# S-109A Introduction to Data Science:

## Homework 5: Logistic Regression, High Dimensionality and PCA, LDA/QDA

**Harvard University**
**Summer 2018**
**Instructors**: Pavlos Protopapas, Kevin Rader

---

## INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- 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.

Names of people you have worked with goes here:

---

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, LinearRegression, RidgeCV
from sklearn.linear_model import LogisticRegressionCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler
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.metrics import r2_score

import math
from scipy.special import gamma

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

alpha = 0.5
```

# 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 `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 a classification models for this data set. We will also use Principal Components Analysis (PCA) to visualize the data and to reduce its dimensions.

# Question 1: 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. Lets explore a few of the genes and see how well they discriminate between cancer classes. Create a single figure with four subplots arranged in a 2x2 grid. Consider the following four genes: `D29963_at`, `M23161_at`, `hum_alu_at`, and `AFFX-PheX-5_at`. For each gene overlay two histograms of the gene expression values on one of the subplots, one histogram for each cancer type. Does it appear that any of these genes discriminate between the two classes well? How are you able to tell?
5. Since our data has dimensions that are not easily visualizable, we want to reduce the dimensionality of the data to make it easier to visualize. Using PCA, find the top two principal components for the gene expression data. Generate a scatter plot using these principal components, highlighting the two cancer types in different colors. How well do the top two principal components discriminate between the two classes? How much of the variance within the data do these two principal components explain?

**Answers:**

**1.1:** 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).

In [4]:

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

**1.2:** 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.

In [24]:

```
min_max_scaler = MinMaxScaler()
train_scaled = min_max_scaler.fit(data_train)
data_train_normalized =pd.DataFrame(train_scaled.transform(data_train), columns=
data_train.columns)

data_train_normalized.head()
```

Out[24]:

| | Cancer_type | AFFX-BioB-5_at | AFFX-BioB-M_at | AFFX-BioB-3_at | AFFX-BioC-5_at | AFFX-BioC-3_at | AFFX-BioDn-5_at | AFFX-BioDn-3_at |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.466192 | 0.739726 | 0.255814 | 0.246154 | 0.433190 | 0.240418 | 0.880427 |
| **1** | 0.0 | 0.658363 | 0.794521 | 0.213953 | 0.421978 | 0.573276 | 0.717770 | 0.741637 |
| **2** | 0.0 | 0.727758 | 0.857143 | 0.586047 | 0.107692 | 0.683190 | 0.649826 | 0.642705 |
| **3** | 0.0 | 0.000000 | 0.622309 | 0.348837 | 0.714286 | 0.200431 | 0.526132 | 0.708897 |
| **4** | 0.0 | 0.702847 | 0.745597 | 0.113953 | 0.224176 | 0.741379 | 0.620209 | 0.713167 |

5 rows × 7130 columns

**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?

Yes, I do forsee a problem trying to fit a model onto a dataset where there are more predictors than obsevrsations (p > n). For example, this means that there might not be one unique solution to the problem. There might be multiple solutions that could fit the data equally well. This could possibly mean that the data is overfit (finding a model that can correctly predict the relationship between the variables might be hard) because it is too flexible since multiple different solutions could fit the same training data.

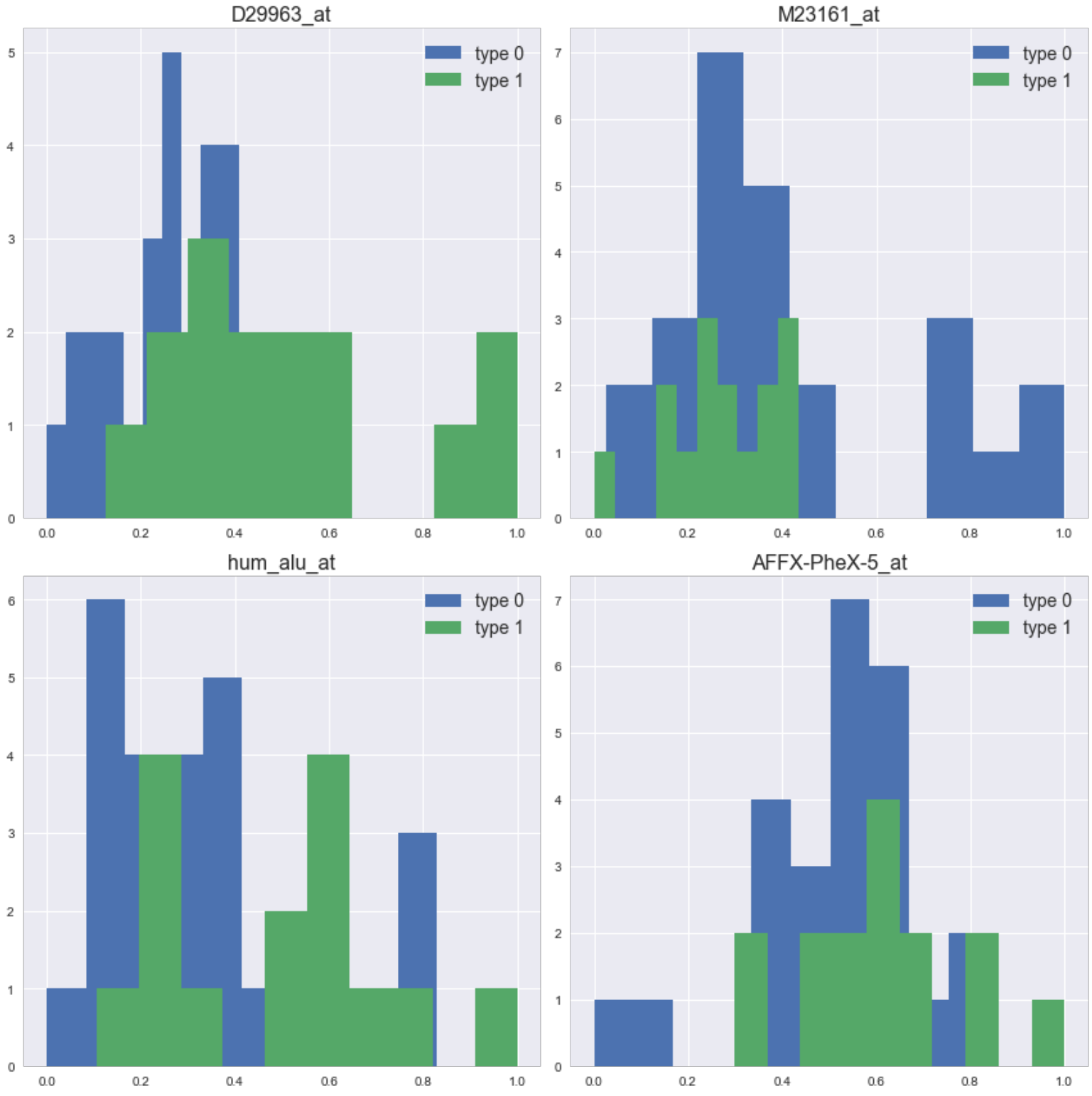
resources

https://stats.stackexchange.com/questions/223486/modelling-with-more-variables-than-data-points (https://stats.stackexchange.com/questions/223486/modelling-with-more-variables-than-data-points)
https://stats.stackexchange.com/questions/10423/number-of-features-vs-number-of-observations (https://stats.stackexchange.com/questions/10423/number-of-features-vs-number-of-observations)
https://stats.stackexchange.com/questions/56141/explanation-of-minimum-observations-for-multiple-regression (https://stats.stackexchange.com/questions/56141/explanation-of-minimum-observations-for-multiple-regression)

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

In [25]:

```
type0 = data_train_normalized.loc[(data_train_normalized['Cancer_type'] == 0.0)]
type1 = data_train_normalized.loc[(data_train_normalized['Cancer_type'] == 1.0)]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12,12))
ax0, ax1, ax2, ax3 = axes.flatten()

ax0.hist(type0['D29963_at'], label='type 0')
ax0.hist(type1['D29963_at'], label='type 1')
ax0.legend(prop={'size': 14})
ax0.set_title('D29963_at', fontsize=16)

ax1.hist(type0['M23161_at'], label='type 0')
ax1.hist(type1['M23161_at'], label='type 1')
ax1.legend(prop={'size': 14})
ax1.set_title('M23161_at',fontsize=16)

ax2.hist(type0['hum_alu_at'], label='type 0')
ax2.hist(type1['hum_alu_at'], label='type 1')
ax2.legend(prop={'size': 14})
ax2.set_title('hum_alu_at',fontsize=16)

ax3.hist(type0['AFFX-PheX-5_at'], label='type 0')
ax3.hist(type1['AFFX-PheX-5_at'], label='type 1')
ax3.legend(prop={'size': 14})
ax3.set_title('AFFX-PheX-5_at',fontsize=16)

fig.tight_layout()
plt.show()
```



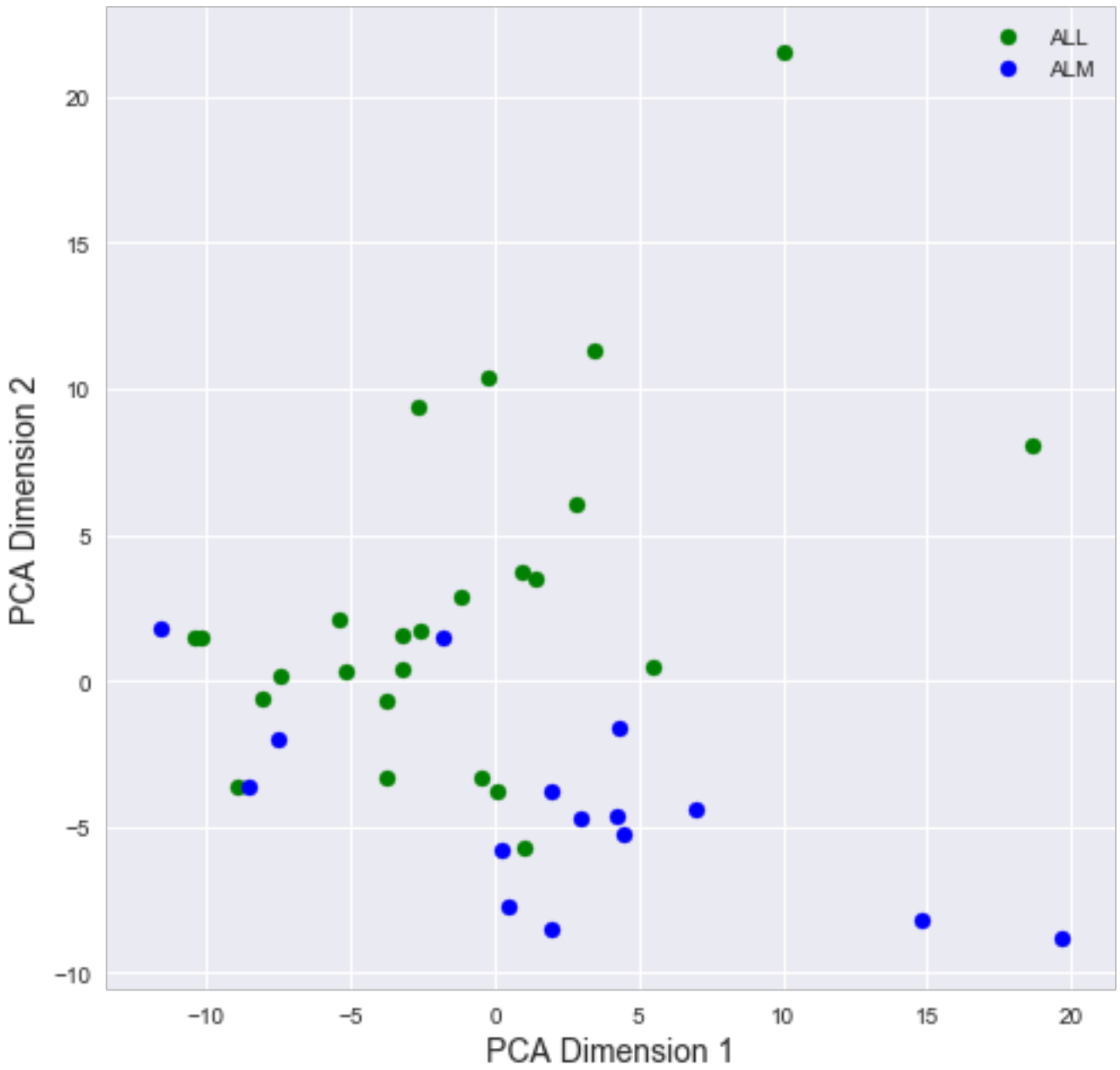

One way to tell if any of these four genes discriminate between the two classes well would be to look at the histograms and see if there is a clear deviation between the distribution of type 0 and type 1. Looking at the graphs above, this does not seem to be the case. None of them are discriminating between the two classes well. For example, looking at the D29963_at gene, there is overlap between type 0 and type 1's expression distribution, therefore it cannot be said that D29963 discriminates. Same with the other three genes.

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

In [49]:

```
y_train = data_train_normalized['Cancer_type']

x_train = data_train_normalized[data_train_normalized.columns[data_train_normali
zed.columns!='Cancer_type'] ]

pca_transformer = PCA(2).fit(x_train)
x_train_2d = pca_transformer.transform(x_train)

colors = ['g', 'b']
label_text=['ALL', 'ALM']

fig, axe = plt.subplots(figsize=(8,8))
for cancer_type in [0,1]:
    cur_df = x_train_2d[y_train==cancer_type]
    plt.scatter(cur_df[:,0], cur_df[:,1], c = colors[cancer_type], label=label_t
ext[cancer_type])
plt.suptitle('Two Principal Components of PCA', fontsize=16);
plt.xlabel('PCA Dimension 1', fontsize=14)
plt.ylabel('PCA Dimension 2',fontsize=14)
plt.legend();


print('% of variance within data that these components explain:', 100*sum(pca_tr
ansformer.explained_variance_ratio_))
```

```
% of variance within data that these components explain: 27.31782945
2185432
```




These top two principal compononents do not discriminate between the two classes very well. As we can see from the graph above, there is a lot of overlap between the two classes, so it is hard to discriminate between the two classes. If they were discriminated well, there would be clear seperation between the two colors.

The top two principal components explain about 27.31% of the variance within the training data set.

# Question 2: 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.

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 the regression model interpreted for a patient as an estimate of the probability that the patient has `Cancer_type`=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 `Cancer_type`=1, and the others into the `Cancer_type`=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 do the training and test classification accuracies 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 to not regularize (use 'C=100000').
4. Plot the quantitative output from the 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.

**Answers:**

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

In [50]:

```
train_x = data_train_normalized['D29963_at']
y_train = data_train_normalized['Cancer_type']

train_x = train_x.values.reshape(-1,1)

linreg = LinearRegression()

OLSModel = linreg.fit(train_x, y_train)
```

Yes, there is a problem with this interpretation. In this part of the question, we are using a linear regression model and the question asks if we can predict using the results of linear regression. The answer to that is no. Since linear regression only fits a linear line to the data, we cannot predict from it. We need a probability between 0 and 1 to do this and linear regression cannot do that. Logistic regression is the one that keeps all of its values between 0 and 1 and does exactly what is needed; outputting an estimate of the probability.

**2.2:** 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 `Cancer_type=1`, and the others into the `Cancer_type=0`. Evaluate the classification accuracy (1 - misclassification rate) of the obtained classification model on both the training and test sets.

In [52]:

```
# normalize using same scale as training set
data_test_normalized = pd.DataFrame(train_scaled.transform(data_test), columns=d
ata_test.columns)

y_test = data_test_normalized['Cancer_type']
x_test = data_test_normalized['D29963_at']
x_test = x_test.values.reshape(-1,1)

print('Classification Accuracy from LineReg')
print('---------')
score = accuracy_score(y_train, OLSModel.predict(train_x).round(), normalize=Tru
e)
print('Training set:', score)
print('Misclassification Rate:', 1-score)
score2 = accuracy_score(y_test, OLSModel.predict(x_test).round(), normalize=True
)
print('---------')
print('Test set:', score2)
print('Misclassification Rate:', 1-score2)
```

```
Classification Accuracy from LineReg
---------
Training set: 0.8
Misclassification Rate: 0.19999999999999996
---------
Test set: 0.7575757575757576
Misclassification Rate: 0.24224242424242243
```

**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? 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').

In [54]:

```
logit = LogisticRegression(C=1000000)
logit.fit(train_x, y_train)

print('Classification Accuracy from LogReg')
print('---------')
score3 = accuracy_score(y_train, logit.predict(train_x).round(), normalize=True)
print('Training set:', score3)
print('Misclassification Rate:', 1-score3)
print('---------')
score4 = accuracy_score(y_test, logit.predict(x_test), normalize=True)
print('Training set:', score4)
print('Misclassification Rate:', 1-score4)
```

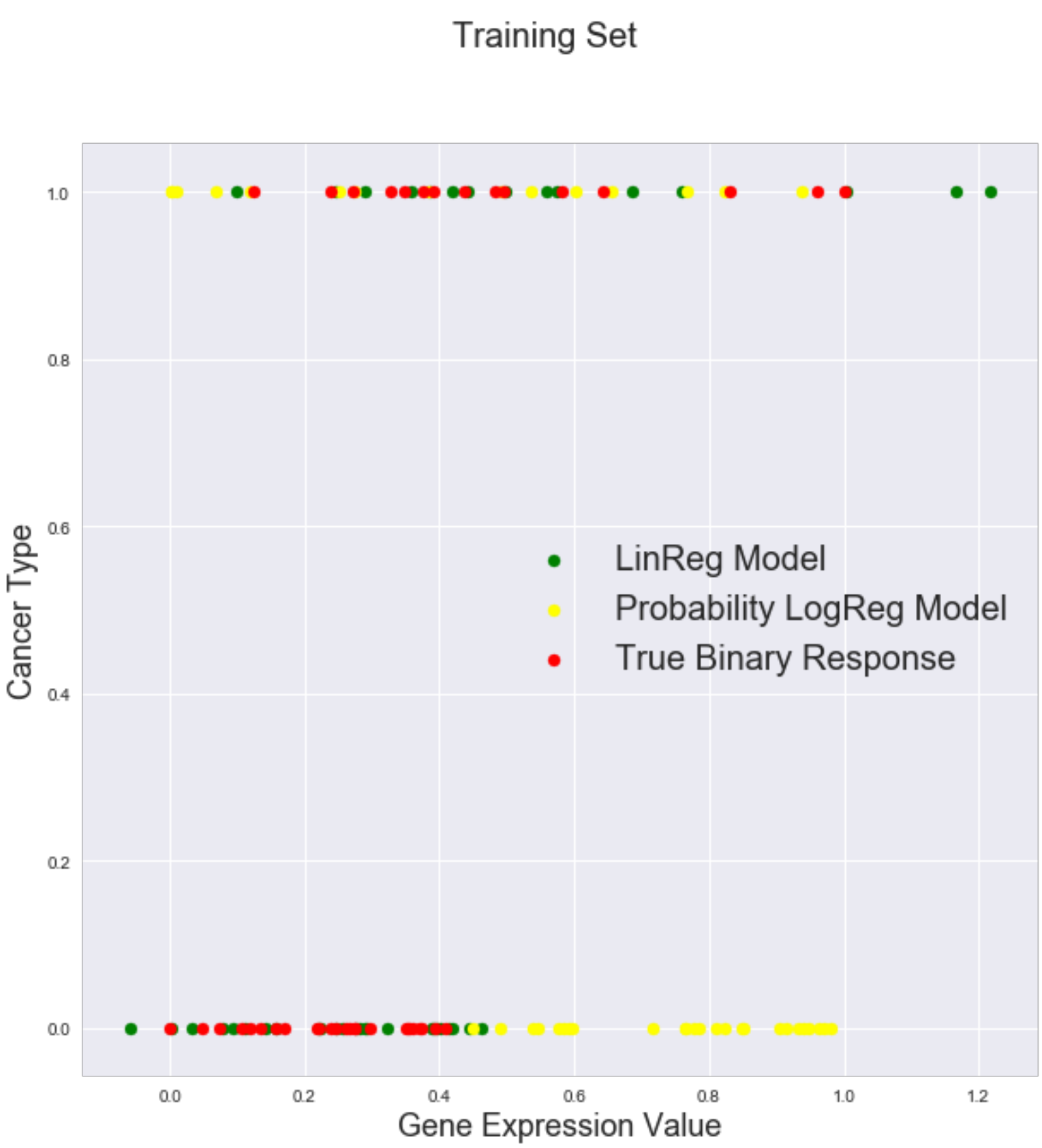
```
Classification Accuracy from LogReg
---------
Training set: 0.8
Misclassification Rate: 0.19999999999999996
---------
Training set: 0.7575757575757576
Misclassification Rate: 0.24242424242424243
```

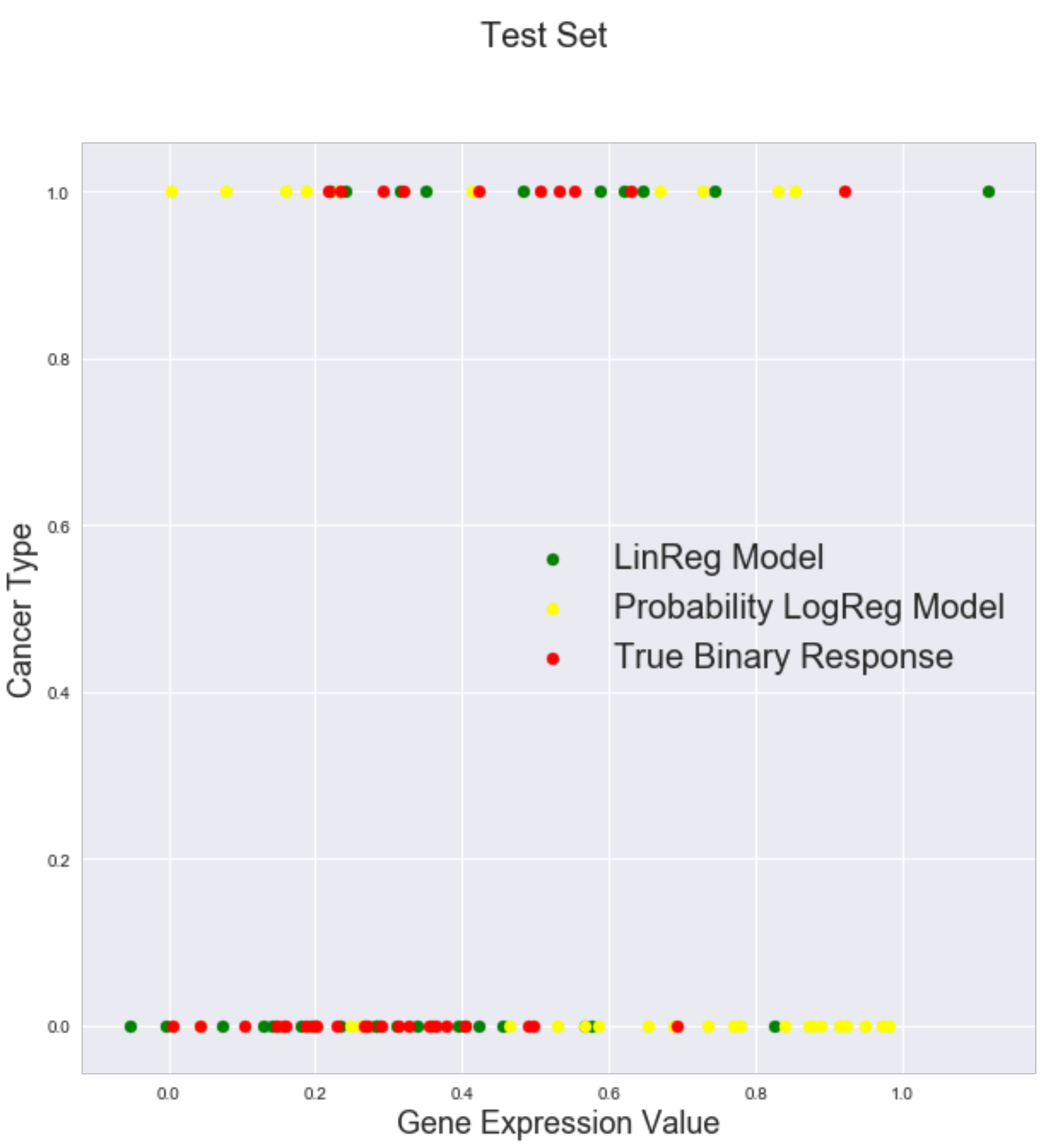
Comparing the classification accuracies from 2.2 (linereg) to 2.3 (logreg), there is no difference. Using two different methods, we were able to get the same classification and misclassification rates for both the training and test set.

**2.4:** Plot the quantitative output from the 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 [212]:

```
label_text = ['LinReg Model', 'Probability LogReg Model', 'True Binary Response'
]

fig, ax1 = plt.subplots(figsize=(10,10))
# quantitive output from linreg model
plt.scatter(OLSModel.predict(train_x), y_train, color='green', label=label_text[
0])
# probability output from logreg model
plt.scatter(logit.predict_proba(train_x)[:,0], y_train, color='yellow', label=la
bel_text[1])
#true binary response from training set points
plt.scatter(train_x, y_train, color='red' , label=label_text[2])
fig.suptitle('Training Set', fontsize=20);
plt.xlabel('Gene Expression Value', fontsize=18)
plt.ylabel('Cancer Type', fontsize=18)
plt.legend(fontsize=20)
plt.show()

fig, ax1 = plt.subplots(figsize=(10,10))
plt.scatter(OLSModel.predict(x_test), y_test, color='green', label=label_text[0]
)
plt.scatter(logit.predict_proba(x_test)[:,0], y_test, color='yellow' , label=lab
el_text[1])
plt.scatter(x_test, y_test, color='red', label=label_text[2])
fig.suptitle('Test Set', fontsize=20);
plt.xlabel('Gene Expression Value', fontsize=18)
plt.ylabel('Cancer Type', fontsize=18)
plt.legend(fontsize=20)
plt.show()
```


Training Set
Cancer Type
1.0
0.8
0.6
0.4
0.2
0.0
LinReg Model
Probability LogReg Model
True Binary Response
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Gene Expression Value




In 2.3, we calcualted the scores and saw that both logreg and linreg produced identical classification rates. However, as said in 2.1, data from linear regression does not allow us to make binary predictions, whereas logistical regression allows us to do exactly that; therefore (conceptually) logreg is the model better suited for binary classification.

However, if we look at the graphs above, it looks like the linear regression model is better suited for binary classification because the green (linreg) and red (true binary response) points tend to overlap with each other, whereas the yellow (logreg) doesn't. The same thing can be said for the training set graph where the green (linreg) and red (true binary response) points tend to overlap also.

So if we need at the graphs, linear regression model is a good model for binary classification, but conceptually, logreg is better suited because it outputs a probability.

# Question 3: 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 question 2 with a single gene (on both the training and test sets)?
2. Use the `visualize_prob` function provided below to visualize the probabilties predicted by the fitted multiple logistic regression model on both the training and test data sets. The function creates a visualization that places the data points on a vertical line based on the predicted probabilities, with the different cancer classes shown in different colors, and with the 0.5 threshold highlighted using a dotted horizontal line. Is there a difference in the spread of probabilities in the training and test plots? Are there data points for which the predicted probability is close to 0.5? If so, what can you say about these points?

In [225]:

```
#--------  visualize_prob
# A function to visualize the probabilities predicted by a Logistic Regression m
odel
# 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
    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 Ty
pe 1')
    neg_handle = ax.plot(np.zeros((nneg,1)), yneg[:,1], 'ro', label = 'Cancer Ty
pe 0')

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

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

**Answers**

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

In [226]:

```
logit1 = LogisticRegression(C=1000000)
logit1.fit(x_train, y_train)

y_test = data_test_normalized['Cancer_type']
test_x = data_test_normalized.drop(['Cancer_type'], axis = 1)

print('Classification Accuracy from Multiple LogReg')
print('---------')
score5 = accuracy_score(y_train, logit1.predict(x_train).round(), normalize=True
)
print('Training set:', score5)
print('Misclassification Rate:', 1-score5)
print('---------')
score6 = accuracy_score(y_test, logit1.predict(test_x).round(), normalize=True)
print('Test set:', score6)
print('Misclassification Rate:', 1-score6)
```

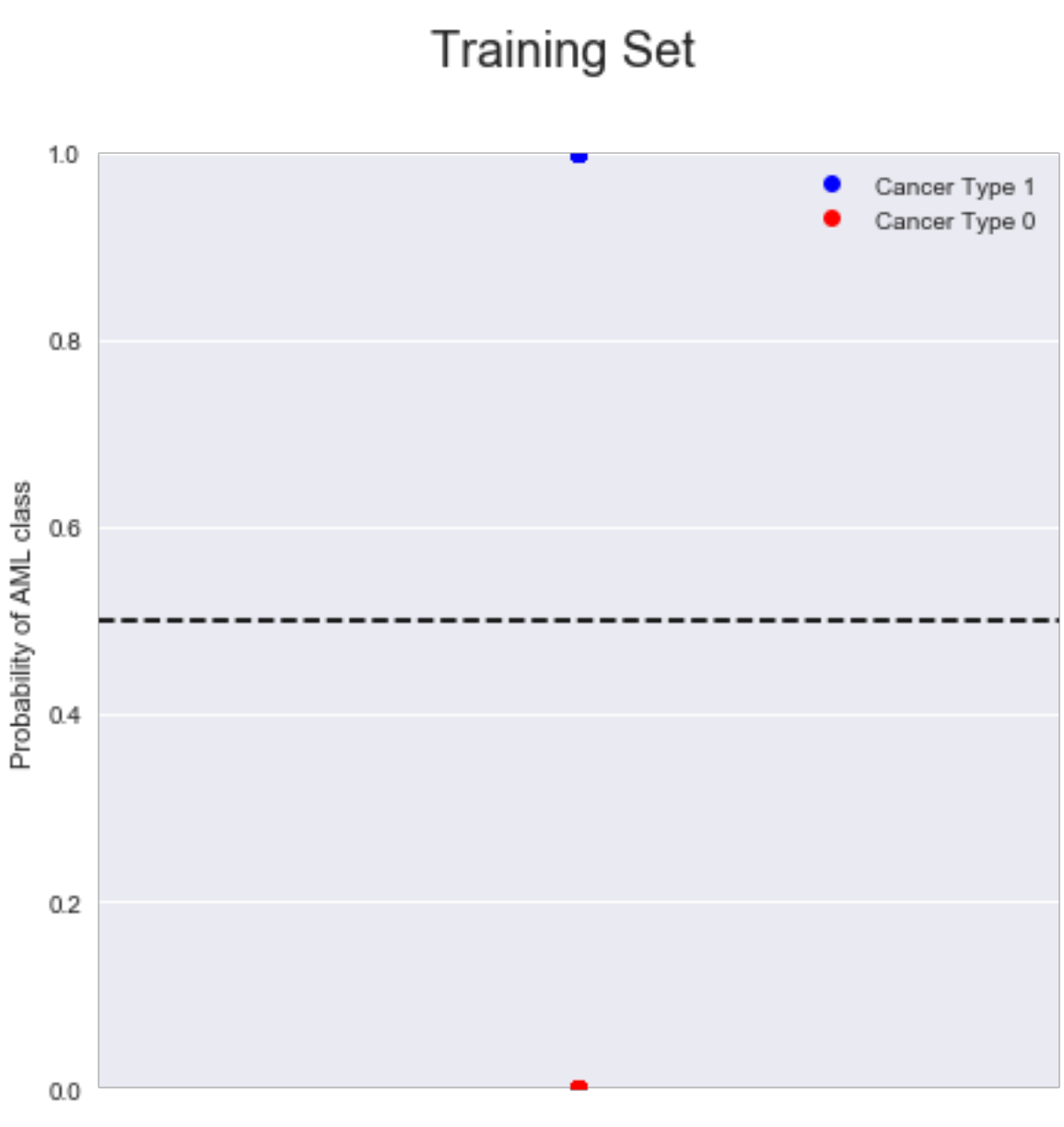
```
Classification Accuracy from Multiple LogReg
---------
Training set: 1.0
Misclassification Rate: 0.0
---------
Test set: 1.0
Misclassification Rate: 0.0
```

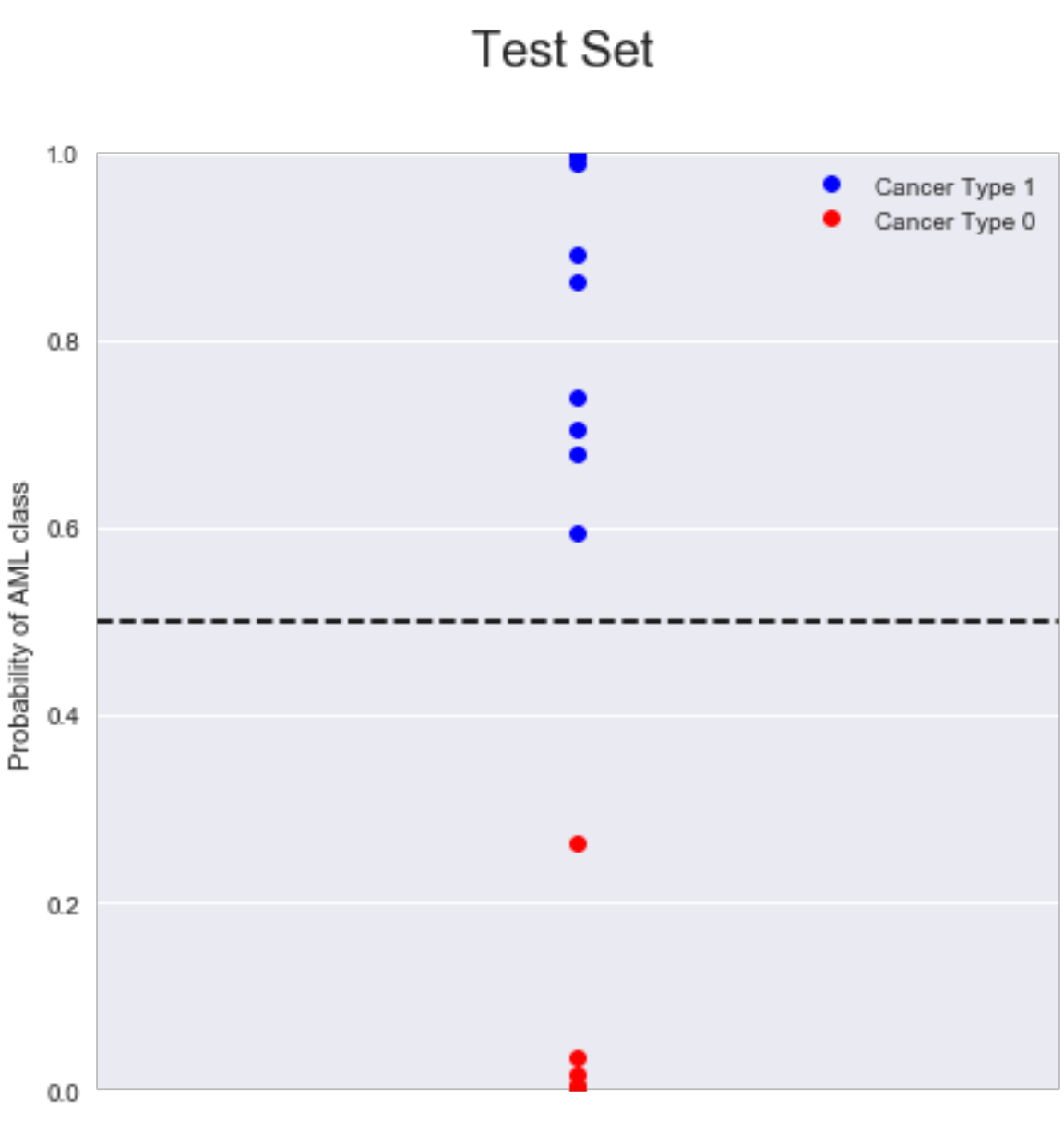
The multiple logreg classification accuracy is 1.0, so it can sucesffully predict it every single time. This is a huge improvement from the LogReg in Q2, where we had an accuracy of .8(training) and .75 (test). This happened beacuse the data set we're using for the multiple logreg has hundreds of predictors/genes, comapred to just one gene/predictor in Q2.

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

In [227]:

```
fig, ax = plt.subplots(figsize=(7,7))
visualize_prob(logit1, x_train, y_train, ax)
fig.suptitle('Training Set', fontsize=20);


fig, ax = plt.subplots(figsize=(7,7))
visualize_prob(logit1, test_x, y_test, ax)
fig.suptitle('Test Set', fontsize=20);
```


Training Set
Probability of AML class
1.0
0.8
0.6
0.4
0.2
0.0
Cancer Type 1
Cancer Type 0




Yes, there is a difference in the spread of the training and test plots. For example, if you look at the training set, the probability of type 1 is 1.0 and type 0 is 0, so the training set is basically perfect. Now if we apply the same model to the test set, the accuracy scores are the same, but if we look at the graph, there is a spread of probabilites for both types. For type 1, there arer a few points closer to 0.6, whereas for type 0, the highest probability found was around ~0.35. There are points close to 0.5, but not completely there for the test set and type 1 (blue). This means that there is a 0.5/50% chance that the model can predict that type of cancer for those points.

## Question 4: 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 1000 boostrap samples/iterations.

**Answer:**

In [304]:

```
def make_bootstrap_sample(dataset_X, dataset_y, size = None):
```

```
    # your code here


    # by default return a bootstrap sample of the same size as the original data
set
    if not size: size = len(dataset_X)

    # if the X and y datasets aren't the same size, raise an exception
    if len(dataset_X) != len(dataset_y):
        raise Exception("Data size must match between dataset_X and dataset_y")


    sample_indices = np.random.choice(size, size=size, replace=True)

    bootstrap_dataset_X = dataset_X.iloc[sample_indices, :]
    bootstrap_dataset_y = dataset_y[sample_indices]

    # return as a tuple your bootstrap samples of dataset_X as a pandas datafram
e
    # and your bootstrap samples of dataset y as a numpy column vector

    return (bootstrap_dataset_X, bootstrap_dataset_y)


def calculate_coefficients(dataset_X, dataset_y, model):

    # your code here

    # fit the model
    model.fit(dataset_X, dataset_y)

    coefficients_dictionary = dict(zip(dataset_X.columns, model.coef_.ravel()))

    # return coefficients  in the variable  coefficients_dictioanry as a diction
ary
    # with the key being the name of the feature as a string
    # the value being the value of the coefficients
    # do not return the intercept as part of this
    return coefficients_dictionary


def get_significant_predictors(regression_coefficients, significance_level):

    # your code here


    # regression_coefficients is a list of dictionaries
    # with the key being the name of the feature as a string
    # the value being the value of the coefficients
    # each dictionary in th list should be the output of calculate_coefficients

    if (len(regression_coefficients) <= 0):
        return []
```

```
    coeff_names = np.array(list(regression_coefficients[0].keys()), dtype='objec
t')
    coeff_samples = [list(coeff_list.values()) for coeff_list in regression_coef
ficients]

    coeff_samples = np.array(coeff_samples)

    # Obtain bottom percentile values
    bottom_percentile = np.percentile(coeff_samples, q=significance_level/2, axi
s=0)

    # Obtain top percentile values
    top_percentile = np.percentile(coeff_samples, q=1-significance_level/2, axis
=0)

    # Coefficients with bottom value greater than 0 or top value less than 0 are
signficant
    significant_index = ((bottom_percentile > 0.0) | (top_percentile < 0.0))

    significant_coefficients = list(coeff_names[significant_index])

    # return the significant coefficients as a list of strings
    return significant_coefficients
```

In [308]:

```
N_bootstrap_samples = 1000
regression_coefficients = []
for i in range(N_bootstrap_samples):
    sample = make_bootstrap_sample(x_train, y_train)
    coefficient_dict = calculate_coefficients(sample[0], sample[1], LogisticRegr
ession(C=1000000))
    regression_coefficients.append(coefficient_dict)

print(len(get_significant_predictors(regression_coefficients, 0.05)))
```

```
1929
```

Around 1929 coeffs estimated by mulitple logisitic regresstion are signfiicantly different from zero.

# Question 5: High Dimensionality

One of the issues you may run into when dealing with high dimensional data is that your 2D and 3D intuition may fail breakdown. For example, distance metrics in high dimensions can have properties that may feel counterintuitive.

Consider the following: You have a hypersphere with a radius of $1$, inside of a hypercube centered at $0$, with edges of length $2$.

1. As a function of $d$, the number of dimensions, how much of the hypercube's volume is contained within the hypersphere?
2. What happens as $d$ gets very large?
3. Using the functions provided below, create a plot of how the volume ratio changes as a function of $d$.
4. What does this tell you about where the majority of the volume of the hypercube resides in higher dimensions?

*HINTS:*

- The volume of a hypercube with edges of length $2$ is $V_c(d) = 2^d$.
- The volume of a hyperphere with a radius of $1$ is $V_s(d) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}$, where $\Gamma$ is Euler's Gamma Function.
- $\Gamma$ is increasing for all $d \geq 1$.

In [309]:

```
def V_c(d):
    """
    Calculate the volumn of a hypercube of dimension d.
    """
    return 2**d

def V_s(d):
    """
    Calculate the volume of a hypersphere of dimension d.
    """
    return math.pi**(d/2)/gamma((d/2)+1)
```

**Answers:**

1.

$V_c(d)$/V_(d) = $(2^d)$ / $\frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}$

The expression above is shown as a function of d.

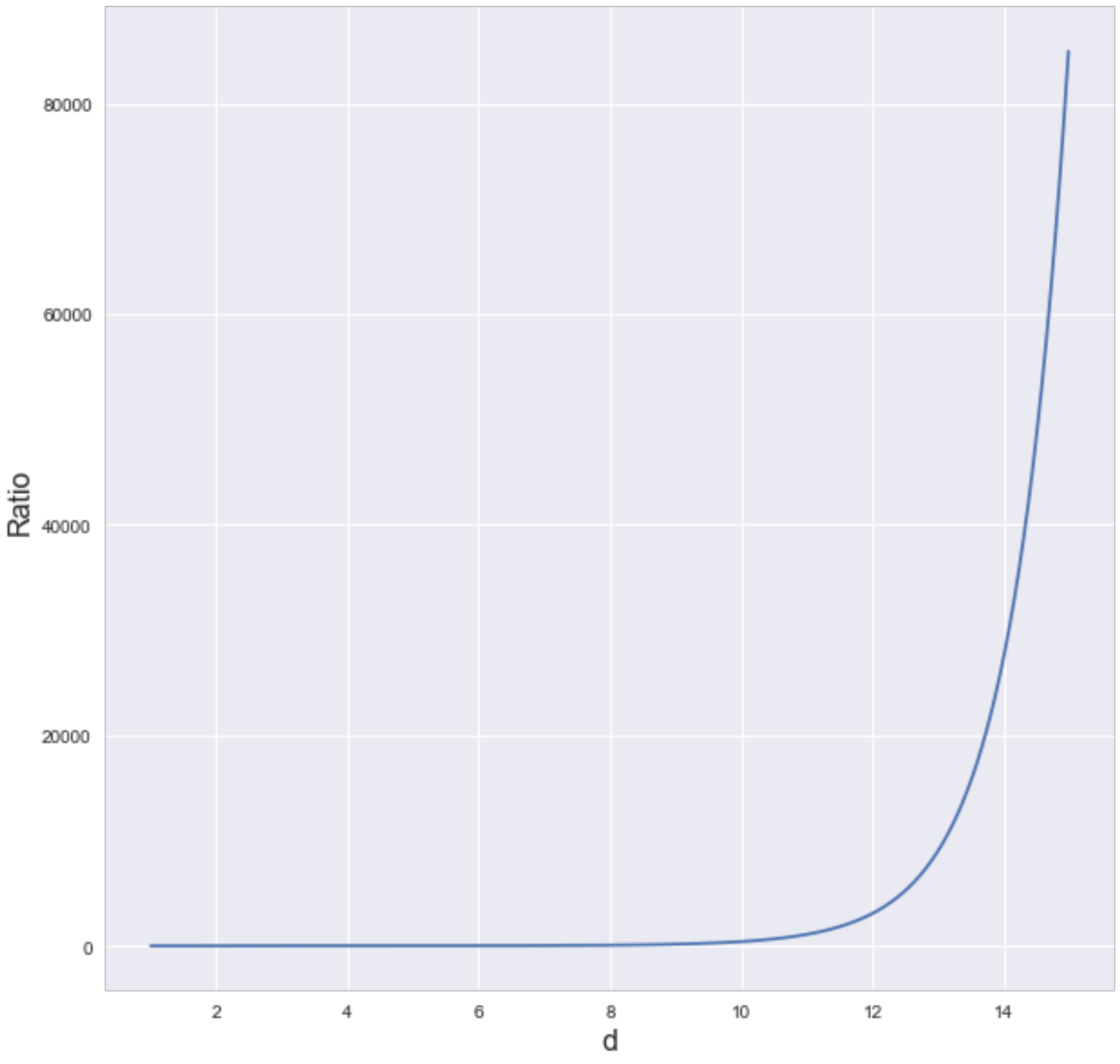
2.

As d gets very large, the ratio of the hypercube volume:hypersphere volume goes towards infinitiy.

3.

In [310]:

```
d = np.arange(1, 15, .01)

fig, ax1 = plt.subplots(figsize=(10,10))
ax1.plot(d, V_c(d)/(V_s(d)))
ax1.set_xlabel('d', fontsize=16)
ax1.set_ylabel('Ratio', fontsize=16)
fig.suptitle('Hypercube vs Hypershere Volume Ratio', fontsize=20);
```




4.

In higher dimensions, this means that the majority of the hypercube volume is completely inside the sphere because the ratio function goes towards infinity, as said in 5.2.

If radius is smaller than edges of length, the hypersphere is fully inside the hypercube. They are equal. If the radius is larger than the edges of length, it is the opposite, hypercube is completely inside the hypersphere.

resource: https://math.stackexchange.com/questions/1996000/intersection-of-hypercube-and-hypersphere (https://math.stackexchange.com/questions/1996000/intersection-of-hypercube-and-hypersphere)

# Question 6: PCA and Dimensionality Reduction

As we saw above, high dimensional problems can have counterintuitive behavior, 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.

1. Using the gene data from Problem 1, how many principal components do we need to capture at least 90% of the variance? How much of the variance do they actually capture? Fit a Logistic Regression model using these principal components. How do the classification accuracy values on both the training and tests sets compare with the models fit in question 3.1?
2. Use the code provided in question 3 to visualize the probabilities predicted by the fitted model 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:**

**6.1:** Using the gene data from Problem 1, how many principal components do we need to capture at least 90% of the variance? How much of the variance do they actually capture? Fit a Logistic Regression model using these principal components. How do the classification accuracy values on both the training and tests sets compare with the models fit in question 3.1?

In [311]:

```
pca = PCA(n_components=0.90).fit(x_train)
x_train_2 = pca.transform(x_train)

print('Principal Componenents Needed for 90% Variance:', pca.n_components_)
print('Variance percentage actually captured (%):', 100*sum(pca.explained_varian
ce_ratio_))

print('---------')
print('---------')

logit2 = LogisticRegression(C=1000000)
logit2.fit(x_train_2, y_train)

print('Classification Accuracy from LogReg after PCA')
print('---------')

score7 = accuracy_score(y_train, logit2.predict(x_train_2))
print('Training set:', score7)
print('Misclassification Rate:', 1-score7)
print('---------')

x_test_2 = pca.transform(test_x)
score8 = accuracy_score(y_test, logit2.predict(x_test_2))
print('Test set:', score8)
print('Misclassification Rate:', 1-score8)
```

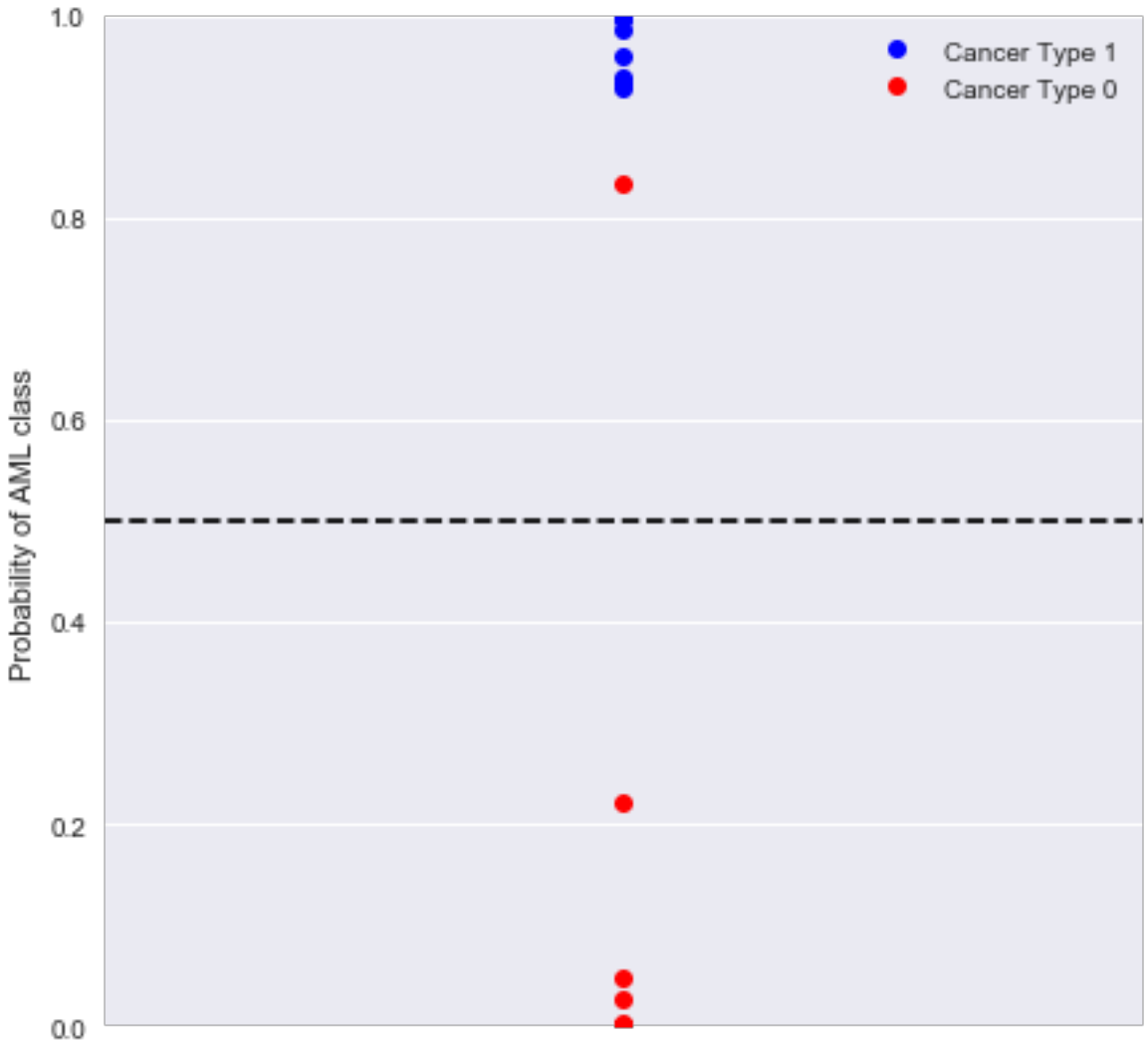
```
Principal Componenents Needed for 90% Variance: 29
Variance percentage actually captured (%): 90.26870362661793
---------
---------
Classification Accuracy from LogReg after PCA
---------
Training set: 1.0
Misclassification Rate: 0.0
---------
Test set: 0.9696969696969697
Misclassification Rate: 0.030303030303030276
```

Compared to 3.1, the classification accuracy for the training set is identical at 1.0. For the test set, they are nearly identical. 3.1 outputted 1.0, whereas the test set in 6.1 outputted 0.97, so they are nearly the same. There is a small chance for miscalculations from the model in 6.1, as shown as the miscalculation rate of 0.0303.

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

In [312]:

```
fig, ax = plt.subplots(figsize=(7,7))
visualize_prob(logit2, x_train_2d, y_train, ax)
fig.suptitle('Training Set', fontsize=20);

fig, ax = plt.subplots(figsize=(7,7))
visualize_prob(logit2, x_test_2d, y_test, ax)
fig.suptitle('Test Set', fontsize=20);
```




Compared to the model in 3.2, the training set graphs are identical. On the other hand, the test set graphs are not similar. For example in 3.2, for type 0 (red), the highest probabiilty found was around 0.32, whereas here in 6.2, there is one point above 0.8. Basically Cancer type 1 (blue) has high probability of correctly predicting, whereas type 0 (Red) doesn't. The difference in the test set's classficiation rate is why the graphs are different between 3.1 and 6.2.

One advantage that a lower dimentional representation provides is that it allows the computer to compute the data quicker(and use less space) because there are less dimensions (predictors) to crunch the data on. This reduces the number of random variables under consideration, makes the data easier to visualize, and the removal of the multi-colinearly improves the performance of the model itself.

resource used:
https://en.wikipedia.org/wiki/Dimensionality_reduction#Advantages_of_dimensionality_reduction (https://en.wikipedia.org/wiki/Dimensionality_reduction#Advantages_of_dimensionality_reduction)

# Multiclass Thyroid Classification

In this problem, you will build a model for diagnosing disorders in a patient's thyroid gland. Given the results of medical tests on a patient, the task is to classify the patient either as:

- *normal* (class 1)
- having *hyperthyroidism* (class 2)
- or having *hypothyroidism* (class 3).

The data set is provided in the file `dataset_hw5_2.csv`. Columns 1-2 contain biomarkers for a patient (predictors):

- Biomarker 1: (Logarithm of) level of basal thyroid-stimulating hormone (TSH) as measured by radioimmuno assay
- Biomarker 2: (Logarithm of) maximal absolute difference of TSH value after injection of 200 micro grams of thyrotropin-releasing hormone as compared to the basal value.

The last column contains the diagnosis for the patient from a medical expert. This data set was obtained from the UCI Machine Learning Repository.

Notice that unlike previous exercises, the task at hand is a 3-class classification problem. We will explore the use of different methods for multiclass classification.

First task: split the data using the code provided below.

# Question 7: Fit Classification Models

1. Generate a 2D scatter plot of the training set, denoting each class with a different color. Does it appear that the data points can be separated well by a linear classifier?
2. Briefly explain the difference between multinomial logistic regression and one-vs-rest (OvR) logistic regression methods for fitting a multiclass classifier (in 2-3 sentences).
3. Fit linear classification models on the thyroid data set using both the methods. You should use $L_2$ regularization in both cases, tuning the regularization parameter using cross-validation. Is there a difference in the overall classification accuracy of the two methods on the test set?
4. Also, compare the training and test accuracies of these models with the following classification methods:

    - Multiclass Logistic Regression with quadratic terms
    - Linear Discriminant Analysis
    - Quadratic Discriminant Analysis
    - k-Nearest Neighbors
      *Note:* you may use either the OvR or multinomial variant for the multiclass logistic regression (with $L_2$ regularization). Do not forget to use cross-validation to choose the regularization parameter, and also the number of neighbors in k-NN.
5. Does the inclusion of the polynomial terms in logistic regression yield better test accuracy compared to the model with only linear terms?

*Hint:* You may use the `KNeighborsClassifier` class to fit a k-NN classification model.

**Answers:**

**7.0:** First task: split the data using the code provided below.

In [125]:

```
np.random.seed(9001)
df = pd.read_csv('data/dataset_hw5_2.csv')
msk = np.random.rand(len(df)) < 0.5
data_train = df[msk]
data_test = df[~msk]

data_train.head()
```

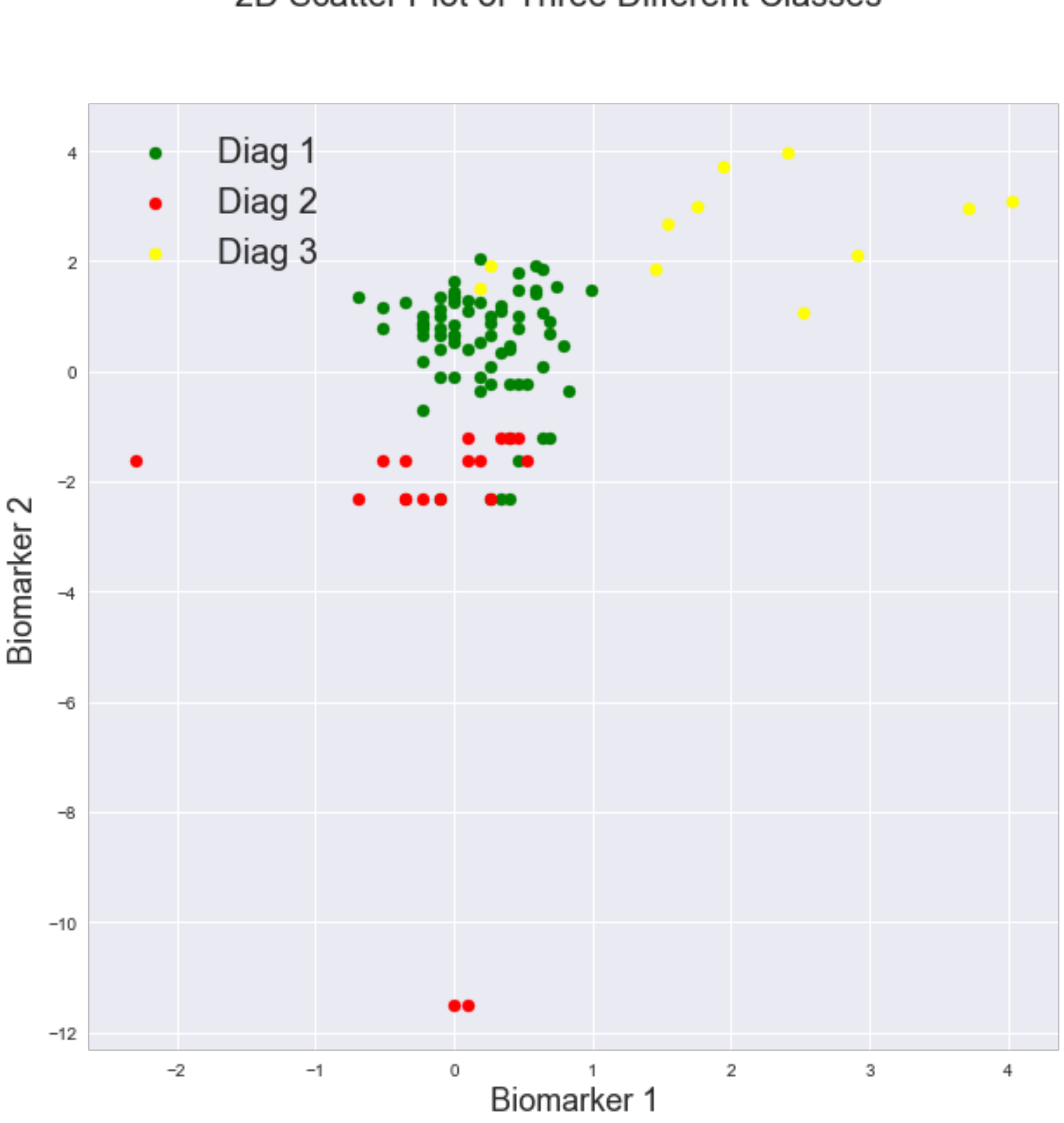
Out[125]:

| | Biomarker 1 | Biomarker 2 | Diagnosis |
|---|---|---|---|
| **0** | 0.262372 | 0.875473 | 1.0 |
| **5** | 0.336479 | 1.098616 | 1.0 |
| **9** | 0.182330 | -1.609488 | 2.0 |
| **12** | -0.223131 | 0.788462 | 1.0 |
| **13** | 0.587792 | 1.458617 | 1.0 |

**7.1:** Generate a 2D scatter plot of the training set, denoting each class with a different color. Does it appear that the data points can be separated well by a linear classifier?

In [313]:

```
# Richard said no need to normalize data due to log scale

class1 = data_train.loc[(data_train['Diagnosis'] == 1.0)]
class2 = data_train.loc[(data_train['Diagnosis'] == 2.0)]
class3 = data_train.loc[(data_train['Diagnosis'] == 3.0)]

label_text = ['Diag 1', 'Diag 2', 'Diag 3']

fig, ax1 = plt.subplots(figsize=(10,10))
plt.scatter(class1['Biomarker 1'], class1['Biomarker 2'], color='green', label=l
abel_text[0])
plt.scatter(class2['Biomarker 1'], class2['Biomarker 2'], color='red' , label=la
bel_text[1])
plt.scatter(class3['Biomarker 1'], class3['Biomarker 2'], color='yellow', label=
label_text[2])
fig.suptitle('2D Scatter Plot of Three Different Classes', fontsize=20);
plt.xlabel('Biomarker 1', fontsize=18)
plt.ylabel('Biomarker 2', fontsize=18)
plt.legend(fontsize=20)
plt.show()
```




Yes, there is very little overlap between the two three different diag/classes, therefore there is a possibility that the data points can be seperated well by a linear classifier.

**7.2:** Briefly explain the difference between multinomial logistic regression and one-vs-rest (OvR) logistic regression methods for fitting a multiclass classifier (in 2-3 sentences).

One major difference between the multinomial logreg and ovr is that with the multinomimal regression, all the predicted probabiliites add up to 1 total, whereas with ovr, they do not add up to one. Instead, with ovr, the probabilities are estimated in seperate models over and over again.

resources: https://stats.stackexchange.com/questions/60087/several-logistic-regressions-vs-multinomial-regression (https://stats.stackexchange.com/questions/60087/several-logistic-regressions-vs-multinomial-regression)

**7.3:** Fit linear classification models on the thyroid data set using both the methods. You should use $L_2$ RIDGE regularization in both cases, tuning the regularization parameter using cross-validation. Is there a difference in the overall classification accuracy of the two methods on the test set?

In [390]:

```
ytrain = data_train['Diagnosis']
xtrain = data_train.drop('Diagnosis', axis=1)
ytest = data_test['Diagnosis']
xtest = data_test.drop('Diagnosis', axis=1)

numbers = (0.001, .005, 1, 5, 10, 100, 500, 1000, 10000)
```

```
# ovr
logregcv = LogisticRegressionCV(Cs=numbers, solver='newton-cg', cv=5, multi_clas
s='ovr')
logreg = logregcv.fit(xtrain, ytrain)
logreg.scores_[2]

# looking at array below, after 5 folds, regularization parameter of 1 (4th from
left) had highest score
```

Out[390]:

```
array([[0.81818182, 0.81818182, 0.86363636, 0.86363636, 0.86363636,
        0.86363636, 0.86363636, 0.86363636, 0.86363636],
       [0.80952381, 0.80952381, 0.95238095, 0.95238095, 0.95238095,
        0.95238095, 0.95238095, 0.95238095, 0.95238095],
       [0.80952381, 0.85714286, 0.85714286, 0.85714286, 0.85714286,
        0.85714286, 0.85714286, 0.85714286, 0.85714286],
       [0.84210526, 0.89473684, 0.78947368, 0.78947368, 0.78947368,
        0.78947368, 0.78947368, 0.78947368, 0.78947368],
       [0.84210526, 0.84210526, 0.94736842, 0.94736842, 0.94736842,
        0.94736842, 0.94736842, 0.94736842, 0.94736842]])
```

In [391]:

```
# multinomial
logregcv2 = LogisticRegressionCV(Cs=numbers, solver='newton-cg', cv=5, multi_cla
ss='multinomial')
logreg2 = logregcv.fit(xtrain, ytrain)
logreg2.scores_[2]

# looking at array below, after 5 folds, regularization parameter of 1 (4th from
left) had highest score
```

Out[391]:

```
array([[0.81818182, 0.81818182, 0.86363636, 0.86363636, 0.86363636,
        0.86363636, 0.86363636, 0.86363636, 0.86363636],
       [0.80952381, 0.80952381, 0.95238095, 0.95238095, 0.95238095,
        0.95238095, 0.95238095, 0.95238095, 0.95238095],
       [0.80952381, 0.85714286, 0.85714286, 0.85714286, 0.85714286,
        0.85714286, 0.85714286, 0.85714286, 0.85714286],
       [0.84210526, 0.89473684, 0.78947368, 0.78947368, 0.78947368,
        0.78947368, 0.78947368, 0.78947368, 0.78947368],
       [0.84210526, 0.84210526, 0.94736842, 0.94736842, 0.94736842,
        0.94736842, 0.94736842, 0.94736842, 0.94736842]])
```

In [392]:

```
# .score automatically picks best paramter and outputs from there!

# ovr

print("OVR Model Training Set Score:", logreg.score(xtrain, ytrain))
print("OVR Model Test Set Score:", logreg.score(xtest, ytest))

# mvr

print("Multinomial Model Training Set Score:", logreg2.score(xtrain, ytrain))
print("Multinomial Model Test Set Score:", logreg2.score(xtest, ytest))
```

```
OVR Model Training Set Score: 0.8431372549019608
OVR Model Test Set Score: 0.8407079646017699
Multinomial Model Training Set Score: 0.8431372549019608
Multinomial Model Test Set Score: 0.8407079646017699
```

There is no difference between the ovr and multinomial models (using our specific data)

**7.4:** Also, compare the training and test accuracies of these models with the following classification methods:

- Multiclass Logistic Regression with quadratic terms
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- k-Nearest Neighbors

*Note:* you may use either the OvR or multinomial variant for the multiclass logistic regression (with $L_2$ regularization). Do not forget to use cross-validation to choose the regularization parameter, and also the number of neighbors in k-NN.

In [460]:

```
# Multiclass Logistic Regression with quadratic terms
# used OVR!

poly = PolynomialFeatures(degree=2)
poly1 = poly.fit(xtrain)

xtrainpoly = poly1.transform(xtrain)
xtestpoly = poly1.transform(xtest)

# function automatically picks best regulatizrion paramter from given list
numbers = (0.001, .005, 1, 5, 10, 100, 500, 1000, 10000)

multireg = LogisticRegressionCV(Cs=numbers, solver='newton-cg', cv=5, multi_clas
s='ovr')
multireg1 = logregcv.fit(xtrainpoly, ytrain)

print("Multiclass OVR Model Training Set Score:", multireg1.score(xtrainpoly, yt
rain))
print("Multiclass OVR Model Test Set Score:", multireg1.score(xtestpoly, ytest))
```

```
Multiclass OVR Model Training Set Score: 0.8823529411764706
Multiclass OVR Model Test Set Score: 0.8849557522123894
```

In [461]:

```
# LDA

fitted_lda = LinearDiscriminantAnalysis().fit(xtrain, ytrain)
print("LDA Model Training Set Score:", fitted_lda.score(xtrain, ytrain))
print("LDA Model Test Set Score:", fitted_lda.score(xtest, ytest))
```

```
LDA Model Training Set Score: 0.8725490196078431
LDA Model Test Set Score: 0.831858407079646
```

In [462]:

```
# QDA (can't do yet, don't have quadratic terms)

fitted_qda = QuadraticDiscriminantAnalysis().fit(xtrain, ytrain)
print("QDA Model Train Set Score:", fitted_qda.score(xtrain, ytrain))
print("QDA Model Test Set Score:", fitted_qda.score(xtest, ytest))
```

```
QDA Model Train Set Score: 0.8725490196078431
QDA Model Test Set Score: 0.8495575221238938
```

In [463]:

```
# k-NN
results = np.zeros((10,3))
for i,n in enumerate(range(1,11)):
    model = KNeighborsClassifier(n_neighbors = n)
    results[i,:] = cross_val_score(model, xtrain, ytrain, cv=3)

results_df = pd.DataFrame(results, index=list(range(1,11)), columns= ["CV1","CV2
","CV3"])
results_df['meanCV'] = np.mean(results, axis=1)

print(results_df)
print('k=3 seems to output highest value')
```

```
         CV1       CV2       CV3    meanCV
1   0.800000  0.852941  0.939394  0.864112
2   0.742857  0.882353  0.909091  0.844767
3   0.885714  0.941176  0.969697  0.932196
4   0.828571  0.941176  0.909091  0.892946
5   0.828571  0.882353  0.909091  0.873338
6   0.828571  0.852941  0.878788  0.853433
7   0.857143  0.852941  0.939394  0.883159
8   0.828571  0.852941  0.909091  0.863535
9   0.885714  0.852941  0.909091  0.882582
10  0.857143  0.794118  0.909091  0.853450
k=3 seems to output highest value
```

In [397]:

```
# k-NN continued

optimal_knn = KNeighborsClassifier(n_neighbors = 3).fit(xtrain, ytrain)
print("3-NN Training KNN train set score:",optimal_knn.score(xtrain, ytrain))
print("3-NN Tuned KNN test set score:",optimal_knn.score(xtest, ytest))
```

```
3-NN Training KNN train set score: 0.9313725490196079
3-NN Tuned KNN test set score: 0.8672566371681416
```

For test set, kNN (k=3) model provided the highest classification score, whereas for the test set, we saw the highest score from the multiclass OVR (with poly) terms model. It can also be said that QDA and LDA produced similar results to each other.

**7.5:** Does the inclusion of the polynomial terms in logistic regression yield better test accuracy compared to the model with only linear terms?

In [408]:

```
print('------')
print('BEFORE polynomial terms (only linear)')
print('------')
print('OVR Model Training Set Score: 0.8431372549019608')
print('OVR Model Test Set Score: 0.8407079646017699')
print('------')
print('AFTER')
print('------')
print("Multiclass OVR Model Training Set Score:", multireg1.score(xtrainpoly, yt
rain))
print("Multiclass OVR Model Test Set Score:", multireg1.score(xtestpoly, ytest))
print('------')
```

```
------
BEFORE polynomial terms (only linear)
------
OVR Model Training Set Score: 0.8431372549019608
OVR Model Test Set Score: 0.8407079646017699
------
AFTER
------
Multiclass OVR Model Training Set Score: 0.8823529411764706
Multiclass OVR Model Test Set Score: 0.8849557522123894
------
```

Yes, it does. The inclusion of polynomial terms yieled slightly higher classification values for both the training and test set. They both went from around 0.84 to 0.88.

# Question 8: Visualize Decision Boundaries

The following code will allow you to visualize the decision boundaries of a given classification model.

In [523]:

```
#--------  plot_decision_boundary
# A function that visualizes the data and the decision boundaries
# Input:
#      x (predictors)
#      y (labels)
#      model (the classifier you want to visualize)
#      title (title for plot)
#      ax (a set of axes to plot on)
#      poly_degree (highest degree of polynomial terms included in the model; No
ne by default)

def plot_decision_boundary(x, y, model, title, ax, poly_degree=None):
    # Create mesh
    # Interval of points for biomarker 1
    min0 = x[:,0].min()
    max0 = x[:,0].max()
    interval0 = np.arange(min0, max0, (max0-min0)/100)
    n0 = np.size(interval0)

    # Interval of points for biomarker 2
    min1 = x[:,1].min()
    max1 = x[:,1].max()
    interval1 = np.arange(min1, max1, (max1-min1)/100)
    n1 = np.size(interval1)

    # Create mesh grid of points
    x1, x2 = np.meshgrid(interval0, interval1)
    x1 = x1.reshape(-1,1)
    x2 = x2.reshape(-1,1)
    xx = np.concatenate((x1, x2), axis=1)

    # Predict on mesh of points
    # Check if polynomial terms need to be included
    if(poly_degree!=None):
        # Use PolynomialFeatures to generate polynomial terms
        poly = PolynomialFeatures(poly_degree,include_bias = True)
        xx_ = poly.fit_transform(xx)
        yy = model.predict(xx_)

    else:
        yy = model.predict(xx)

    yy = yy.reshape((n0, n1))

    # Plot decision surface
    x1 = x1.reshape(n0, n1)
```

```
    x2 = x2.reshape(n0, n1)

    ax.contourf(x1, x2, yy, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot scatter plot of data
    yy = y.reshape(-1,)
    ax.scatter(x[yy==1,0], x[yy==1,1], c='blue', label='Normal', cmap=plt.cm.coo
lwarm)
    ax.scatter(x[yy==2,0], x[yy==2,1], c='cyan', label='Hyper', cmap=plt.cm.cool
warm)
    ax.scatter(x[yy==3,0], x[yy==3,1], c='red', label='Hypo', cmap=plt.cm.coolwa
rm)

    # Label axis, title
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Biomarker 1')
    ax.set_ylabel('Biomarker 2')
```

**Note:** The provided code uses `sklearn`'s `PolynomialFeatures` to generate higher-order polynomial terms, with degree `poly_degree`. Also, if you have loaded the data sets into `pandas` data frames, you may use the `as_matrix` function to obtain a `numpy` array from the data frame objects.

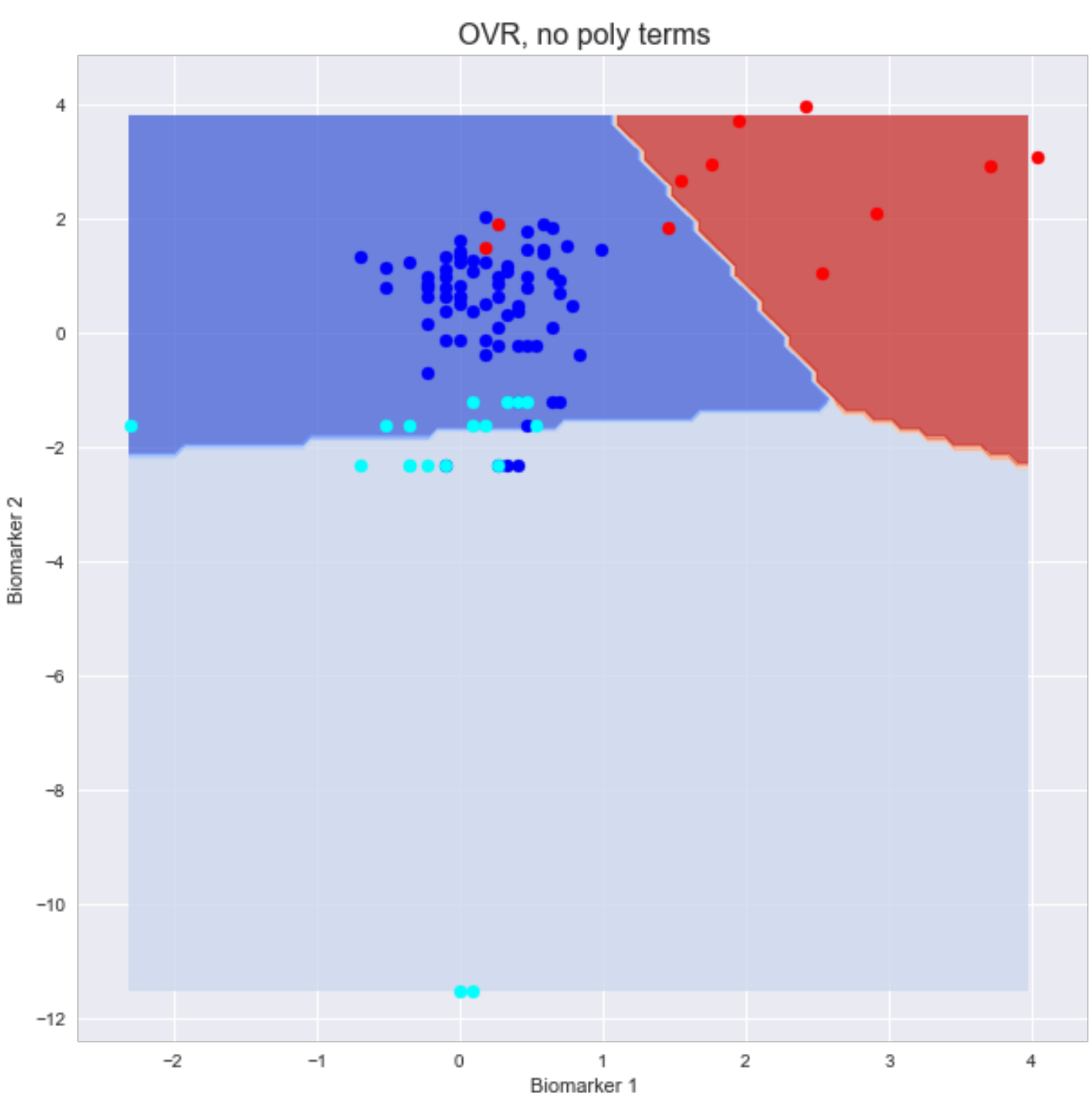
1. Use the above code to visualize the decision boundaries for each of the model fitted in the previous question.
2. Comment on the difference in the decision boundaries (if any) for the OvR and multinomial logistic regression models. Is there a difference between the decision boundaries for the linear logistic regression models and LDA. What about the decision boundaries for the quadratic logistic regression and QDA? Give an explanation for your answer.
3. QDA is a generalization of the LDA model. What's the primary difference that makes QDA more general? How does that manifest in the plots you generated?

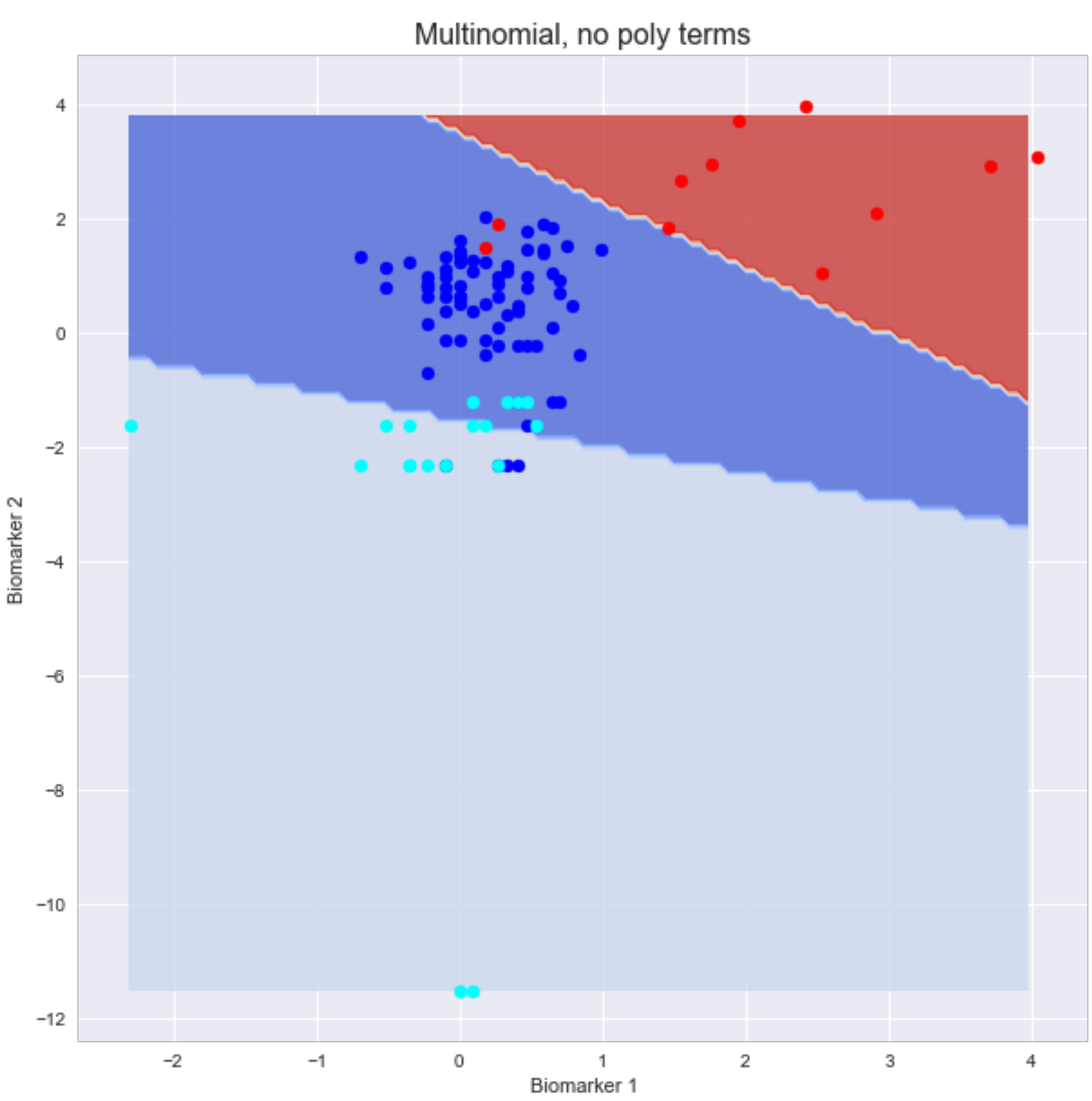
**Answers:**

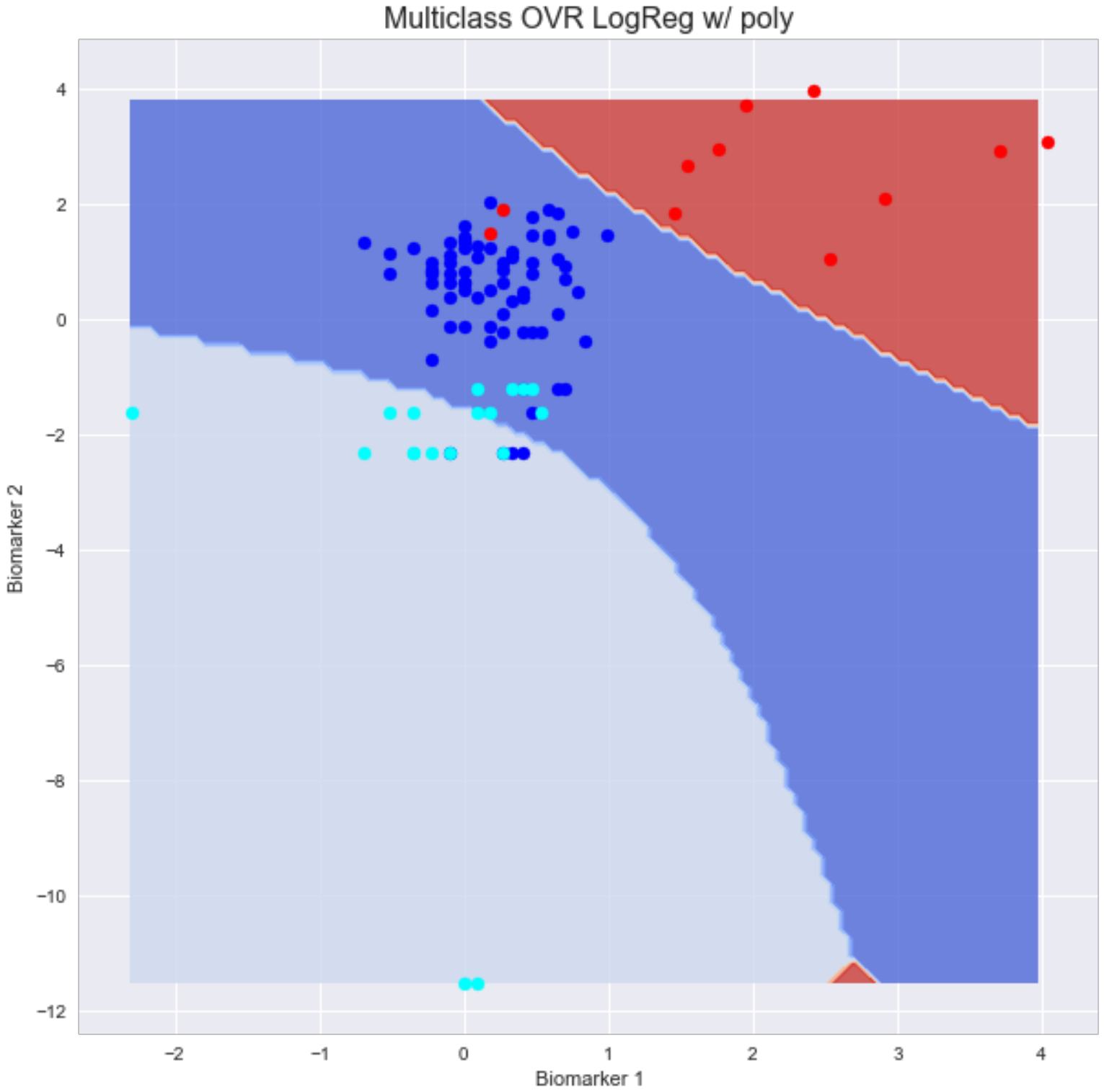
**8.1:** Use the above code to visualize the decision boundaries for each of the model fitted in the previous question.

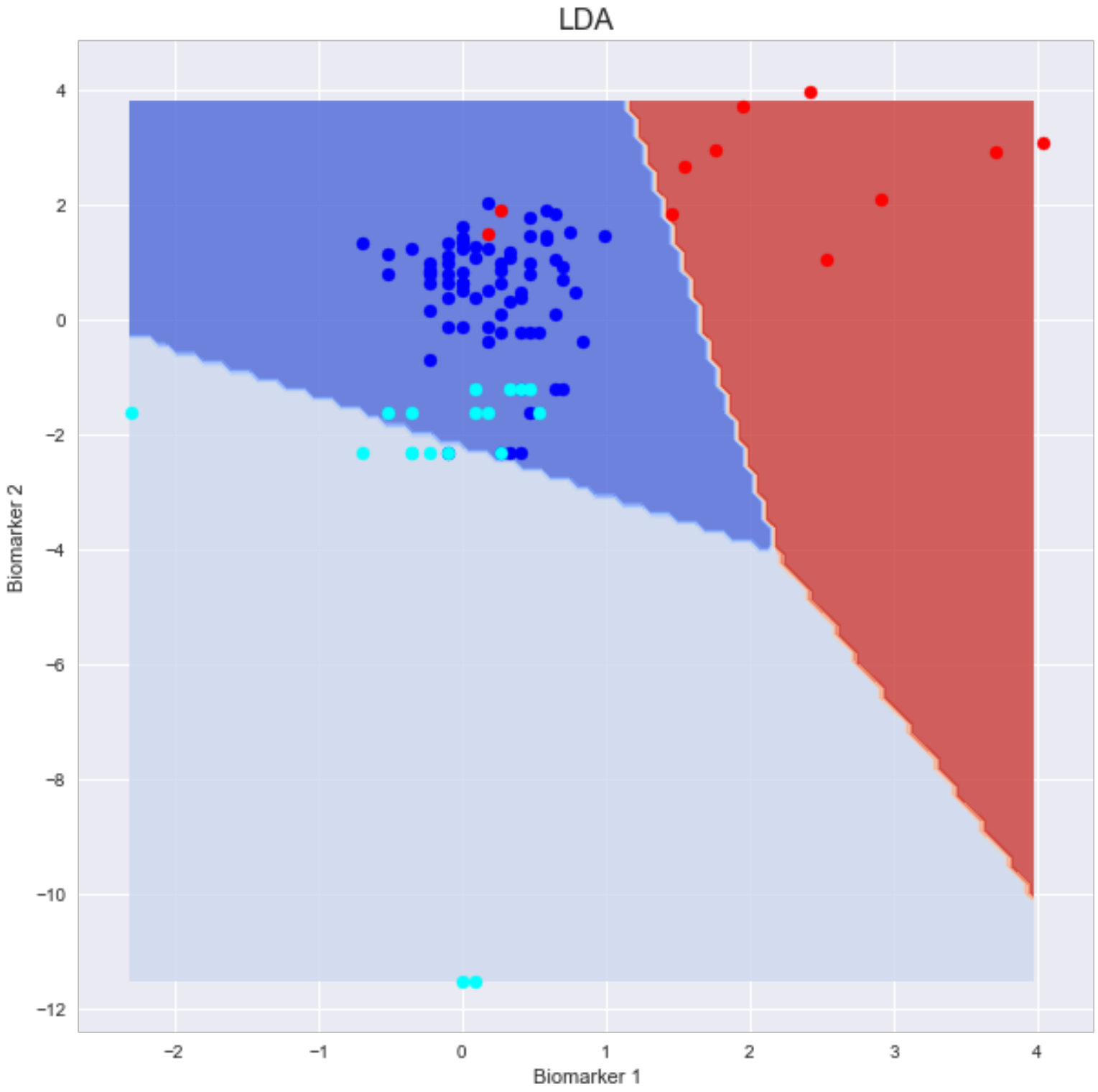
In [525]:

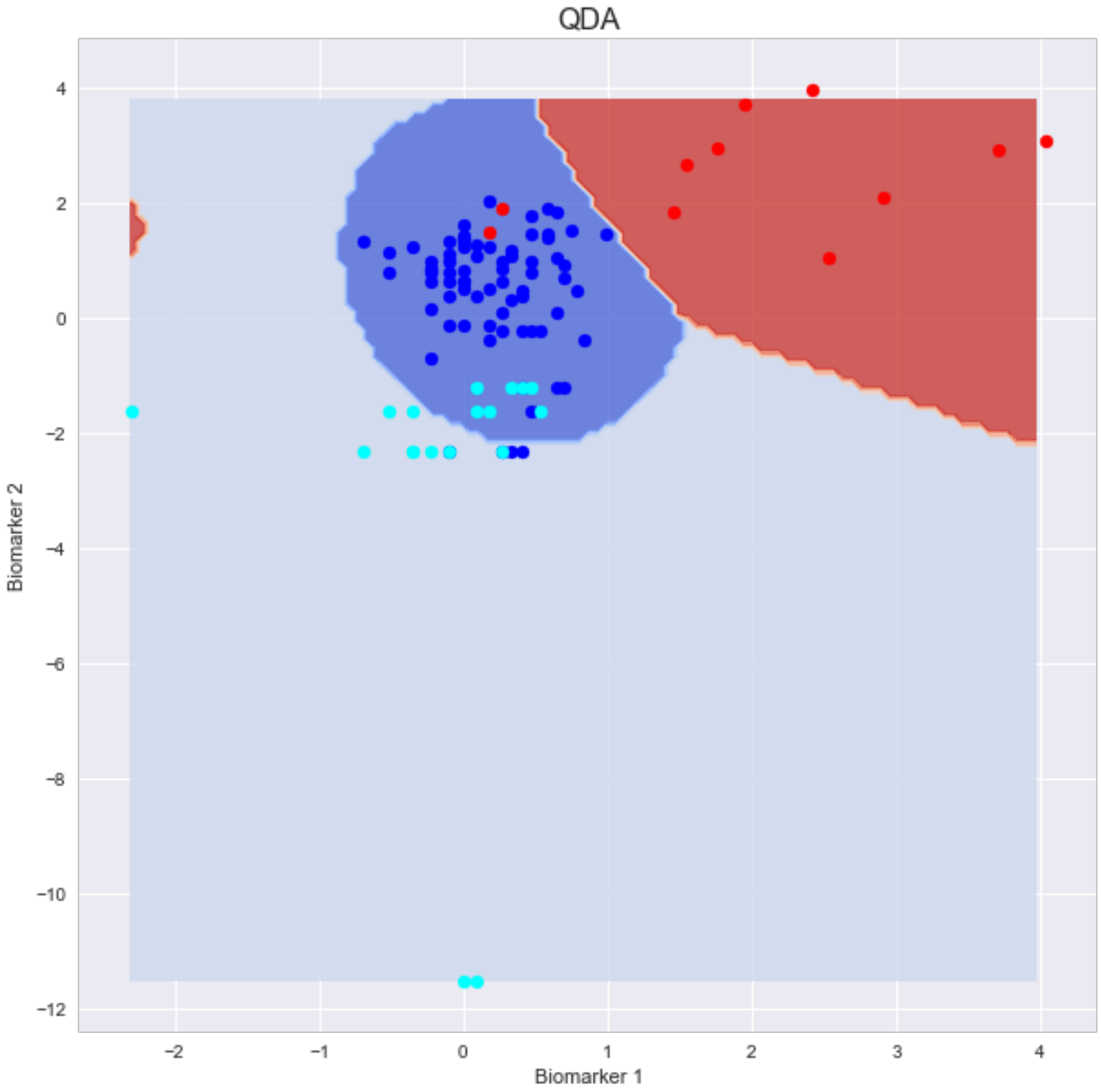
```
xtest1 = xtest.values
xtrain1 = xtrain.values
ytest1 = ytest.values
ytrain1 = ytrain.values

ovr = LogisticRegression(C=1 ,solver='newton-cg', multi_class='ovr')
ovr_fitted = ovr.fit(xtrain,ytrain)

fig, ax = plt.subplots(figsize=(10,10))
title = 'OVR, no poly terms'
plot_decision_boundary(xtrain1, ytrain1, ovr_fitted, title, ax, poly_degree=None
)

mn = LogisticRegression(C=1 ,solver='newton-cg', multi_class='multinomial')
mn_fitted = mn.fit(xtrain,ytrain)

fig, ax = plt.subplots(figsize=(10,10))
title = 'Multinomial, no poly terms'
plot_decision_boundary(xtrain1, ytrain1, mn_fitted, title, ax, poly_degree=None)

fitted = LogisticRegression(C=1, solver='newton-cg', multi_class='ovr')
fitted_multi = fitted.fit(xtrainpoly, ytrain)

fig, ax = plt.subplots(figsize=(10,10))
title = 'Multiclass OVR LogReg w/ poly'
plot_decision_boundary(xtrain1, ytrain1, fitted_multi, title, ax, poly_degree=2)

fig, ax = plt.subplots(figsize=(10,10))
title = 'LDA'
plot_decision_boundary(xtrain1, ytrain1, fitted_lda, title, ax, poly_degree=None
)

fig, ax = plt.subplots(figsize=(10,10))
title = 'QDA'
plot_decision_boundary(xtrain1, ytrain1, fitted_qda, title, ax, poly_degree=None
)

fig, ax = plt.subplots(figsize=(10,10))
title = 'k-NN, k=3'
plot_decision_boundary(xtrain1, ytrain1, optimal_knn, title, ax, poly_degree=Non
e)
```

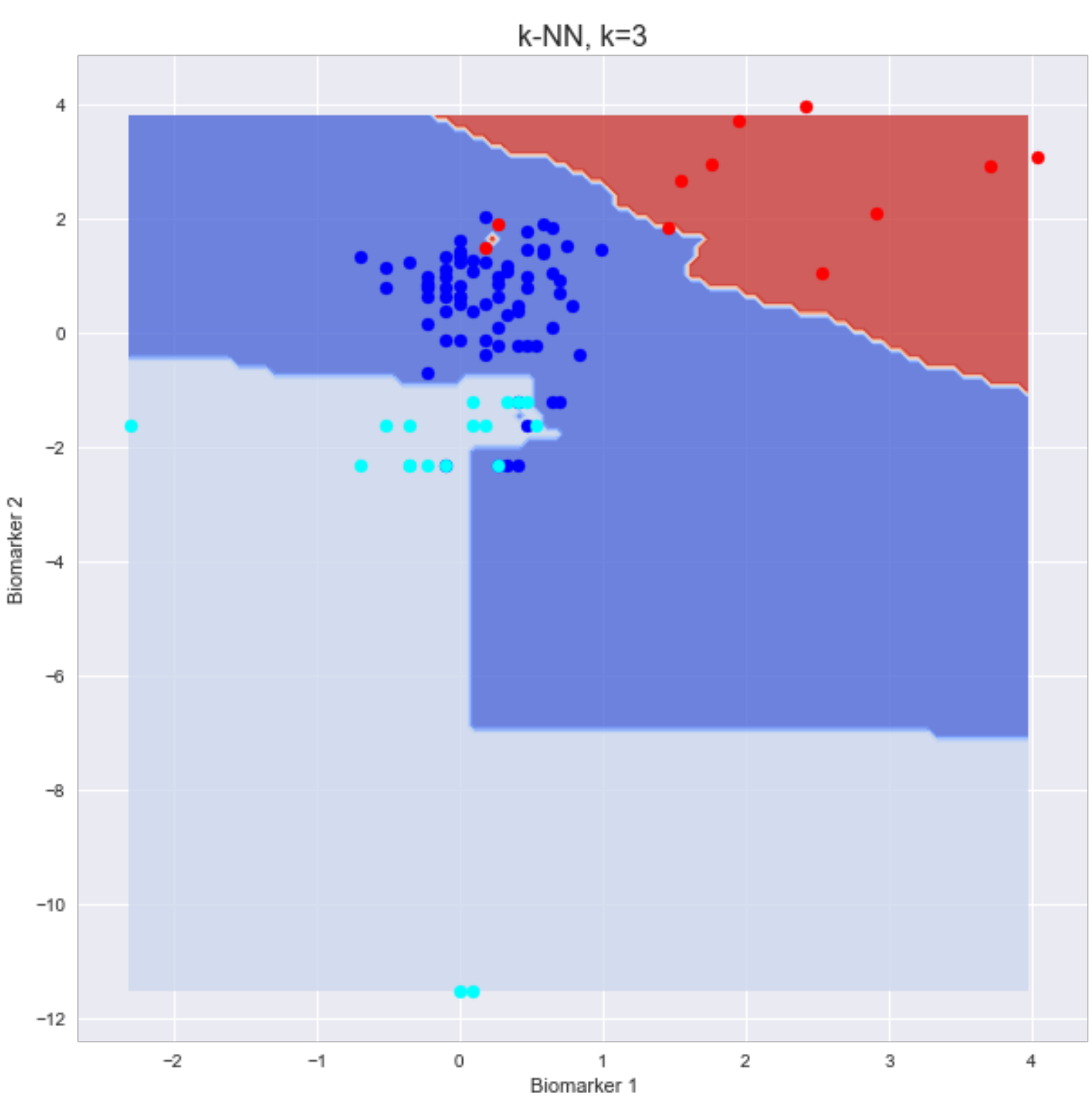

OVR, no poly terms
Biomarker 2
Biomarker 1
4
2
0
−2
−4
−6
−8
−10
−12
−2
−1
0
1
2
3
4


Multinomial, no poly terms
Biomarker 2
Biomarker 1
4
2
0
−2
−4
−6
−8
−10
−12
−2
−1
0
1
2
3
4


Multiclass OVR LogReg w/ poly
Biomarker 2
4
2
0
−2
−4
−6
−8
−10
−12
−2
−1
0
1
2
3
4
Biomarker 1


LDA
Biomarker 2
Biomarker 1
4
2
0
−2
−4
−6
−8
−10
−12
−2
−1
0
1
2
3
4


QDA
Biomarker 2
4
2
0
−2
−4
−6
−8
−10
−12
−2
−1
0
1
2
3
4
Biomarker 1




**8.2:** Comment on the difference in the decision boundaries (if any) for the OvR and multinomial logistic regression models. Is there a difference between the decision boundaries for the linear logistic regression models and LDA. What about the decision boundaries for the quadratic logistic regression and QDA? Give an explanation for your answer.

Looking at the difference between OvR and multinomial (no poly terms), it looks like the decsision boundaries for the multinomial are parallel from one another, whereas OVR produces completely different boundaries. This is odd, even though they gave us the same classification scores earlier. The reason for this difference is due to the mathematical equation that multinomial logreg comes from. All the probabilites of multinomial logreg also add up to one. Not the case for ovr. Multinomial sets one of the categores in the response variable as the reference group, and then fits seperate logreg models to predict other cases based off of it.

resource: https://stats.stackexchange.com/questions/310904/in-multinomial-logistic-regression-why-do-the-decision-boundaries-tend-to-be-pa (https://stats.stackexchange.com/questions/310904/in-multinomial-logistic-regression-why-do-the-decision-boundaries-tend-to-be-pa)

Yes, there is a difference between the linear logistic regression model boundary graph and the one for LDA. They both produce linear desicison boundaries, but the one for LDA is slightly different. This is because LDA takes a different appraoch than logreg. LDA models the distribution of the predictors X given the categories that Y takes on. Linear coeffs are estimated differently. MLE for logistic models is based on Gaussian assumptions for LDA. LDA makes more restrictive Gaussian assumptions.

resource: class pdf notes and http://people.math.umass.edu/~anna/stat697F/Chapter4_2.pdf (http://people.math.umass.edu/~anna/stat697F/Chapter4_2.pdf)

Looking at quadratic vs QDA, they do not take on linear boundaries. The quadratic graph uses OVR with poly terms of two degrees, whereas QDA does not specify. If we notice the boundaries for QDA, around the dark blue points, the blue boundary is nearly circular. This is due to the equation for QDA, based on its predictors, that it maximizes over the K classes.

resource: class pdf notes

**8.3:** QDA is a generalization of the LDA model. What's the primary difference that makes QDA more general? How does that manifest in the plots you generated?

The primary difference between QDA and LDA is that LDA assumes the covariances of the MVN (multivariate normal distribution) distributions within classes to be equal, whereas in QDA, they are allowed to be different. This allows the expression to become quadratic.

We see this in our graph from the linearity of the descision boundaries. For LDA, all the boundaries are linear; no curves, whereas for QDA, they are not linear at all. The boundaries are allowed to be quadratic and non-linear.