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]:
```python
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 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 [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...

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

	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

Out[4]:
```
(40, 7130)
```

In [5]:
```python
def normalize(df_scale, df_fit=data_train, feature_range=(0,1)):
    fit_df = df_fit.loc[df_fit.index, df_fit.columns!='Cancer_type']
    scale_df = df_scale.loc[df_scale.index, df_scale.columns!='Cancer_type']
    scaler = MinMaxScaler().fit(fit_df)
    scaled_array = scaler.transform(scale_df)
    scaled_df = pd.DataFrame(scaled_array, index = scale_df.index, columns=scale_
    scaled_df['Cancer_type'] = df['Cancer_type']
    return(scaled_df)

train_normed = normalize(data_train)
test_normed = normalize(data_test)
display(train_normed.describe())
display(test_normed.describe())
```

	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	AFFX-CreX-5_at	Cre
count	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.
mean	0.640347	0.719472	0.369477	0.512253	0.529472	0.550653	0.654253	0.581803	0.
std	0.182889	0.186767	0.237206	0.243956	0.231075	0.214332	0.216589	0.228836	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.596530	0.631115	0.201744	0.325824	0.386853	0.408101	0.547153	0.484127	0.
50%	0.653025	0.745597	0.323256	0.553846	0.584052	0.542683	0.683986	0.634921	0.
75%	0.731762	0.844423	0.500000	0.720330	0.683728	0.713850	0.753381	0.739796	0.
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.

8 rows × 7130 columns

	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	AFFX-CreX-5_at	Cre
count	33.000000	33.000000	33.000000	33.000000	33.000000	33.000000	33.000000	33.000000	33.
mean	0.641055	0.733737	0.371388	0.409524	0.512409	0.479516	0.744937	0.606473	0.
std	0.165245	0.189658	0.335398	0.239065	0.298177	0.309424	0.174231	0.222876	0.
min	0.112100	0.174168	-0.562791	-0.026374	-0.096983	-0.198606	0.489680	-0.024943	0.
25%	0.560498	0.618395	0.255814	0.232967	0.312500	0.240418	0.620641	0.519274	0.
50%	0.683274	0.767123	0.374419	0.373626	0.566810	0.496516	0.716726	0.625850	0.
75%	0.749110	0.872798	0.504651	0.558242	0.698276	0.736934	0.839146	0.752834	0.
max	0.873665	1.013699	1.116279	0.923077	1.314655	0.954704	1.243416	0.970522	0.

8 rows × 7130 columns

Answer: *We do see a lot of variation in the minimum and maximum values for different variables in* *the raw training set which points to possible difference in measurement units for each of these gene types. So we use `MinMaxScaler()` and fit it on our training set and then transform both train and test set using that fit. We can see the output for normalized train and test sets above produces all variables in the range $(0, 1)$ for the train set and very closely in $(0, 1)$ for the test set.*

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

In [6]:
```
# your code here
train_normed.shape
```

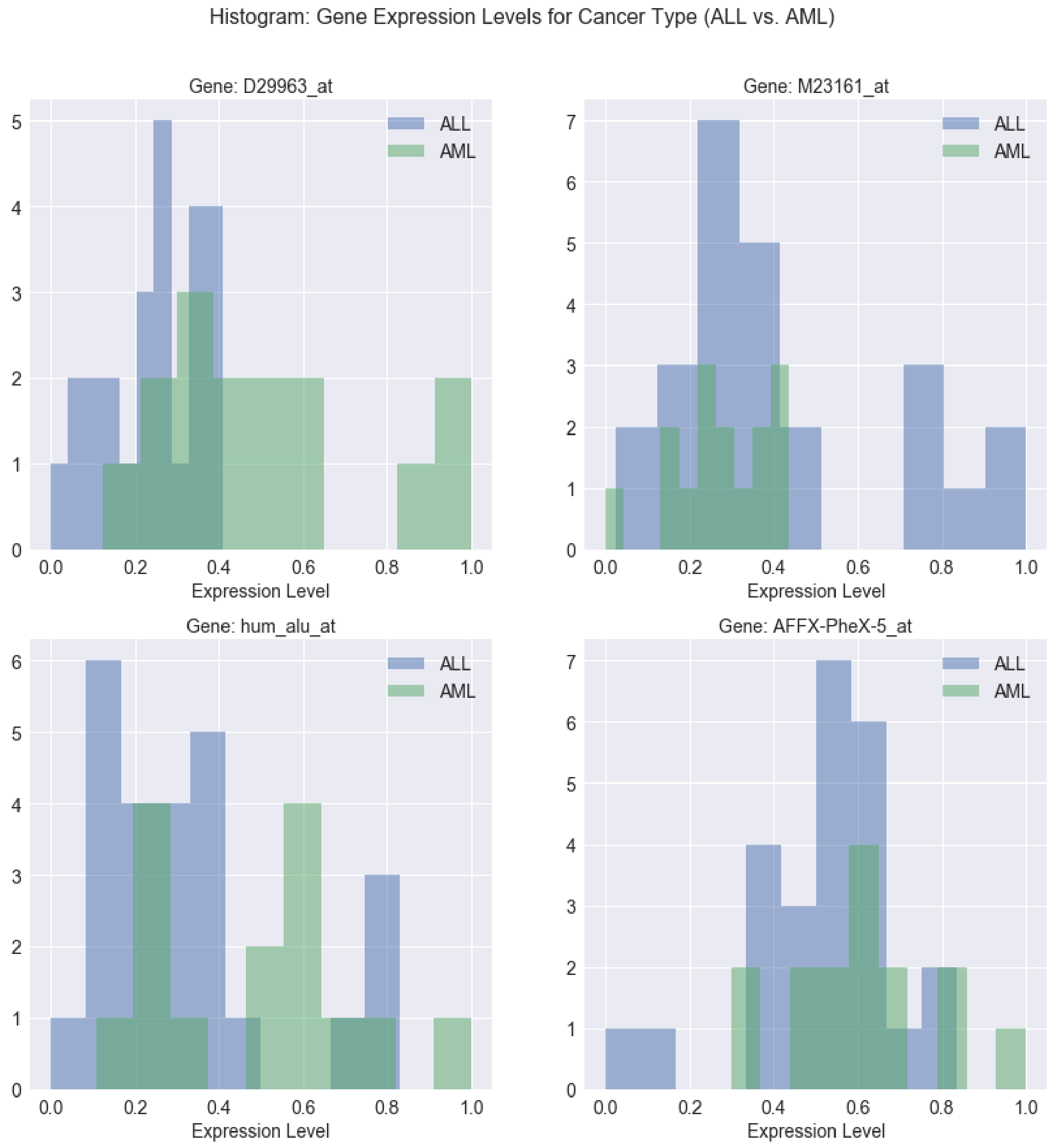
Out[6]:
```
(40, 7130)
```

Answer: *With $p > n$, there can be many solutions to find a model with zero error. Meaning we will get an R^2 of 1 on the train set. But that model will be overfit on the train set and that not all variables might be important for prediction. Thus, we need to find ways to reduce the dimensions and fit the classification model to just a few predictors.*

1.3: Let's explore a few of the genes...

In [7]:
```
# your code here
gene_plot = ['D29963_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5_at']

fig, ax = plt.subplots(2, 2, figsize = (14,14))
i=0
j=0
k=0
for gene in gene_plot:
    #ax[i,j].set_title('Gene: %s'%gene)
    for a in range(0,2):
        can_type = 'ALL' if a==0 else 'AML'
        ax[i,j].hist(train_normed[train_normed['Cancer_type']==a][gene], alpha=0.
    ax[i,j].set_xlabel('Expression Level', fontsize=14)
    ax[i,j].set_title('Gene: %s'%gene, fontsize=14)
    ax[i,j].legend(loc='best', fontsize=14)
    ax[i,j].tick_params(labelsize=14)
    if k in [1,3]:
        i=i+1
        j=j-1
    else: j=j+1
    k=k+1
fig.suptitle('Histogram: Gene Expression Levels for Cancer Type (ALL vs. AML)', y
```

Out[7]:
```
Text(0.5,0.95,'Histogram: Gene Expression Levels for Cancer Type (ALL vs. AM
L)')
```



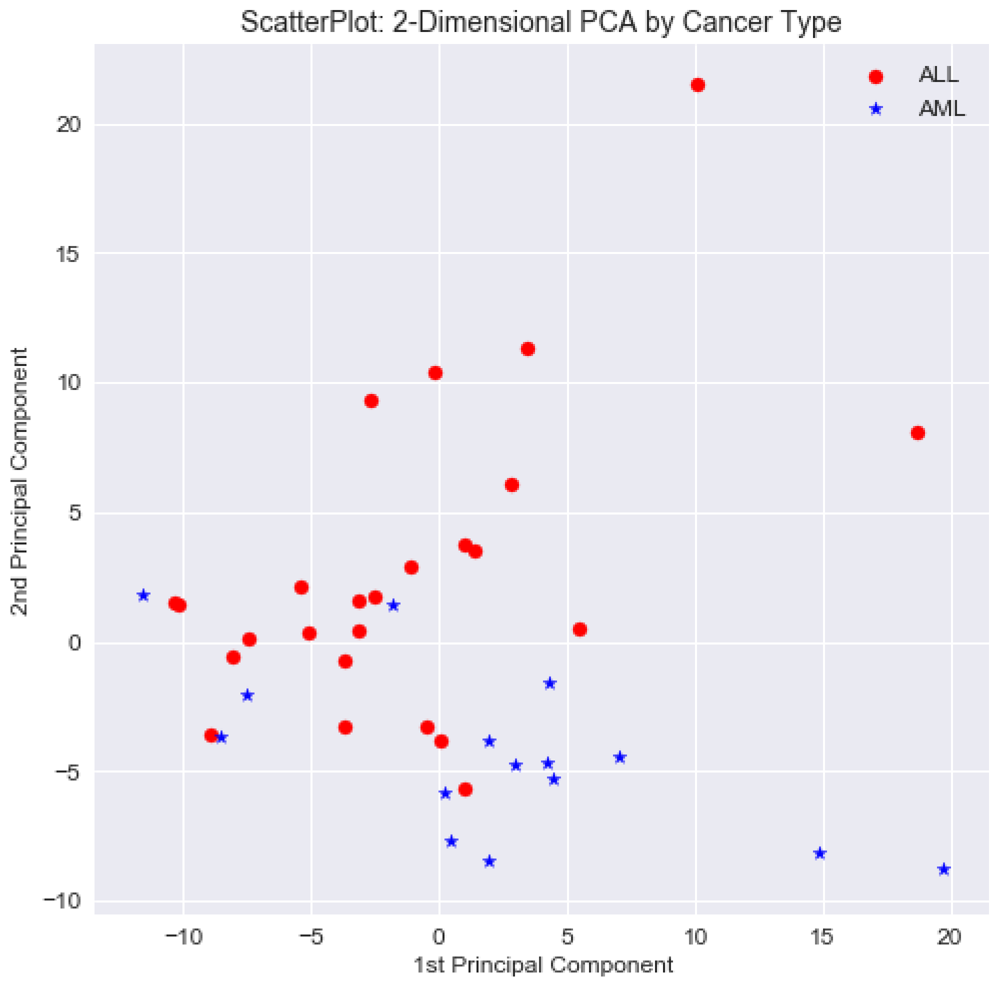
Answer: *We see that gene `D29963_at` and `hum_alu_at` seem to have a higher mean for AML as compared to ALL. Gene `M23161_at` seems to have lower mean for AML than ALL. It also has very low counts for AML compared to ALL. `AFFX-PheX-5_at` doesn't show any visible difference between the two classes.*

1.4: Since our data has dimensions that are not easily visualizable...

In [8]:
```
# your code here
X_train = train_normed.loc[train_normed.index, train_normed.columns!='Cancer_type
y_train = train_normed['Cancer_type']
X_test = test_normed.loc[test_normed.index, test_normed.columns!='Cancer_type']
y_test = test_normed['Cancer_type']

pca2d = PCA(2).fit(X_train)
x_train_2d = pd.DataFrame(pca2d.transform(X_train), index=X_train.index, columns=
x_test_2d = pd.DataFrame(pca2d.transform(X_test), index=X_test.index, columns=['P

a = np.cumsum(pca2d.explained_variance_ratio_)[1]
print(a)

can_type = ['ALL','AML']
marker = ['o','*']
color = ['red', 'blue']

fig_pc, ax_pc = plt.subplots(1,1, figsize=(8,8))
for a in range(0,2):
    ax_pc.scatter(x_train_2d.loc[y_train==a, x_train_2d.columns=='PCA1'], x_train_
                  label=can_type[a], marker=marker[a], color=color[a])
ax_pc.set_xlabel('1st Principal Component', fontsize=12)
ax_pc.set_ylabel('2nd Principal Component', fontsize=12)
ax_pc.set_title('ScatterPlot: 2-Dimensional PCA by Cancer Type', fontsize=14)
ax_pc.tick_params(labelsize=12)
ax_pc.legend(loc='best', fontsize=12)
```

```
0.2731782945208861
```

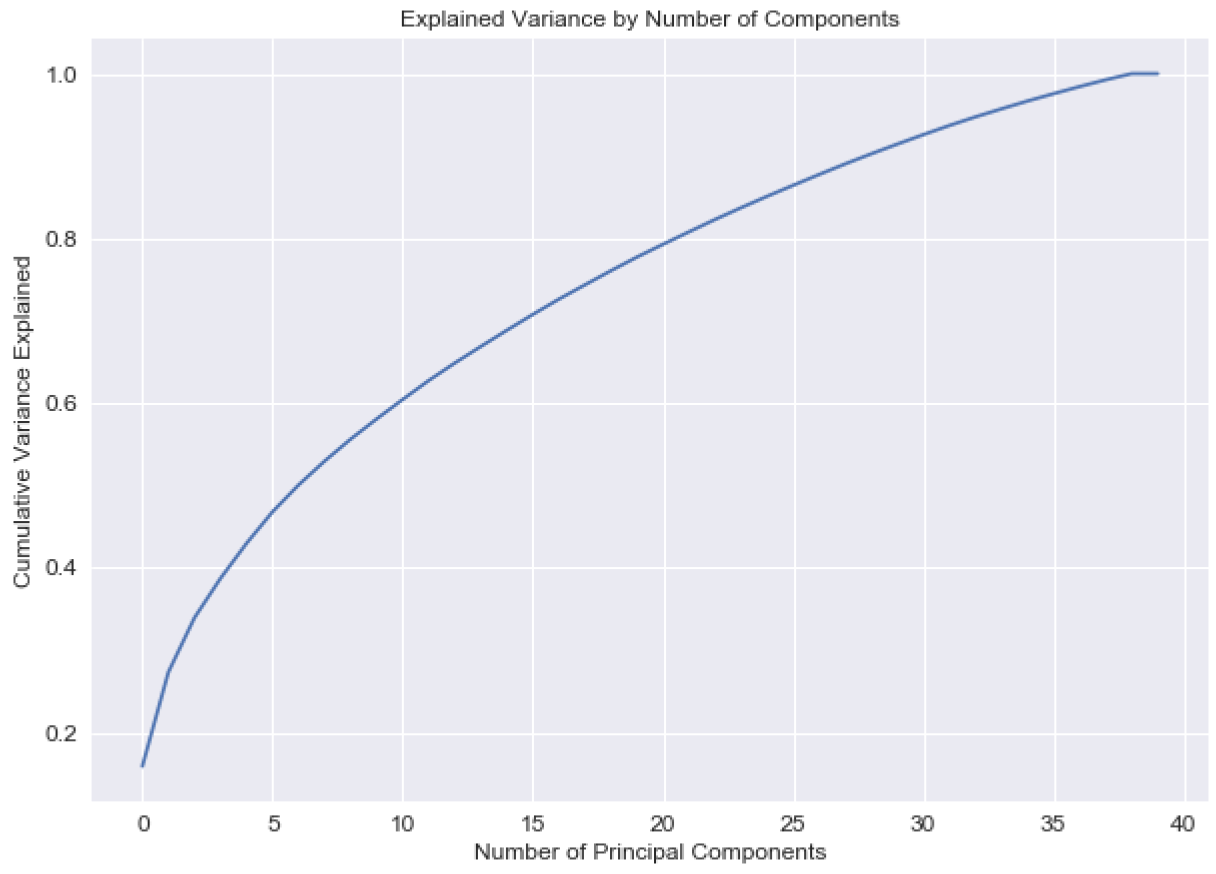
Out[8]:
```
<matplotlib.legend.Legend at 0xef9539c828>
```



Answer: *The* $1st$ *two principal components explain 27% of the total variance in the entire predictor set. We can also see that the two principal components show a fairly clear discrimination between the two* `Cancer_type` *classes. We could somewhat imagine a classification boundary with a slope at 45 degree angle and intersecting the* $1st$ *component axis at almost 0.*

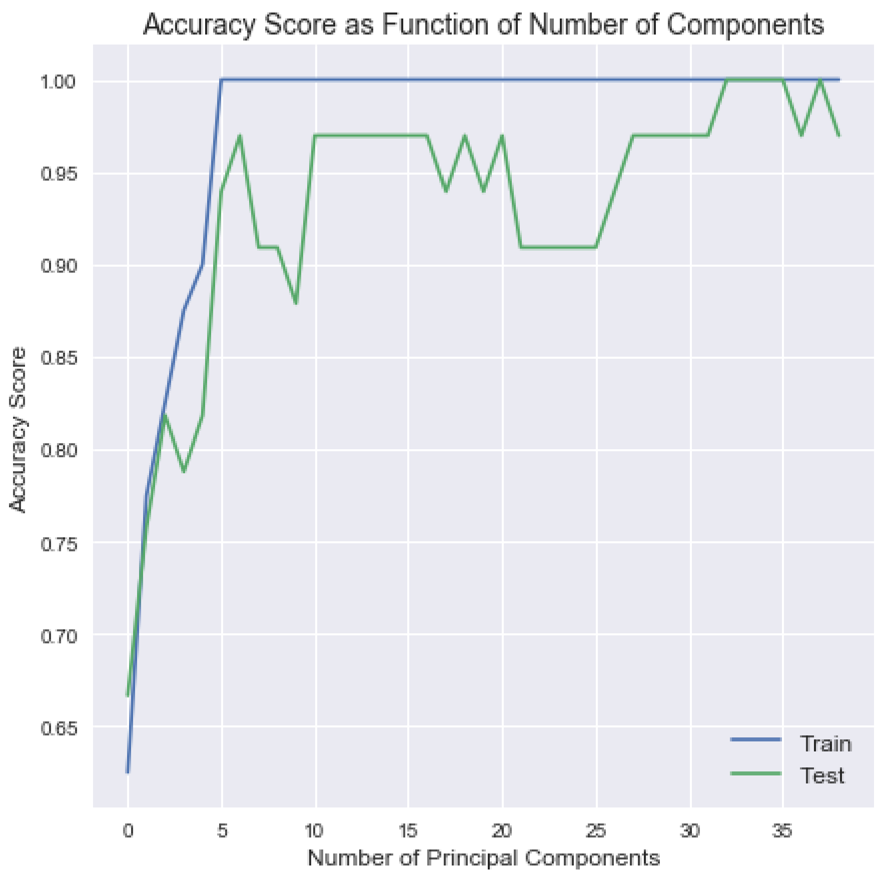
1.5: Plot the cumulative variance explained in the feature set...

In [9]:
```
# your code here
pca_cols = []
for i in range(1,41):
    pca_cols.append('PCA%s'%i)

pca40d = PCA(40).fit(X_train)
x_train_40d = pd.DataFrame(pca40d.transform(X_train), index=X_train.index, column
x_test_40d = pd.DataFrame(pca40d.transform(X_test), index=X_test.index, columns=p

var_exp40 = np.cumsum(pca40d.explained_variance_ratio_)

fig_40, ax_40 = plt.subplots(1,1, figsize=(10,7))
ax_40.plot(var_exp40)
ax_40.set_xlabel('Number of Principal Components', fontsize=12)
ax_40.set_ylabel('Cumulative Variance Explained', fontsize=12)
ax_40.set_title('Explained Variance by Number of Components', fontsize=12)
ax_40.tick_params(labelsize=12)
```



In [10]:
```
# plot for train and test accuracies to determine optimum number of components
train_acc = []
test_acc = []
for i in range(1, len(x_train_40d.columns)):
    log_reg1 = LogisticRegression(C=100000).fit(x_train_40d[x_train_40d.columns[:
    y_train_pred = log_reg1.predict(x_train_40d[x_train_40d.columns[:i]])
    y_test_pred = log_reg1.predict(x_test_40d[x_test_40d.columns[:i]])
    train_acc.append(accuracy_score(y_train, y_train_pred))
    test_acc.append(accuracy_score(y_test, y_test_pred))

fig, ax = plt.subplots(1,1, figsize=(7,7))
ax.plot(train_acc, label='Train')
ax.plot(test_acc, label='Test')
ax.set_xlabel('Number of Principal Components', fontsize=12)
ax.set_ylabel('Accuracy Score', fontsize=12)
ax.set_title('Accuracy Score as Function of Number of Components', fontsize=14)
ax.legend(loc='best', fontsize=12)
```

Out[10]:
```
<matplotlib.legend.Legend at 0xef92683e10>
```



In [11]:
```
pc_select = x_train_40d.columns[:7]
var_select = var_exp40[:7]
display(pc_select)
display(var_select)
```

```
Index(['PCA1', 'PCA2', 'PCA3', 'PCA4', 'PCA5', 'PCA6', 'PCA7'], dtype='object')
```

```
array([0.15889035, 0.27317829, 0.33914119, 0.3870151 , 0.42957185,
       0.46789439, 0.50053032])
```

In [12]:
```
var_at_90 = var_exp40[var_exp40>0.9][0]
var_90 = var_exp40[var_exp40<=var_at_90]
display(var_90)
len(var_90)
```

```
array([0.15889035, 0.27317829, 0.33914119, 0.3870151 , 0.42957185,
       0.46789439, 0.50053032, 0.52950387, 0.55619709, 0.58074806,
       0.60440367, 0.62741071, 0.64863686, 0.66873622, 0.68841059,
       0.70768316, 0.72622259, 0.74356872, 0.76077827, 0.7773256 ,
       0.79281807, 0.80798491, 0.82297995, 0.83725127, 0.85111525,
       0.86445647, 0.87761542, 0.89045139, 0.90268704])
```

Out[12]:
```
29
```

Answer: Notice that we are only able to get 40 components since the explained variance ratio is always going to be 1 after 40 components, as we have only 40 observations. *Even though the variance explained by principal components is independent of the response variable, 2 components might not be enough since we can see that the train and test accuracies continue to increase after that. However, we see that the first 7 components give us the highest train and test accuracies until we increase the number of components to 33. So, although the first 7 components don't give us the highest possible test accuracy, the accuracy score is still very good (which is what we would want in a case where we are trying to predict a life threatening disease) and seems to plateau after that.*

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

In [13]:
```
# your code here
lm_reg = OLS(y_train, sm.add_constant(X_train['D29963_at']))
OLSModel = lm_reg.fit()
display(OLSModel.summary())

fig, ax = plt.subplots(1,1,figsize=(7,7))
ax.hist(OLSModel.predict(sm.add_constant(X_train['D29963_at'])))
ax.set_xlabel('Predicted Values', fontsize=12)
ax.set_title('Histogram for Predicted Values (OLS)', fontsize=14)
ax.tick_params(labelsize=12)
```

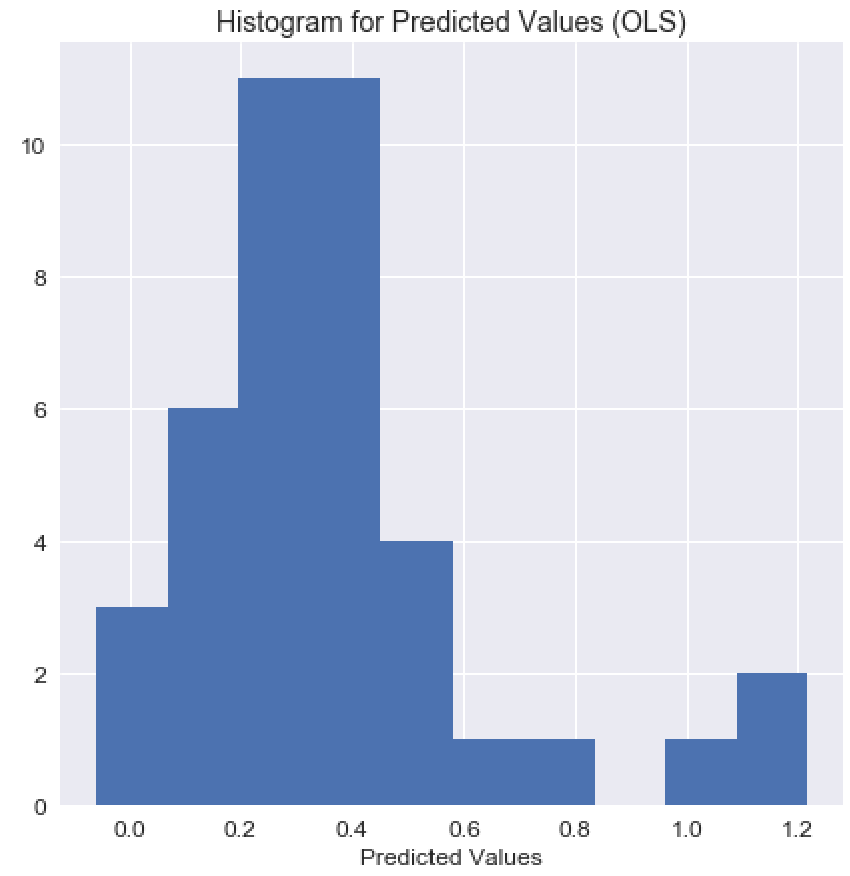
OLS Regression Results

Dep. Variable:	Cancer_type	**R-squared:**	0.329
Model:	OLS	**Adj. R-squared:**	0.311
Method:	Least Squares	**F-statistic:**	18.61
Date:	Wed, 24 Oct 2018	**Prob (F-statistic):**	0.000110
Time:	22:26:52	**Log-Likelihood:**	-19.769
No. Observations:	40	**AIC:**	43.54
Df Residuals:	38	**BIC:**	46.92
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.0587	0.119	-0.492	0.625	-0.300	0.183
D29963_at	1.2764	0.296	4.314	0.000	0.677	1.875

Omnibus:	5.833	**Durbin-Watson:**	0.838
Prob(Omnibus):	0.054	**Jarque-Bera (JB):**	5.011
Skew:	0.775	**Prob(JB):**	0.0816
Kurtosis:	2.222	**Cond. No.**	5.15

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.



Answer: *We can clearly see that the predicted values go outside the range of* $(0, 1)$*. This is because the model is trying to formulate a unit change in the response variable in terms of unit change in the gene* `D29963_at` *, meaning that its treating our response variable as quantitative. Also, as we get closer to 1 or 0, increase or decrease in the gene expression level might not yield us much different results in reality but the model will still give it a significance even if our response goes beyond* $(0, 1)$*.*

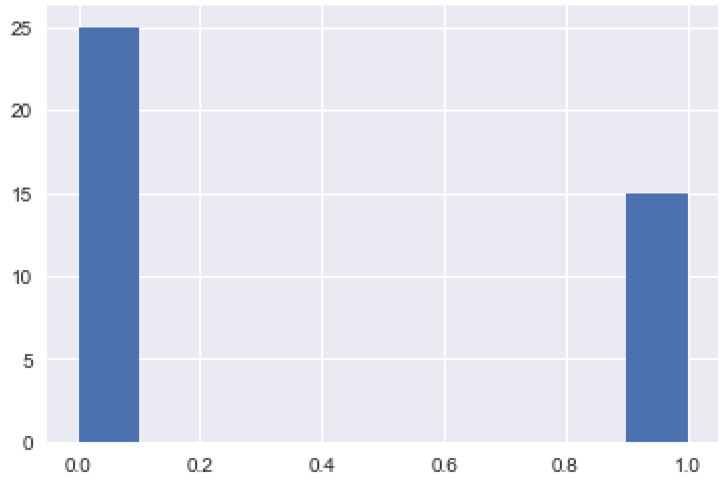
2.2: The fitted linear regression model can be converted to a classification model...

In [14]:
```
# your code here
y_train_pred = OLSModel.predict(sm.add_constant(X_train['D29963_at']))
y_test_pred = OLSModel.predict(sm.add_constant(X_test['D29963_at']))

y_train_pred[y_train_pred>=0.5] = 1
y_train_pred[y_train_pred<0.5] = 0
y_test_pred[y_test_pred>=0.5] = 1
y_test_pred[y_test_pred<0.5] = 0

print('Train Set Accuracy Score for OLS: %s'%accuracy_score(y_train, y_train_pred
print('Test Set Accuracy Score for OLS: %s'%accuracy_score(y_test, y_test_pred))
```

```
Train Set Accuracy Score for OLS: 0.8
Test Set Accuracy Score for OLS: 0.7575757575757576
```

In [15]:
```
plt.hist(y_train)
```

Out[15]:
```
(array([25.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 15.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



Answer: *We can see in the quick histogram that we created for* `y_train` *(and the histogram of the predicted values in 2.1), that the spread between the two Cancer types is not 80/20. So our model is not classifying everything as one or the other type. Also, the test accuracy is 75% which is fairly close to train accuracy, considering we are only using one predictor.*

2.3: Next, fit a simple logistic regression model to the training set...

In [16]:
```
# your code here
logreg = LogisticRegression(C=100000, solver='newton-cg')
logreg_model = logreg.fit(X_train[['D29963_at']], y_train)

print('Train Set Accuracy Score for Logistic Regression: %s'%accuracy_score(y_tra
print('Test Set Accuracy Score for Logistic Regression: %s'%accuracy_score(y_test
```

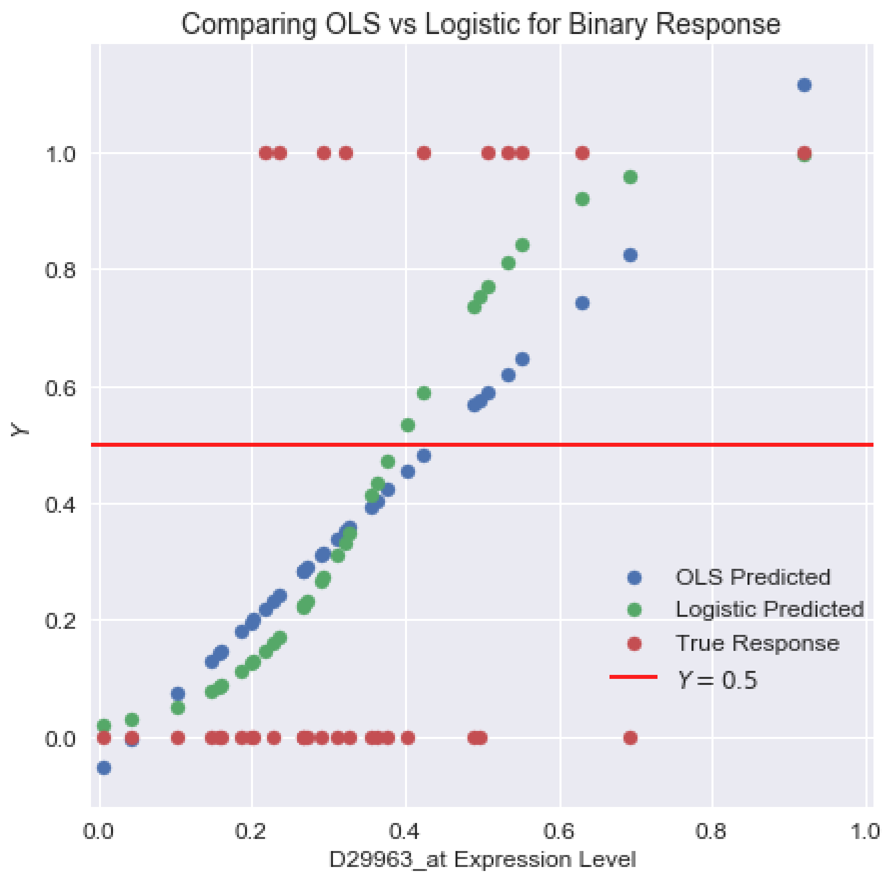
```
Train Set Accuracy Score for Logistic Regression: 0.8
Test Set Accuracy Score for Logistic Regression: 0.7575757575757576
```

Answer: *The train and test accuracy scores are exactly the same using logistic regression and linear regression. This is because, the logistic regression only acts as a function to make a sigmoidal curve out of a line so that the values don't go beyond $(0, 1)$. Essentially, in logistic regression we predict the* `Log-odds` *of $Y = 1$. Hence, with same set of predictors, we will always get the same accuracy for linear and logistic regression after converting our predicted response for linear regression to binary (as done in 2.2).*

2.4: Create a figure with 4 items ...

In [17]:
```
# your code here
fig, ax = plt.subplots(1,1, figsize=(7,7))
ax.scatter(X_test[['D29963_at']], OLSModel.predict(sm.add_constant(X_test['D29963_
ax.scatter(X_test[['D29963_at']], pd.DataFrame(logreg_model.predict_proba(X_test[
ax.scatter(X_test[['D29963_at']], y_test, label='True Response')
ax.hlines(y=0.5, color='red', xmin=-0.01, xmax=1.01, label='$Y=0.5$')
ax.set_xlabel('D29963_at Expression Level', fontsize=12)
ax.set_ylabel('$Y$', fontsize=12)
ax.set_xlim(-0.01,1.01)
ax.set_title('Comparing OLS vs Logistic for Binary Response', fontsize=14)
ax.tick_params(labelsize=12)
ax.legend(loc=(0.65,0.14), fontsize=12)
```

Out[17]:
```
<matplotlib.legend.Legend at 0xef93fa4080>
```



Answer: *We can see that, for binary response, the logistic regression model is the better suited one as it gives us values within the range of $(0, 1)$. It is also easier to interpret logistic regression as the rate of increase or decrease in the* `Log-odds` *goes down as we approach closer to $(0, 1)$ unlike in linear regression.*

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 [18]:
```
#--------  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...

In [19]:
```
# your code here
log_reg_full = LogisticRegression(C=100000)
log_model_full = log_reg_full.fit(X_train, y_train)
print('Train Set Accuracy Score for Logistic Regression: %s'%accuracy_score(y_tra
print('Test Set Accuracy Score for Logistic Regression: %s'%accuracy_score(y_test
```

```
Train Set Accuracy Score for Logistic Regression: 1.0
Test Set Accuracy Score for Logistic Regression: 1.0
```

Answer: *The accuracy scores for model with all predictors is clearly higher than that for the model with just one predictor in Question 2. However, we are surely getting an overfit model with all predictors included, as the number of predictors is greater than the number of observations.*

3.2: How many of the coefficients estimated by this multiple logistic regression...

In [20]:
```
# bootstrapping code
# your code here
N=1000
betas_boot = [[np.nan]*len(X_train.columns)]*1000
for i in range(N):
    sample_indices = np.random.choice(list(X_train.index), size=X_train.shape[0],

    X_train_resample = X_train.loc[sample_indices,:]
    y_train_resample = y_train[sample_indices]

    bootstrap_model = LogisticRegression(C=100000).fit(X_train_resample, y_train_
    cur_betas = list(bootstrap_model.coef_[0])
    betas_boot[i] = cur_betas
```

In [21]:
```
betas_bs_df = pd.DataFrame(betas_boot, columns=X_train.columns)
np.percentile(betas_bs_df['D29963_at'],q=[2.5, 97.5])
```

Out[21]:
```
array([0.03380448, 0.06854877])
```

In [22]:
```
sig_betas = []
for i in betas_bs_df.columns:
    pct = np.percentile(betas_bs_df[i],q=[2.5, 97.5])
    if pct[0]>0 or pct[1]<0:
        sig_betas.append(i)

len(sig_betas)
```

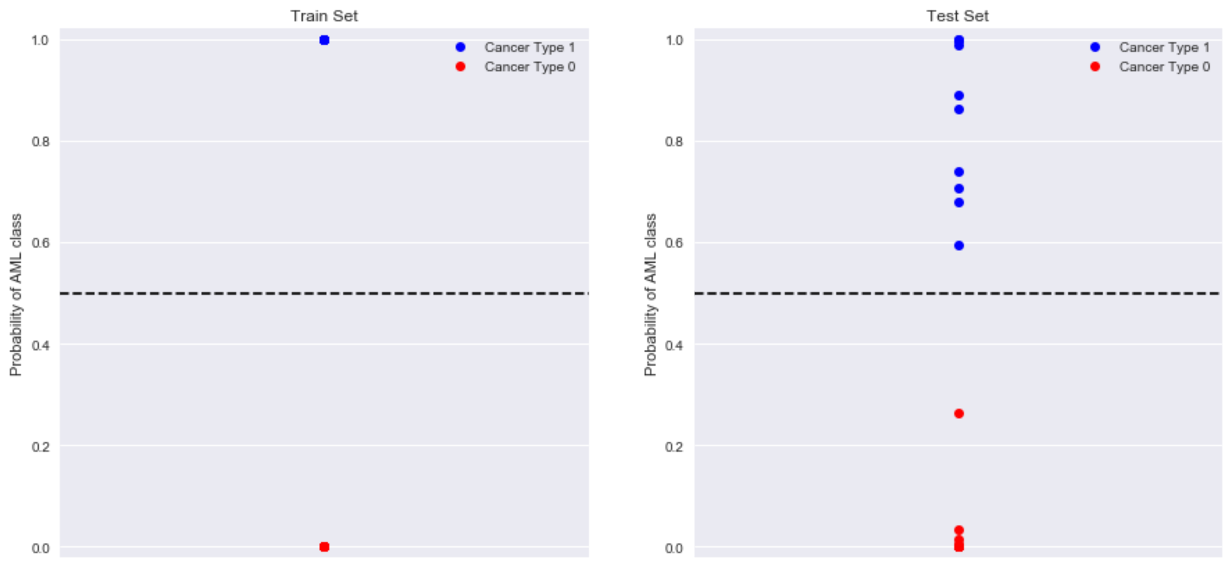
Out[22]:
```
1885
```

Answer: *Only 1885 variables have a confidence interval, at 5% significance, that does not include zero. This means that other predictors contain zero in their 95% confidence interval and are not significantly different.*

3.3: Use the visualize_prob function provided below ...

In [23]:
```python
""" Plot classification model """

# your code here
fig1, ax1 = plt.subplots(1, 2, figsize=(15,7))
visualize_prob(log_model_full, X_train, y_train, ax=ax1[0])
visualize_prob(log_model_full, X_test, y_test, ax=ax1[1])
ax1[0].set_ylim(ymin=-0.02, ymax=1.02)
ax1[0].set_title('Train Set', fontsize=12)
ax1[1].set_ylim(ymin=-0.02, ymax=1.02)
ax1[1].set_title('Test Set', fontsize=12)
```

Out[23]:
```
Text(0.5,1,'Test Set')
```



Answer: *The predicted probabilities in the training set are always either 0 or 1, which again points to the fact that the model is being overfit on the train set. However, we see the predicted probabilities are more spread out in the test set even though the classification accuracy is the same. So the points that are near the 0.5 threshold have less probability to be in their respective class and could be in a different class if the threshold were to change from 0.5 to something else.*

3.4: Open question: Comment on the classification accuracy...

Answer: *Due to a very high number of predictors compared to the number of observations, we have too many solutions that give us zero error. Thus, with predicted probabilities either exactly the same as true values, the model is overfit to our training set. However, our classification accuracy mainly depends on the predicted probabilities using our model on the test set and the threshold we are going to use to create a binary response. We also see with bootstrapping that not all betas are significant and that most of them can be eliminated.*

We should try to look at the variables that are signficantly different than zero using bootstrap and find the principal components of just these predictors. Then select the principal components that make most sense, fit a logistic regression model with K-fold cross-validation to train set and then get test set accuracies. Also, more data points could help address issues with dimensionality too. It would also be interesting to see the results at different thresholds.

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...

In [24]:
```
# your code here

log_reg1 = LogisticRegression(C=100000).fit(x_train_40d[pc_select], y_train)
log_reg2 = LogisticRegression(C=100000).fit(x_train_40d[pca_cols[:len(var_90)]], 

accuracy = dict()

accuracy[r'$Selected Components$'] = [np.nan]*2
accuracy[r'$Var Explained = 0.9$'] = [np.nan]*2

results = pd.DataFrame(accuracy)
results.rename({0: 'Accuracy on Train Set', 1: 'Accuracy on Test Set'}, inplace=T

sc_select_train = accuracy_score(y_train, log_reg1.predict(x_train_40d[pc_select]
sc_select_test = accuracy_score(y_test, log_reg1.predict(x_test_40d[pc_select]))
sc_var90_train = accuracy_score(y_train, log_reg2.predict(x_train_40d[pca_cols[:l
sc_var90_test = accuracy_score(y_test, log_reg2.predict(x_test_40d[pca_cols[:len(

results[r'$Selected Components$'][:] = [sc_select_train, sc_select_test]
results[r'$Var Explained = 0.9$'][:] = [sc_var90_train, sc_var90_test]

results
```

Out[24]:

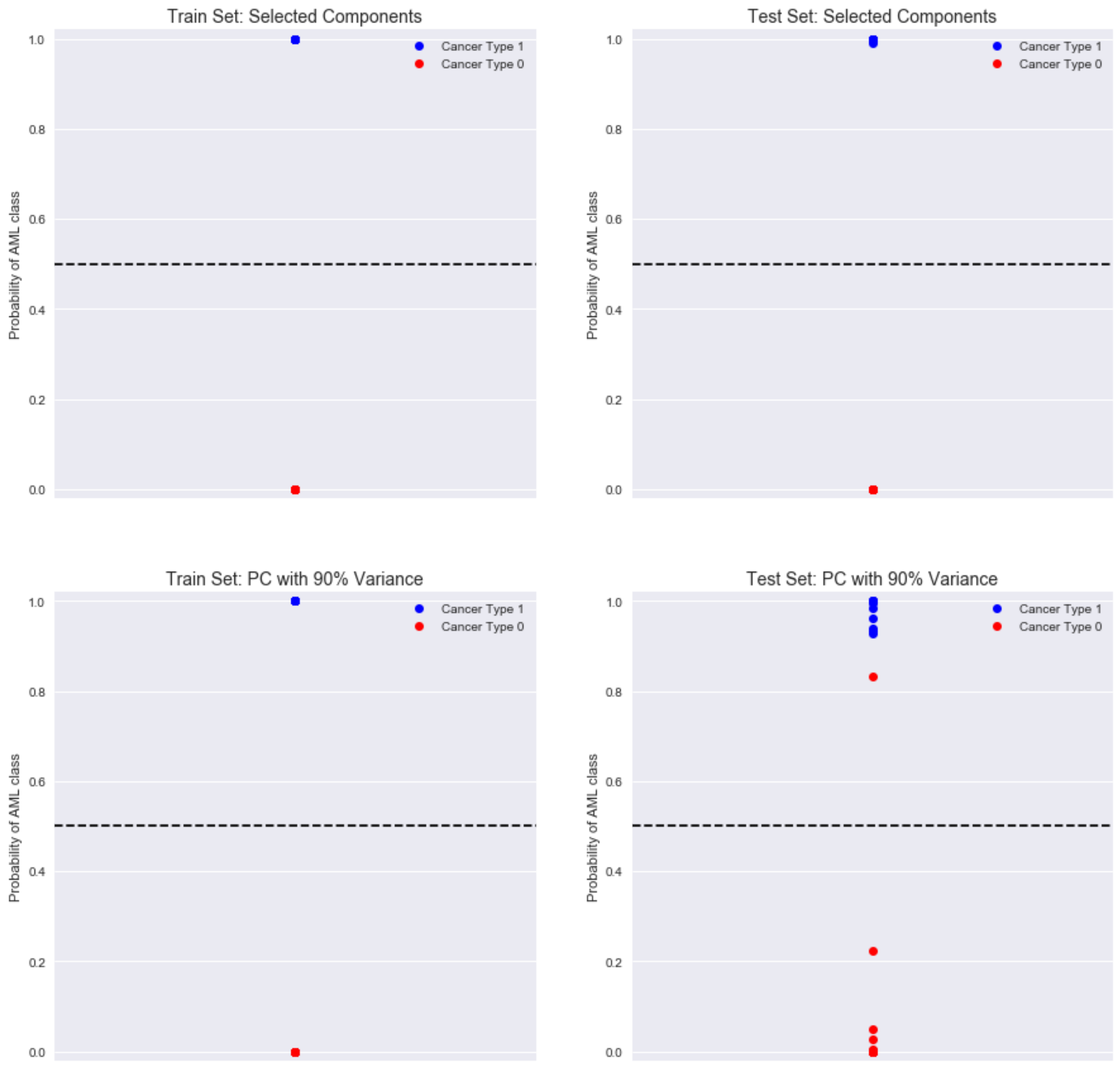
	$Selected Components$	$Var Explained = 0.9$
Accuracy on Train Set	1.000000	1.000000
Accuracy on Test Set	0.969697	0.969697

Answer: *The accuracies for the train set are the same as that in question 3 and the accuracies for test set are slightly less, but still significantly high. Which means that even though we are getting zero error on our training set, there is a greater chance that the model can be used to generalize on additional data with much less number of dimensions compared to the full model in question 3.* **Note** *that we get the same test accuracy with much less (7) components as with components that explain 90% of varaince (29).*

4.2: Use the code provided in question 3...

In [25]:
```python
# your code here
fig1, ax1 = plt.subplots(2, 2, figsize=(15,15))
visualize_prob(log_reg1, x_train_40d[pc_select], y_train, ax=ax1[0,0])
visualize_prob(log_reg1, x_test_40d[pc_select], y_test, ax=ax1[0,1])
visualize_prob(log_reg2, x_train_40d[pca_cols[:len(var_90)]], y_train, ax=ax1[1,0
visualize_prob(log_reg2, x_test_40d[pca_cols[:len(var_90)]], y_test, ax=ax1[1,1])
ax1[0,0].set_ylim(ymin=-0.02, ymax=1.02)
ax1[0,0].set_title('Train Set: Selected Components', fontsize=14)
ax1[0,1].set_ylim(ymin=-0.02, ymax=1.02)
ax1[0,1].set_title('Test Set: Selected Components', fontsize=14)
ax1[1,0].set_ylim(ymin=-0.02, ymax=1.02)
ax1[1,0].set_title('Train Set: PC with 90% Variance', fontsize=14)
ax1[1,1].set_ylim(ymin=-0.02, ymax=1.02)
ax1[1,1].set_title('Test Set: PC with 90% Variance', fontsize=14)
```

Out[25]:
```
Text(0.5,1,'Test Set: PC with 90% Variance')
```



Answer: *The spread of probabilities is narrower for both the models in question 4 as compared to the one in question 3. Lower dimensionality means that there is a unique solution to find the minimum error with just 7 components and that our model is more interpretable. It is also faster to fit* *and test a model with lower dimensions and PCA can also help address some issues arising due to multi-collinearity in the original predictor set.*

```
In [ ]:
```