CS109A Introduction to Data Science:

Homework 5: Logistic Regression, High Dimensionality and PCA

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

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

Out[1]:

INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas https://canvas.harvard.edu/courses/42693/pages/homework-policies-and-submission-instructions (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:

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.pyplot as plt
import matplotlib as mpl

%matplotlib inline

import seaborn as sns
sns.set()

from IPython.display import display
```

In [3]:
```
# define figure defaults
mpl.rc('axes', labelsize=14, titlesize=14)
mpl.rc('figure', figsize=[6,4], titlesize=16)
mpl.rc('legend', fontsize=12)
mpl.rc('lines', linewidth=2, color='k')
mpl.rc('xtick', labelsize=14)
mpl.rc('ytick', labelsize=14)
```

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 Let's 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 [4]:
```
sns.set(style="white")
```

In [5]:
```
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...

In [6]:
```
# use pandas describe to look at mean, std, and min-max range of genes
data_train.describe()
```

Out[6]:

	Cancer_type	AFFX-BioB-5_at	AFFX-BioB-M_at	AFFX-BioB-3_at	AFFX-BioC-5_at	AFFX-BioC-3_at	AFFX-BioDn-5_at	A
count	40.00000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	
mean	0.37500	-116.125000	-163.350000	-9.125000	209.075000	-250.325000	-379.925000	
std	0.49029	102.783364	95.437871	101.998539	111.000205	107.218776	123.026449	
min	0.00000	-476.000000	-531.000000	-168.000000	-24.000000	-496.000000	-696.000000	-1
25%	0.00000	-140.750000	-208.500000	-81.250000	124.250000	-316.500000	-461.750000	
50%	0.00000	-109.000000	-150.000000	-29.000000	228.000000	-225.000000	-384.500000	
75%	1.00000	-64.750000	-99.500000	47.000000	303.750000	-178.750000	-286.250000	
max	1.00000	86.000000	-20.000000	262.000000	431.000000	-32.000000	-122.000000	

8 rows × 7130 columns

In [7]:
```
# initialize/fit scaler and normalize data
scaler = MinMaxScaler()
scaler = scaler.fit(data_train.values)
scaled_train = data_train.copy()
scaled_train[data_train.columns] = scaler.transform(data_train.values)

# repeat normalization fit on training data on test data
scaled_test = data_test.copy()
scaled_test[data_test.columns] = scaler.transform(data_test.values)
```

```
C:\Users\winsl0w\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: D
ataConversionWarning: Data with input dtype int64 was converted to float64 by M
inMaxScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [8]:
```
# inspect values again to ensure proper range
scaled_train.describe()
```

Out[8]:

	Cancer_type	AFFX-BioB-5_at	AFFX-BioB-M_at	AFFX-BioB-3_at	AFFX-BioC-5_at	AFFX-BioC-3_at	AFFX-BioDn-5_at	AFFX-BioDn-3_at
count	40.00000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000
mean	0.37500	0.640347	0.719472	0.369477	0.512253	0.529472	0.550653	0.654253
std	0.49029	0.182889	0.186767	0.237206	0.243956	0.231075	0.214332	0.216589
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.00000	0.596530	0.631115	0.201744	0.325824	0.386853	0.408101	0.547153
50%	0.00000	0.653025	0.745597	0.323256	0.553846	0.584052	0.542683	0.683986
75%	1.00000	0.731762	0.844423	0.500000	0.720330	0.683728	0.713850	0.753381
max	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

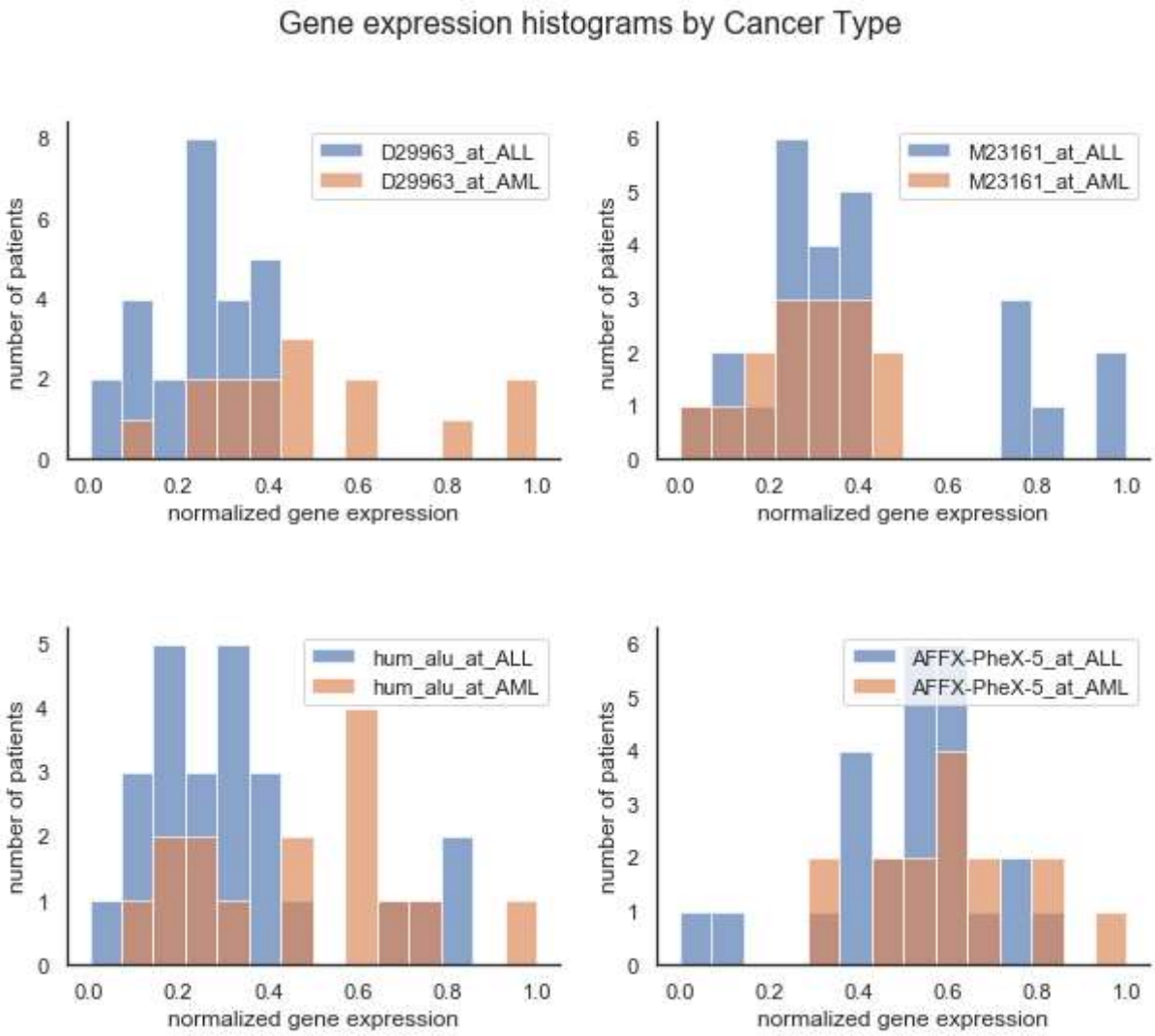
8 rows × 7130 columns

1.2: Notice that the resulting training set contains...

This is a problem because there is not enough information in the training set to fit this many parameters. With fewer data points than parameters, there will be no unique solution for the model.

1.3: Let's 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 [9]:
```
# group by cancer type
by_cancer = scaled_train.groupby("Cancer_type")

# open figure and define genes to index
plt.figure(figsize=(10,8))
plt.suptitle("Gene expression histograms by Cancer Type")
genes = ["D29963_at","M23161_at","hum_alu_at","AFFX-PheX-5_at"]
bins = np.linspace(0,1,15)

for i, g in enumerate(genes):

    plt.subplot(2,2,i+1)
    plt.xlabel("normalized gene expression")
    plt.ylabel("number of patients")

    for j in range(len(by_cancer.groups)):
        if j:
            leg_lab = g + "_AML"
        else:
            leg_lab = g + "_ALL"
        plt.hist(by_cancer.get_group(j)[g], label=leg_lab, alpha=0.65, bins=bins)
    plt.legend()
    sns.despine()

plt.subplots_adjust(hspace=0.5)
```



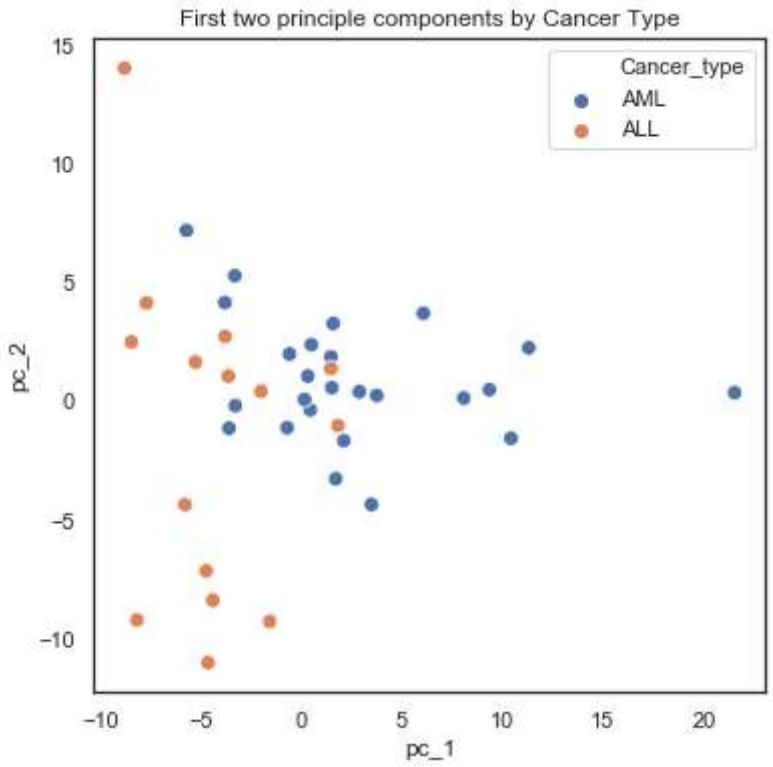
It looks like higher expression of `D29963_at` might be linked to AML, since some AML patients (about half) show higher levels of expression than is seen an any ALL patients. `M23161_at` looks like it might be linked to ALL, since some ALL patients show much higher expression than any AML patient. The gene `hum_alu_at` could be weakly correlated to AML, but it is not clear. There is no striking difference in the distributions of `AFFX-PheX-5` expression for AML and ALL patients.

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?

In [52]:
```
# find principle components for predictors
x = scaled_train.copy().drop(columns="Cancer_type", axis=1)
pca = PCA(n_components=x.shape[1])
pca.fit(x)

# transform data and store in new dataframe
columns = ['pc_%i' % i for i in range(x.shape[0])]
df_pca = pd.DataFrame(pca.transform(x), columns=columns, index=x.index)

# add cancer type to the df
cancer_list = ["AML"]* df_pca.shape[0]
cancer_list = list(map(lambda i, j: i if not j else "ALL", cancer_list, scaled_tr
df_pca["Cancer_type"] = cancer_list
```

In [11]:
```
# plot results colored by cancer type
plt.figure(figsize=(6,6))
sns.scatterplot(df_pca["pc_1"],df_pca["pc_2"], hue=df_pca["Cancer_type"], s=60);\
plt.title("First two principle components by Cancer Type")
```

Out[11]:
```
Text(0.5,1,'First two principle components by Cancer Type')
```



In [12]:
```
print("The first two principle components explain %.2f%% and %.2f%% of the varian
      % tuple(pca.explained_variance_ratio_[:2]))
```

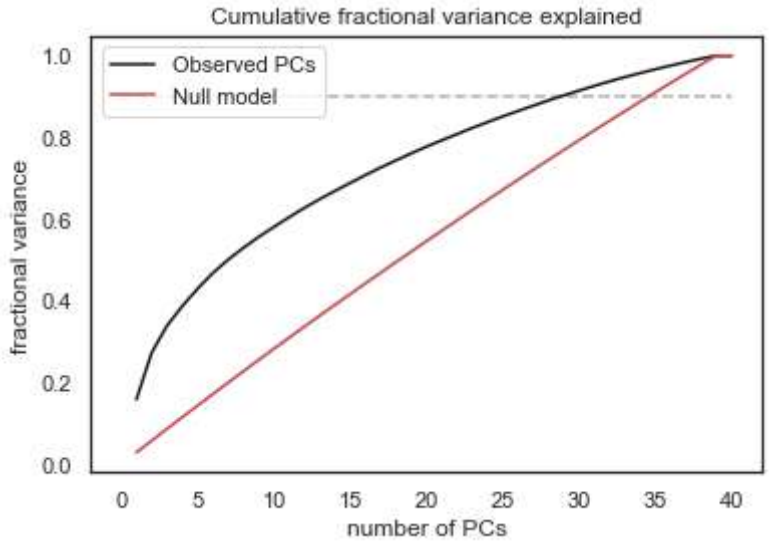
```
The first two principle components explain 0.16% and 0.11% of the variance, res
pectively
```

Based on the graph, the first two PCs do not seem to separate the cancer types particularly well. Together the first two principle components explain nearly 90% of the variance.

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 [77]:
```
# sum variance explained
cum_var = np.cumsum(pca.explained_variance_ratio_)
plt.figure(figsize=(6,4))
plt.plot(np.arange(1,x.shape[0]+1,1), cum_var,c='k', label="Observed PCs")

# randomly reshuffle data to assess upper limit due to sampling error
# shuffle each column of data matrix and do PCA
x_shuffle = scaled_train.copy().drop(columns="Cancer_type", axis=1).values
np.apply_along_axis(np.random.shuffle,0,x_shuffle)
pca_shuffle = PCA(n_components=x_shuffle.shape[1])
pca_shuffle.fit(x_shuffle)
cum_var_shuffle = np.cumsum(pca_shuffle.explained_variance_ratio_)

# plot results
plt.plot(np.arange(1,x.shape[0]+1,1),cum_var_shuffle,c='r', label="Null model")

plt.hlines(0.9,0,40,color=(.7,.7,.7), linestyle='--')
plt.legend()
plt.xlabel("number of PCs")
plt.ylabel("fractional variance")
plt.title("Cumulative fractional variance explained");
```



In [74]:
```
print("%i PCs explain variance above the sampling error" %\
      (np.where(cum_var<cum_var_shuffle)[0][0]))
```

```
38 PCs explain variance above the sampling error
```

One option is to compare the variance explained by our PCs to the null model: that all of our features are truly independent but correlated due to finite sampling. To generate the null model, we can shuffle the rows of each column independently to break the structure of the data. Using this approach, we get 38 PCs above the sampling error.

In [14]:
```
print("A minimum of %i PCs are needed to capture 90%% of the variance" % (np.wher
```

```
A minimum of 29 PCs are needed to capture 90% of the variance
```

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?

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 $y = 0.5$.

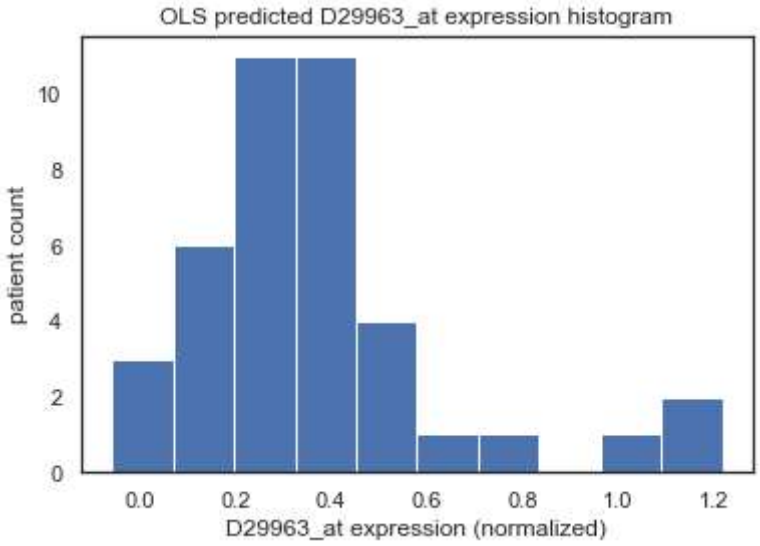
Based on these plots, does one of the models appear better suited for binary classification than the other? Explain in 3 sentences or fewer.

Answers:

2.1 Fit a simple linear regression model to the training set using the single gene predictor `D29963_at` to predict cancer type 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?

In [15]:
```
# Fit OLS Model
gene_x = sm.add_constant(scaled_train["D29963_at"].values)
ols_mdl = OLS(scaled_train["Cancer_type"].values,gene_x).fit()
```

In [16]:
```
# predict values from training set and plot histogram of results
pred_y = ols_mdl.predict(gene_x)
plt.figure
plt.hist(pred_y)
plt.xlabel("D29963_at expression (normalized)")
plt.ylabel("patient count")
plt.title("OLS predicted D29963_at expression histogram");
```



One problem with this model is that the model is not constrained to be bounded between 0-1, the range of a probability. We could potentially get probability values at $\pm\infty$, which are uninterpretable.

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 [17]:
```
# calculate training accuracy on the predictions generate above
train_accuracy = np.sum((pred_y>0.5)==scaled_train["Cancer_type"].values)/len(pre
print("training accuracy = %.2f" % train_accuracy)
```

```
training accuracy = 0.80
```

In [18]:
```
# calculate test accuracy on the predictions generate above
pred_y = ols_mdl.predict(sm.add_constant(scaled_test["D29963_at"].values))
test_accuracy = np.sum((pred_y>0.5)==scaled_test["Cancer_type"].values)/len(pred_
print("test accuracy = %.2f" % test_accuracy)
```

```
test accuracy = 0.76
```

In [19]:
```
# assess baseline performance of stupid models
dumb_train_accuracy = np.sum(scaled_train["Cancer_type"].values)/len(scaled_train
dumb_test_accuracy = np.sum(scaled_test["Cancer_type"].values)/len(scaled_test["C

print("Train data:\n All ones accuracy = %.2f, \t All zeros accuracy = %.2f" % (d
print("Test data:\n All ones accuracy = %.2f, \t All zeros accuracy = %.2f" % (du
```

```
Train data:
 All ones accuracy = 0.38,       All zeros accuracy = 0.62
Test data:
 All ones accuracy = 0.30,       All zeros accuracy = 0.70
```

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 [20]:
```
# format design matrix, and intialize/fit logistic model
scaled_train_x = scaled_train["D29963_at"].values.reshape(-1,1)
scaled_train_y =scaled_train["Cancer_type"].values
logit = LogisticRegression(C=100000, solver='newton-cg')
logit = logit.fit(scaled_train_x, scaled_train_y)

# generate predictions for train and test data
scaled_test_x = scaled_test["D29963_at"].values.reshape(-1,1)
scaled_test_y = scaled_test["Cancer_type"].values
pred_train_y = logit.predict(scaled_train_x)
pred_test_y = logit.predict(scaled_test_x)

# calculate accuracies
train_accuracy = np.sum((pred_train_y==scaled_train_y))/len(pred_train_y)
test_accuracy = np.sum((pred_test_y==scaled_test_y))/len(pred_test_y)

# print results
print("training accuracy = %.2f" % train_accuracy)
print("test accuracy = %.2f" % test_accuracy)
```

```
training accuracy = 0.80
test accuracy = 0.76
```

Both the training and test accuracy are very close to the accuracies of the OLS model. One plausible reason that the accuracy does not change much is that there are not many gene expression values close to where the models cross the boundary line at p=0.5. The gene histogram above for `D29963_at` shows that the expression pattern appears bimodal, meaning that most points will not change their predicted class.

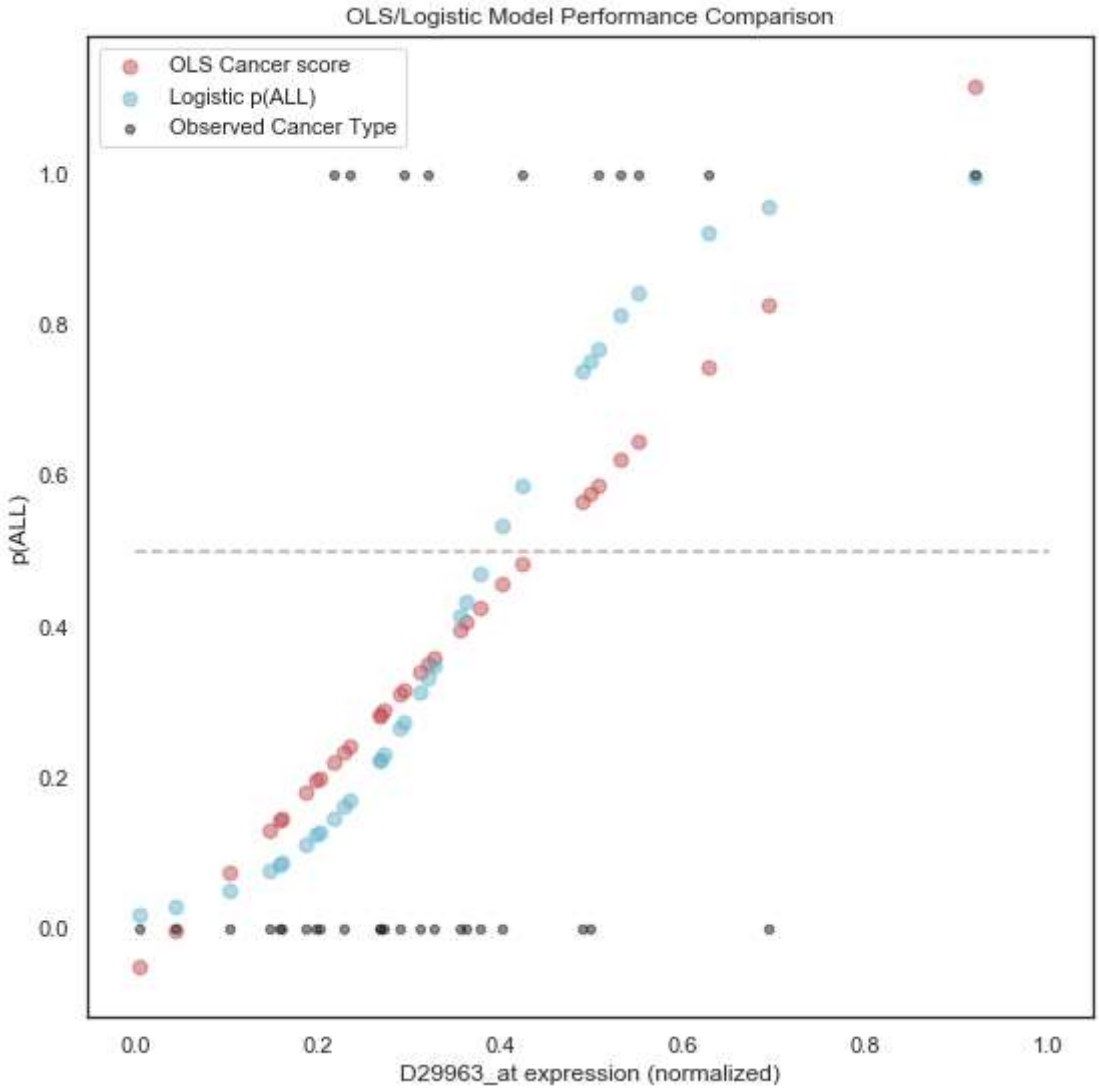
2.4 Create a figure with 4 items displayed on the same plot:

- 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 [21]:
```
# get predictions
gene_expression = scaled_test["D29963_at"].values
ols_y = ols_mdl.predict(sm.add_constant(gene_expression))
log_y = logit.predict_proba(scaled_test_x)

# plot data
plt.figure(figsize=(9,9))
plt.scatter(gene_expression, ols_y, s=50, c='r', label="OLS Cancer score", alpha=
plt.scatter(gene_expression, log_y[:,1], s=50, c='c', label="Logistic p(ALL)", al
plt.scatter(gene_expression, scaled_test_y, s=20, c='k', label="Observed Cancer T
plt.hlines(0.5,0,1, color=(.7,.7,.7), linestyle='--')
plt.xlabel("D29963_at expression (normalized)")
plt.ylabel("p(ALL)")
plt.legend()
plt.title("OLS/Logistic Model Performance Comparison");
```



The logistic model seems better for classification. The shape of the curve will push the values closer

to p=0 or p=1 for most values of `D29963_at` expression.

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 [22]:
```
#--------  visualize_prob
# A function to visualize the probabilities predicted by a Logistic Regression mo
# 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 Typ
    neg_handle = ax.plot(np.zeros((nneg,1)), yneg[:,1], 'ro', label = 'Cancer Typ

    # 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 [23]:
```
# format design matrix, and intialize/fit logistic model
scaled_train_x = scaled_train.drop(columns="Cancer_type").values
scaled_train_y =scaled_train["Cancer_type"].values
logit = LogisticRegression(C=100000, solver='newton-cg')
logit = logit.fit(scaled_train_x, scaled_train_y)

# generate predictions for train and test data
scaled_test_x = scaled_test.drop(columns="Cancer_type").values
scaled_test_y = scaled_test["Cancer_type"].values
pred_train_y = logit.predict(scaled_train_x)
pred_test_y = logit.predict(scaled_test_x)

# calculate accuracies
train_accuracy = np.sum((pred_train_y==scaled_train_y))/len(pred_train_y)
test_accuracy = np.sum((pred_test_y==scaled_test_y))/len(pred_test_y)

# print results
print("training accuracy = %.2f" % train_accuracy)
print("test accuracy = %.2f" % test_accuracy)
```

```
training accuracy = 1.00
test accuracy = 1.00
```

The accuracies for both training and test data are perfect.

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 [24]:
```
# bootstrapping code
def bootstrap_log_params(x, y, n):

    # intialize model and coefficient placeholder
    clf = LogisticRegression(C=100000, solver='newton-cg')
    bs_coefs = np.empty((n,x.shape[1]),dtype="float64")

    for i in range(n):

        if i%100 == 0:
            print("iteration = %i" % i)

        # get random row indices to sample and store in tmp matrix
        idx = np.random.randint(0,x.shape[0],size=x.shape[0])
        tmp_x = x[idx,:]
        tmp_y = y[idx]

        # fit model to tmp data and store coefs
        clf = clf.fit(tmp_x, tmp_y)
        bs_coefs[i,:] = clf.coef_

    return(bs_coefs)
```

In [25]:
```
# bootstrap resample data with replacement to get confidence interval
bootstrapped_params = bootstrap_log_params(scaled_train_x, scaled_train_y, 1000)
```

```
iteration = 0
iteration = 100
iteration = 200
iteration = 300
iteration = 400
iteration = 500
iteration = 600
iteration = 700
iteration = 800
iteration = 900
```

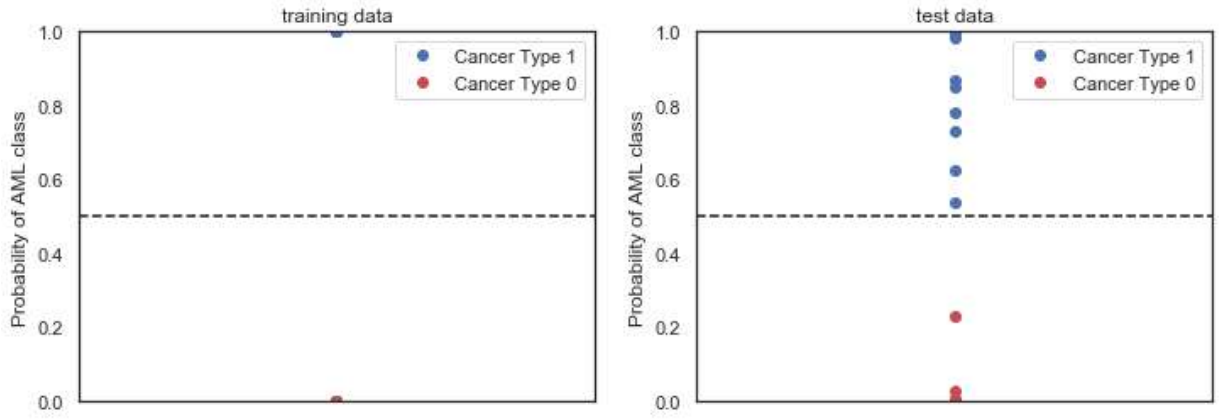
In [26]:
```
# calculate 95% confidence interval and see if zero is contained in it
lb, ub = np.percentile(bootstrapped_params, (2.5,97.5), axis=0)
is_significant = (lb < 0) & (ub > 0)
print("%i coefficients significantly different from zero" % np.sum(is_significant
```

```
5199 coefficients significantly different from zero
```

your answer here

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?

In [101]:
```
# initialize axes and visualize model results from Q3.1
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
ax = plt.gca()
visualize_prob(logit, scaled_train_x, scaled_train_y, ax)
plt.title("training data")
plt.subplot(1,2,2)
ax = plt.gca()
visualize_prob(logit, scaled_test_x, scaled_test_y, ax)
plt.title("test data");
```



The spread of probabilities is very different between the training and test data. The points in both sets fall on the correct side of the classification boundary. However, the training data points are all very close to p=0 or p=1 while many of test points are close to the p=0.5 classification boundary. What we can say about the points close to the boundary is that the p(AML) and p(ALL) are not very different, meaning that the model is less certain about the classification of those points. In general this difference between the two plots suggests that our model is fit less accurately to the test data than the training data, even though the accuracy score is identical between the two.

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.

The plot above suggests that our model is likely overfit to the training data, which is not particularly surprising considering our large value of the regularization parameter, C, and the high number of features relative to our number of data points. We should use the same strategies we generally use to handle overfitting: cross validation or regularization (via a smaller value of C). We could also try to reduce the number of features via dimensional reduction or feature selection through forward selection or lasso regression.

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 [136]:
```
# define num components from Q1.5
n_q15 = 38

# PCA with number of components from Q1.5
x_train = scaled_train.copy().drop(columns="Cancer_type", axis=1).values
x_test = scaled_test.copy().drop(columns="Cancer_type", axis=1).values
pca_q15 = PCA(n_components=n_q15)
pca_q15.fit(x)
x_train_q15 = pca.transform(x_train)[:,:n_q15]
x_test_q15 = pca.transform(x_test)[:,:n_q15]
clf_q15 = LogisticRegression(C=100000, solver='newton-cg')
clf_q15 = clf_q15.fit(x_train_q15, scaled_train_y)

# report training and test accuracies
print("Train accuracy: %.2f" % clf_q15.score(x_train_q15, scaled_train_y))
print("Train accuracy: %.2f" % clf_q15.score(x_test_q15, scaled_test_y))
```

```
Train accuracy: 1.00
Train accuracy: 1.00
```

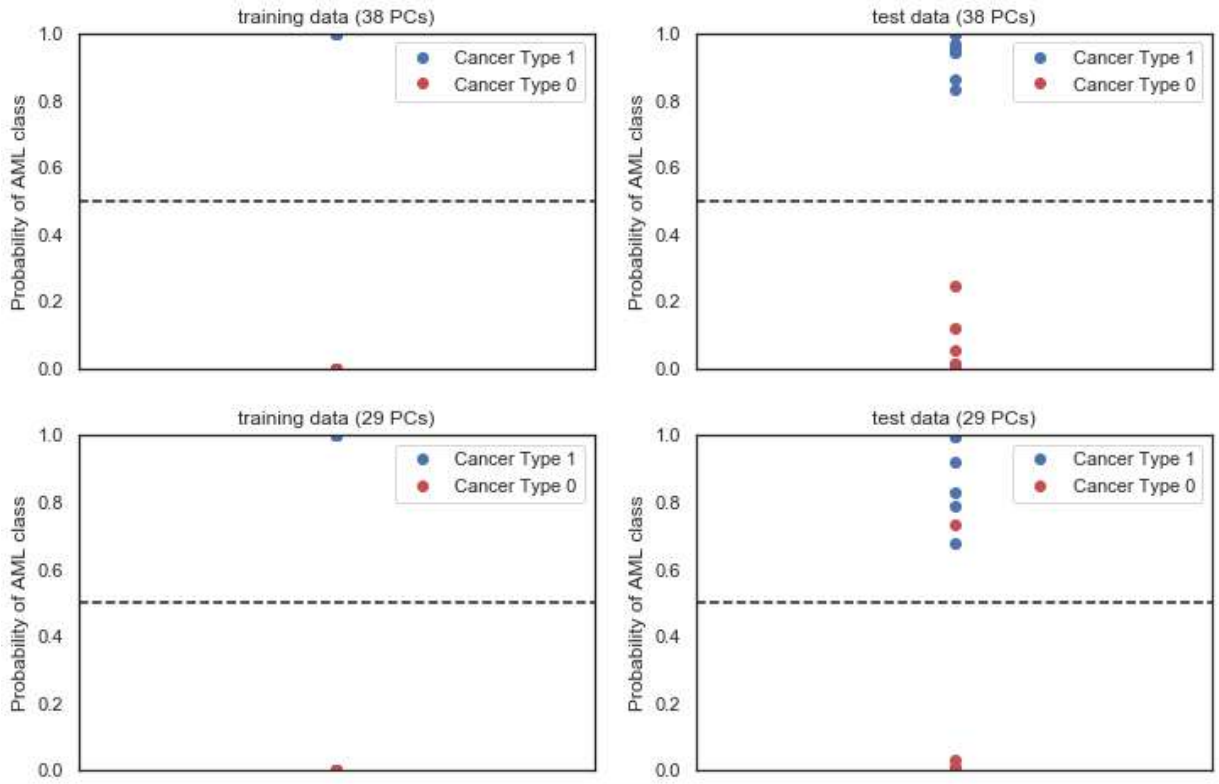
In [130]:
```
# PCA with number of components needed to capture 90% of variance
n_90 = 29
pca_v90 = PCA(n_components=n_90)
pca_v90.fit(x)
x_train_v90 = pca.transform(x_train)[:,:n_90]
x_test_v90 = pca.transform(x_test)[:,:n_90]
clf_v90 = LogisticRegression(C=100000, solver='newton-cg')
clf_v90 = clf_v90.fit(x_train_v90, scaled_train_y)

# report training and test accuracies
print("Train accuracy: %.2f" % clf_v90.score(x_train_v90, scaled_train_y))
print("Test accuracy: %.2f" % clf_v90.score(x_test_v90, scaled_test_y))
```

```
Train accuracy: 1.00
Train accuracy: 0.97
```

The model accuracies are very comparable to the accuracies obtained with the high dimensional data, with only a single misclassification on the test data.

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?

In [137]:
```
# initialize axes and visualize model results
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
visualize_prob(clf_q15, x_train_q15, scaled_train_y, plt.gca())
plt.title("training data (%i PCs)" % n_q15)
plt.subplot(2,2,2)
visualize_prob(clf_q15, x_test_q15, scaled_test_y, plt.gca())
plt.title("test data (%i PCs)" % n_q15)
plt.subplot(2,2,3)
visualize_prob(clf_v90, x_train_v90, scaled_train_y, plt.gca())
plt.title("training data (%i PCs)" % n_90)
plt.subplot(2,2,4)
visualize_prob(clf_v90, x_test_v90, scaled_test_y, plt.gca())
plt.title("test data (%i PCs)" % n_90);
```



Here PCA offers the benefit of simplifying the model without really negatively affecting the performance (predicted probabilities actually seem better with 38 PCs than the full data set). The lower dimensional representations capture most of the variance in the original data set, but removes the multicolinearity in the data since each principal component is guaranteed to be orthogonal to the others. Multicolinearity is a problem because it leads to higher variance in parameter estimates due to ambiguity in accurately assigning coefficient magnitudes of colinear predictors. In general this should produce more stability in the model since there is less error in our parameter estimates. In this particular case, we also have many more predictors than observations in high-dimensional model, which means that we don't have enough information to accurately estimate coefficients for each predictor.