CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 5

Logistic Regression and PCA

Harvard University
Fall 2017
Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave, Margo Levine

---

INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- Do not include your name(s) in the notebook if you are submitting as a group.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.

Your partner's name (if you submit separately): Ryan Mitchell

Enrollment Status (109A, 121A, 209A, or E109A): 109A

Import libraries:

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
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.utils import resample
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

Cancer Classification from Gene Expressions

In this homework assignment, 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 `dataset_hw5.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 parts, we will use logistic regression to build a classification model for this data set. We will also use principal components analysis (PCA) to visualize the data and to reduce its dimensions.

Part (a): Data Exploration

1. 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).
2. 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.
3. Notice that the results training set contains more predictors than observations. Do you foresee a problem in fitting a classification model to such a data set?
4. A convenient tool to visualize the gene expression data is a heat map. Arrange the rows of the training set so that the 'AML' rows are grouped together and the 'ALL' rows are together. Generate a heat map of the data with expression values from the following genes: `D49818_at`, `M23161_at`, `hum_alu_at`, `AFFX-PheX-5_at`, `M15990_at`. By observing the heat map, comment on which of these genes are useful in discriminating between the two classes.

5. We can also visualize this data set in two dimensions using PCA. Find the top two principal components for the gene expression data. Generate a scatter plot using these principal components, highlighting the AML and ALL points in different colors. How well do the top two principal components discriminate between the two classes?

In [3]:

```
np.random.seed(9001)
df = pd.read_csv('./data/dataset_hw5.csv')
msk = np.random.rand(len(df)) < 0.5
data_train = df[msk]
data_test = df[~msk]
```

In [4]:

```
data_train.describe()
```

Out[4]:

	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
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000
mean	0.375000	-125.281250	-165.250000	5.000000	175.375000	-262.625000	-401.500000
std	0.491869	113.857646	74.106288	143.113084	104.963204	133.640526	155.560796
min	0.000000	-476.000000	-326.000000	-410.000000	-24.000000	-535.000000	-810.000000
25%	0.000000	-178.750000	-214.000000	-77.250000	93.250000	-367.750000	-496.750000
50%	0.000000	-115.000000	-162.500000	7.000000	173.000000	-259.000000	-398.000000
75%	1.000000	-61.000000	-99.500000	67.000000	258.750000	-200.000000	-310.250000
max	1.000000	86.000000	-36.000000	312.000000	328.000000	114.000000	-122.000000

8 rows × 7130 columns

In [5]:

```python
# Scale the data to vary between 0 and 1
from sklearn.preprocessing import MinMaxScaler
keys = list(data_train.keys())
scaler = MinMaxScaler(copy=True, feature_range=(0, 1))
data_train = pd.DataFrame(scaler.fit_transform(data_train), columns=keys)

data_test = pd.DataFrame(scaler.fit_transform(data_test), columns=keys)
data_test.describe()
```

Out[5]:

	Cancer_type	AFFX-BioB-5_at	AFFX-BioB-M_at	AFFX-BioB-3_at	AFFX-BioC-5_at	AFFX-BioC-3_at	AFFX-BioDn-5_at	AFFX-BioDn-3_a
count	41.000000	41.000000	41.000000	41.000000	41.000000	41.000000	41.000000	41.00000
mean	0.317073	0.699636	0.723938	0.502644	0.500548	0.678759	0.650246	0.68097
std	0.471117	0.192733	0.212567	0.180343	0.250661	0.259243	0.244640	0.17995
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.634483	0.654440	0.402098	0.353319	0.512702	0.462046	0.56807
50%	0.000000	0.737931	0.756757	0.487762	0.462527	0.762125	0.702970	0.69592
75%	1.000000	0.820690	0.872587	0.566434	0.683084	0.856813	0.849835	0.78645
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

8 rows × 7130 columns

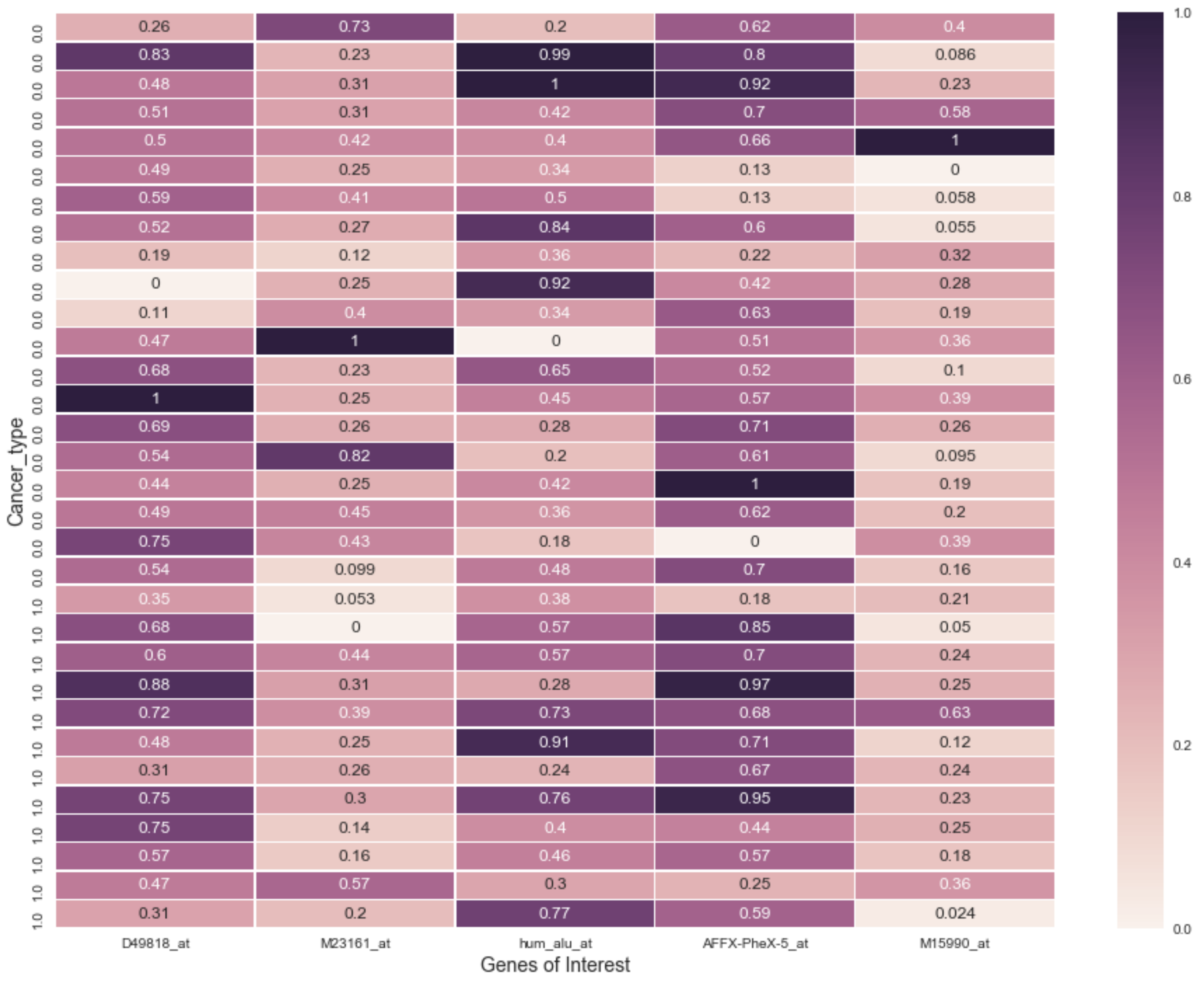
In [6]:

```python
# Create test and train X and Y dataframes
Xtrain = data_train.copy()
Xtest  = data_test.copy()

# Create response
ytrain = data_train['Cancer_type']
ytest  = data_test['Cancer_type']
```

In [7]:

```python
#  Arrange the rows of the training set so that the 'AML' rows are grouped
#  together and the 'ALL' rows are together.
#  Generate a heat map of the data with expression values from the following genes:
#      D49818_at, M23161_at, hum_alu_at, AFFX-PheX-5_at, M15990_at.

import seaborn as sns
from IPython.display import display, HTML
```

```
plt.figure(figsize=(16, 12))

hm_columns = ['Cancer_type', 'D49818_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5_at

#  Group rows by 'Cancer_type'

Xtrain_hm = Xtrain[hm_columns].sort_values('Cancer_type')

#  Set row index to 'Cancer_type' for heat map plotting purposes
Xtrain_hm.set_index('Cancer_type', inplace=True)

#  Generate heat map, label cells with standardized gene expression value
ax = sns.heatmap(Xtrain_hm, annot=True, linewidths = .5)
ax.set_ylabel('Cancer_type', fontsize = 14)
ax.set_xlabel("Genes of Interest", fontsize = 14)
plt.show()

#  Output mean for each of these genes grouped by 'Cancer_type'
print ("Mean of heatmap plotted genes grouped by 'Cancer_type':")
display (Xtrain.groupby(['Cancer_type'])['D49818_at', 'M23161_at', 'hum_alu_at', 'AF

#  Output standard deviation for each of these genes grouped by 'Cancer_type'
print ("Standard deviations of heatmap plotted genes grouped by 'Cancer_type':")
Xtrain.groupby(['Cancer_type'])['D49818_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5
```



```
Mean of heatmap plotted genes grouped by 'Cancer_type':
```

	D49818_at	M23161_at	hum_alu_at	AFFX-PheX-5_at	M15990_at
Cancer_type					
0.0	0.504676	0.374248	0.465162	0.55364	0.266351
1.0	0.573192	0.255162	0.530630	0.62931	0.231240

```
Standard deviations of heatmap plotted genes grouped by 'Cancer_type':
```

```
Out[7]:
```

	D49818_at	M23161_at	hum_alu_at	AFFX-PheX-5_at	M15990_at
Cancer_type					
0.0	0.235649	0.229664	0.278974	0.261095	0.226137
1.0	0.190286	0.160412	0.222520	0.246693	0.156332

```
In [8]:
```

```
#Find the top two principal components for the gene expression data.
pca = PCA(n_components = 2)
X2 = pca.fit_transform(Xtrain[Xtrain.columns.difference(['Cancer_type'])])

dfpca = pd.DataFrame({"target":Xtrain.Cancer_type})
dfpca['pc1'] = X2[:,0]
dfpca['pc2'] = X2[:,1]
dfpca.head()
```

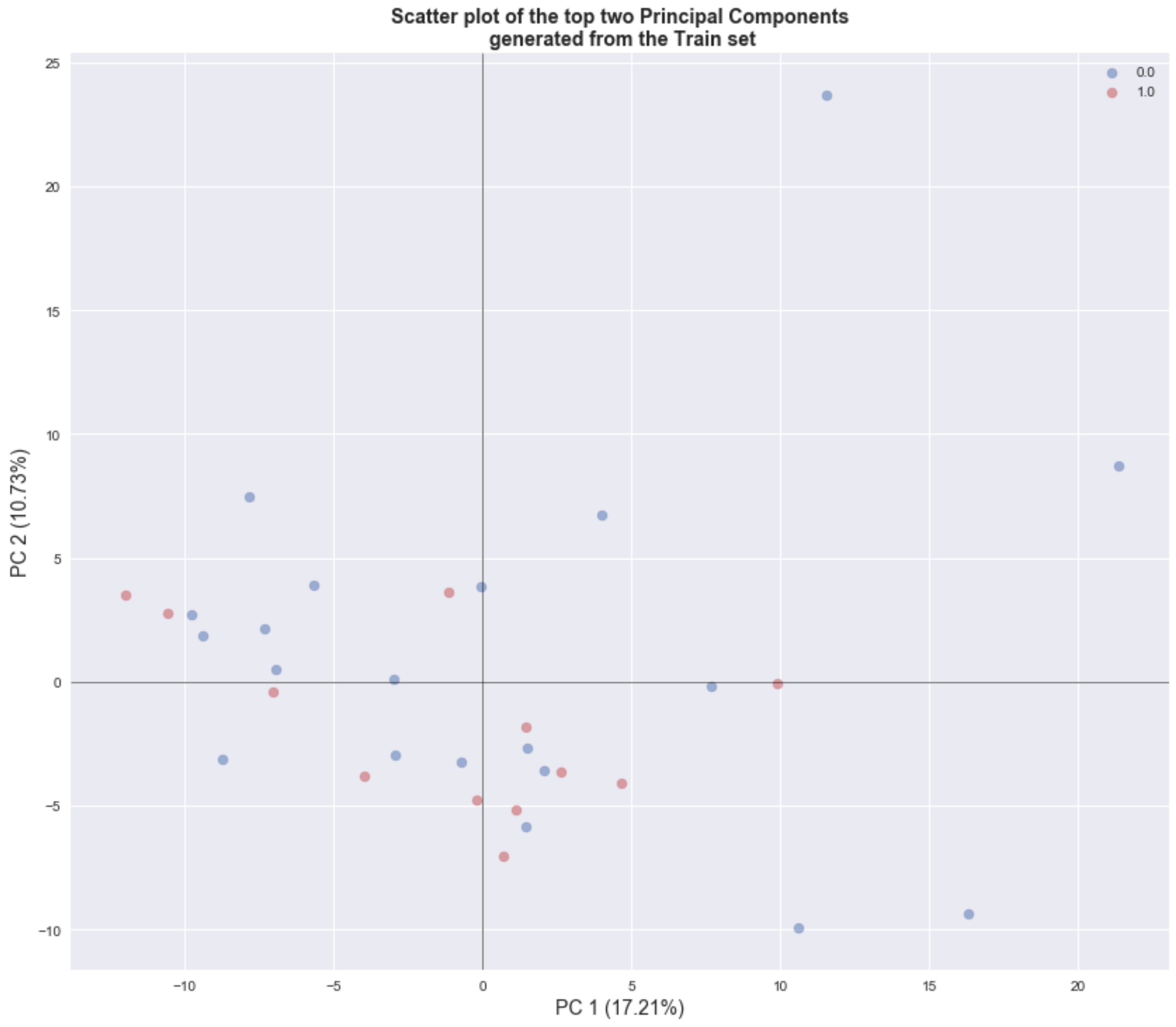
```
Out[8]:
```

	target	pc1	pc2
0	0.0	7.667012	-0.181414
1	0.0	-8.705269	-3.125841
2	0.0	21.341975	8.695756
3	0.0	11.527633	23.669014
4	0.0	-7.842507	7.473386

```
In [9]:
```

```
#Generate a scatter plot using these principal components, highlighting the AML and
c0=sns.color_palette()[0]; c2=sns.color_palette()[2]
colors = [c0, c2]
plt.figure(figsize=(14, 12))
#
for label, color in zip(dfpca['target'].unique(), colors):
    mask = dfpca['target']==label
    plt.scatter(dfpca[mask]['pc1'],
                dfpca[mask]['pc2'],
                c = color,
                label = label,
                alpha = 0.5,
                s = 50)

# Add the axis labels
plt.xlabel('PC 1 (%.2f%%)' % (pca.explained_variance_ratio_[0]*100), fontsize = 14)
plt.ylabel('PC 2 (%.2f%%)' % (pca.explained_variance_ratio_[1]*100), fontsize = 14)
plt.axhline(y = 0, color = 'k', linewidth = 0.5)
plt.axvline(x = 0, color = 'k', linewidth = 0.5)
plt.title('Scatter plot of the top two Principal Components\n generated from the Tra
plt.legend()
```

```
Out[9]:

<matplotlib.legend.Legend at 0x118aa6780>
```



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

A: Yes, Because there are more significantly more dimensions (7129) than training samples (32), the model's predictive ability may exhibit high variance (overfitting). The interpretability of the model will suffer too. It will be more challenging, if not impossible, to easily ascertain the most important predictors of the model.

By observing the heat map, comment on which of these genes are useful in discriminating between the two classes.

From examining the heat map above, we didn't discern any clear patterns so we also computed the per class means and standard deviations for each gene directly below the heat map. For each of the five genes, the combination of the per class means being sufficiently close together along with the per class standard deviations being large enough to overlap into the other cancer class mean value makes us question the usefulness of these five genes for discriminating between the two classes.

If we were to choose the best gene for this purpose it would be the **M23161_at** cancer gene since it exhibits the largest difference in per cancer class mean values along with the lowest per cancer class standard deviation.

How well do the top two principal components discriminate between the two classes?

A: Based on examining the scatter plot of the top two principal components above, the top two principal components don't do a very good job of discriminating between the two classes. There is no discernable clustering of each cancer class. Rather, both classes still look randomly distributed in the scatter plot with no clear boundary between the two cancer classes.

There are a few points of cancer type ALL that are high in both principal components (towards the upper right quadrant of the chart), while there are no neighboring AML samples, but this may be a coincidence and not a significant occurrence.

Part (b): Linear Regression vs. Logistic Regression

Begin by analyzing the differences between using linear regression and logistic regression for classification. For this part, you shall work with a single gene predictor: `M23161_at`.

1. Fit a simple linear regression model to the training set using the single gene predictor `D29963_at`. We could interpret the scores predicted by regression model interpreted for a patient as an estimate of the probability that the patient has the `ALL` type cancer (class 1). Is there a problem with this interpretation?
2. The fitted linear regression model can be converted to a classification model (i.e. a model that predicts one of two binary labels 0 or 1) by classifying patients with predicted score greater than 0.5 into the `ALL` type (class 1), and the others into the `AML` type (class 0). Evaluate the classification accuracy (1 - misclassification rate) of the obtained classification model on both the training and test sets.
3. Next, fit a simple logistic regression model to the training set. How does the training and test calssification accuracy of this model compare with the linear regression model? Remember, you need to set the regularization parameter for sklearn's logistic regression function to be a very large value in order not to regularize (use 'C=100000').

4. Plot the quantitative output from linear regression model and the probabilistic output from the logistic regression model (on the training set points) as a function of the gene predictor. Also, display the true binary response for the training set points in the same plot. Based on these plots, does one of the models appear better suited for binary classification than the other? Explain.

In [10]:

```
# Fit a simple linear regression model to the training set using the
# single gene predictor D29963_at.
# Evaluate the classification accuracy on both the training and test sets.

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix

Xtrain_D29963 = Xtrain['D29963_at'].reshape(-1, 1)
Xtest_D29963  = Xtest['D29963_at'].reshape(-1, 1)

linreg = LinearRegression().fit(Xtrain_D29963, ytrain)

linreg_preds_train = (linreg.predict(Xtrain_D29963))
linreg_preds_test  = (linreg.predict(Xtest_D29963))

print("Train R2 Score: "+str(r2_score(ytrain, linreg_preds_train)))

print("Test R2 Score: "+str(r2_score(ytest, linreg_preds_test)))


print ("Train Classification accuracy is: " + str(accuracy_score(ytrain, np.round(l
print ("\nTest Classification accuracy is: " + str(accuracy_score(ytest, np.round(l

print ("\n\nTrain Confusion Matrix:")
print (confusion_matrix(ytrain, np.round(linreg_preds_train), labels=None))
print ("\nTest Confusion Matrix:")
print (confusion_matrix(ytest, np.round(linreg_preds_test), labels=None))
```

```
Train R2 Score: 0.100344879474
Test R2 Score: 0.329972737867
Train Classification accuracy is: 0.71875

Test Classification accuracy is: 0.853658536585


Train Confusion Matrix:
[[19  1]
 [ 8  4]]

Test Confusion Matrix:
[[27  1]
 [ 5  8]]
```

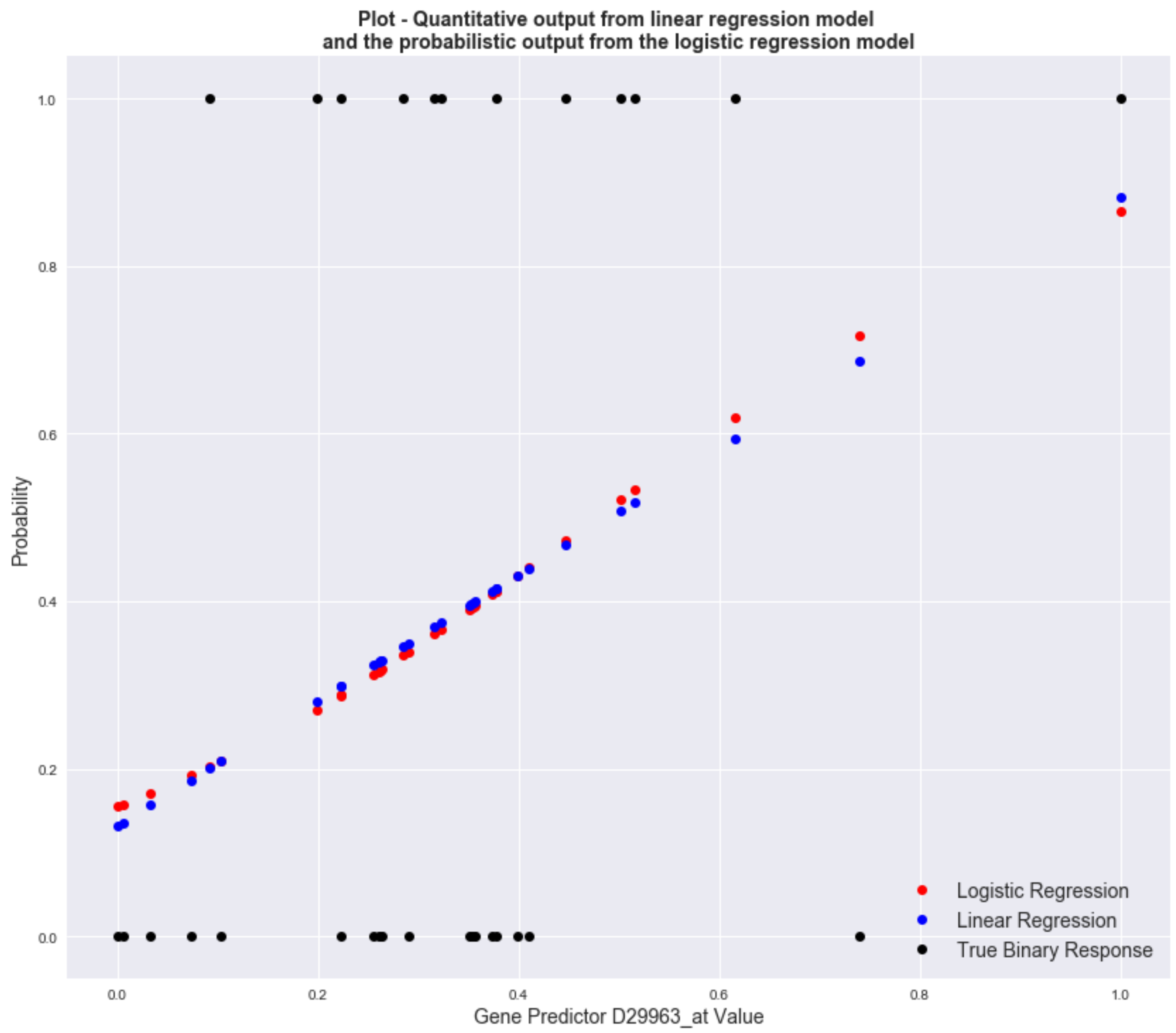
In [11]:

```
# Create a logistic regression estimator
logreg_one_pred = LogisticRegression(C=100000)
logreg_one_pred.fit(Xtrain_D29963, ytrain)
logreg_preds_train = (logreg_one_pred.predict(Xtrain_D29963))
logreg_preds_test  = (logreg_one_pred.predict(Xtest_D29963))

print ("Logistic Regression - Train Classification accuracy is: "
       + str(logreg_one_pred.score(Xtrain_D29963, ytrain)))
print ("\nLogistic Regression - Test Classification accuracy is: "
       + str(logreg_one_pred.score(Xtest_D29963, ytest)))
print ("\n\nLogistic Regression - Train Confusion Matrix:")
print (confusion_matrix(ytrain, np.round(logreg_preds_train), labels=None))
print ("\nLogistic Regression - Test Confusion Matrix:")
print (confusion_matrix(ytest, np.round(logreg_preds_test), labels=None))

# LogisticRegression.predict_proba gives us estimates for each classifier
# to compare it to the LinearRegression estimates, we need to use the second classi
# probability (the probability it was 1.0, instead of 0.0)
logreg_pred_proba_train = logreg_one_pred.predict_proba(Xtrain_D29963)[:,1]
```

```
Logistic Regression - Train Classification accuracy is: 0.71875

Logistic Regression - Test Classification accuracy is: 0.829268292683


Logistic Regression - Train Confusion Matrix:
[[19  1]
 [ 8  4]]

Logistic Regression - Test Confusion Matrix:
[[26  2]
 [ 5  8]]
```

In [12]:

```
# Create a scatter plot showing the linear and logistic regression probability pred
plt.figure(figsize=(14, 12))
plt.plot(Xtrain_D29963, logreg_pred_proba_train, 'ro', label = 'Logistic Regression
plt.plot(Xtrain_D29963, linreg_preds_train, 'ro', c='b', label = 'Linear Regression
plt.plot(Xtrain_D29963, ytrain, 'ro', c='black', label = 'True Binary Response')
plt.title('Plot - Quantitative output from linear regression model \nand the probab
from the logistic regression model', fontweight='bold', fontsize=14)
plt.xlabel('Gene Predictor D29963_at Value', fontsize = 14)
plt.ylabel('Probability', fontsize = 14)
plt.legend(fontsize = 14)
plt.show()
```



Is there a problem with this interpretation? (Interpret the scores predicted by regression model interpreted for a patient as an estimate of the probability)

Yes. The linear regression score indicates how FAR from true the value was, which is almost meaningless (or sometimes entirely meaningless) in a classification problem.

Even in a classification problem with only two classes, it may be that the calculated prediction for most values is somewhere around the threshold -- there is relatively little variance in the predictions, although that little variance is very important and can be used for an excellent classification model.

Another problem with this interpretation is the linear regression model's predictions are not probabilities (unlike the logistic regression's predictions) and that's what a classification model should predict for each category. Furthermore, if we wanted to extend this model to predict more than two cancer classes, which have no natural ordering, the linear regression model's predictions would make no sense at all.

Evaluate the classification accuracy (1 - misclassification rate) of the obtained classification model on both the training and test sets.

There were 23 correctly out of 32 correctly predicted samples in the training dataset, for a classification accuracy of 0.719. There were 35 out of 41 correctly predicted samples in the test dataset, for a classification accuracy of 0.854.

How does the Logistic Regression training and test classification accuracy of this model compare with the linear regression model?

The scores are lower for the logistic regression model, although the problems are the same -- they are not higher (lower in the case of the train score, only slightly higher in the case of the test score) than simply guessing "ALL" every time, in which case you'd score 0.667. The test data score is higher than the train data score, indicating that the model is not well fitted. However, it should be noted that the test and train scores in the logistic regression model do have somewhat better agreement, which may or may not be significant.

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

The linear regression plot is, as one would expect, exactly linear. The logistic regression plot has a very slight logistic curve to it, which can go above and below the linear regression to fit the data. However, it is not apparent from the plot whether this improves its predictive powers in this particular binary classification.

Part (c): Multiple Logistic Regression

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 Part (b) with a single gene (on both the training and test sets)?
2. "Use the `visualize_prob` from `HW5_functions.py` 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 `ALL` and `AML` classes shown in different colors, and with the 0.5 threshold highlighted using a dotted horizontal line. Is there a difference in the spread of probabilities in the training and test plots? Are there data points for which the predicted probability is close to 0.5? If so, what can you say about these points?"

In [13]:

```
# Remove the response variable from Xtrain and Xtest
Xtrain = Xtrain[Xtrain.columns.difference(['Cancer_type'])]
Xtest  = Xtest[Xtest.columns.difference(['Cancer_type'])]
```

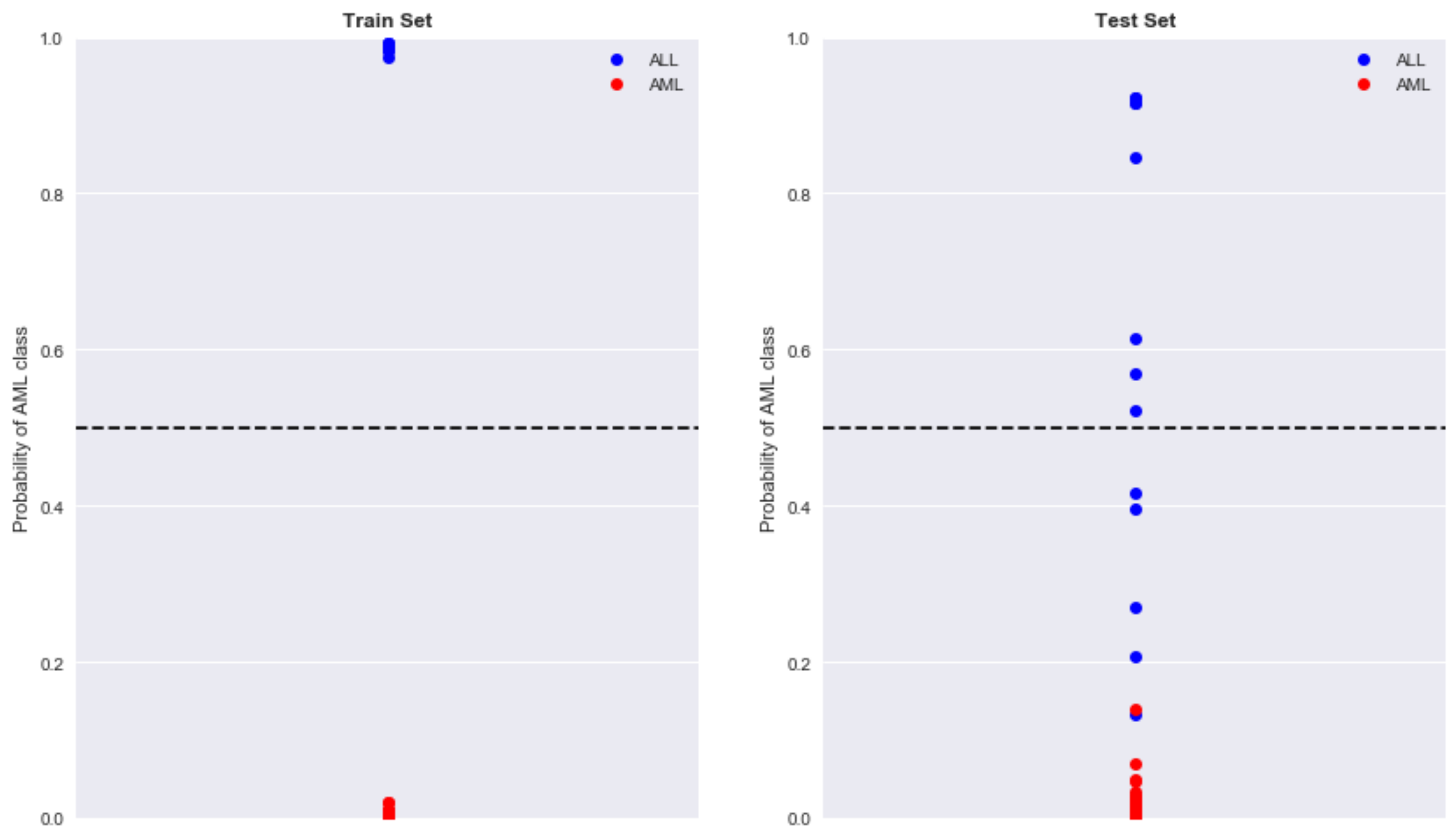
In [14]:

```
from HW5_functions import visualize_prob
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix

# Create a logistic regression on all predictors, get train and test predictions
logreg = LogisticRegression().fit(Xtrain, ytrain)
train_preds = logreg.predict(Xtrain)
test_preds = logreg.predict(Xtest)

# Get the test and train classification scores
print ("\nMultiple Logistic Regression Train Classification accuracy is: " +
       str(logreg.score(Xtrain, ytrain)))
print ("Multiple Logistic Regression Test Classification accuracy is:  " +
       str(logreg.score(Xtest, ytest)))

# Create a true/false positive/negative matrix for test and train predictions
print ("\nMultiple Logistic Regression Train Confusion Matrix:")
print (confusion_matrix(ytrain, np.round(train_preds), labels=None))
print ("\nMultiple Logistic Regression Test Confusion Matrix:")
print (confusion_matrix(ytest, np.round(test_preds), labels=None))
```

```
Multiple Logistic Regression Train Classification accuracy is: 1.0
Multiple Logistic Regression Test Classification accuracy is:  0.87804
8780488

Multiple Logistic Regression Train Confusion Matrix:
[[20  0]
 [ 0 12]]

Multiple Logistic Regression Test Confusion Matrix:
[[28  0]
 [ 5  8]]
```

In [15]:

```
from HW5_functions import visualize_prob

# Plot the predicted probability visualization for test and train data
fig = plt.figure(figsize=(14, 18))
ax = fig.add_subplot(221)
visualize_prob(logreg, Xtrain, ytrain, ax)
plt.title('Train Set', fontweight='bold')

ax = fig.add_subplot(222)
visualize_prob(logreg, Xtest, ytest, ax)
plt.title('Test Set', fontweight='bold')
plt.show()
```



Q: How does the classification accuracy of this model compare with the models fitted in Part (b) with a single gene (on both the training and test sets)?

This simple multi logistic regression is extremely overfitted. Because there are so many parameters, we are able to build a model that fits the training dataset perfectly. However, the score on the test set is lower than the score where just a single gene was considered.

Q: Is there a difference in the spread of probabilities in the training and test plots?

Yes, there is a significant difference in the spread of probabilities between the training and test plots. The AML labeled training data is tightly clustered near probability 1.0 and the ALL labeled training data is tightly clustered near probability 0.0 in the training plot.

There is far greater spread of probabilities of AML and ALL labeled data in the test plot. The AML labeled test data is spread out significantly across most of the probability range (~0.1 to 0.9) in the test plot instead of being tightly clustered near probability 1.0 in the training plot. The ALL labeled test data is spread out more between probability 0.0 and 0.1. While this is not nearly as much spread as the AML labeled test data, it is no longer tightly clustered as compared to the ALL labeled data in the training plot.

Q: Are there data points for which the predicted probability is close to 0.5?

Yes, in the test plot there are some AML labeled data points that whose probabilities are near 0.5 (0.4 - 0.6). The train data is extremely well separated and contains no values between ~5 and 95%

Q: If so, what can you say about these points?

Because these points are all AML labeled data points, it might be worthwhile to reduce the classification threshold for AML from 0.5. From looking at the test plot, changing the threshold to 0.2 would most likely reduce misclassifation error.

Part (d): Analyzing Significance of Coefficients

How many of the coefficients estimated by the multiple logistic regression in the previous problem are significantly different from zero at a *significance level of 95%*?

Hint: To answer this question, use *bootstrapping* with 100 boostrap samples/iterations.

In [16]:

```
from sklearn.utils import resample

# Get a sampling of coefficients for all predictors
coefficients = []
for i in range(100):
    Xtrain_sampled = resample(data_train, n_samples=100)
    ytrain_sampled = Xtrain_sampled['Cancer_type']
    Xtrain_sampled = Xtrain_sampled[Xtrain_sampled.columns.difference(['Cancer_type
    logreg = LogisticRegression().fit(Xtrain_sampled, ytrain_sampled)
    coefficients.append(logreg.coef_[0])

coefficients = np.array(coefficients)
coefficients_df = pd.DataFrame(np.array(coefficients), columns = list(Xtrain.keys()

# Get a dataframe with statistics in the columns and predictors in the rows
coef_stats = coefficients_df.describe().transpose()
print("Number all Stats:")
print(len(coef_stats))

# Find the upper and lower bounds of significance at 95%
coef_stats['lower_95'] = coef_stats['mean'] - coef_stats['std']*2
coef_stats['upper_95'] = coef_stats['mean'] + coef_stats['std']*2

coef_stats['significant'] = coef_stats.apply(lambda row: 0 if row['lower_95'] < 0 a
significant_stats = coef_stats.loc[coef_stats['significant'] == 1]
print("Number significant stats:")
print(len(significant_stats))

significant_stats.head()
```

```
Number all Stats:
7129
Number significant stats:
5104
```

Out[16]:

	count	mean	std	min	25%	50%	75%	max
AB000114_at	100.0	-0.028255	0.002911	-0.035092	-0.030273	-0.027806	-0.026505	-0.021864
AB000115_at	100.0	-0.012271	0.002369	-0.017572	-0.013806	-0.012661	-0.011047	-0.005168
AB000220_at	100.0	-0.020155	0.003173	-0.030917	-0.022022	-0.020191	-0.018376	-0.013361
AB000409_at	100.0	-0.008521	0.003018	-0.016616	-0.010361	-0.008815	-0.006389	0.000223
AB000449_at	100.0	-0.020076	0.001750	-0.025734	-0.021022	-0.020125	-0.019223	-0.015512

Q: How many of the coefficients estimated by the multiple logistic regression in the previous problem

are significantly different from zero at a *significance level of 95%*?

There are 5,164 significant stats at a significance level of 95% (two standard deviations on either side away from 0)

Part (e): Dimensionality Reduction using PCA

A reasonable approach to reduce the dimensionality of the data is to use PCA and fit a logistic regression model on the first set of principal components contributing to 90% of the variance in the predictors.

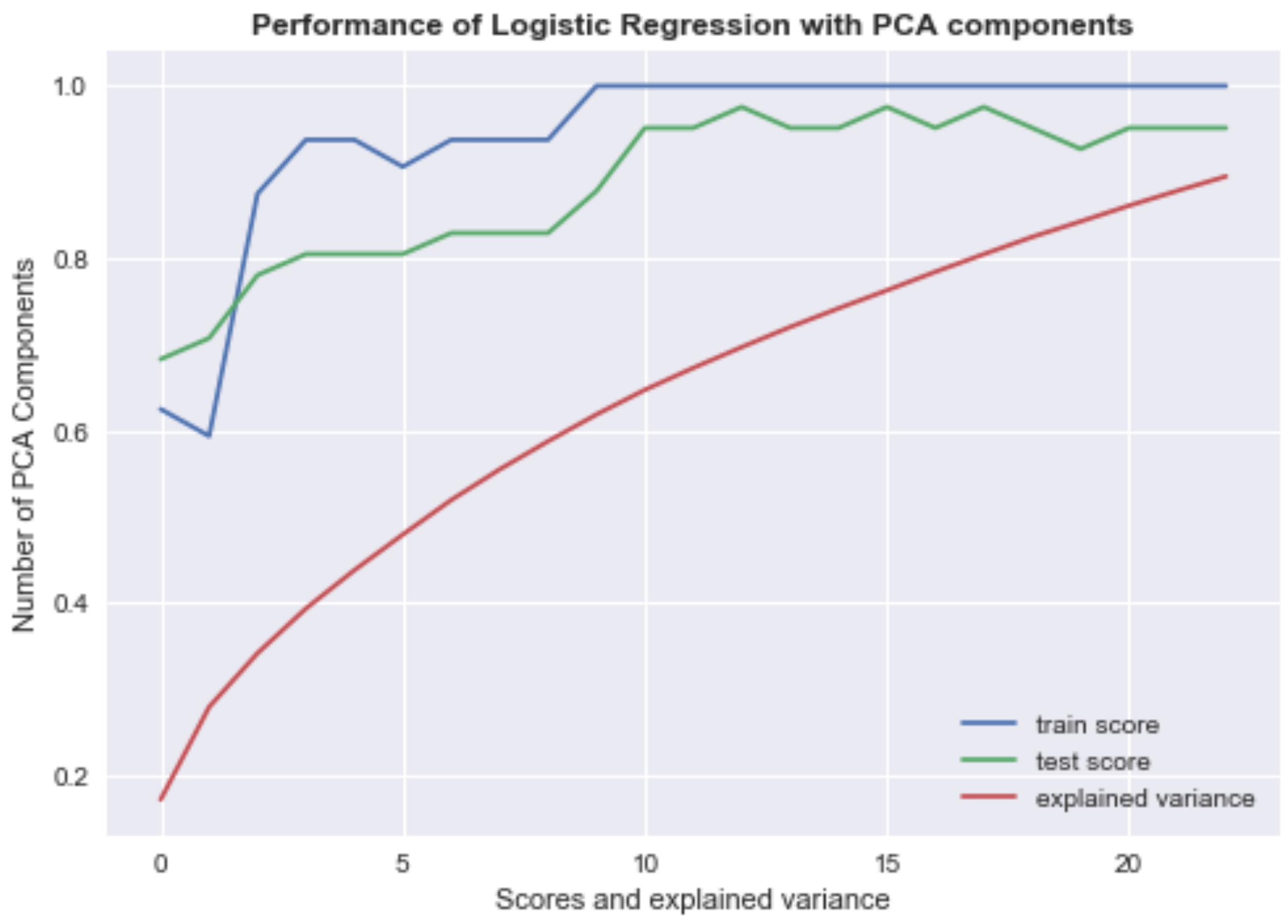
1. How do the classification accuracy values on both the training and tests sets compare with the models fitted in Parts (c) and (d)?
2. Re-fit a logistic regression model using 5-fold cross-validation to choose the number of principal components, and comment on whether you get better test performance than the model fitted above (explain your observations).
3. Use the code provided in Part (c) to visualize the probabilities predicted by the fitted models on both the training and test sets. How does the spread of probabilities in these plots compare to those for the models in Part (c) and (d)?

In [17]:

```
# Collect logistic regression scores across many PCA components
scores = {'train': [], 'test': []}
explainedVariances = []
for n_comp in range(1,24):
    pca = PCA(n_components = n_comp)
    pca.fit(Xtrain)
    # Get the cumulative explained variance of all of the components
    explainedVariances.append(pca.explained_variance_ratio_.cumsum()[-1])
    Xtrain_pca = pca.transform(Xtrain)
    Xtest_pca = pca.transform(Xtest)
    # Fit to a logistic regression model for scoring
    logreg = LogisticRegression()
    logreg.fit(Xtrain_pca, ytrain)

    scores['train'].append(logreg.score(Xtrain_pca, ytrain))
    scores['test'].append(logreg.score(Xtest_pca, ytest))

# Get the best scores and corresponding number of PCA componets for the test and tr
maxIdx = np.argmax(scores['test'])
print('Best test score:')
print(scores['test'][maxIdx])
print('Number of components:')
print(maxIdx+1)

maxTrainIdx = np.argmax(scores['train'])
print("Best train score:")
print(scores['train'][maxTrainIdx])
print('Number of components:')
print(maxTrainIdx+1)
```

```
# Plot the scores for test and train data, along with explained variance of the PCA
# by the number of PCA components
plt.xlabel("Scores and explained variance")
plt.ylabel("Number of PCA Components")
plt.plot(scores['train'], label='train score')
plt.plot(scores['test'], label='test score')
plt.plot(explainedVariances, label='explained variance')
plt.title('Performance of Logistic Regression with PCA components', fontweight='bolc
plt.legend(loc='lower right')
plt.show()
```

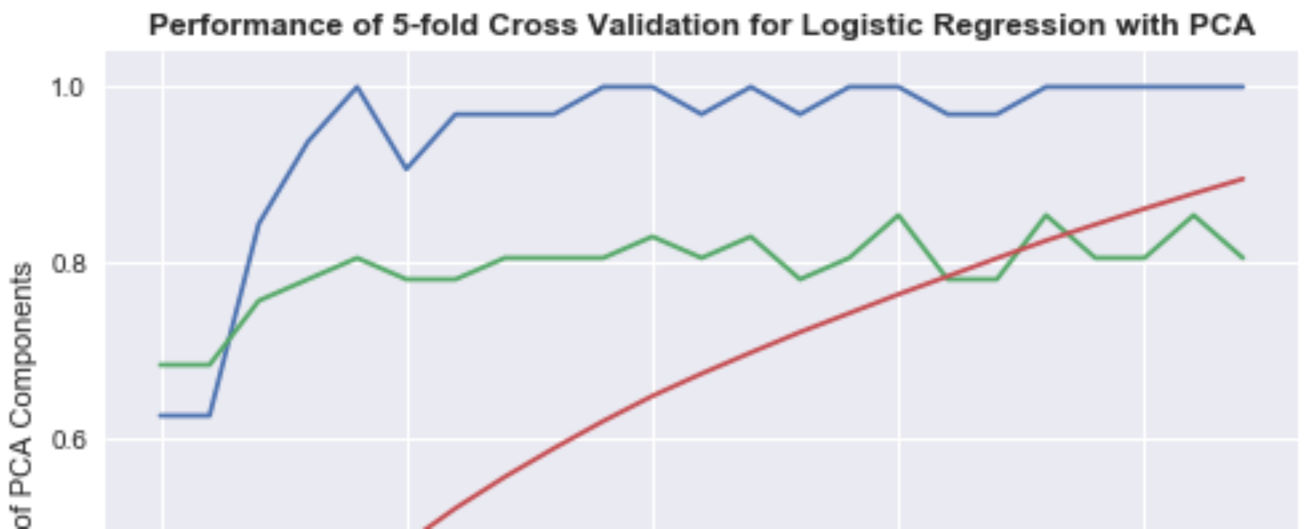
```
Best test score:
0.975609756098
Number of components:
13
Best train score:
1.0
Number of components:
10
```

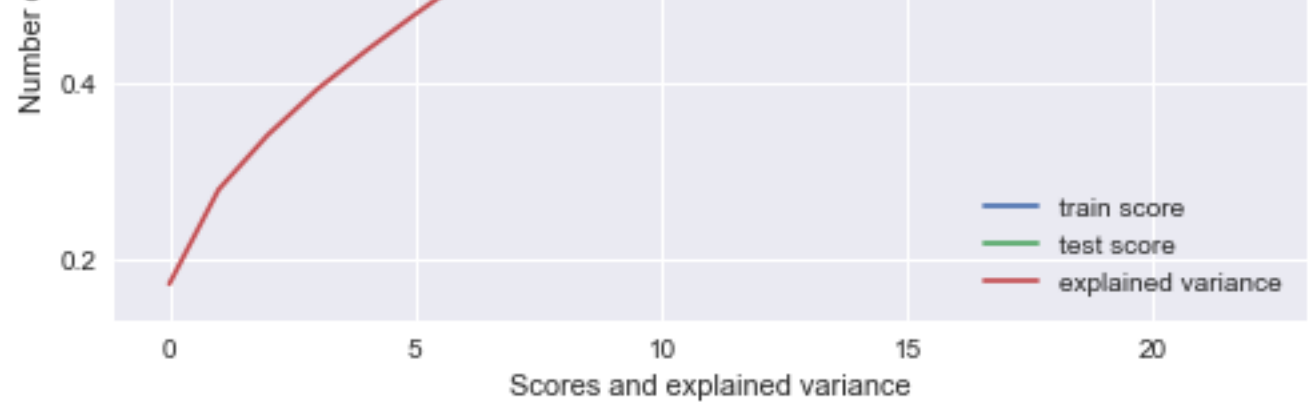


In [18]:

```
# Similar to the approach above, use cross validation in lieu of bootstrapping
cv_scores = {'train': [], 'test': []}
cv_variances = []
for n_comp in range(1,24):
    pca = PCA(n_components = n_comp)
    pca.fit(Xtrain)
    # Get the cumulative explained variance of all of the components
    cv_variances.append(pca.explained_variance_ratio_.cumsum()[-1])
    Xtrain_pca = pca.transform(Xtrain)
    Xtest_pca = pca.transform(Xtest)

    # Fit to a logistic regression with 5 fold cross validation
    logreg = LogisticRegressionCV(cv=5)
    logreg.fit(Xtrain_pca, ytrain)

    cv_scores['train'].append(logreg.score(Xtrain_pca, ytrain))
    cv_scores['test'].append(logreg.score(Xtest_pca, ytest))

maxIdx = np.argmax(cv_scores['test'])
print('Best test score:')
print(cv_scores['test'][maxIdx])
print('Number of components:')
print(maxIdx+1)

maxTrainIdx = np.argmax(cv_scores['train'])
print('Best train score:')
print(cv_scores['train'][maxTrainIdx])
print('Number of components:')
print(maxTrainIdx+1)

plt.xlabel("Scores and explained variance")
plt.ylabel("Number of PCA Components")
plt.plot(cv_scores['train'], label='train score')
plt.plot(cv_scores['test'], label='test score')
plt.plot(cv_variances, label='explained variance')
plt.title('Performance of 5-fold Cross Validation for Logistic Regression with PCA',
plt.legend(loc='lower right')
plt.show()
```

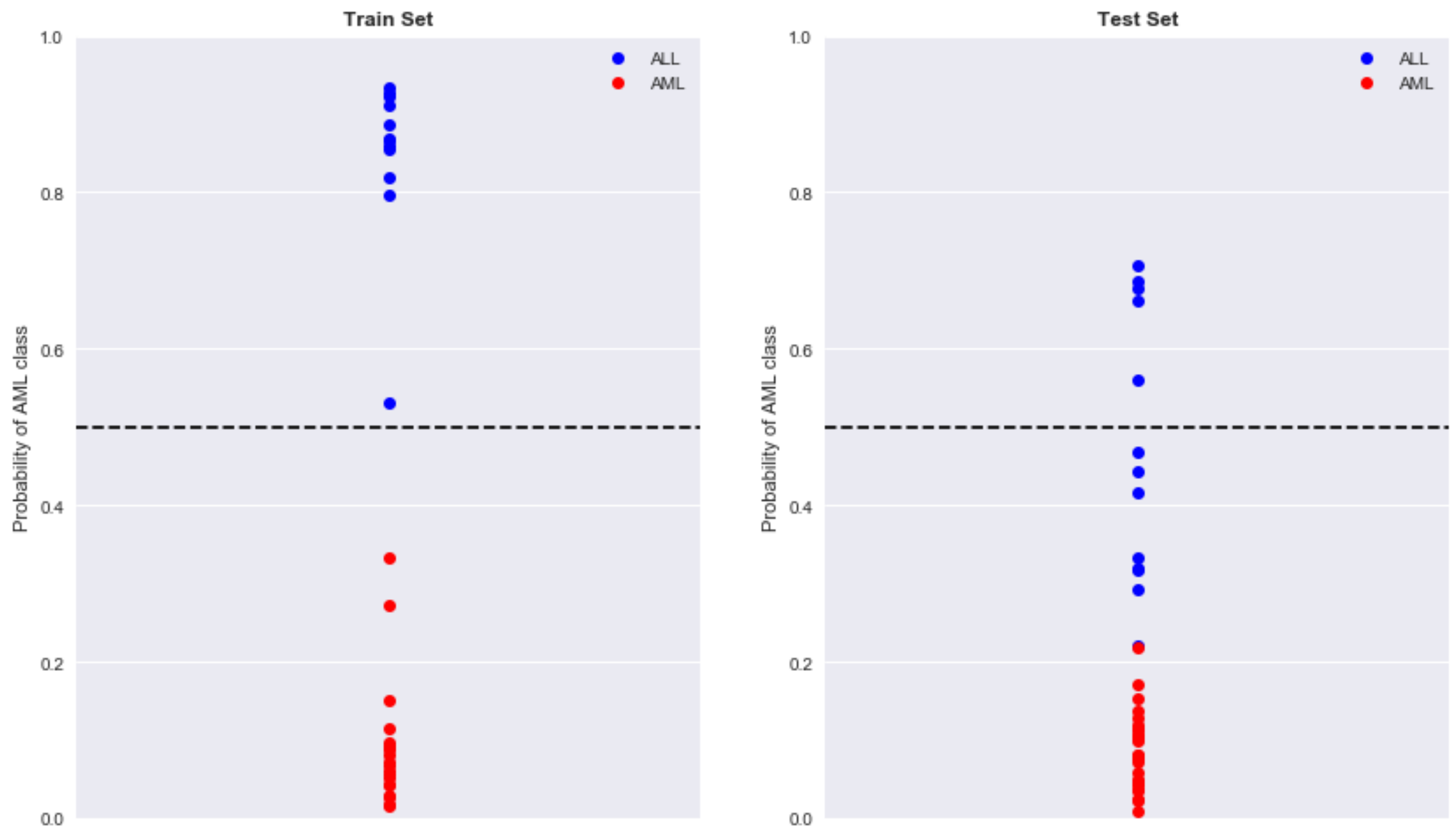
```
Best test score:
0.853658536585
Number of components:
16
Best train score:
1.0
Number of components:
5
```





In [19]:

```
pca.fit(Xtrain)
Xtrain_pca = pca.transform(Xtrain)
Xtest_pca = pca.transform(Xtest)

# Plot the performance on logistic regression with 5 fold cross validation
logreg = LogisticRegressionCV(cv=5)
logreg.fit(Xtrain_pca, ytrain)

fig = plt.figure(figsize=(14, 18))
ax = fig.add_subplot(221)
visualize_prob(logreg, Xtrain_pca, ytrain, ax)
plt.title('Train Set', fontweight='bold')

ax = fig.add_subplot(222)
visualize_prob(logreg, Xtest_pca, ytest, ax)
plt.title('Test Set', fontweight='bold')
plt.show()
```



Q: How do the classification accuracy values on both the training and tests sets compare with the

models fitted in Parts (c) and (d)?

The logistic regression without cross validation performed much better than any of the models seen so far. 0.976 on the test data for 18 PCA components. However, this peak in performance appears to be a spike, and may be an artifact of the data. The logistic regression exactly fit the test data after 8 PCA components, indicating overfitting.

Q: Re-fit a logistic regression model using 5-fold cross-validation to choose the number of principal components, and comment on whether you get better test performance than the model fitted above (explain your observations).

The test score was 0.854 with the 5 fold cross validation (at 11 components), and the test data was fit perfectly earlier at 4 components (although this appears to be a lucky early spike). The performance was, overall, poorer than the logistic regression without cross validation. The data appears to have been extremely overfit, which caused poor test performance.

Q: Use the code provided in Part (c) to visualize the probabilities predicted by the fitted models on both the training and test sets. How does the spread of probabilities in these plots compare to those for the models in Part (c) and (d)?

Again, the cross validation fits the train data perfectly, while it fails on the test dataset. Unlike the earlier model, however, the spread of probabilities is much greater in the training dataset, indicating that less overfitting may be occurring than in part c. This fact is not immediately obvious simply by looking at the scores on the test datasets, alongside the fact that they perfectly fit the training set. This is likely the effect of the cross validation attempting to reduce overfitting on such a small training set.

```
In [ ]:
```