S-109A Introduction to Data Science:

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

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 [1]:

```
import numpy as np
import pandas as pd

import statsmodels.api as sm
from statsmodels.api import OLS

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

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 [2]:

```
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 [3]:

```
# See the data
data_train.sample(5)
```

Out[3]:

	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	AFFX-CreX-5_at	AFFX-CreX-3_at	...	U4
61	1	-112	-185	24	170	-197	-400	-215	-227	100	...	
58	0	-12	-172	12	172	-137	-205	358	-104	-25	...	
28	1	7	-100	-57	132	-377	-478	-351	-290	283	...	
65	1	-157	-370	-77	340	-438	-364	-216	-210	-86	...	
69	1	-58	-217	63	95	-191	-230	-86	-152	-6	...	

5 rows × 7130 columns

In [4]:

```
# See the data
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_a
count	40.00000	40.000000	40.000000	40.000000	40.000000	40.000000	40.00000(
mean	0.37500	-116.125000	-163.350000	-9.125000	209.075000	-250.325000	-379.92500(
std	0.49029	102.783364	95.437871	101.998539	111.000205	107.218776	123.02644(
min	0.00000	-476.000000	-531.000000	-168.000000	-24.000000	-496.000000	-696.00000(
25%	0.00000	-140.750000	-208.500000	-81.250000	124.250000	-316.500000	-461.75000(
50%	0.00000	-109.000000	-150.000000	-29.000000	228.000000	-225.000000	-384.50000(
75%	1.00000	-64.750000	-99.500000	47.000000	303.750000	-178.750000	-286.25000(
max	1.00000	86.000000	-20.000000	262.000000	431.000000	-32.000000	-122.00000(

8 rows × 7130 columns

In [5]:

```
# Here we normalize the data

for column in list(data_train.columns)[1:]:
    series = data_train[column]
    a = series.min()
    b = series.max()
    data_train.loc[:, column] = data_train[column].apply(lambda x: (x-a)/(b-a))

    # Apply the same transformation to test data
    data_test.loc[:, column] = data_test[column].apply(lambda x: (x-a)/(b-a))

    # Do the same with the entire dataset
    # It will be used in Question 1.5 for visualization purporses only
    series = df[column]
    a = series.min()
    b = series.max()
    df.loc[:, column] = df[column].apply(lambda x: (x-a)/(b-a))
```

```
/home/lucasresck/anaconda3/envs/fds/lib/python3.8/site-packages/panda
s/core/indexing.py:966: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  self.obj[item] = s
```

In [6]:

```
data_train.describe()
```

Out[6]:

	Cancer_type	AFFX-BioB-5_at	AFFX-BioB-M_at	AFFX-BioB-3_at	AFFX-BioC-5_at	AFFX-BioC-3_at	AFFX-BioDn-5_at	AFF BioD 3_
count	40.00000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.0000(
mean	0.37500	0.640347	0.719472	0.369477	0.512253	0.529472	0.550653	0.6542!
std	0.49029	0.182889	0.186767	0.237206	0.243956	0.231075	0.214332	0.2165!
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	0.00000	0.596530	0.631115	0.201744	0.325824	0.386853	0.408101	0.5471!
50%	0.00000	0.653025	0.745597	0.323256	0.553846	0.584052	0.542683	0.6839!
75%	1.00000	0.731762	0.844423	0.500000	0.720330	0.683728	0.713850	0.7533!
max	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000(

8 rows × 7130 columns

In [7]:

```
data_test.describe()
```

Out[7]:

	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	AFF BioD 3_
count	33.000000	33.000000	33.000000	33.000000	33.000000	33.000000	33.000000	33.0000(
mean	0.303030	0.641055	0.733737	0.371388	0.409524	0.512409	0.479516	0.7449:
std	0.466694	0.165245	0.189658	0.335398	0.239065	0.298177	0.309424	0.1742:
min	0.000000	0.112100	0.174168	-0.562791	-0.026374	-0.096983	-0.198606	0.4896(
25%	0.000000	0.560498	0.618395	0.255814	0.232967	0.312500	0.240418	0.6206-
50%	0.000000	0.683274	0.767123	0.374419	0.373626	0.566810	0.496516	0.7167:
75%	1.000000	0.749110	0.872798	0.504651	0.558242	0.698276	0.736934	0.8391-
max	1.000000	0.873665	1.013699	1.116279	0.923077	1.314655	0.954704	1.2434:

8 rows × 7130 columns

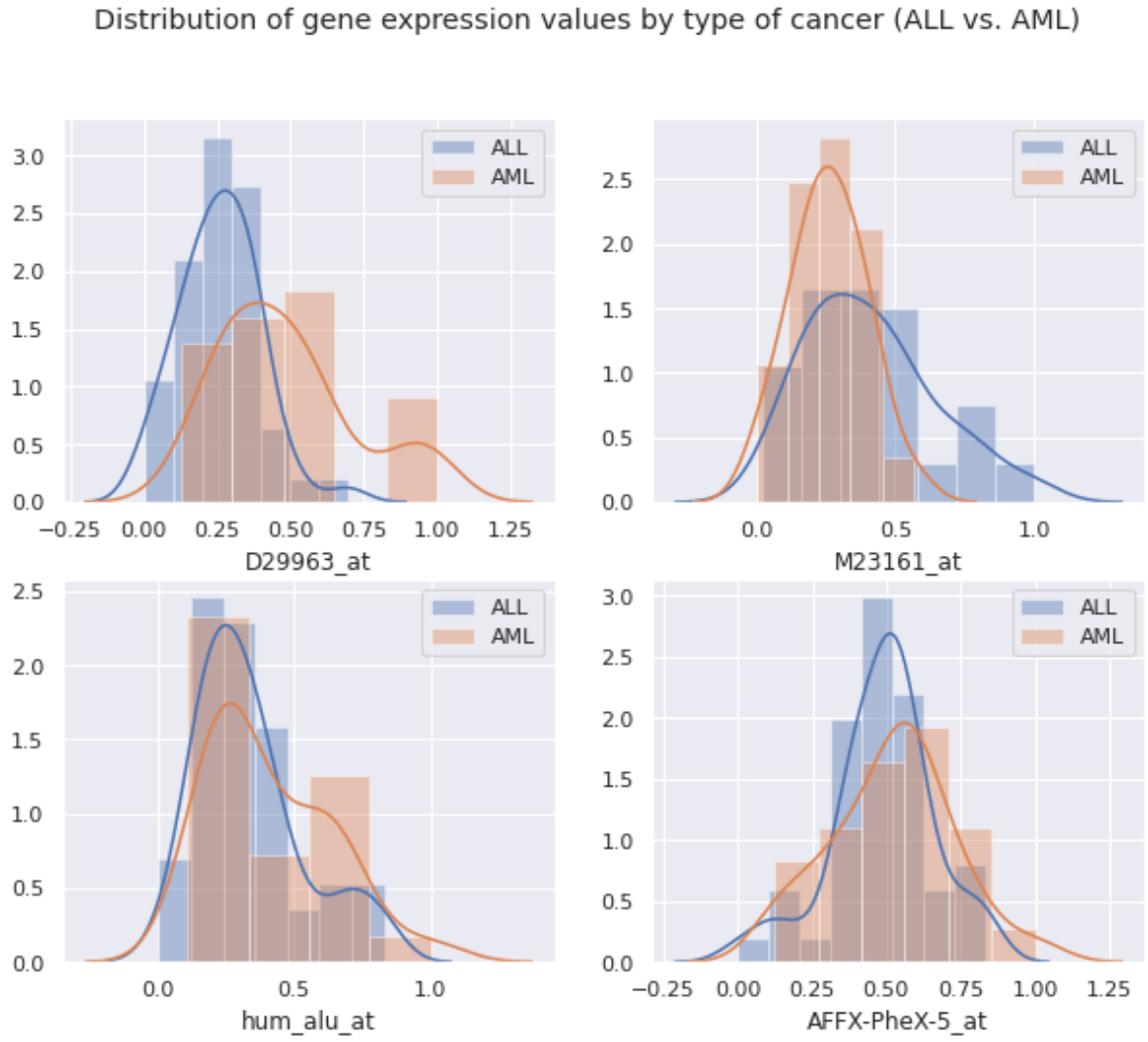
1.3: Notice that the results training set contains more predictors than observations. Do you foresee a problem in fitting a classification model to such a data set?

We can foresee that fitting a classification model will result in an overfit at the dataset. We can allso foresee multicollinearity among many features and problems inverting matrices.

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 [8]:

```
fig, axes = plt.subplots(2, 2, figsize=(10, 8))
all_df = df[df.Cancer_type == 0]
aml_df = df[df.Cancer_type == 1]
genes = ['D29963_at', 'M23161_at', 'hum_alu_at', 'AFFX-PheX-5_at']
for i, column in enumerate(genes):
    sns.distplot(all_df[column], ax = axes.reshape(4)[i], label='ALL')
    sns.distplot(aml_df[column], ax = axes.reshape(4)[i], label='AML')
    axes.reshape(4)[i].legend()
fig.suptitle(
    'Distribution of gene expression values by type of cancer (ALL vs. AML)'
)
plt.show()
```

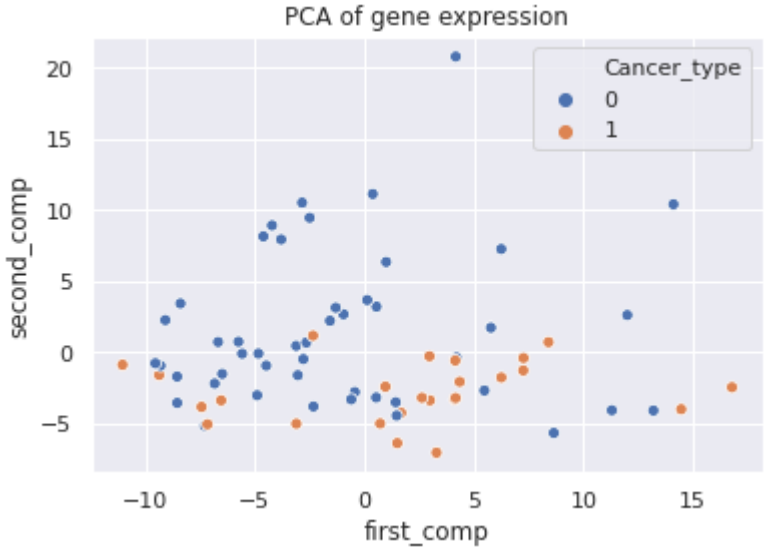


We see that the distributions of the gene expression are not the same for the two types of cancer and the four genes considerated. For example, *D29963_at* has different means, modes and variances. *M23161_at* also has a significant difference.

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 [9]:

```
pca = PCA(n_components=2).fit(df.drop(columns='Cancer_type'))
pca_X = pca.transform(df.drop(columns='Cancer_type'))
pca_df = pd.DataFrame(pca_X, columns=['first_comp', 'second_comp'])

sns.scatterplot(x=pca_df.first_comp, y=pca_df.second_comp, hue=df.Cancer_type)
plt.title('PCA of gene expression')
plt.show()
```



In [10]:

```
pca.explained_variance_ratio_.sum()
```

Out[10]:

```
0.2547289013937412
```

The PCA does a good job discriminating between the two classes of cancer, considering that we had many many features. The two first components explain around 25% of the variance of the data.

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 [11]:

```
X_train = data_train['D29963_at'].to_numpy()
y_train = data_train['Cancer_type'].to_numpy()
ols_model = OLS(y_train, sm.add_constant(X_train)).fit()
```

One problem with this intepretation is that we could have an estimation of probability greater than 1 or less than 0. Another ptoblem is that we are assuming that the data can be classified with a linear regression (it's the assumption, it could be not true).

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 [12]:

```
prediction = ols_model.predict(sm.add_constant(X_train))
missclassification = (prediction > 0.5) != y_train
print('Classification accuracy on the training set: {}'.format(
    1 - missclassification.sum()/len(y_train)
))
```

```
Classification accuracy on the training set: 0.8
```

In [13]:

```
X_test = data_test['D29963_at'].to_numpy()
y_test = data_test['Cancer_type'].to_numpy()
prediction = ols_model.predict(sm.add_constant(X_test))
missclassification = (prediction > 0.5) != y_test
print('Classification accuracy on the test set: {:.4f}'.format(
    1 - missclassification.sum()/len(y_test)
))
```

```
Classification accuracy on the test set: 0.7576
```

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 [14]:

```
lr_model = LogisticRegression(C=100000)
lr_model = lr_model.fit(X_train.reshape(-1, 1), y_train)
```

In [15]:

```
missclassification = lr_model.predict(X_train.reshape(-1, 1)) != y_train
print('Classification accuracy on the training set: {}'.format(
    1 - missclassification.sum()/len(y_train)
))
```

```
Classification accuracy on the training set: 0.8
```

In [16]:

```
missclassification = lr_model.predict(X_test.reshape(-1, 1)) != y_test
print('Classification accuracy on the test set: {:.4f}'.format(
    1 - missclassification.sum()/len(y_test)
))
```

```
Classification accuracy on the test set: 0.7576
```

Both models perform the same on the training and test data!

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 [17]:

```
ax = sns.scatterplot(
    X_train,
    ols_model.predict(sm.add_constant(X_train)),
    label='Linear regression'
)
sns.scatterplot(
    X_train,
    lr_model.predict_proba(X_train.reshape(-1, 1))[:, 1],
    ax=ax,
    label='Logistic regression'
)
sns.scatterplot(X_train, y_train, ax=ax, label='True outputs')
plt.title('Linear regression vs logistic regression for classification')
plt.xlabel('Gene expression')
plt.ylabel('Value')
plt.show()
```

Logistic regression intuitively seems to be more appropriated to this problem. We also have that problem with linear regression already mentioned: if linear regression estimations are being interpreted as probabilites, we have probabilities greater than 1 and less than 0, what doesn't make sense.

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 [18]:

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

    # 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 [19]:

```
X_train = data_train.drop(columns='Cancer_type').to_numpy()
y_train = data_train['Cancer_type'].to_numpy()
X_test = data_test.drop(columns='Cancer_type').to_numpy()
y_test = data_test['Cancer_type'].to_numpy()
```

In [20]:

```
mlr_model = LogisticRegression(C=100000).fit(X_train, y_train)
```

In [21]:

```
missclassification = mlr_model.predict(X_train) != y_train
print('Classification accuracy on the training set: {}'.format(
    1 - missclassification.sum()/len(y_train)
))
```

```
Classification accuracy on the training set: 1.0
```

In [22]:

```
missclassification = mlr_model.predict(X_test) != y_test
print('Classification accuracy on the test set: {:.4f}'.format(
    1 - missclassification.sum()/len(y_test)
))
```

```
Classification accuracy on the test set: 1.0000
```

This model has accuracy of 1 for both training and test dataset, much better than the last models.

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 [23]:

```
fig, ax = plt.subplots()
visualize_prob(mlr_model, X_train, y_train, ax)
plt.show()
```

In [24]:

```
fig, ax = plt.subplots()
visualize_prob(mlr_model, X_test, y_test, ax)
plt.show()
```

The points for the training dataset are concentrated in the extremes, maybe because of overfitting. It would explain part of the fact that they are not so concentrated in the plot for the test dataset. We do not have points close to probability 0.5.

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 [25]:

```
# From Homework 4, Question 1

# dataset_x should be a pandas dataframe

## accepts dataset inputs as numpy arrays
def make_bootstrap_sample(dataset_X, dataset_y, size = None):
    dataset_X = dataset_X.copy()
    dataset_y = dataset_y.copy()

    # by default return a bootstrap sample of the same size as the original dataset
    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")

    samples_indices = np.random.randint(0, len(dataset_X), len(dataset_X))

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

    # return as a tuple your bootstrap samples of dataset_X as a pandas dataframe
    # 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):

    features = list(dataset_X.columns)
    coef_ = list(model.coef_[0])
    coefficients_dictionary = dict(zip(features, coef_))

    # return coefficients  in the variable  coefficients_dictionary as a dictionary
    # 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):
    features = list(regression_coefficients[0].keys())
    significant_coefficients = []
    for feature in features:
        values = [coefficients[feature] for coefficients in regression_coefficients
        if np.quantile(values, significance_level/2) > 0:
            significant_coefficients.append(feature)
        else:
            if np.quantile(values, 1 - significance_level/2) < 0:
                significant_coefficients.append(feature)

    # 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 the list should be the output of calculate_coefficients

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

In [26]:

```
X_train = data_train.drop(columns='Cancer_type')
y_train = data_train.Cancer_type.to_numpy()[..., np.newaxis]

regression_coefficients = []
for i in range(1000):
    bootstrap_sample = make_bootstrap_sample(X_train, y_train)
    model = LogisticRegression(C=100000)
    model = model.fit(bootstrap_sample[0], bootstrap_sample[1].reshape(-1))
    coefficients = calculate_coefficients(
        bootstrap_sample[0],
        bootstrap_sample[1],
        model
    )
    regression_coefficients.append(coefficients)
significant_bootstrap = get_significant_predictors(
    regression_coefficients,
    0.05
)

print(
    ('{:.2f}% of the predictors are significantly ' +
    'different from zero at a significance level of 95%.').format(
        len(significant_bootstrap)/len(X_train.columns) * 100)
)
```

```
26.23% of the predictors are significantly different from zero at a si
gnificance level of 95%.
```

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 $V_s(d) = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}$, where Γ is Euler's Gamma Function.
- Γ is increasing for all $d \geq 1$.

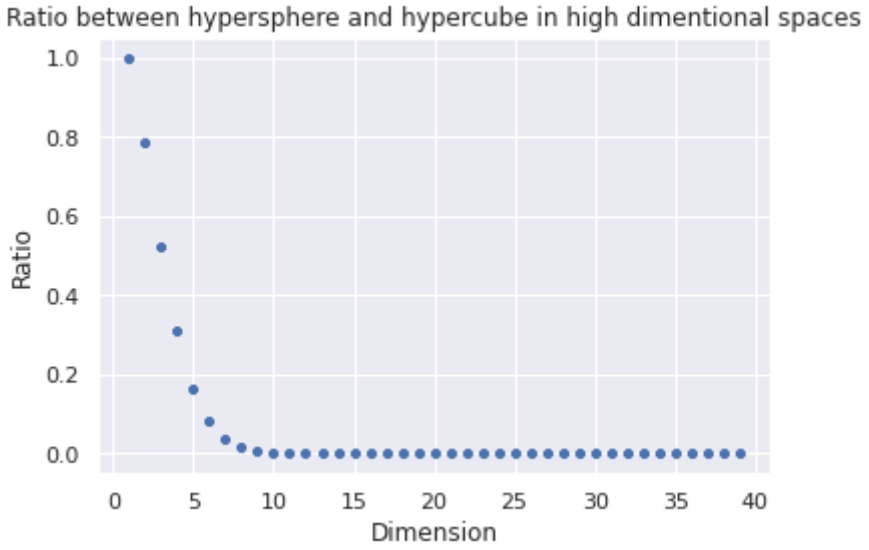
In [27]:

```
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:

In [28]:

```
x = list(range(1, 40))
y = [V_s(d)/V_c(d) for d in x]

sns.scatterplot(x, y)
plt.title('Ratio between hypersphere and hypercube in high dimentional spaces')
plt.xlabel('Dimension')
plt.ylabel('Ratio')
plt.show()
```



---

1. As a function of d (the number of dimensions), $\frac{V_s(d)}{V_c(d)} = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)2^d}$ of the hypercube is contained within the hypersphere.
2. Algebric manipulation:

$$\frac{V_s(d)}{V_c(d)} = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)2^d} = \frac{1}{\Gamma\left(\frac{d}{2}+1\right)}\left(\frac{\sqrt{\pi}}{2}\right)^d$$

When d goes to infinity, the gamma function increases, so its fraction is bounded. The other factor goes to zero, because the fraction is less than 1. So the ratio also goes to zero.

4. The plot provides evidence of the fact that the majority of the volume of the hypercube resides outside the hypersphere in higher dimensions.

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 [29]:

```
pca = PCA().fit(data_train.drop(columns='Cancer_type'))

summation = 0
number = 0
for i, variance in enumerate(pca.explained_variance_ratio_):
    summation += variance
    if summation >= 0.9:
        number = i + 1
        break
print(('We need the first {} components in order ' +
       'to explain {:.4f}% of the variance of the data.').format(
    number,
    100*sum(pca.explained_variance_ratio_[:number])
))
```

```
We need the first 29 components in order to explain 90.2687% of the va
riance of the data.
```

In [30]:

```
X_train = pca.transform(data_train.drop(columns='Cancer_type'))[:, :number]
y_train = data_train.Cancer_type.to_numpy()
X_test = pca.transform(data_test.drop(columns='Cancer_type'))[:, :number]
y_test = data_test.Cancer_type.to_numpy()

mlr_pca_model = LogisticRegression(C=10000, max_iter=1000)
mlr_pca_model = mlr_pca_model.fit(X_train, y_train)

missclassification = mlr_pca_model.predict(X_train) != y_train
print('Classification accuracy on the training set: {}'.format(
    1 - missclassification.sum()/len(y_train)
))

missclassification = mlr_pca_model.predict(X_test) != y_test
print('Classification accuracy on the test set: {:.4f}'.format(
    1 - missclassification.sum()/len(y_test)
))
```

```
Classification accuracy on the training set: 1.0
Classification accuracy on the test set: 0.9697
```

We need the first 29 components to explain 90 of the variance of the training data. The classification accuracy decreases for test data, but it's still good, compared to the those in Question 3.1.

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~~ **higher** dimensional representation yields comparable predictive power, what advantage does the lower dimensional representation provide?

In [31]:

```
fig, ax = plt.subplots()
visualize_prob(mlr_pca_model, X_train, y_train, ax)
plt.show()
```

In [32]:

```
fig, ax = plt.subplots()
visualize_prob(mlr_pca_model, X_test, y_test, ax)
plt.show()
```

The difference between these graphs and those from question 3.2 is relative to test data: the last plot is more spread than the first one. Well, it's expected, because the accuracy is less than before.

The lower dimensional representation provides less overfitting, collinearity reduction, computational time improvements and better intuitive interpretation (for 2, 3 and 4 dimensions).

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.

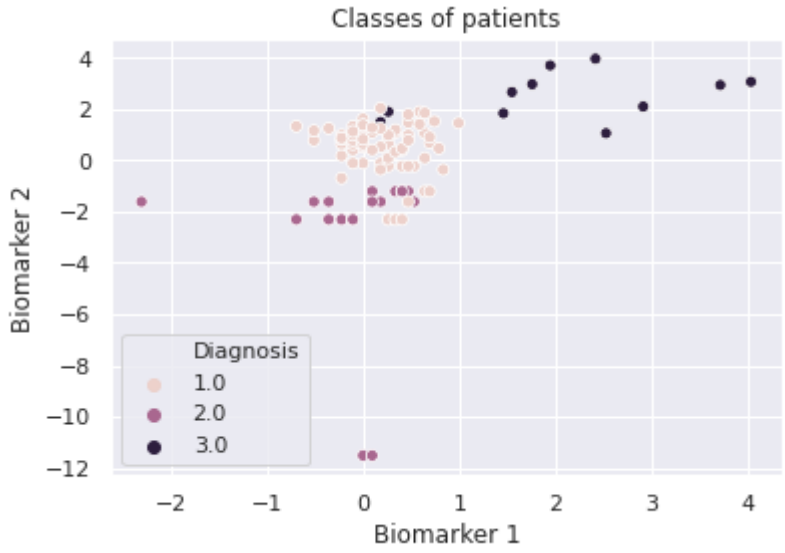
In [33]:

```
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]
```

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 [34]:

```
sns.scatterplot(
    x='Biomarker 1',
    y='Biomarker 2',
    hue='Diagnosis',
    data=data_train,
    legend='full'
)
plt.title('Classes of patients')
plt.show()
```



Data seems to be separable 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).

While multinomial logistic regression regresses all categories against one specific category (a pivot, e.g. *normal*, class 1), OvR try to regress all possibilities of pivots. Multinomial is more efficient, because of less parameters.

7.3: 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?

In [35]:

```
# See the data
data_train.describe()
```

Out[35]:

	Biomarker 1	Biomarker 2	Diagnosis
count	102.000000	102.000000	102.000000
mean	0.344213	0.136095	1.392157
std	0.830087	2.201512	0.677164
min	-2.302485	-11.512925	1.000000
25%	-0.105349	-1.076236	1.000000
50%	0.222351	0.641859	1.000000
75%	0.470010	1.273894	2.000000
max	4.032469	3.970292	3.000000

In [36]:

```
# Here we standardize the data
for column in data_train.drop(columns='Diagnosis').columns:
    mean = data_train[column].mean()
    var = data_train[column].var()
    data_train.loc[:, column] = data_train[column].apply(
        lambda x: (x - mean)/np.sqrt(var)
    )

    # Apply the same transformation to test data
    data_test.loc[:, column] = data_test[column].apply(
        lambda x: (x - mean)/np.sqrt(var)
    )
```

```
/home/lucasresck/anaconda3/envs/fds/lib/python3.8/site-packages/panda
s/core/indexing.py:1048: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  self.obj[item_labels[indexer[info_axis]]] = value
```

In [37]:

```
# See the data again
data_train.describe()
```

Out[37]:

	Biomarker 1	Biomarker 2	Diagnosis
count	1.020000e+02	1.020000e+02	102.000000
mean	1.099338e-16	-2.884403e-17	1.392157
std	1.000000e+00	1.000000e+00	0.677164
min	-3.188459e+00	-5.291373e+00	1.000000
25%	-5.415846e-01	-5.506811e-01	1.000000
50%	-1.468062e-01	2.297352e-01	1.000000
75%	1.515469e-01	5.168264e-01	2.000000
max	4.443218e+00	1.741620e+00	3.000000

In [38]:

```
X_train = data_train.drop(columns='Diagnosis').to_numpy()
y_train = data_train.Diagnosis.to_numpy()
X_test = data_test.drop(columns='Diagnosis').to_numpy()
y_test = data_test.Diagnosis.to_numpy()

ovr_lr_model = LogisticRegressionCV(
    Cs=10, cv=5, penalty='l2', multi_class='ovr'
)
ovr_lr_model = ovr_lr_model.fit(X_train, y_train)

missclassification = ovr_lr_model.predict(X_train) != y_train
print('The classification accuracy of OvR in training data is {:.4f}%.'.format(
    (1 - missclassification.sum()/len(y_train))*100
))
```

```
The classification accuracy of OvR in training data is 85.2941%.
```

In [39]:

```
missclassification = ovr_lr_model.predict(X_test) != y_test
print('The classification accuracy of OvR in test data is {:.4f}%.'.format(
    (1 - missclassification.sum()/len(y_test))*100
))
```

```
The classification accuracy of OvR in test data is 84.0708%.
```

In [40]:

```
multi_lr_model = LogisticRegressionCV(
    Cs=10, cv=5, penalty='l2', multi_class='multinomial'
)
multi_lr_model = multi_lr_model.fit(X_train, y_train)

missclassification = multi_lr_model.predict(X_train) != y_train
print('The classification accuracy of multinomial in training data is {:.4f}%.'.for
    (1 - missclassification.sum()/len(y_train))*100
))
```

```
The classification accuracy of multinomial in training data is 88.235
3%.
```

In [41]:

```
missclassification = multi_lr_model.predict(X_test) != y_test
print('The classification accuracy of multinomial in test data is {:.4f}%.'.format(
    (1 - missclassification.sum()/len(y_test))*100
))
```

```
The classification accuracy of multinomial in test data is 87.6106%.
```

Multinomial logistic regression did better in both training and test 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.

Multiclass Logistic Regression with quadratic terms

In [42]:

```
X_train_quad = PolynomialFeatures(2, include_bias=False)
X_train_quad = X_train_quad.fit_transform(X_train)
X_test_quad = PolynomialFeatures(2, include_bias=False)
X_test_quad = X_test_quad.fit_transform(X_test)
```

In [43]:

```
multi_quad_lr_model = LogisticRegressionCV(
    Cs=10, cv=5, penalty='l2', multi_class='multinomial', max_iter=150
)
multi_quad_lr_model = multi_quad_lr_model.fit(X_train_quad, y_train)

missclassification = multi_quad_lr_model.predict(X_train_quad) != y_train
print('The classification accuracy of multinomial in training data is {:.4f}%.'.for
    (1 - missclassification.sum()/len(y_train))*100
))
```

```
The classification accuracy of multinomial in training data is 89.215
7%.
```

In [44]:

```
missclassification = multi_quad_lr_model.predict(X_test_quad) != y_test
print('The classification accuracy of multinomial in test data is {:.4f}%.'.format(
    (1 - missclassification.sum()/len(y_test))*100
))
```

```
The classification accuracy of multinomial in test data is 90.2655%.
```

Linear Discriminant Analysis

In [45]:

```
kf = KFold(n_splits=5, shuffle=True, random_state=434)
shrinkages = list(np.linspace(0, 1))
mean_accuracies = []
for shrinkage in shrinkages:
    accuracies = []
    for train_index, test_index in kf.split(X_train):
        lda_model = LinearDiscriminantAnalysis(
            shrinkage=shrinkage, solver='lsqr'
        )
        lda_model = lda_model.fit(X_train, y_train)
        prediction = lda_model.predict(X_train[test_index])
        missclassification = (prediction != y_train[test_index]).sum()
        accuracy = 1 - missclassification/len(y_train[test_index])
        accuracies.append(accuracy)
    mean_accuracies.append(np.mean(accuracies))

shrinkage = shrinkages[np.argmax(mean_accuracies)]
print('Selected shrinkage {}, with mean accuracy equal to {:.4f}.'.format(
    shrinkage,
    np.max(mean_accuracies)
))
```

```
Selected shrinkage 0.0, with mean accuracy equal to 0.8724.
```

In [46]:

```
lda_model = LinearDiscriminantAnalysis(shrinkage=shrinkage, solver='lsqr')
lda_model = lda_model.fit(X_train, y_train)
missclassification = (lda_model.predict(X_train) != y_train).sum()/len(y_train)
print('The classification accuracy of LDA in training data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of LDA in training data is 87.2549%.
```

In [47]:

```
missclassification = (lda_model.predict(X_test) != y_test).sum()/len(y_test)
print('The classification accuracy of LDA in test data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of LDA in test data is 84.0708%.
```

Quadratic Discriminant Analysis

In [48]:

```
kf = KFold(n_splits=5, shuffle=True, random_state=736)
shrinkages = list(np.linspace(0, 1))
mean_accuracies = []
for shrinkage in shrinkages:
    accuracies = []
    for train_index, test_index in kf.split(X_train):
        qda_model = QuadraticDiscriminantAnalysis(reg_param=shrinkage)
        qda_model = qda_model.fit(X_train, y_train)
        prediction = qda_model.predict(X_train[test_index])
        missclassification = (prediction != y_train[test_index]).sum()
        accuracy = 1 - missclassification/len(y_train[test_index])
        accuracies.append(accuracy)
    mean_accuracies.append(np.mean(accuracies))

shrinkage = shrinkages[np.argmax(mean_accuracies)]
print('Selected shrinkage {}, with mean accuracy equal to {:.4f}.'.format(
    shrinkage,
    np.max(mean_accuracies)
))
```

```
Selected shrinkage 0.061224489795918366, with mean accuracy equal to
0.8824.
```

In [49]:

```
qda_model = QuadraticDiscriminantAnalysis(reg_param=shrinkage)
qda_model = qda_model.fit(X_train, y_train)
prediction = qda_model.predict(X_train)
missclassification = (prediction != y_train).sum()/len(y_train)
print('The classification accuracy of QDA in training data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of QDA in training data is 88.2353%.
```

In [50]:

```
missclassification = (qda_model.predict(X_test) != y_test).sum()/len(y_test)
print('The classification accuracy of QDA in test data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of QDA in test data is 85.8407%.
```

k-Nearest Neighbors

In [51]:

```
kf = KFold(n_splits=5, shuffle=True, random_state=1000)
ks = list(range(1, int(np.floor(len(y_train)*4/5) + 1)))
mean_accuracies = []
for k in ks:
    accuracies = []
    for train_index, test_index in kf.split(X_train):
        knn_model = KNeighborsClassifier(n_neighbors=k).fit(
            X_train[train_index],
            y_train[train_index]
        )
        prediction = knn_model.predict(X_train[test_index])
        missclassification = (prediction != y_train[test_index]).sum()
        accuracy = 1 - missclassification/len(y_train[test_index])
        accuracies.append(accuracy)
    mean_accuracies.append(np.mean(accuracies))

k = np.argmax(mean_accuracies) + 1
print('Selected {} neighbors, with mean accuracy equal to {:.4f}.'.format(
    k,
    mean_accuracies[k-1]
))
```

```
Selected 3 neighbors, with mean accuracy equal to 0.9029.
```

In [52]:

```
knn_model = KNeighborsClassifier(n_neighbors=k).fit(X_train, y_train)
missclassification = (knn_model.predict(X_train) != y_train).sum()/len(y_train)
print('The classification accuracy of kNN in training data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of kNN in training data is 94.1176%.
```

In [53]:

```
missclassification = (knn_model.predict(X_test) != y_test).sum()/len(y_test)
print('The classification accuracy of kNN in test data is {:.4f}%.'.format(
    (1 - missclassification)*100
))
```

```
The classification accuracy of kNN in test data is 88.4956%.
```

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

Multinomial logistic regression with the polynomial terms did slightly better in test data.

Question 8: Visualize Decision Boundaries

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

In [54]:

```
#--------  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; None

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 = False)
        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.coolwa
    ax.scatter(x[yy==2,0], x[yy==2,1], c='cyan', label='Hyper', cmap=plt.cm.coolwar
    ax.scatter(x[yy==3,0], x[yy==3,1], c='red', label='Hypo', cmap=plt.cm.coolwarm)

    # Label axis, title
    ax.set_title(title)
    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.

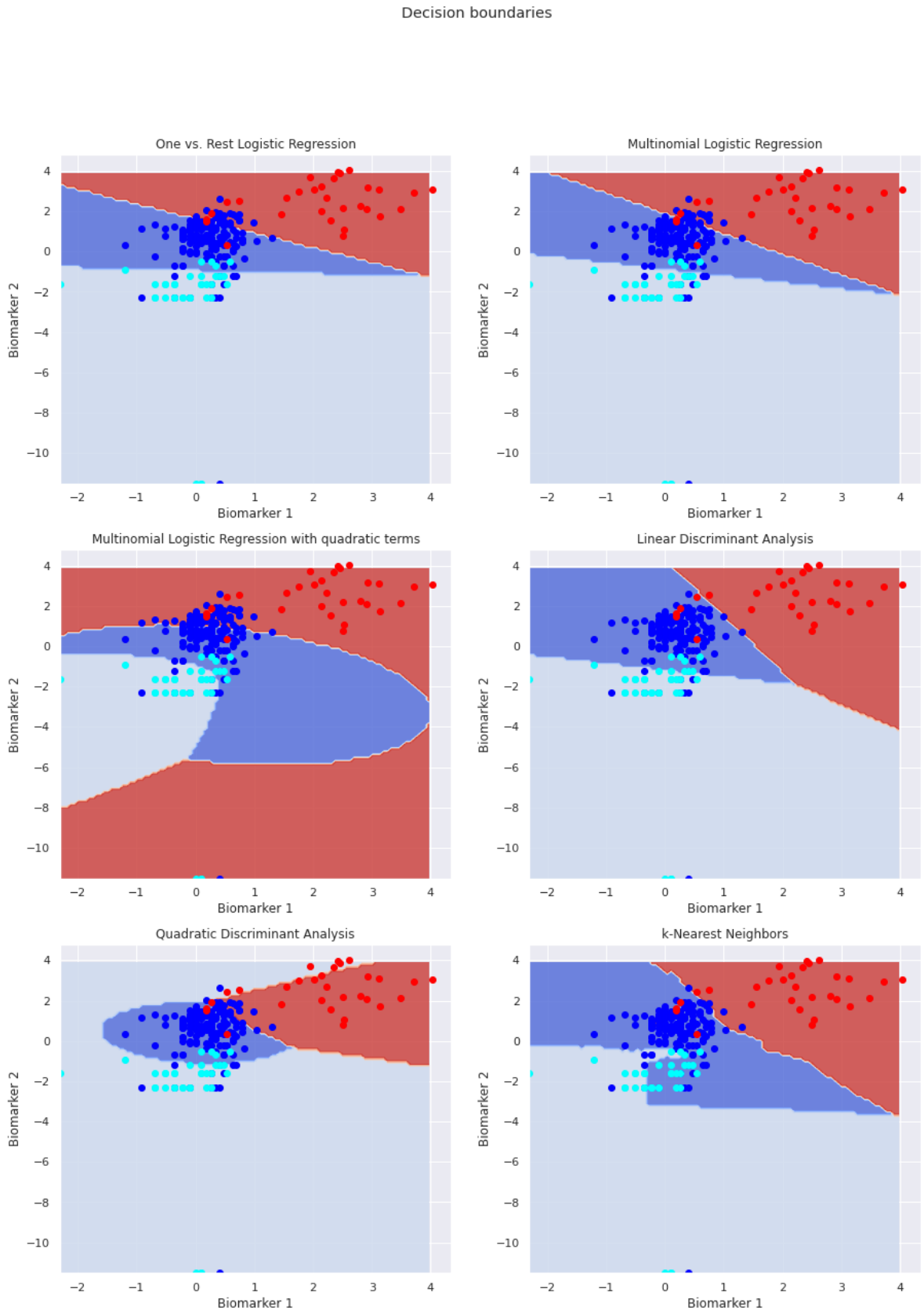
1. Use the above code to visualize the decision boundaries for each of the model fitted in the previous question.
2. Comment on the difference in the decision boundaries (if any) for the OvR and multinomial logistic regression models. Is there a difference between the decision boundaries for the linear 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 [55]:

```
models = [
    ovr_lr_model, multi_lr_model, multi_quad_lr_model,
    lda_model, qda_model, knn_model
]

fig, axes = plt.subplots(3, 2, figsize=(15, 20))
fig.suptitle('Decision boundaries')
for i, model in enumerate(models):
    X = df.drop(columns='Diagnosis').to_numpy()
    y = df.Diagnosis.to_numpy()
    title = [
        'One vs. Rest Logistic Regression',
        'Multinomial Logistic Regression',
        'Multinomial Logistic Regression with quadratic terms',
        'Linear Discriminant Analysis',
        'Quadratic Discriminant Analysis',
        'k-Nearest Neighbors'
    ][i]
    if i == 2:
        plot_decision_boundary(X, y, model, title, axes.reshape(6)[i], 2)
    else:
        plot_decision_boundary(X, y, model, title, axes.reshape(6)[i])
```



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.

- OvR and multinomial logistic regression: they are almost the same.
- Linear logistic regression models and LDA: they are almost the same too; the difference is the fact that the boundaries are more inclined.
- Quadratic logistic regression and QDA: they are very different. In the LR, we see that the gray area specializes in light blue dots, while the other areas are spread. In the QDA, we see the opposite: both blue and red specialize, and gray area is spread.

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 that makes QDA more general is the fact that LDA assumes that the covariances of the classes are the same, and QDA don't. That is, in LDA, each class is equally spread. So, in the plot of LDA, we see boundaries of better distributed areas, different from the plot of QDA.
