cs109a_hw5_109_submit

October 25, 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

In [2]:
```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/master
HTML(styles)
```

Out[2]:
```
<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.

Homework Group #16

In [3]:
```
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 [4]: np.random.seed(9002)
        df = pd.read_csv('https://raw.githubusercontent.com/michelkana/cs109a/master/dataset_hw5
        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 [5]: def normalize_columns(df, cols, scaler):
            df_copy = df.copy()
            df_copy[cols] = scaler.transform(df[cols])
            return df_copy
```

```
In [6]: response_col = 'Cancer_type'
        predictors = data_train.columns.difference([response_col])
        scaler = MinMaxScaler().fit(data_train[predictors])
        data_train_scaled = normalize_columns(data_train, predictors, scaler)
        data_test_scaled = normalize_columns(data_test, predictors, scaler)
```

```
In [7]: # separate into predictors and response
        ## train
        y_train = data_train_scaled[response_col]
        X_train_scaled = data_train_scaled[predictors]
        ## test
        y_test = data_test_scaled[response_col]
        X_test_scaled = data_test_scaled[predictors]
```

```
In [6]: # verify transformation
        X_train_scaled.describe()
```

```
Out[6]:        A28102_at  AB000114_at  AB000115_at  AB000220_at  AB000381_s_at  \
        count  40.000000    40.000000    40.000000    40.000000      40.000000
        mean    0.225047     0.379253     0.133660     0.377114       0.614879
        std     0.184174     0.239350     0.217092     0.198466       0.210569
```

```
min     0.000000     0.000000     0.000000     0.000000       0.000000
25%     0.089725     0.235477     0.026899     0.272388       0.482794
50%     0.193655     0.356846     0.056630     0.353234       0.617409
75%     0.313684     0.508299     0.098749     0.404229       0.736842
max     1.000000     1.000000     1.000000     1.000000       1.000000

       AB000409_at  AB000410_s_at  AB000449_at  AB000450_at  AB000460_at  \
count    40.000000      40.000000    40.000000    40.000000    40.000000
mean      0.338714       0.642135     0.232975     0.338544     0.366077
std       0.187751       0.260907     0.215510     0.233942     0.211425
min       0.000000       0.000000     0.000000     0.000000     0.000000
25%       0.238484       0.515249     0.089877     0.176699     0.245512
50%       0.304439       0.681380     0.182209     0.317476     0.353501
75%       0.402010       0.857143     0.301227     0.407282     0.438241
max       1.000000       1.000000     1.000000     1.000000     1.000000

          ...      Z84721_cds2_at  Z84722_at  Z86000_at  Z93784_at  \
count     ...           40.000000  40.000000  40.000000  40.000000
mean      ...            0.301265   0.724295   0.307447   0.283810
std       ...            0.222715   0.160970   0.206939   0.199026
min       ...            0.000000   0.000000   0.000000   0.000000
25%       ...            0.124274   0.676128   0.190579   0.166584
50%       ...            0.255927   0.750353   0.273651   0.213722
75%       ...            0.403569   0.801128   0.390149   0.316881
max       ...            1.000000   1.000000   1.000000   1.000000

       Z94753_s_at  Z95624_at  Z96810_at  Z97054_xpt2_at  Z97074_at  \
count    40.000000  40.000000  40.000000       40.000000  40.000000
mean      0.390125   0.375121   0.411335        0.397629   0.408433
std       0.215197   0.214607   0.234657        0.236164   0.218074
min       0.000000   0.000000   0.000000        0.000000   0.000000
25%       0.246865   0.211353   0.260593        0.236676   0.245719
50%       0.332288   0.335749   0.389831        0.352755   0.422945
75%       0.464734   0.555556   0.550847        0.509711   0.525257
max       1.000000   1.000000   1.000000        1.000000   1.000000

       hum_alu_at
count   40.000000
mean     0.387442
std      0.246407
min      0.000000
25%      0.207247
50%      0.333076
75%      0.609126
max      1.000000

[8 rows x 7129 columns]
```

The variance is indeed very large accross genes. After scaling this issue is solved.

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

```
In [7]: X_train_scaled.head()
```

```
Out[7]:      A28102_at  AB000114_at  AB000115_at  AB000220_at  AB000381_s_at  \
         0     0.095644     0.531120     0.097688     0.348259       0.554656
         2     0.064394     0.298755     0.058046     0.363184       0.639676
         5     0.292614     0.282158     0.052383     0.343284       0.404858
         9     0.053977     0.634855     0.051439     0.233831       0.951417
         10    0.033144     0.435685     0.058046     0.233831       0.761134

             AB000409_at  AB000410_s_at  AB000449_at  AB000450_at  AB000460_at  \
         0       0.041876       0.524880     0.047853     0.357282     0.309515
         2       0.490787       0.857143     0.359509     0.357282     0.295512
         5       0.401173       0.455859     0.171779     0.100971     0.234470
         9       0.396985       0.060995     0.426994     0.943689     0.856373
         10      0.317420       0.897271     0.180368     0.318447     0.180610

                ...      Z84721_cds2_at  Z84722_at  Z86000_at  Z93784_at  Z94753_s_at  \
         0      ...            0.197924   0.803244   0.223612   0.426456     0.592476
         2      ...            0.076476   0.747532   0.104769   0.475814     0.451411
         5      ...            0.125161   0.769394   0.107115   0.176703     0.253918
         9      ...            0.266351   1.000000   0.886630   0.656466     0.927900
         10     ...            0.000000   0.700282   0.117279   0.120434     0.376176

             Z95624_at  Z96810_at  Z97054_xpt2_at  Z97074_at  hum_alu_at
         0    0.463768   0.000000        0.272809   0.547945    0.166547
         2    0.222222   0.326271        0.317977   0.407534    0.225260
         5    0.178744   0.182203        0.199639   0.476027    0.398496
         9    0.314010   0.266949        0.702800   0.453767    0.145881
         10   0.062802   0.656780        0.412827   0.248288    0.329248

         [5 rows x 7129 columns]
```

High numbers of predictors make things harder in three key ways:

- Matrices are often not invertable
- There's a bigger chance of multicollinearity
- There's a danger of overfitting

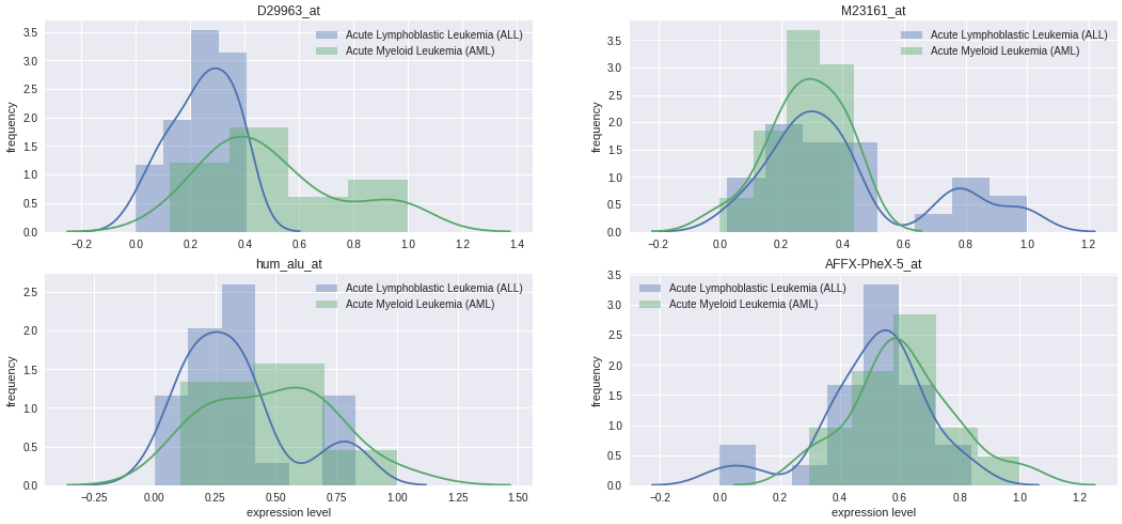
There are 7130 predictors and only 40 samples. The classification model might be unidentifiable. Regularization and PCA could be required.

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

```
In [0]: def plot_genes_histograms(genes_list, df):
            count = len(genes_list)
            if (count % 2 != 0) | (count<2):
                return
```

```
    genes_arr = np.array(genes_list).reshape(-1, 2)
    fig, ax = plt.subplots(count//2, 2, figsize=(18, 4*count//2))
    for (i,j), gene in np.ndenumerate(genes_arr):
        sns.distplot(df[df['Cancer_type'] == 0][gene],
                     label='Acute Lymphoblastic Leukemia (ALL)', ax=ax[i][j])
        sns.distplot(df[df['Cancer_type'] == 1][gene],
                     label='Acute Myeloid Leukemia (AML)', ax=ax[i][j])
        ax[i][j].set_title(gene)
        ax[i][j].legend()
        ax[i][j].set_xlabel('')
        ax[i][j].set_ylabel('frequency')
    ax[i][j].set_xlabel('expression level')
    ax[i][j-1].set_xlabel('expression level')
```

```
In [9]: genes_list = ['D29963_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5_at']
        plot_genes_histograms(genes_list, data_train_scaled)
```



The histogram overlay shows the distribution of gene expression levels. If the probability density function of ALL does not overlay with the function for AML, then we can conclude that that gene discriminates between both classes of cancer. Unfortunately there is a significant overlay of the pair-PDFs for all four genes. None of the genes clearly discrininate between ALL and AML. However we can say that low levels of D29963_at, low levels of hum_alu_at, very high levels of M23161_at, very low levels of AFFX-PheX-5_at provide a higher probability that the patient develops ALL. For very high levels of D29963_at the probability is higher that the cancer type is AML.

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

```
In [0]: pca_transformer = PCA(2).fit(X_train_scaled)
        X_train_2d = pca_transformer.transform(X_train_scaled)
        var_explained = pca_transformer.explained_variance_ratio_
```

```
In [13]: colors = ['r','c']
         label_text = ["Acute Lymphoblastic Leukemia (ALL)", "Acute Myeloid Leukemia (AML)"]

         markshapes = ['x','o']
         # and we loop over the different groups
         for cancer_type in [0,1]:
             df_per_type = X_train_2d[y_train==cancer_type]
             plt.scatter(df_per_type[:,0], df_per_type[:,1],
                         marker=markshapes[cancer_type],
                         c = colors[cancer_type],
                         label=label_text[cancer_type])

         plt.xlabel("PCA Dimension 1")
         plt.ylabel("PCA Dimention 2")
         plt.title("top two principal components for the gene expression data")
         plt.legend();
```

Both disease conditions are not very well discriminated by the top two PCA components. Low values (< 4) in dimension 2 predict both ALL and AML conditions with relatively equal probability. High values (>4) in dimension 2 could be used to predict ALL with high accuracy. The combination of values with dimension 1 greater than 3, and dimension 2 less than zero could be used to classify AML.

```
In [14]: print("Variance explained by each PCA component:", var_explained)
         print("Total Variance Explained:", np.sum(var_explained))
```

```
Variance explained by each PCA component: [0.15889035 0.11428795]
Total Variance Explained: 0.2731782945208864
```

The first PCA dimension captures 15.88% of the variance in the data, and the second PCA dimension adds another 11.43%. Together, we've got just 27.31% of the total variation in the training data. We migh need additional PCA dimensions.

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

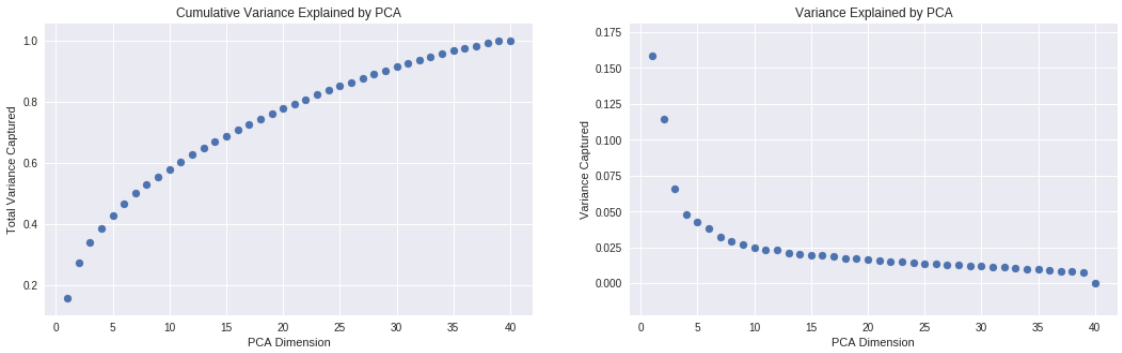
```
In [0]: pca_transformer = PCA(50).fit(X_train_scaled)
        X_train_50d = pca_transformer.transform(X_train_scaled)
        var_explained_50d = pca_transformer.explained_variance_ratio_
```

```
In [16]: var_explained_cum = np.cumsum(var_explained_50d)

         # display as a table
         var_explained_df = pd.DataFrame()
         var_explained_df['dimensions'] = np.array(list(range(1,len(var_explained_cum)+1)))
         var_explained_df['cumulative_var_explained'] = var_explained_cum
         display(var_explained_df)
```

	dimensions	cumulative_var_explained
0	1	0.158890
1	2	0.273178
2	3	0.339141
3	4	0.387015
4	5	0.429572
5	6	0.467894
6	7	0.500530
7	8	0.529504
8	9	0.556197
9	10	0.580748
10	11	0.604404
11	12	0.627411
12	13	0.648637
13	14	0.668736
14	15	0.688411
15	16	0.707683
16	17	0.726223
17	18	0.743569
18	19	0.760778
19	20	0.777326
20	21	0.792818
21	22	0.807985
22	23	0.822980
23	24	0.837251

```
24          25                  0.851115
25          26                  0.864456
26          27                  0.877615
27          28                  0.890451
28          29                  0.902687
29          30                  0.914478
30          31                  0.925960
31          32                  0.937067
32          33                  0.947493
33          34                  0.957267
34          35                  0.966780
35          36                  0.975667
36          37                  0.984278
37          38                  0.992505
38          39                  1.000000
39          40                  1.000000
```

```
In [17]: fig, ax = plt.subplots(1, 2, figsize=(18, 5))
         ax[0].scatter(range(1, 41), var_explained_cum)
         ax[0].set_xlabel("PCA Dimension")
         ax[0].set_ylabel("Total Variance Captured")
         ax[0].set_title("Cumulative Variance Explained by PCA");
         ax[1].scatter(range(1, 41), var_explained_50d)
         ax[1].set_xlabel("PCA Dimension")
         ax[1].set_ylabel("Variance Captured")
         ax[1].set_title("Variance Explained by PCA");
```



Adding more components has no effect on the first two principal components. It is safe to add more components in order to achieve greater variance coverage of the feature set.

Looking at the plot, more variance is captured by increasing the number of PCA components. We can have a maximum number of 40 components since we have 40 samples in the training dataset. Therefore we can consider adding components as long as the explained variance increases significantly. For example, with 22 dimensions, our model can explain 80% of the original variance. In order to explain 100% from there, we have to almost double the number of dimensions. This would have drawback on interpretability and increases the danger of overfitting.

But if we look at the graph of variance captured ("Variance Explained by PCA"), the slope of the curve flattens out after about 10 dimensions. Adding additional dimensions after around this point only adds around 2% to the total variance captured. So, even beyond 10 components, when we add additional components, we would be adding complexity without explaining that much more variance.

So even while we only capture 58% of the variance (under two-thirds), 10 dimensions might be a good tradeoff between variance and additional model complexity.

However, we would ordinarily perform cross-validation to help determine the optimal number of components.

By looking at the plot, 29 components are required in order to explain at least 90% of the variability in the feature set.

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

```
In [10]: model_D29963at_ols = OLS(data_train_scaled.Cancer_type.values,
                                  sm.add_constant(data_train_scaled.D29963_at)
                                 ).fit()
         model_D29963at_ols.summary()

Out[10]: <class 'statsmodels.iolib.summary.Summary'>
         """
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.329
Model:                            OLS   Adj. R-squared:                  0.311
Method:                 Least Squares   F-statistic:                     18.61
Date:                Thu, 25 Oct 2018   Prob (F-statistic):           0.000110
Time:                        03:26:56   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 specif
"""
```

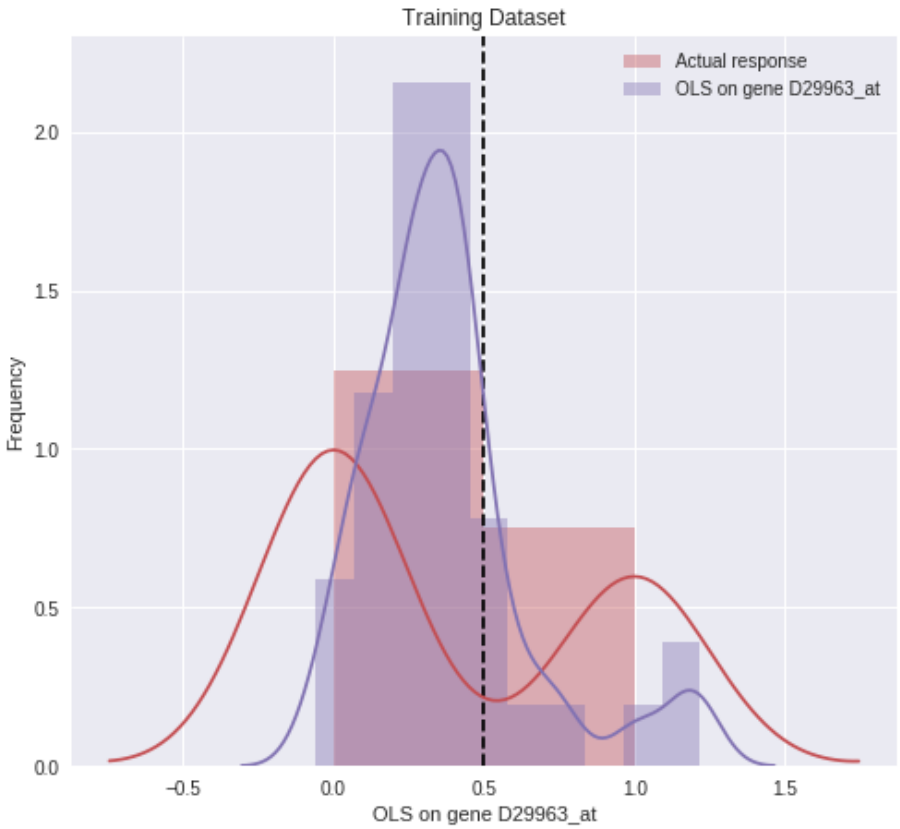
In [0]:
```
# get predictions for train and test
lin_y_pred_train = model_D29963at_ols.predict(sm.add_constant(data_train_scaled.D29963_a
lin_y_pred_test = model_D29963at_ols.predict(sm.add_constant(data_test_scaled.D29963_at)
```

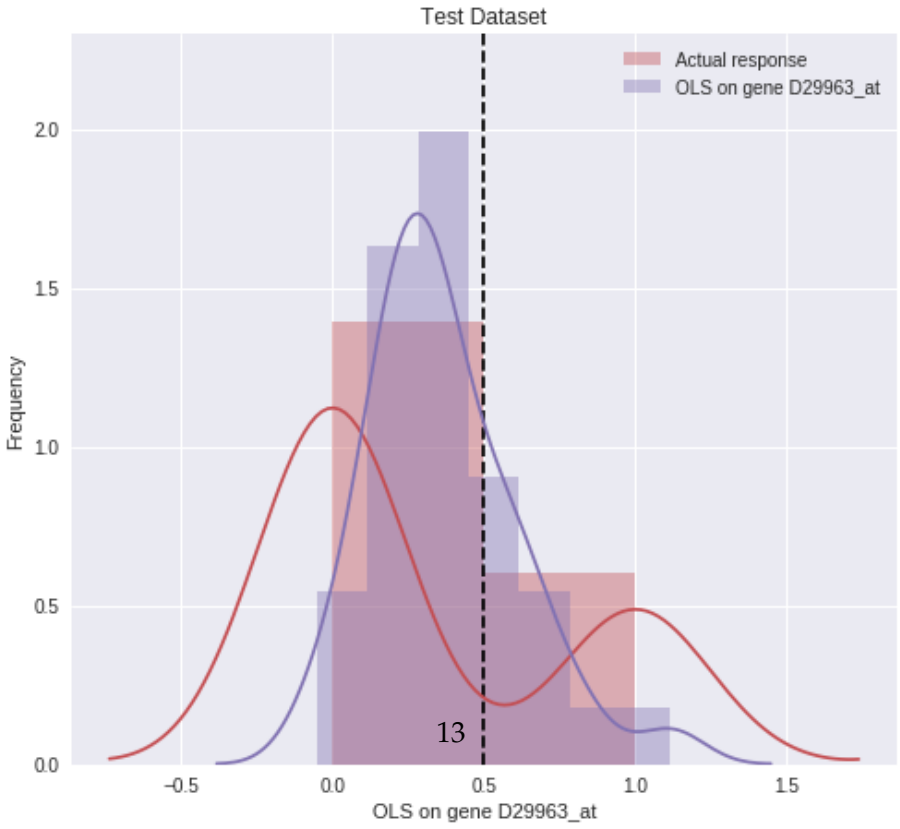
In [0]:
```
# helper functions
def plot_hist(data, label, color, ax):
  sns.distplot(data, color=color, label=label, ax=ax)

def set_ax_title_label_legend(ax, title = None, xlabel="predictor", ylabel="response"):
  if title is None:
    ax.set_title(ylabel + " vs. " + xlabel)
  else:
    ax.set_title(title)
  ax.set_xlabel(xlabel)
  ax.set_ylabel(ylabel)
  ax.legend()

def plot_overlay_hist(data_to_overlay, title, labels, colors, ax):
  for i, data in enumerate(data_to_overlay):
    plot_hist(data, labels[i], colors[i], ax)
    # set labels
```

```
            # call labels/legends function
            set_ax_title_label_legend(ax,
                title=title,
                xlabel=labels[i],
                ylabel="Frequency")

        def plot_overlay_hist_series(data_list, titles, labels, colors, axes):
          if len(axes) < len(data_list):
            raise Exception('Fewer axes ({}) than data groups ({})'.format(
              len(axes),
              len(data_list)))
          for i, data_to_overlay in enumerate(data_list):
            plot_overlay_hist(data_to_overlay, titles[i], labels, colors, axes[i])
```

In [116]:
```
        # histogram of predictions
        def plot_simple_linear_hist():
          ## set up plots
          nrows = 2
          ncols = 1
          fsize = (8, 16)
          fig, ax = plt.subplots(nrows, ncols, figsize=fsize)

          ## data to iterate over
          colors = [ 'C2', 'C3', ]
          y_train_to_plot = [ data_train_scaled.Cancer_type, lin_y_pred_train, ]
          y_test_to_plot = [ data_test_scaled.Cancer_type, lin_y_pred_test, ]
          y_labels = [  "Actual response", "OLS on gene D29963_at", ]
          plot_labels = [ "Training Dataset", "Test Dataset", ]
          plot_overlay_hist_series(
              [y_train_to_plot, y_test_to_plot],
              plot_labels,
              y_labels,
              colors,
              ax)

          # add vertical lines
          threshold = 0.5
          ymin, ymax = 0, 2.3
          for a in ax:
            a.vlines(threshold, 0, ymax, colors='k', linestyles='dashed')
            a.set_ylim(ymin, ymax) # reset frame

          fig.suptitle("Cancer type: ALL[0] or ALM[1]", fontsize=18)

        plot_simple_linear_hist()
```

Cancer type: ALL[0] or ALM[1]





Although the response is binary, the OLS regession model's range of outputs are continuous. This leads to major issues for interpreting the output values.

For example, in the plots above the true response is trivially bimodal at zero and one in both training and test sets. But the histograms for the OLS predictions are in other ranges. In the training set, there are peak counts for values that are near 0.4 and 1.2. In the test set, we see the most number of predictions near 0.4. What does it mean to be 0.4 between two cancer types? And moreso, what does it mean to be 1.2 like a certain cancer type? Are observations with response value 1.2 three times *more similar* to ALM (cancer type 1)?

The fundamental problem is that regression model can produce predictions that fall outside the bounds of $(0, 1)$, and thus produces values that do not fit the definition of probabilities, i.e. there is no such thing as a probabilities greater than one or less than zero. The 1.2 we see above cannot be a probability.

As one consequence, the usual way of discussing false positives and false negatives, which relies on Bayes' Theorem, breaks down because of this, since the inputs to Bayes' theorem (and thus the outputs) are no longer probabilities.

In a medical context, where a missed diagnosis can be very expensive, and series of tests need to be designed around the rate of false positives and false negatives, this becomes detrimental.

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

```
In [19]: # get quantitative response
         y_train_D29963at_predicted_ols_qr = model_D29963at_ols.predict(
             sm.add_constant(data_train_scaled.D29963_at))
         # convert to binary response
         y_train_D29963at_predicted_ols = y_train_D29963at_predicted_ols_qr >= 0.5
         accuracy_train_D29963at_ols = (np.sum(
             y_train_D29963at_predicted_ols == data_train_scaled.Cancer_type.values) /
                                        len(data_train_scaled.Cancer_type.values))

         print("Classification accuracy of the OLS classification model on train set is ",
               accuracy_train_D29963at_ols)
```

```
Classification accuracy of the OLS classification model on train set is  0.8
```

```
In [20]: # get quantitative response
         y_test_D29963at_predicted_ols_qr = model_D29963at_ols.predict(
             sm.add_constant(data_test_scaled.D29963_at))
         # convert to binary response
         y_test_D29963at_predicted_ols = y_test_D29963at_predicted_ols_qr >= 0.5
         accuracy_test_D29963at_ols = (np.sum(
             y_test_D29963at_predicted_ols == data_test_scaled.Cancer_type.values) /
                                       len(data_test_scaled.Cancer_type.values))

         print("Classification accuracy of the OLS classification model on test set is ",
               accuracy_test_D29963at_ols)
```

```
Classification accuracy of the OLS classification model on test set is  0.7575757575757576
```

The classification based on linear regression predicts the cancer type with 75.75% accuracy on test set while reaching 80% on training set. However, we see the model generating predictions greater than one and less than zero, which are confusing to interpret.

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

```
In [11]: BIG_C = 10**5
         columns = ['D29963_at']
         fitted_lr = LogisticRegression(
             C=BIG_C).fit(
             X_train_scaled[columns],
             y_train)

         # predict
         y_pred_train = fitted_lr.predict(X_train_scaled[columns])
         y_pred_test = fitted_lr.predict(X_test_scaled[columns])

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

         print("Coefficients:")
         print(fitted_lr.coef_)
         print("Intercepts:")
         print(fitted_lr.intercept_)
         print("Train: {}%".format(train_score))
         print("Test: {}%".format(test_score))
```

```
Coefficients:
[[10.26387493]]
Intercepts:
[-3.99462202]
Train: 80.0%
Test: 75.75757575757575%
```

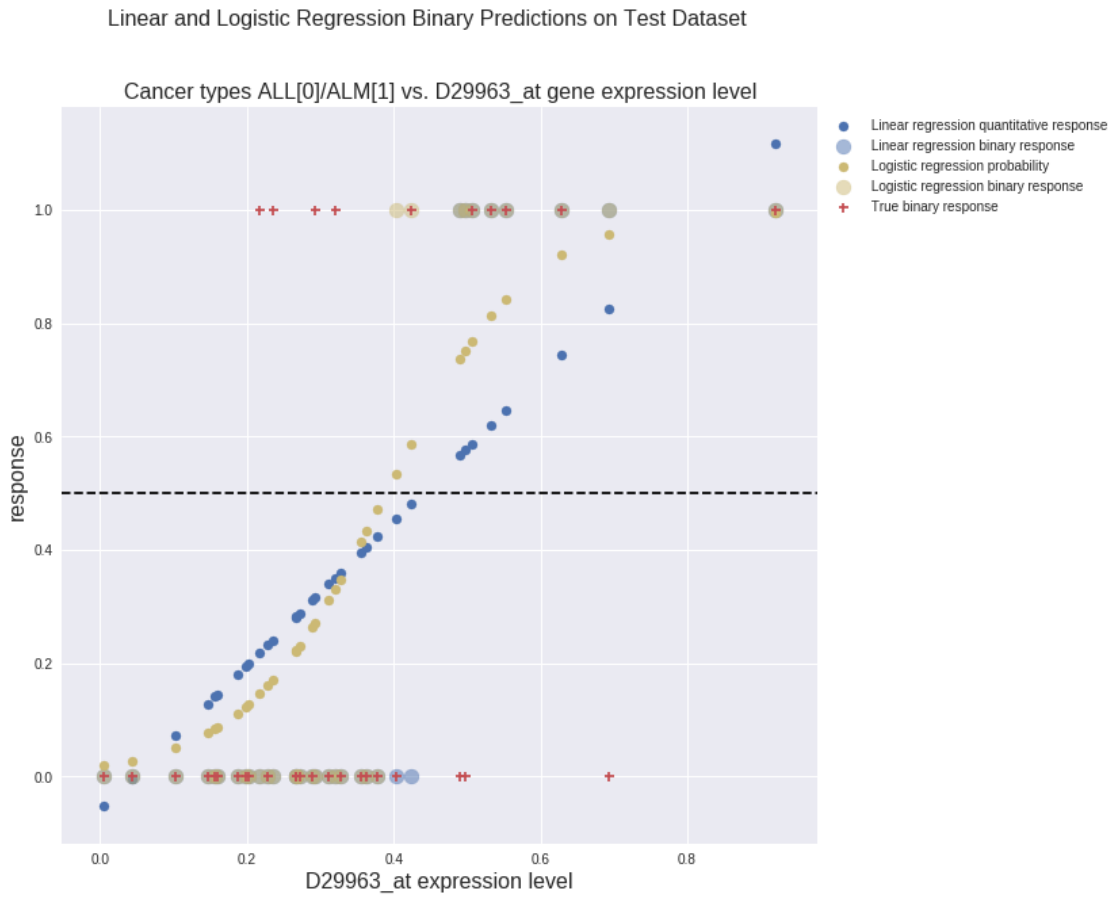
Strangely, the classification accuracy of both models seems to be the same. This probably occurs because the straight regression line fit and the logistic function divide the plane of data points in a similar fashion. Data points falling in the middle of the plane, i.e. with middle expression levels for this gene, would be most likely to fall on different sides of the boundary, depending on the function. It is possible there aren't many points in this area of the graph.

2.4: Create a figure with 4 items ...

```
In [22]: # your code here

         fig, ax = plt.subplots(1,1, figsize=(10, 10))

         colors = ['C0', 'C4', 'C2']

         binary_size = 120
```

```
# linear regression
# quantitative response
plt.scatter(data_test_scaled.D29963_at,
            y_test_D29963at_predicted_ols_qr,
            color=colors[0],
            label = ' Linear regression quantitative response')
# binary response
plt.scatter(data_test_scaled.D29963_at,
            y_test_D29963at_predicted_ols,
            color=colors[0],
            alpha=0.5,
            s=binary_size,
            label = ' Linear regression binary response')

# logistic regression
# probabilities
y_prob_test = fitted_lr.predict_proba(
    data_test_scaled.D29963_at.values.reshape(-1,1))[:,1]
plt.scatter(data_test_scaled.D29963_at,
            y_prob_test,
            color=colors[1],
            label = ' Logistic regression probability')
# binary response
plt.scatter(data_test_scaled.D29963_at,
            y_pred_test,
            s=binary_size,
            color=colors[1],
            alpha=0.5,
            label = ' Logistic regression binary response')

# true test response
# binary response
plt.scatter(data_test_scaled.D29963_at,
            y_test,
            color=colors[2],
            marker='+',
            alpha=1,
            label = ' True binary response')

# add y tick marks for cancer types?
ax.axhline(y = 0.5, color = 'k', linestyle = '--')
plt.ylabel("response", fontsize=16)
plt.xlabel("{} expression level".format(columns[0]), fontsize=16)
plt.legend(loc="best", bbox_to_anchor=(1,1))
plt.suptitle(
    "Linear and Logistic Regression Binary Predictions on Test Dataset",
    fontsize=16)
```

```
plt.title(
    "\nCancer types ALL[0]/ALM[1] vs. D29963_at gene expression level",
    fontsize=16)
print("") # suppress message
```



Based on the plot of the binary model predictions, it is hard to tell whether the logistic regression or the linear regression fares better in predicting the correct binary response in the test set. The linear regression seems to have one more false negative, and the logistic regression seems to have one more false positive. However, both models misclassify several of the same points.

The logistic regession produces probabilities, which is an advantage, while the linear regression outputs a quantitative response that can exceed the bounds of $(0, 1)$, which is a disadvantage. But purely for the purposes of predicting the binary response in the test, they have fared roughly the same.

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 [8]: #--------  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...

```
In [12]: fitted_lr = LogisticRegression(C=BIG_C).fit(X_train_scaled, y_train)

         # predict
         y_pred_train = fitted_lr.predict(X_train_scaled)
         y_pred_test = fitted_lr.predict(X_test_scaled)

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

         # report
         print("Coefficients:")
         print(fitted_lr.coef_)
         print("Intercepts:")
         print(fitted_lr.intercept_)
         print("Train: {}%".format(train_score))
         print("Test: {}%".format(test_score))

Coefficients:
[[ 0.04018829 -0.03813132  0.01562291 ... -0.00195913  0.00041904
   0.06724911]]
Intercepts:
[0.0034058]
Train: 100.0%
Test: 100.0%


In [25]: # classification counts
         # overview of false positives and false negatives
         pd.crosstab(y_test, y_pred_test,
                     margins=True,
                     rownames=['Actual'],
                     colnames=['Predicted'])

Out[25]: Predicted   0   1  All
         Actual
         0          23   0   23
         1           0  10   10
         All        23  10   33
```

By fitting our logistic regression model with all 7129 predictors, we have improved the train and test score to 100%. A big question is whether we are now in danger of overfitting, which such a large ratio of predictors to data points. We are obviously facing confounding as the results obtained with one predictor is different from those obtained with multiple predictors, indicating

some correlations among them. A study of coefficients would bring more light into this observation.

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

```
In [13]: #Creating model
         model = LogisticRegression(C=BIG_C)

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

         #Conduct bootstraping iterations
         for i in range(bootstrap_iterations):
             sample = data_train_scaled.sample(frac=1, replace=True)
             y_train_sample = sample['Cancer_type']
             X_train_sample = sample.drop(['Cancer_type'], axis=1)
             model.fit(X_train_sample, y_train_sample)
             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_scaled.columns[i])

In [14]: #print('Columns :', significant_cols)
         print('Count of 95% statistically significant coefficients :', coeffs_count)
         print(("Discussion: Out of 7130 coefficients, " +
                "only {} of them are significant at p > 0.05.\n" +
                "That is  only {:.2f}% of them.").format(coeffs_count, coeffs_count/7130*100))

Count of 95% statistically significant coefficients : 1859
Discussion: Out of 7130 coefficients, only 1859 of them are significant at p > 0.05.
That is  only 26.07% of them.
```

3.3: Use the visualize_prob function provided below ...

```
In [28]: """ 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(fitted_lr, X_train_scaled, y_train, ax1)
ax1.set_title('Training Dataset')

#Plot Testing
visualize_prob(fitted_lr, X_test_scaled, y_test, ax2)
ax2.set_title('Testing Dataset')
ax2.set_ylabel('')
plt.show()
```

The probabilities in the training set are concentrated solely at 0 and 1. The probabilities for the data in the test set are more spread out, even though the accuracy for the test set was 100%. A few of the points classified as ALM (cancer type 1) received probabilities fairly close to the threshold of 0.5. These points correspond to samples which are close to the classification boundary. Their class can be highly impacted when the boundary is slightly moved. The associated patients are good candidates for false positives or false negatives.

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

Although the classification accuracy is 100% both on train and test sets, boostraping shows that only around 1/4 of coefficients are significant to the response. In order to generalize the model, it might be safe to perform regularization, or better, principal components analysis in order to account for the high dimensionality of our data and its possible negative impact on the prediction quality and interpretability.

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 [29]: selected_dim = 10
         ninety_percent_dim = 29
         models_to_fit = [selected_dim, ninety_percent_dim]
         models_comparison = {}

         # Logistic PCR:
         # get principal components for different specific dimensions,
         # then train and fit a logistic regression model for each
         for i, dim in enumerate(models_to_fit):
             cur_model = None
             results = {}
             # fit model
             results['dim'] = dim
             pca_transformer = PCA(dim).fit(X_train_scaled)
             cur_X_train = pca_transformer.transform(X_train_scaled)
             cur_X_test =  pca_transformer.transform(X_test_scaled)
             cur_model = LogisticRegression(C=BIG_C).fit(cur_X_train, y_train)

             # predict
             y_pred_train = cur_model.predict(cur_X_train)
             y_pred_test = cur_model.predict(cur_X_test)

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

             # store results and model info for later
             results['model'] = cur_model # model used
             results['X_train'] = cur_X_train
             results['X_test'] = cur_X_test
             results['train_score'] = train_score
             results['test_score'] = test_score
             results['y_pred_train'] = y_pred_train
             results['y_pred_test'] = y_pred_test
             models_comparison[dim] = results
```

```
        # report scores
        for d in models_comparison.keys():
            print("{} dimension PCA - train: {:.2f}% test: {:.2f}%".format(
              models_comparison[d]['dim'],
              models_comparison[d]['train_score'],
              models_comparison[d]['test_score'],
            ))
```

```
10 dimension PCA - train: 100.00% test: 90.91%
29 dimension PCA - train: 100.00% test: 96.97%
```

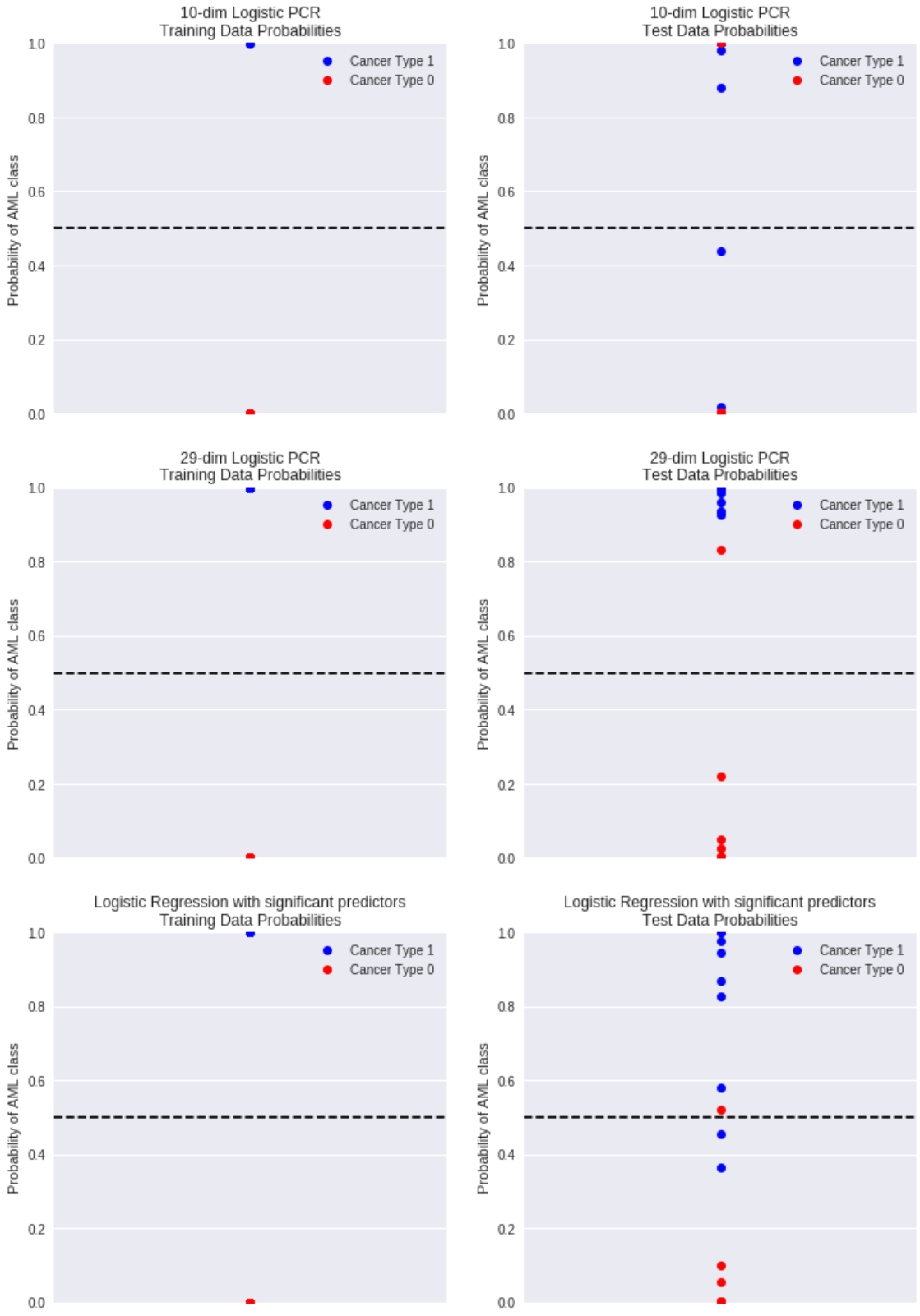
Previously by fitting a logistic regression model with all 7129 predictors, we obtained 100% classification accuracy on train and test. Now using principal components analysis, we fit a logistic regression model using principal components as predictors.

When using 10 components we achieve 100% classification accuracy on train set and 87.87% on test set. Using 29 components does not change the classification accuracy on training set, but the classification accuracy is improved to 96.97%. With 10 components, 58.07% of the variance is captured, against 90% captured by 29 components. It seems that our principal components regression has introduced few new false positives and/or false negatives compared to the logistic regression model from all 7129 predictors. By allowing some loss in prediction accuracy (from 100% down to 87.87% on test set), we have gained in better interpretability by using mostly significant prodictors. We recall that only 1/4 of predictors were found to be significant at $p > 0.05$ via cross-validation.

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

```
In [30]: # function to visualize the probabilities
         # predicted by a set of fitted models with PCR
         def visualize_prob_train_test(model, model_desc,
                                       X_train, y_train,
                                       X_test, y_test,
                                       ax1, ax2):
           visualize_prob(model, X_train, y_train, ax1)
           ax1.set_title("{}\nTraining Data Probabilities".format(model_desc))
           visualize_prob(model, X_test, y_test, ax2)
           ax2.set_title("{}\nTest Data Probabilities".format(model_desc))

         ## visualize PCR models with 10 and 29 components

         # set up figure
         plot_rows = len(models_to_fit)+1
         plot_cols = 2
         fig, axes = plt.subplots(plot_rows, plot_cols, figsize=(plot_cols*6, plot_rows*6))

         count = 0
         for m in models_to_fit:
           # model metadata
           model = models_comparison[m]['model']
```

```
  dim = models_comparison[m]['dim']
  model_desc = "{}-dim Logistic PCR".format(dim)

  # plot
  ax1, ax2 = axes[count][:]
  X_train = models_comparison[m]['X_train']
  X_test = models_comparison[m]['X_test']
  visualize_prob_train_test(model, model_desc,
                            X_train, y_train,
                            X_test, y_test,
                            ax1, ax2)

  y_pred_test = models_comparison[m]['y_pred_test']
  count += 1

## visualize logistic regression model with significant predictors

lr_model_significant_cols = LogisticRegression(C=BIG_C).fit(
    X_train_scaled[significant_cols],
    y_train)
train_score_significant_cols = lr_model_significant_cols.score(
    X_train_scaled[significant_cols],
    y_train)
test_score_significant_cols = lr_model_significant_cols.score(
    X_test_scaled[significant_cols],
    y_test)
ax1, ax2 = axes[count][:]
visualize_prob_train_test(lr_model_significant_cols,
      'Logistic Regression with significant predictors',
      X_train_scaled[significant_cols], y_train,
      X_test_scaled[significant_cols], y_test,
      ax1, ax2)
```


10-dim Logistic PCR
Training Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class
1.0
0.8
0.6
0.4
0.2
0.0
10-dim Logistic PCR
Test Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class
29-dim Logistic PCR
Training Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class
29-dim Logistic PCR
Test Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class
Logistic Regression with significant predictors
Training Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class
Logistic Regression with significant predictors
Test Data Probabilities
Cancer Type 1
Cancer Type 0
Probability of AML class

In [31]:
```
## display an overview of false positives and false negatives for PCR models with 10 an
count = 0
display("Predicted class vs Actual class")
for m in models_to_fit:
  # model metadata
  model = models_comparison[m]['model']
  dim = models_comparison[m]['dim']
  model_desc = "{}-dim Logistic PCR".format(dim)

  y_pred_test = models_comparison[m]['y_pred_test']
  display(model_desc)
  display(pd.crosstab(
      y_test,
      y_pred_test,
      margins=True,
      rownames=['Actual'],
      colnames=['Predicted']))
  count += 1

## display an overview of false positives and false negatives
# for logistic regression model with significant predictors
display('Logistic Regression with significant predictors')
display(pd.crosstab(
    y_test,
    lr_model_significant_cols.predict(
        X_test_scaled[significant_cols]),
    margins=True,
    rownames=['Actual'],
    colnames=['Predicted']))
```

```
'Predicted class vs Actual class'
```

```
'10-dim Logistic PCR'
```

Predicted	0	1	All
Actual			
0	22	1	23
1	2	8	10
All	24	9	33

```
'29-dim Logistic PCR'
```

Predicted	0	1	All
Actual			
0	22	1	23

```
1           0  10   10
All        22  11   33


'Logistic Regression with significant predictors'


Predicted   0  1  All
Actual
0          22  1   23
1           2  8   10
All        24  9   33
```

The spread of probabilities appears to be the same accross all three models for training data, though the higher dimensional models seem to generate a larger number of predictions indicating uncertainty. All labels for class 0 are clustered around the probability 0, while labels for class 1 are found at probability 1. Such a spread produces a classification accuracy of 100% on the training data.

The probabilities on test data are spread with more variability for the logistic regression model fit on significant predictors (at $p < 0.05$). This model produces 2 false negatives and 1 false positive. Both logistic models fit with 10 principal components produces 4 false negatives and 1 false positive. The probabilities obtained with those PCR models are mostly clustered near 0 and 1 with few data points spread towards the classification boundary at 0.5.

The plots below shows how the explained variance ("Variance Explained by PCA") and cumulative explained variance ("Cumulative Variance Explained by PCA") changes with increasing dimensions. Lower dimensional representation with 10 components yields worse predictive power (test score 88%) compared to higher dimensions with 29 components (test score 96.97%). Lower dimension was chosen by eyeballing the plot of variance explained and looking at the elbow area.

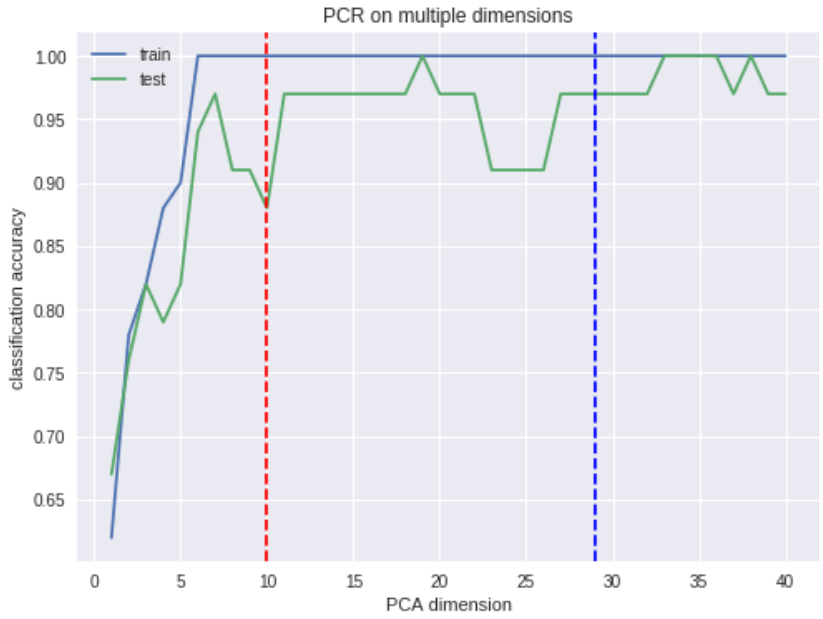
The test performance for 10 dimensions is slightly worse then the performance of nearby dimensions (a local minimum). It would have been possible to increase the amount of dimensions slightly in order to improve performance, while still staying close to the elbow region of the variance captured graph. A model selection process utilizing cross-validaton would have better suggested the best options for the number of PCR dimensions.

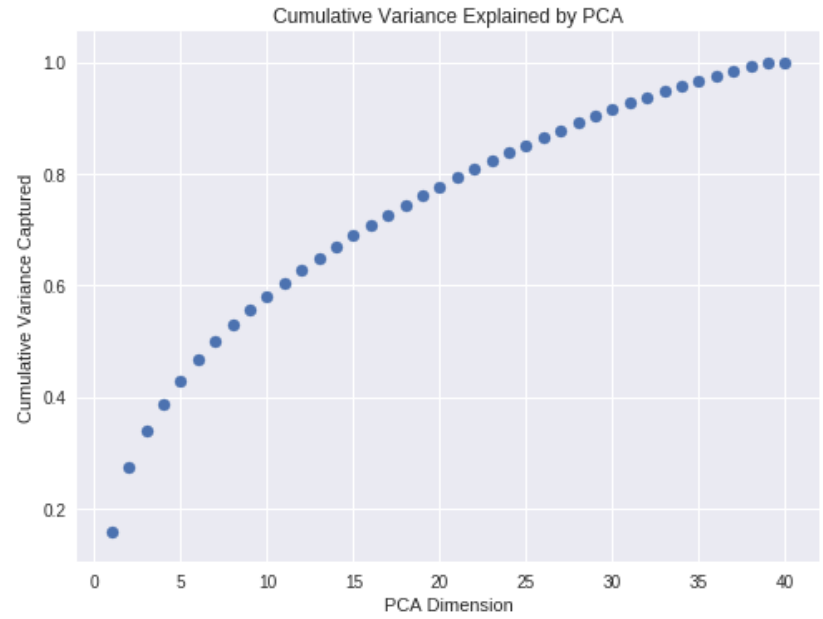
```
In [0]: def pca_logistic(dim):
            pca_transformer = PCA(dim).fit(X_train_scaled)
            X_train_pca = pca_transformer.transform(X_train_scaled)
            X_test_pca = pca_transformer.transform(X_test_scaled)
            model = LogisticRegression(C=1000000).fit(X_train_pca,y_train)
            train_score = model.score(X_train_pca,y_train)
            test_score = model.score(X_test_pca,y_test)
            #train_score = accuracy_score(y_train, model.predict(X_train_pca))
            #test_score = accuracy_score(y_test, model.predict(X_test_pca))
            clf = LogisticRegressionCV(cv=5).fit(X_train_pca, y_train)
            cv_score = clf.score(X_train_pca, y_train)
            return (train_score,
                    test_score,
                    pca_transformer.explained_variance_ratio_,
```

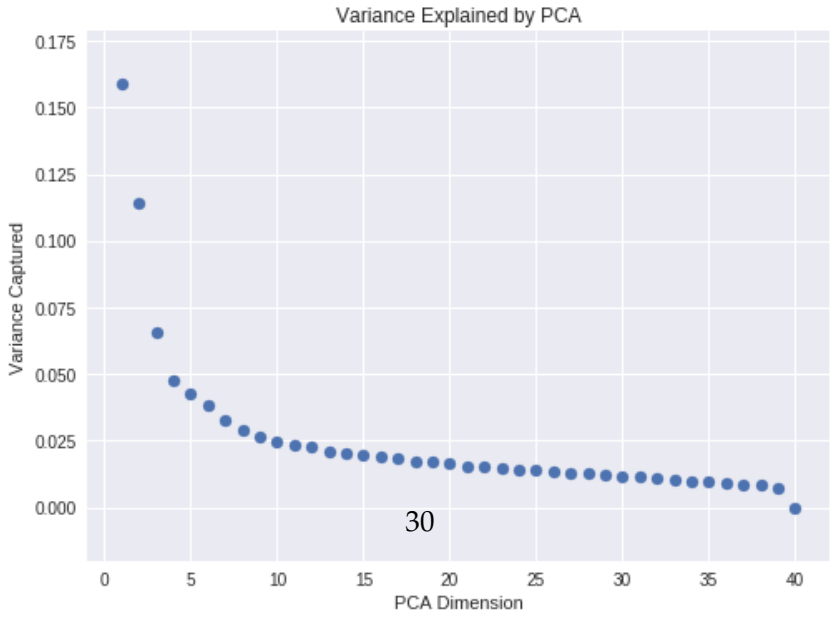
```
                    model.coef_,
                    cv_score)
```

In [34]:
```
def plot_pca_scores(max_dim=40):

    train_scores = []
    test_scores = []
    cv_scores = []
    dim_range = range(1, max_dim+1)

    for dim in dim_range:
        (train_score,
         test_score,
         var_explained,
         coefs,
         cv_score) = pca_logistic(dim)
        train_scores.append(round(train_score, 2))
        test_scores.append(round(test_score, 2))
        cv_scores.append(round(cv_score,2))

    fig, ax = plt.subplots(3,1,figsize=(8,20))
    ax[0].plot(dim_range, train_scores, label='train')
    ax[0].plot(dim_range, test_scores, label='test')
    ax[0].axvline(x=10, color='r', linestyle='--')
    ax[0].axvline(x=29, color='b', linestyle='--')
    ax[0].set_xlabel('PCA dimension')
    ax[0].set_ylabel('classification accuracy')
    ax[0].set_title('PCR on multiple dimensions')
    ax[0].legend()

    var_explained_cum = np.cumsum(var_explained)
    ax[1].scatter(dim_range, var_explained_cum)
    ax[1].set_xlabel("PCA Dimension")
    ax[1].set_ylabel("Cumulative Variance Captured")
    ax[1].set_title("Cumulative Variance Explained by PCA")

    ax[2].scatter(dim_range, var_explained)
    ax[2].set_xlabel("PCA Dimension")
    ax[2].set_ylabel("Variance Captured")
    ax[2].set_title("Variance Explained by PCA")


    print("Classification accuracy on test for each dimension: ", test_scores)
    print("Variance captured by each single dimension: ", var_explained)
    print("Cumulative variance captured at each dimension: ", var_explained_cum)
    print("Cross-validation score for each dimension: ", cv_scores)

plot_pca_scores()
```

```
Classification accuracy on test for each dimension:  [0.67, 0.76, 0.82, 0.79, 0.82, 0.94, 0.97,
Variance captured by each single dimension:  [1.58890345e-01 1.14287949e-01 6.59628918e-02 4.787
 4.25567520e-02 3.83225402e-02 3.26359261e-02 2.89735451e-02
 2.66932266e-02 2.45509698e-02 2.36556039e-02 2.30070468e-02
 2.12261465e-02 2.00993660e-02 1.96743636e-02 1.92725678e-02
 1.85394343e-02 1.73461259e-02 1.72095498e-02 1.65473368e-02
 1.54924718e-02 1.51668361e-02 1.49950438e-02 1.42713203e-02
 1.38639772e-02 1.33412133e-02 1.31589544e-02 1.28359687e-02
 1.22356477e-02 1.17908904e-02 1.14823727e-02 1.11070763e-02
 1.04255621e-02 9.77443029e-03 9.51248818e-03 8.88740202e-03
 8.61113419e-03 8.22663504e-03 7.49497253e-03 4.90296184e-32]
Cumulative variance captured at each dimension:  [0.15889035 0.27317829 0.33914119 0.3870151  0.
 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 0.91447793
 0.9259603  0.93706738 0.94749294 0.95726737 0.96677986 0.97566726
 0.98427839 0.99250503 1.         1.        ]
Cross-validation score for each dimension:  [0.62, 0.85, 0.88, 0.85, 0.9, 0.98, 0.98, 0.95, 0.98
```


PCR on multiple dimensions
train
test
classification accuracy
PCA dimension


Cumulative Variance Explained by PCA
Cumulative Variance Captured
PCA Dimension


Variance Explained by PCA
Variance Captured
PCA Dimension