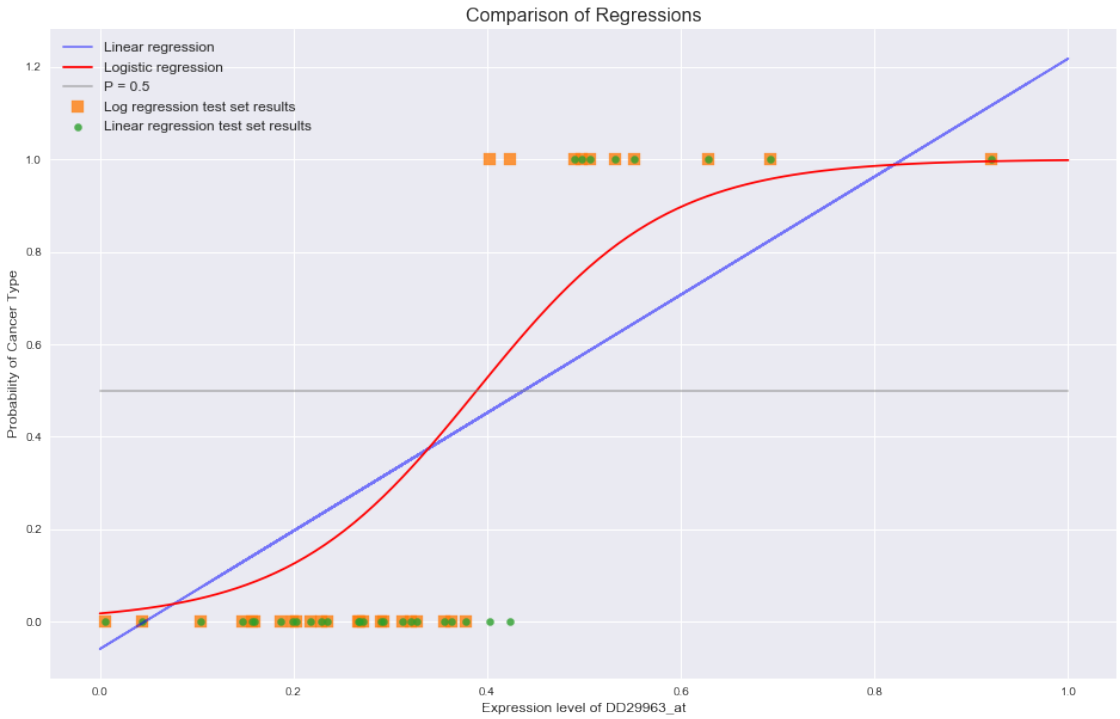



**Answer:** This scatterplot shows that the probability swings at an expression level just below 0.4, as opposed to an expression level just above 0.4 (which was what we saw in 2.2).

**2.4:** Create a figure with 4 items displayed on the same plot: - the quantitative response from the linear regression model as a function of the gene predictor `D29963_at`. - the predicted probabilities of the logistic regression model as a function of the gene predictor `D29963_at`.
- the true binary response for the test set points for both models in the same plot. - a horizontal line at $y = 0.5$.

Based on these plots, does one of the models appear better suited for binary classification than the other? Explain in 3 sentences or fewer.

```
In [25]: #predict v. predict_praba
         x_values = np.linspace(0,1,100).reshape(-1,1)
         y_values = fitted_logit.predict_proba(x_values)[:,1]

In [26]: plt.figure(figsize = [16,10])

         # Quantative response from linear regression
         plt.plot(x_train_single, y_pred_single, color = 'b', alpha = 0.5, label = 'Linear regr

         # Logistic regression predicted possibilites
         plt.plot(x_values, fitted_logit.predict_proba(x_values)[:,1], color ='r', label = 'Log

         # True binary response?
         plt.scatter(x_test_single, y_pred_log_test, color = 'tab:orange',alpha = 0.8, s=100, m
         plt.scatter(x_test_single, y_pred_test_recode, color ='tab:green', alpha = 0.8,s=150,

         # Horizontal line at y = 0.5
         plt.plot([0,1], [0.5,0.5], color = 'tab:gray', alpha = 0.5, label = 'P = 0.5')

         #Add title and labels
         plt.title('Comparison of Regressions',fontsize = 16)
         plt.xlabel('Expression level of DD29963_at', fontsize = 12)
         plt.ylabel('Probability of Cancer Type', fontsize = 12)
         plt.legend(fontsize = 12)

Out[26]: <matplotlib.legend.Legend at 0x275fdc03080>
```




**Answer**: The logistic regression appears to be best suited for binary classification. The linear regression can produce outcomes outside of the range (0,1), whereas the result of a logistic regression will always fall in that range.

Question 3 [30pts]: Multiple Logistic Regression

**3.1** Next, fit a multiple logistic regression model with all the gene predictors from the data set. How does the classification accuracy of this model compare with the models fitted in question 2 with a single gene (on both the training and test sets)?

**3.2** How many of the coefficients estimated by this multiple logistic regression in the previous part are significantly different from zero at a *significance level of 5%*? Use the same value of C=100000 as before.

**Hint:** To answer this question, use *bootstrapping* with 1000 boostrap samples/iterations.

**3.3** Use the `visualize_prob` function provided below (or any other visualization) to visualize the probabilties predicted by the fitted multiple logistic regression model on both the training and test data sets. The function creates a visualization that places the data points on a vertical line based on the predicted probabilities, with the different cancer classes shown in different colors, and with the 0.5 threshold highlighted using a dotted horizontal line. Is there a difference in the spread of probabilities in the training and test plots? Are there data points for which the predicted probability is close to 0.5? If so, what can you say about these points?

**3.4** Open question: Comment on the classification accuracy of the train and test sets. Given the results above how would you assess the generalization capacity of your trained model? What other tests or approaches would you suggest to better guard against the false sense of security on the accuracy of the model as a whole.

```
In [27]: #--------  visualize_prob
         # A function to visualize the probabilities predicted by a Logistic Regression model
```

```
# Input:
#      model (Logistic regression model)
#      x (n x d array of predictors in training data)
#      y (n x 1 array of response variable vals in training data: 0 or 1)
#      ax (an axis object to generate the plot)

def visualize_prob(model, x, y, ax):
    # Use the model to predict probabilities for x
    y_pred = model.predict_proba(x)

    # Separate the predictions on the label 1 and label 0 points
    ypos = y_pred[y==1]
    yneg = y_pred[y==0]

    # Count the number of label 1 and label 0 points
    npos = ypos.shape[0]
    nneg = yneg.shape[0]

    # Plot the probabilities on a vertical line at x = 0,
    # with the positive points in blue and negative points in red
    pos_handle = ax.plot(np.zeros((npos,1)), ypos[:,1], 'bo', label = 'Cancer Type 1')
    neg_handle = ax.plot(np.zeros((nneg,1)), yneg[:,1], 'ro', label = 'Cancer Type 0')

    # Line to mark prob 0.5
    ax.axhline(y = 0.5, color = 'k', linestyle = '--')

    # Add y-label and legend, do not display x-axis, set y-axis limit
    ax.set_ylabel('Probability of AML class')
    ax.legend(loc = 'best')
    ax.get_xaxis().set_visible(False)
    ax.set_ylim([0,1])
```

**Answers:** **3.1:** Next, fit a multiple logistic regression model with all the gene predictors from the data set. How does the classification accuracy of this model compare with the models fitted in question 2 with a single gene (on both the training and test sets)?

In [28]:
```
x_train, y_train = prepare_data(data_train_norm)
x_test, y_test = prepare_data(data_test_norm)
x_train_cst = sm.add_constant(x_train)
x_test_cst = sm.add_constant(x_test)

#Training
model = LogisticRegression(C=100000).fit(x_train_cst, y_train)

#Predict
y_pred_train = model.predict(x_train_cst)
y_pred_test = model.predict(x_test_cst)
```

```
        #Perfromance Evaluation
        train_score = accuracy_score(y_train, y_pred_train)*100
        test_score = accuracy_score(y_test, y_pred_test)*100

        print("Training Set Accuracy:",str(train_score)+'%')
        print("Testing Set Accuracy:",str(test_score)+'%')
```

```
Training Set Accuracy: 100.0%
Testing Set Accuracy: 100.0%
```

**Answer:** The classification accuracy of this model is far higher than the classification accuracy for the two previous models.

**3.2** How many of the coefficients estimated by this multiple logistic regression in the previous part are significantly different from zero at a *significance level of 5%*? Use the same value of C=100000 as before.

**Hint:** To answer this question, use *bootstrapping* with 1000 boostrap samples/iterations.

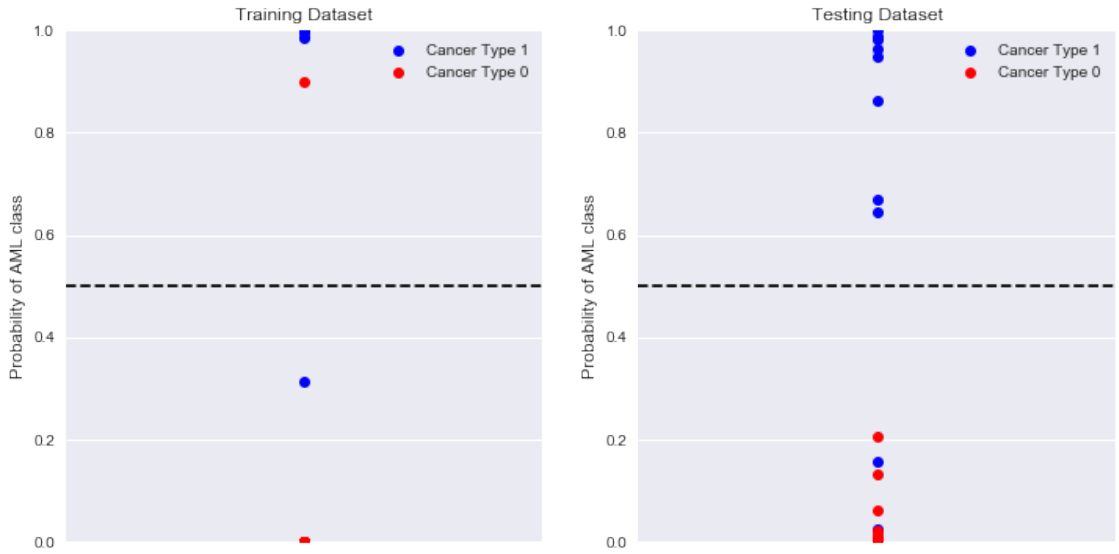
```
In [29]: #Creating model
         model = LogisticRegression(C=100000, fit_intercept = True).fit(x_train_cst, y_train)

         #Initializing variables
         bootstrap_iterations = 1000
         coeffs = np.zeros((bootstrap_iterations, data_train_norm.shape[1]-1))

         #Conduct bootstraping iterations
         for i in range(bootstrap_iterations):
             temp = data_train_norm.sample(frac=1, replace=True)
             response_variable = temp['Cancer_type']
             temp = temp.drop(['Cancer_type'], axis=1)
             model.fit(temp, response_variable)
             coeffs[i,:] = model.coef_

         #Find Significant Columns, Count
         coeffs_count, significant_cols = 0, []
         for i in range(coeffs.shape[1]):
             coeff_samples = coeffs[:,i]
             lower_bound = np.percentile(coeff_samples, 2.5)
             upper_bound = np.percentile(coeff_samples, 97.5)
             if lower_bound>0 or upper_bound<0:
                 coeffs_count += 1
                 significant_cols.append(data_train_norm.columns[i])

         #print('Columns :', significant_cols)
         print('Count of 95% statistically significant coefficients :', coeffs_count)
```

```
Count of 95% statistically significant coefficients : 1865
```

**Answer**: 1,865 coefficients are significantly different from zero at a *significance level of 5%*.

**3.3:** Use the `visualize_prob` function provided below to visualize the probabilties predicted by the fitted multiple logistic regression model on both the training and test data sets. The function creates a visualization that places the data points on a vertical line based on the predicted probabilities, with the different cancer classes shown in different colors, and with the 0.5 threshold highlighted using a dotted horizontal line. Is there a difference in the spread of probabilities in the training and test plots? Are there data points for which the predicted probability is close to 0.5? If so, what can you say about these points?

```
In [31]: """ Plot classification model """
         #Create Plot
         fig = plt.figure(figsize=(12,6))
         ax1 = fig.add_subplot(121)
         ax2 = fig.add_subplot(122)

         #Plot Training
         visualize_prob(model, x_train, y_train, ax1)
         ax1.set_title('Training Dataset')

         #Plot Testing
         visualize_prob(model, x_test, y_test, ax2)
         ax2.set_title('Testing Dataset')
         plt.show()
```




**Answer**: The training set probabilities are all either 0 or 1. The test set probabilities are spread between 0 and 0.25 for ALL, and 0.6 and 1 for AML. There are several points close to 0.5 for AML.

**3.4** Open question: Comment on the classification accuracy of tain and test set? Given the results above how would you assest the generalization capacity of your trained model? What other tests would you suggest to better guard against false sense of security on the accuracy of the model as a whole.

**Answer**: Based on these results, we achieve a perfect model on the training data and a good, but not great model on the test data. It provides a generally correct classification, but has a much larger range of results than the train data. This indicates we may have overfit our model to the training data, and should aim the reduce this overfitting. We can do PCA analysis to reduce the number of components we use to create the model.

Question 4 [20 pts]: PCR: Principal Components Regression

High dimensional problems can lead to problematic behavior in model estimation (and make prediction on a test set worse), thus we often want to try to reduce the dimensionality of our problems. A reasonable approach to reduce the dimensionality of the data is to use PCA and fit a logistic regression model on the smallest set of principal components that explain at least 90% of the variance in the predictors.

**4.1** Fit two separate Logistic Regression models using principal components as the predictors: (1) with the number of components you selected from problem 1.5 and (2) with the number of components that explain at least 90% of the variability in the feature set. How do the classification accuracy values on both the training and tests sets compare with the models fit in question 3?

**4.2** Use the code provided in question 3 (or your choice of visualization) to visualize the probabilities predicted by the fitted models in the previous part on both the training and test sets. How does the spread of probabilities in these plots compare to those for the model in question 3.2? If the lower dimensional representation yields comparable predictive power, what advantage does the lower dimensional representation provide?

**Answers:** **4.1** Fit two separate Logistic Regression models using principal components as the predictors: (1) with the number of components you selected from problem 1.5 and (2) with the number of components that explain at least 90% of the variability in the feature set. How do the classification accuracy values on both the training and tests sets compare with the models fit in question 3?

```
In [32]: x_train, y_train = prepare_data(data_train_norm)
         x_test, y_test = prepare_data(data_test_norm)
         x_train_cst = sm.add_constant(x_train)
         x_test_cst = sm.add_constant(x_test)

         #Create and fit PCA object
         pca = PCA()
         pca.fit(x_train_cst)

         #Transforming x_train and x_test
         x_train_pca = pca.transform(x_train_cst)
         x_test_pca = pca.transform(x_test_cst)

         #Find number of components that explain predefined variance threshold
         sum_variance, component_count = 0, 0
         while sum_variance < 0.9:
             sum_variance += pca.explained_variance_ratio_[component_count]
             component_count += 1

         print('Number of Principal Components that explain >=90% of Variance: ', component_cou
         print('Total Variance Explained by '+str(component_count)+' components:', str(sum_vari
```

```
Number of Principal Components that explain >=90% of Variance:  29
Total Variance Explained by 29 components: 90.26870362661795%
```

```
In [33]: pca15= PCA(11)
         pca15.fit(x_train_cst)
         x_train15_pca=pca15.transform(x_train_cst)
         x_test15_pca = pca15.transform(x_test_cst)
         logreg15 = LogisticRegression(C=1000000).fit(x_train15_pca,y_train)

         acc_score_train15 = logreg15.score(x_train15_pca, y_train)*100
         acc_score_test15= logreg15.score(x_test15_pca,y_test)*100

         print("Training Set Accuracy:",str(acc_score_train15)+'%')
         print("Testing Set Accuracy:",str(acc_score_test15)+'%')
```

```
Training Set Accuracy: 100.0%
Testing Set Accuracy: 96.96969696969697%
```

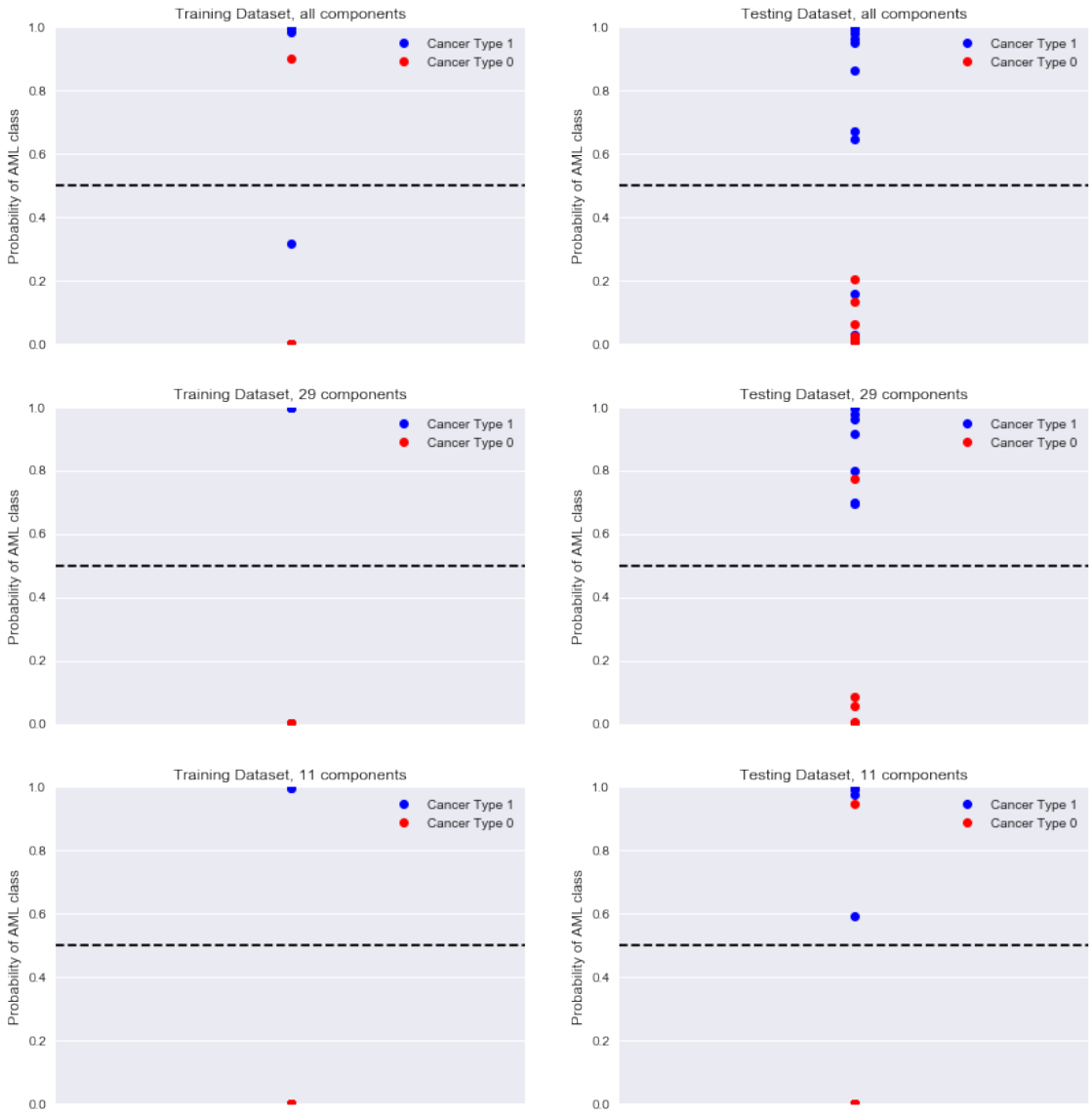
```
In [34]: #Create and fit PCA object
         pca = PCA(n_components=component_count)
         pca.fit(x_train)

         #Transforming x_train and x_test
         x_train_pca = pca.transform(x_train)
         x_test_pca = pca.transform(x_test)

         #Add constant to x_train and x_test
         x_train_pca_cst = sm.add_constant(x_train_pca)
         x_test_pca_cst = sm.add_constant(x_test_pca)

         #Training
         model90 = LogisticRegression(C=100000).fit(x_train_pca_cst, y_train)

         #Predict
         y_pred_train = model90.predict(x_train_pca_cst)
         y_pred_test = model90.predict(x_test_pca_cst)

         #Perfromance Evaluation
         train_score = accuracy_score(y_train, y_pred_train)*100
         test_score = accuracy_score(y_test, y_pred_test)*100

         print("Training Set Accuracy:",str(train_score)+'%')
         print("Testing Set Accuracy:",str(test_score)+'%')
```

```
Training Set Accuracy: 100.0%
Testing Set Accuracy: 96.96969696969697%
```

**Answer**: The accuracy scores are the same with 11 components versus 29 components. This shows us that adding 18 components does not enhance the accuracy of our model.

**4.2:** Use the code provided in question 3 to visualize the probabilities predicted by the fitted models on both the training and test sets. How does the spread of probabilities in these plots compare to those for the model in question 3.2? If the lower dimensional representation yields comparable predictive power, what advantage does the lower dimensional representation provide?

```
In [36]: #Create Plot
         fig = plt.figure(figsize=(14,15))
         ax1 = fig.add_subplot(321)
         ax2 = fig.add_subplot(322)
         ax3 = fig.add_subplot(323)
         ax4 = fig.add_subplot(324)
         ax5 = fig.add_subplot(325)
         ax6 = fig.add_subplot(326)

         #Plot Training
         visualize_prob(model, x_train, y_train, ax1)
         ax1.set_title('Training Dataset, all components')

         #Plot Testing
         visualize_prob(model, x_test, y_test, ax2)
         ax2.set_title('Testing Dataset, all components')

         #Plot Training
         visualize_prob(model90, x_train_pca_cst, y_train, ax3)
         ax3.set_title('Training Dataset, 29 components')

         #Plot Testing
         visualize_prob(model90, x_test_pca_cst, y_test, ax4)
         ax4.set_title('Testing Dataset, 29 components')

         #Plot Training
         visualize_prob(logreg15, x_train15_pca, y_train, ax5)
         ax5.set_title('Training Dataset, 11 components')

         #Plot Testing
         visualize_prob(logreg15, x_test15_pca, y_test, ax6)
         ax6.set_title('Testing Dataset, 11 components')
         plt.show()
```




**Answer**: The spreads with PCA for 11 and 29 components are generally the same as for the model with all the components. The original model was overfit with all the components. We are able to use PCA to transform the model into a simpler model with less overfitting, and it performs just as well on our test data.