CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 7

LDA/QDA and Decision Trees

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

---

INSTRUCTIONS

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

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

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

CS109A

Import libraries:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegressionCV
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
#import pydotplus
#import io
from sklearn.tree import export_graphviz
from IPython.display import Image
from IPython.display import display
%matplotlib inline
```

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 `hw7_dataset.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 following code:

In [2]:
```
np.random.seed(9001)
df = pd.read_csv('hw7_dataset.csv')
msk = np.random.rand(len(df)) < 0.5
data_train = df[msk]
data_test = df[~msk]
```

Question 1: 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 training and test sets?
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?

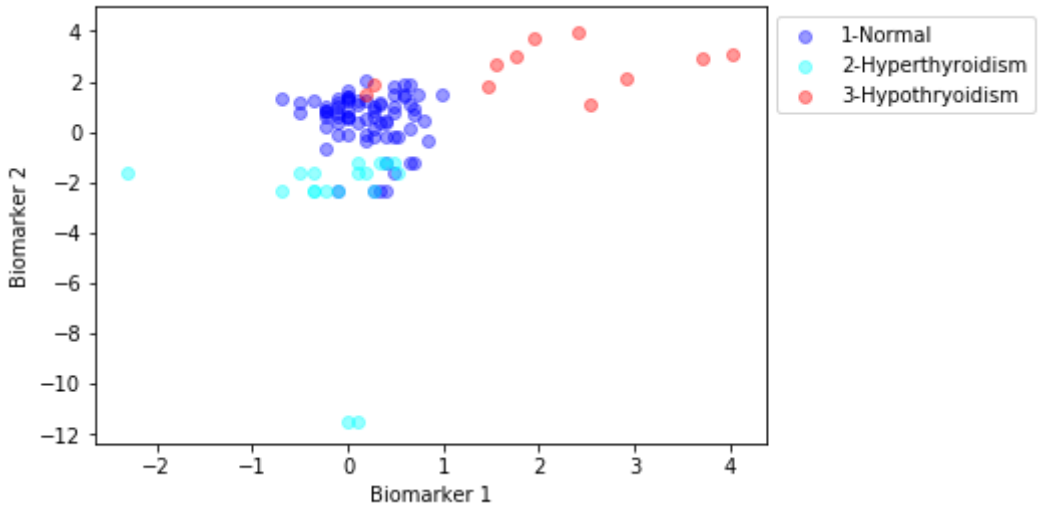
Hint: You may use the `KNeighborsClassifier` class to fit a k-NN classification model.

In [3]:
```
data_train.head()
```

Out[3]:

	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

In [4]:
```
# part 1, generate 2d plot of training set, different colors for each cl
ass

# conver into numpy array, split X and Y
data_train_nda = data_train.as_matrix()

X_train = data_train_nda[:, :-1]
Y_train = data_train_nda[:, -1]

data_test_nda = data_test.as_matrix()

X_test = data_test_nda[:, :-1]
Y_test = data_test_nda[:, -1]

# scatter plot, 3 colors
plt.scatter(X_train[Y_train == 1, 0], X_train[Y_train == 1, 1],
                                color='blue', alpha=0.4, label = '1-Norma
l')
plt.scatter(X_train[Y_train == 2, 0], X_train[Y_train == 2 , 1],
                                color='cyan', alpha=0.4, label = '2-Hypert
hyroidism')
plt.scatter(X_train[Y_train == 3, 0], X_train[Y_train == 3, 1],
                                color='red', alpha=0.4, label = '3-Hypothr
yoidism')

plt.xlabel('Biomarker 1')
plt.ylabel('Biomarker 2')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left', ncol=1)
plt.show()
```



In [5]:
```
# part 3, fit linear classification using MLR and OvR. use L2 regulariza
tion and cross-valid

data_train_nda = data_train.as_matrix()
X_train = data_train_nda[:, :-1]
Y_train = data_train_nda[:, -1]

data_test_nda = data_test.as_matrix()
X_test = data_test_nda[:, :-1]
Y_test = data_test_nda[:, -1]

# train mlr model, 5 fold CV
logreg_mlr = LogisticRegressionCV(cv=5, penalty='l2', multi_class='multi
nomial')
logreg_mlr.fit(X_train, Y_train)
print("MLR score: ", logreg_mlr.score(X_test,Y_test))

# train ovr model, 5 fold CV
logreg_ovr = LogisticRegressionCV(cv=5, penalty='l2', multi_class='ovr')
logreg_ovr.fit(X_train, Y_train)
print("OvR score: ", logreg_ovr.score(X_test,Y_test))
```

```
MLR score:  0.884955752212
OvR score:  0.867256637168
```

In [6]:
```
# part 4, train and test accuract for
    # Multiclass logistic regression with quadratic terms
    # linear discriminant analysis
    # quadratic discriminant analysis
    # k-NN

# Multiclass logistic regression with quadratic terms

# prepare data
import copy
poly_train_data = copy.deepcopy(data_train)
poly_test_data = copy.deepcopy(data_test)
data_train_nda_poly = np.asmatrix(poly_train_data)
data_test_nda_poly = np.asmatrix(poly_test_data)

X_train_poly = data_train_nda_poly[:, :-1]
Y_train_poly = data_train_nda_poly[:, -1]
X_test_poly = data_test_nda_poly[:, :-1]
Y_test_poly = data_test_nda_poly[:, -1]

# get polynomial terms
poly = PolynomialFeatures(2)
X_train_poly_final = poly.fit_transform(X_train_poly)
X_test_poly_final = poly.fit_transform(X_test_poly)

# train mlr model, 5 fold CV
logreg_mlr_poly = LogisticRegressionCV(cv=5, penalty='l2',
multi_class='multinomial')
logreg_mlr_poly.fit(X_train_poly_final, Y_train_poly)
print("MLR score with Polynomial: ", logreg_mlr_poly.score(X_test_poly_f
inal,Y_test_poly))
```

```
print()

# linear discriminant analysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)
print("LDA score: ", lda.score(X_test,Y_test))
print()

# quadratic discriminant analysis
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, Y_train)
print("QDA score: ", qda.score(X_test,Y_test))
print()

# k-NN

scores = []
num_neighbors = range(1, 80)
for n in num_neighbors:
    knn = KNeighborsClassifier(n_neighbors=n)
    cv_scores = cross_val_score(knn, X_train, Y_train, cv=5, scoring='ac
curacy')
    scores.append(cv_scores.mean())

index_of_k = scores.index(max(scores))
best_k = num_neighbors[index_of_k]
print("Best k value for k-NN is ", best_k)

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, Y_train)
print("K-NN score with k = {}: {}".format(best_k, knn.score(X_test,Y_tes
t)))
```

```
/Users/joshkuppersmith/anaconda/lib/python3.6/site-packages/sklearn/uti
ls/validation.py:526: DataConversionWarning: A column-vector y was pass
ed when a 1d array was expected. Please change the shape of y to (n_sam
ples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
MLR score with Polynomial:  0.902654867257

LDA score:  0.83185840708

QDA score:  0.849557522124

Best k value for k-NN is  3
K-NN score with k = 3: 0.8672566371681416
```

ANALYSIS

Does it appear that the data points can be separated well by a linear classifier?

Yes, the data actually fits into 3 categories quite nicely where each color is generally clumped together. A linear classifier like an SVM might work well to classify this data.

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).

Multinomial regression selects one category as a reference, then trains a model for each of the remaining categories to predict them based off of just the reference category. OvR instead goes through each category and trains a model based on all other categories to predict the selected category. This model is a bit more sophisticated because it bases our predictions on more data, but that does not necessarily mean it will be a better predictor.

Is there a difference in the overall classification accuracy of the two methods on the training and test sets?

There is a bit of a difference. MLR has classification rate of 0.884955752212, and OvR has classification rate of 0.867256637168, which means there is a meaningful difference between the two, and MLR is the better model.

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

MLR score with Polynomial: 0.902654867257

LDA score: 0.83185840708

QDA score: 0.849557522124

K-NN score with k = 3: 0.8672566371681416

So in order of best to worst, these models rank MLR with Polynomial, k-NN, QDA, then LDA, so there it appears that a simple linear boundary may not be the best way to classify the data, although it is not terrible.

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

Yes, it yields a slightly higher test accuracy, and these polynomial terms likely capture a non-linear trend in the data without overfitting on the train data.

Question 2: Visualize Decision Boundaries

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

In [7]:
```
#--------  plot_decision_boundary
# A function that visualizes the data and the decision boundaries
```

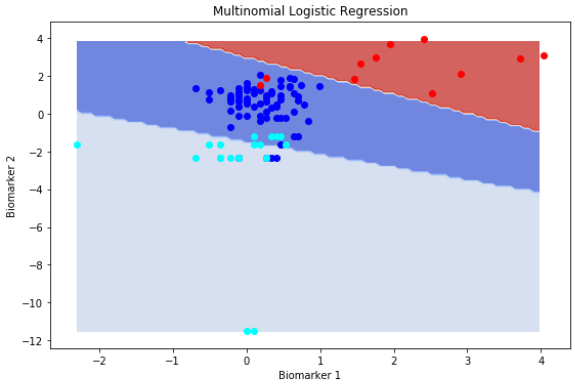
```
# Input:
#      x (predictors)
#      y (labels)
#      model (the classifier you want to visualize)
#      title (title for plot)
#      ax (a set of axes to plot on)
#      poly_degree (highest degree of polynomial terms included in the m
odel; None by default)

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)
        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=pl
t.cm.coolwarm)
    ax.scatter(x[yy==2,0], x[yy==2,1], c='cyan', label='Hyper',
cmap=plt.cm.coolwarm)
    ax.scatter(x[yy==3,0], x[yy==3,1], c='red', label='Hypo', cmap=plt.c
m.coolwarm)

    # Label axis, title
```

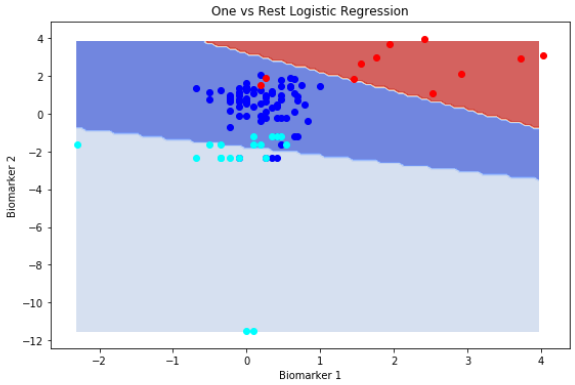
```
    ax.set_title(title)
    ax.set_xlabel('Biomarker 1')
    ax.set_ylabel('Biomarker 2')
```

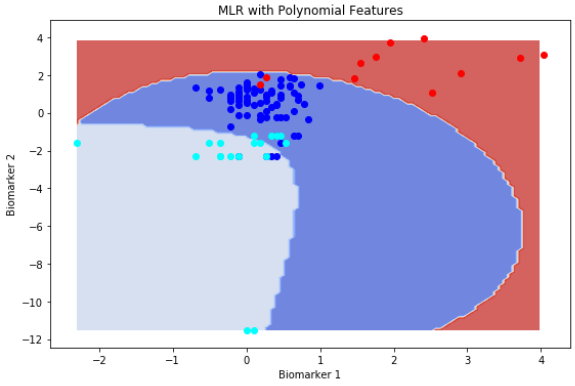
Note: The provided code uses `sklearn`'s `PolynomialFeatures` to generate higher-order polynomial terms, with degree `poly_degree`. Also, if you have loaded the data sets into `pandas` data frames, you may use the `as_matrix` function to obtain a `numpy` array from the data frame objects.

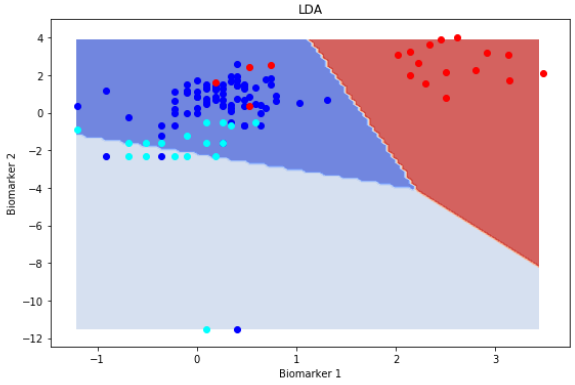
1. Use the above code to visualize the decision boundaries for each of the model fitted in the previous part.
2. Comment on the difference in the decision boundaries (if any) for the OvR and multinomial logistic regression models. Is there a difference between the decision boundaries for the linear logistic regression models and LDA. What about the decision boundaries for the quadratic logistic regression and QDA? Give an explanation for your answer.

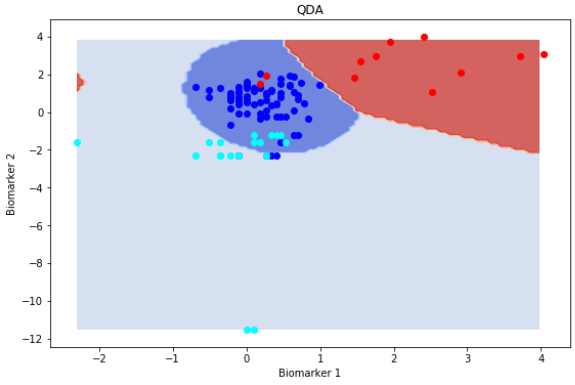
In [8]:
```
# part 1, use above code to visualize decision boundaries for all part a
 models

f, (ax1, ax2, ax3) = plt.subplots(3,2,figsize=(20,20))

# multinomial logistic regression
plot_decision_boundary(X_train, Y_train, logreg_mlr, "Multinomial Logist
ic Regression", ax=ax1[0], poly_degree=None)

# one-vs-rest (OvR) logistic regression
plot_decision_boundary(X_train, Y_train, logreg_ovr, "One vs Rest Logist
ic Regression", ax=ax1[1], poly_degree=None)

# Multiclass Logistic Regression with quadratic terms
plot_decision_boundary(X_train, Y_train, logreg_mlr_poly, "MLR with Poly
nomial Features", ax=ax2[0], poly_degree=2)

# Linear Discriminant Analysis
plot_decision_boundary(X_test, Y_test, lda, "LDA", ax=ax2[1], poly_degre
e=None)

# Quadratic Discriminant Analysis
plot_decision_boundary(X_train, Y_train, qda, "QDA", ax=ax3[0], poly_deg
ree=None)

# k-Nearest Neighbors
plot_decision_boundary(X_train, Y_train, knn, "K Nearest Neighbors",
ax=ax3[1], poly_degree=None)
```

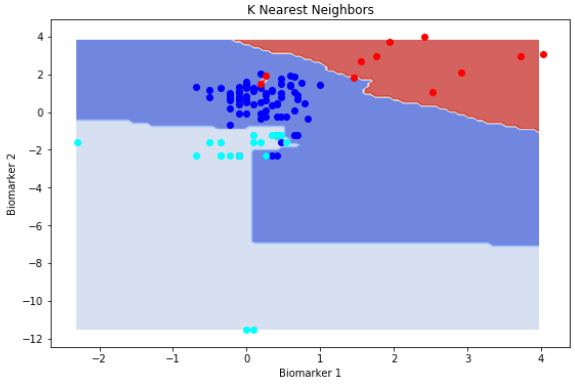

Multinomial Logistic Regression
Biomarker 1
Biomarker 2


One vs Rest Logistic Regression
Biomarker 1
Biomarker 2

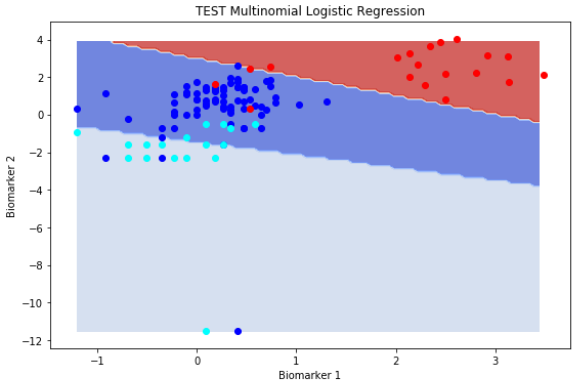

MLR with Polynomial Features
Biomarker 1
Biomarker 2

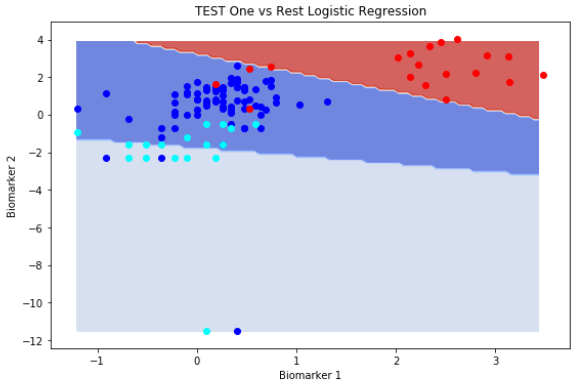

LDA
Biomarker 1
Biomarker 2

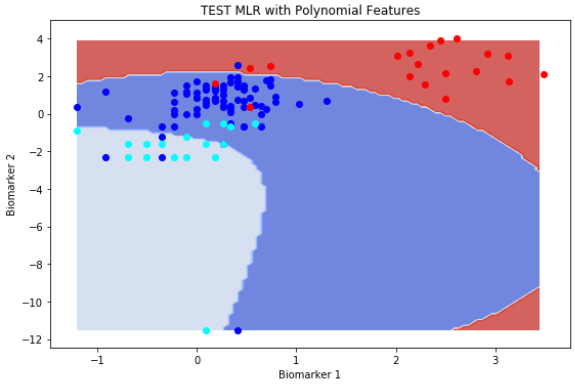

QDA
Biomarker 1
Biomarker 2

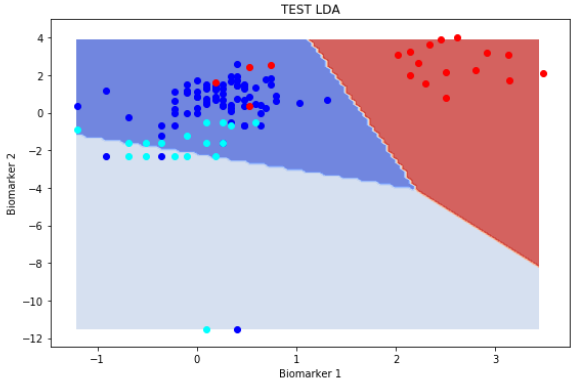

K Nearest Neighbors
Biomarker 1
Biomarker 2

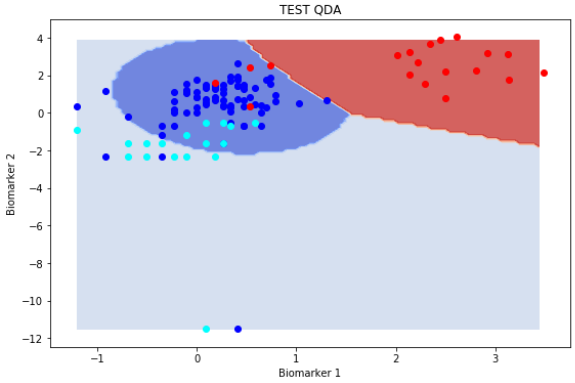
In [9]:
```
# Now for Test data

f2, (ax1, ax2, ax3) = plt.subplots(3,2,figsize=(20,20))

# multinomial logistic regression
plot_decision_boundary(X_test, Y_test, logreg_mlr, "TEST Multinomial Log
istic Regression", ax=ax1[0], poly_degree=None)

# one-vs-rest (OvR) logistic regression
plot_decision_boundary(X_test, Y_test, logreg_ovr, "TEST One vs Rest Log
istic Regression", ax=ax1[1], poly_degree=None)

# Multiclass Logistic Regression with quadratic terms
plot_decision_boundary(X_test, Y_test, logreg_mlr_poly, "TEST MLR with P
olynomial Features", ax=ax2[0], poly_degree=2)

# Linear Discriminant Analysis
plot_decision_boundary(X_test, Y_test, lda, "TEST LDA", ax=ax2[1], poly_
degree=None)

# Quadratic Discriminant Analysis
plot_decision_boundary(X_test, Y_test, qda, "TEST QDA", ax=ax3[0], poly_
degree=None)

# k-Nearest Neighbors
plot_decision_boundary(X_test, Y_test, knn, "TEST K Nearest Neighbors",
ax=ax3[1], poly_degree=None)
```

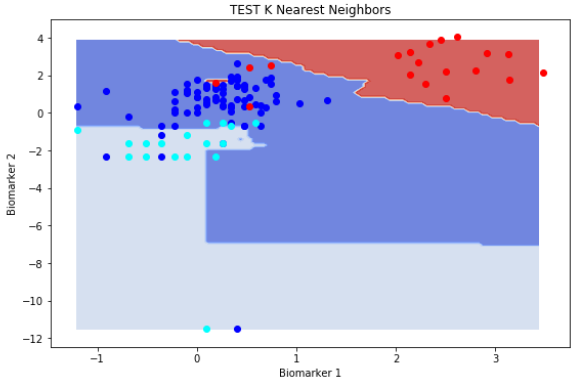

TEST Multinomial Logistic Regression
Biomarker 2
Biomarker 1


TEST One vs Rest Logistic Regression
Biomarker 2
Biomarker 1


TEST MLR with Polynomial Features
Biomarker 2
Biomarker 1


TEST LDA
Biomarker 2
Biomarker 1


TEST QDA
Biomarker 2
Biomarker 1


TEST K Nearest Neighbors
Biomarker 2
Biomarker 1

ANALYSIS

Comment on the difference in the decision boundaries (if any) for the OvR and multinomial logistic regression models.

The differences are slight, but they exist. The top boundary is very similar between the two but the lower boundary has a gentler slope for the OvR model. The intercepts are similar. Give an explanation for your answer. Since both of these methods train linear models, I am not surprised that the differences are so small, since they should generally fit the same linear trends, they just train the data in a slightly different way, as explained in the analysis for question 1.

Is there a difference between the decision boundaries for the linear logistic regression models and LDA. Give an explanation for your answer.

Yes, for linear logistic models, we simply get two linear boundaries separating the categories. But for LDA, we're able to capture more sophisticated patterns because we still find linear relationships, but we are able to calculate means and covariances to generate 3 (K=3) lines to divide this dataset into 3 categories rather than 2 lines for logistic regression. Here, this more sophisticated approach of LDA doesn't seem to do a better job classifying than logistic regression, but with a case like 3 distinct and nearby clusters, one can see why an LDA could be much more useful than the 2 lines from logistic regression.

What about the decision boundaries for the quadratic logistic regression and QDA? Give an explanation for your answer.

These are very different. Quadratic logistic has a large circle, outside of which is red, and a sort of half-moon blue area within the circle. QDA circles the blue points, finds a partition on the right and left for red points, and the rest is the final category. Both QDA and Quadratic logistic regression will generate quadratic curves as boundaries. Quadratic logistic regression simply finds a most oftimal coefficient for an x^2 term in the same way that a linear logistic regression finds a coefficient for x and an intercept. QDA on the other hand uses the LDA formulation, but relaxes the rules to allow the covariance of MVN distributions to be different. Like LDA, QDA allows more variation in the boundaries, so we see slightly more complex boundaries than the simple quadratic curves in quadratic logistic regression.

Question 3: Fit Decision Trees

We next try out decision trees for thyroid classification. For the following questions, you may use the *Gini* index as the splitting criterion while fitting the decision tree.

1. Fit a decision tree model to the thyroid data set with (maximum) tree depths 2, 3, ..., 10. Make plots of the training and test accuracies as a function of the tree depth. Is there a depth at which the fitted decision tree model achieves near-perfect classification on the training set? If so, what can you say about the test accuracy of this model?
2. Use 5-fold cross-validation to find the optimal tree depth. How does the performance of a decision tree fitted with this depth compare with the models fitted in Part 2(a)?
3. Use the code provided in Part 2(c) to visualize the decision boundary of the fitted decision tree. How is the decision boundary of the decision tree model different from the other methods? Given an explanation for your observation.
4. Use the `export_graphviz` function in `sklearn` to generate a visualization of the tree diagram for the fitted model. Based on the visualization, explain *in words* how the fitted model diagnoses 'hypothyroidism' for a patient.

Note: Look at the `export_graphviz` function in the `sklearn.tree` module.

You can get a graphic for this visualization by pasting the generated graphviz file in the text box at http://www.webgraphviz.com/ (http://www.webgraphviz.com/) , or you can do it on your own computer.

If you choose the do the latter, you will have to install `GraphViz` and `pydot` to use the decision tree rendering code. For this, you may execute the following commands in a terminal:

```
$pip install graphviz
$pip install pydot
```

Hint: You may use the `DecisionTreeClassifier` class to fit a decision tree classifier and the `max_depth` attribute to set the tree depth. You may use the `cross_val_score` function for cross-validation with decision trees.

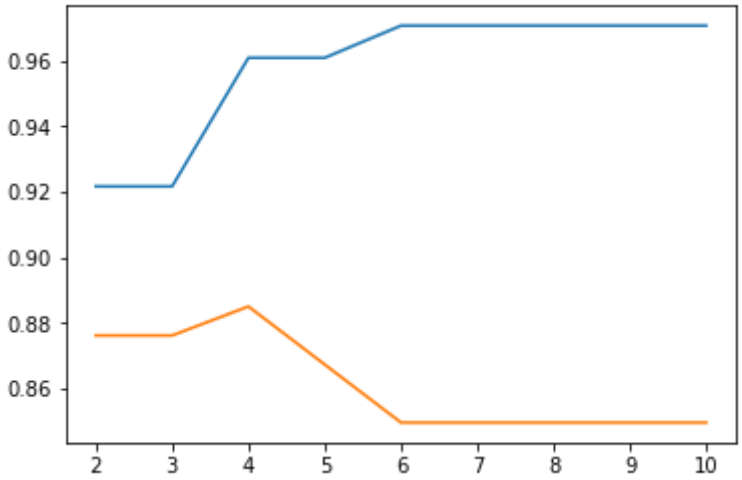
In [10]:
```
# part 1, fit decision tree model with depths 2-10
# plot train/test accuracy vs tree depth

# create array with depths we want to test
depths = [2, 3, 4, 5, 6, 7, 8, 9, 10]

# create empty arrays to hold accuracy scores later
train_scores = []
test_scores = []

# for each depth specified above, fit a tree classifier onto the train
# and find the accuracy score on the train and test sets
for depth in depths:
    dt = DecisionTreeClassifier(max_depth = depth)
    dt.fit(X_train, Y_train)
    train_scores.append(dt.score(X_train, Y_train))
    test_scores.append(dt.score(X_test, Y_test))

# format lists into data series
depths_se = pd.Series(depths)
train_scores_se = pd.Series(train_scores)
test_scores_se = pd.Series(test_scores)

# create a dataframe for data presentation
col_names = ['Depths', 'Train Scores', 'Test Scores']
depths_scores_df = pd.DataFrame(columns = col_names)

depths_scores_df['Depths'] = depths_se.values
depths_scores_df['Train Scores'] = train_scores_se.values
depths_scores_df['Test Scores'] = test_scores_se.values

depths_scores_df
```

Out[10]:

	Depths	Train Scores	Test Scores
0	2	0.921569	0.876106
1	3	0.921569	0.876106
2	4	0.960784	0.884956
3	5	0.960784	0.867257
4	6	0.970588	0.849558
5	7	0.970588	0.849558
6	8	0.970588	0.849558
7	9	0.970588	0.849558
8	10	0.970588	0.849558

In [11]:
```
# plot train and test accuracy scores on one chart for comparison
plt.plot(depths, train_scores)
plt.plot(depths, test_scores)
# plt.xscale('log')
```

Out[11]: [<matplotlib.lines.Line2D at 0x1213bab00>]



In [12]:
```
# part 2, use 5-fold cross validation to find optimal tree depth
from sklearn.cross_validation import KFold

# first pull number of observations in the training set
n = data_train.shape[0]

# create an empty array to hold accuracy scores later
scores = []

# for each train-validation split within the training set, run the follo
wing code
# goal is to use average accuracy scores to decide which depth makes for
 the best model
for f_train, f_test in KFold(n, n_folds=5, shuffle=False, random_state=1
1):
    # create secondary train and validation sets out of the original tra
in set
    mini_train = data_train.iloc[f_train]
    mini_test = data_train.iloc[f_test]

    # convert dataframes to numpy arrays as in part 1
    mini_train_nda = mini_train.as_matrix()

    X_mini_train = mini_train_nda[:, :-1]
    Y_mini_train = mini_train_nda[:, -1]

    mini_test_nda = mini_test.as_matrix()

    X_mini_test = mini_test_nda[:, :-1]
    Y_mini_test = mini_test_nda[:, -1]

    # create empty arrays to hold scores for this cross validation
    train_scores_cv = []
    test_scores_cv = []

    # for each depth, fit the tree classifier onto the train set
    for depth in depths:
        dt_cv = DecisionTreeClassifier(max_depth = depth)
        dt_cv.fit(X_mini_train, Y_mini_train)

        # calculate scores on train and validation sets
        train_scores_cv.append(dt_cv.score(X_mini_train, Y_mini_train))
        test_scores_cv.append(dt_cv.score(X_mini_test, Y_mini_test))

    scores.append(test_scores_cv)

# average the accuracy scores across the different train-validation spli
ts
scores = np.array(scores)
np.mean(scores, axis=0)
# print(avg_train_scores)
# print(avg_test_scores)
```

```
/Users/joshkuppersmith/anaconda/lib/python3.6/site-packages/sklearn/cro
ss_validation.py:44: DeprecationWarning: This module was deprecated in
 version 0.18 in favor of the model_selection module into which all the
refactored classes and functions are moved. Also note that the interfac
e of the new CV iterators are different from that of this module. This
 module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
Out[12]: array([ 0.91142857,  0.88238095,  0.88238095,  0.87238095,  0.86238095,
                 0.85285714,  0.86238095,  0.86238095,  0.86238095])
```

In [13]:
```
from sklearn.model_selection import cross_val_score

# create an empty array to hold accuracy scores later
scores = []

# for each depth, fit the tree classifier onto the train set
# use the cross_val_score method to directly perform cv and get the accu
racy scores
for depth in depths:
    dt = DecisionTreeClassifier(max_depth = depth)
    cv_scores = cross_val_score(dt, X_train, Y_train, cv=5)
    print(cv_scores)
    scores.append(np.mean(cv_scores))

# format into data series
depths_se = pd.Series(depths)
scores_se = pd.Series(scores)

# create a dataframe for data presentation
col_names = ['Depths', 'Scores']
depths_scores_df = pd.DataFrame(columns = col_names)

depths_scores_df['Depths'] = depths_se.values
depths_scores_df['Scores'] = scores_se.values

depths_scores_df
```

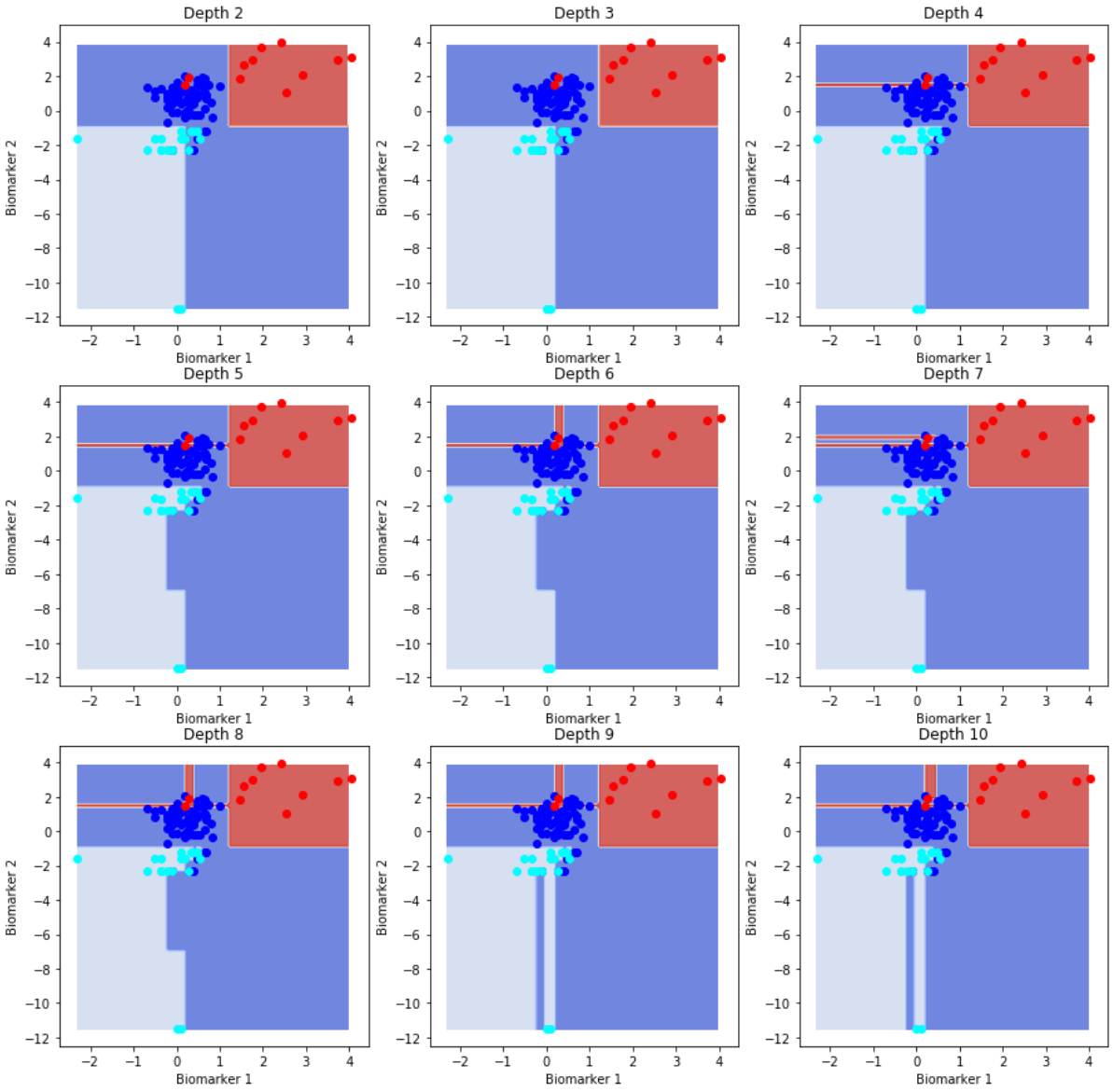
```
[ 0.90909091  0.9047619   0.85714286  0.89473684  1.        ]
[ 0.90909091  0.85714286  0.85714286  0.84210526  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.85714286  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
[ 0.86363636  0.85714286  0.80952381  0.78947368  1.        ]
```

Out[13]:

	Depths	Scores
0	2	0.913147
1	3	0.893096
2	4	0.863955
3	5	0.873479
4	6	0.863955
5	7	0.863955
6	8	0.863955
7	9	0.863955
8	10	0.863955

In [14]:
```
# part 3, visualize boundary of fitted tree

#--------  fit_and_plot_dt
# Fit decision tree with on given data set with given depth, and plot th
e data/model
# Input:
#      fname (string containing file name)
#      depth (depth of tree)

def fit_and_plot_dt(x, y, depth, title, ax, plot_data=True, fill=True, c
olor='Greens'):
    # FIT DECISION TREE MODEL
    dt = DecisionTreeClassifier(max_depth = depth)
    dt.fit(x, y)

    # PLOT DECISION TREE BOUNDARY
    ax = plot_decision_boundary(x, y, dt, title, ax, poly_degree=None)
    return ax
```

In [15]:
```
# plot decision tree boundaries on a scatter plot of training data, for
 each depth
fig, ax = plt.subplots(3, 3, figsize=(15, 15))
ind = 0
for i in depths:
    y_ind = ind // 3
    ax[y_ind, (ind - y_ind*3)] = fit_and_plot_dt(X_train, Y_train, i, 'D
epth {}'.format(i), ax[y_ind, (ind - y_ind*3)])
    # ax[ind].set_xlim(-6, 6)
    # ax[ind].set_ylim(-6, 6)
    ind += 1
```



In [16]:
```
# part 4, visualize the tree diagram of the fitted model. use export_graphviz
dt2 = DecisionTreeClassifier(max_depth = 2)
dt2.fit(X_train, Y_train)

export_graphviz(dt2)
```

```
/Users/joshkuppersmith/anaconda/lib/python3.6/site-packages/sklearn/tree/export.py:386: DeprecationWarning: out_file can be set to None starting from 0.18. This will be the default in 0.20.
  DeprecationWarning)
```

ANALYSIS

Is there a depth at which the fitted decision tree model achieves near-perfect classification on the training set? If so, what can you say about the test accuracy of this model?

The decision tree model first achieves its highest classification accuracy on the training set at a maximum depth of 6. Adding more layers of depth does not improve the accuracy. For the testing set however, a tree depth of 6 is when the accuracy drops the most (from .876106 to 0.849558). This decline in accuracy suggests we have overfit the model on the training set, and that our test accuracy is no longer optimal.

How does the performance of a decision tree fitted with this depth compare with the models fitted in Part 2(a)?

The performance of a decision tree fitted with max depth of 6 is definitely on the low end in terms of its accuracy. It is actually equal to the score from the QDA model, which means that it predicted the existence of thyroid disorders Only QDA is worse, while the others are better.

MLR score with Polynomial: 0.902654867257

LDA score: 0.83185840708

QDA score: 0.849557522124

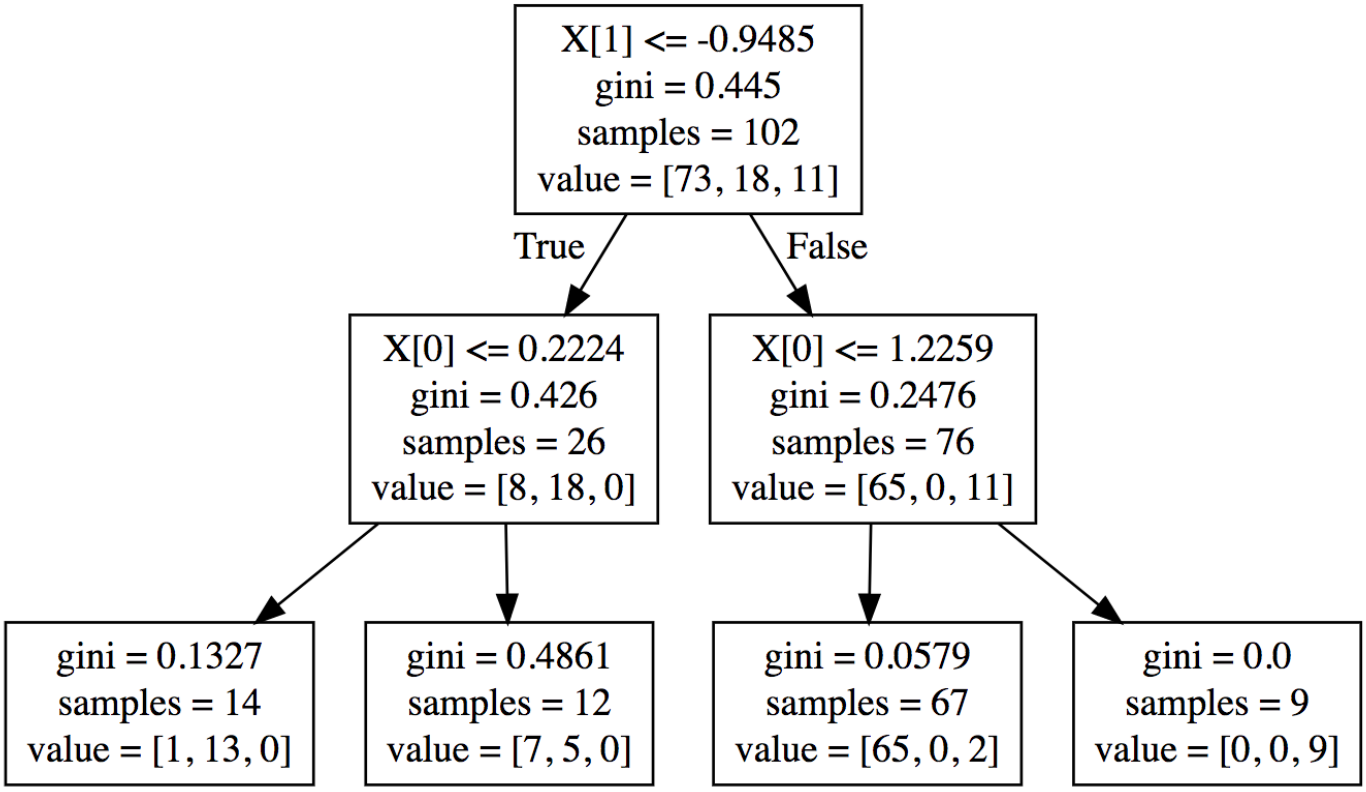
Best k value for k-NN is 3 K-NN score with k = 3: 0.8672566371681416

How is the decision boundary of the decision tree model different from the other methods? Given an explanation for your observation.

Each decision boundary of the decision tree model is a straight line, and more specifically parallel to either the x or y-axis. The linear logistic regression models, and the LDA model, also have straight decision boundary lines, but they have slopes that are affected by the distribution of data points. k-NN comes closest in terms of also have boundary lines parallel to the axes, but for small k (few neighbors), the boundary lines still appear to be sloped.

Based on the visualization, explain in words how the fitted model diagnoses 'hypothyroidism' for a patient.

This visualization shows that Biomarker 2 is a stronger predictor since it is higher up on the tree, therefore we split our decision first. Biomarker 1 is likely a weaker predictor, since we don't do our initial break on this marker. In this depth 2 tree, at the leafs, we see groups that are divided by their classification, and we can determine which by counting how many points are classified in each category. There are only 9 classified as red, so this is the far right leaf. It also looks like only 12 are categorized with the gray backround, so this is probably the second leaf from the left. Then, the far left and 3rd from the left leafs represent 81 points that are classified as blue.



Question 4: Too many models to choose from!

We have so far seen six different ways of fitting a classification model for thyroid classification problem: **linear logistic regression**, **logistic regression with polynomial terms**, **LDA**, **QDA**, **k-NN** and **decision tree**. Which of these methods should one use in practice? To answer this question, we now look at the pros and cons of each method.

1. Compare and contrast the six models based on each of the following criteria (a supporting table to summarize your thoughts can be helpful):
    - Classification performance
    - Complexity of decision boundary
    - Memory storage
    - Ease of interpretability
2. If you were a clinician who had to use the classifier to diagnose thyroid disorders in patients, which among the six methods would you be most comfortable in using?

ANALYSIS:

Compare all 6 models based on Classification accuracy

1) Decision Tree best score: 0.913147

2) Logistic Regression with Polynomial score: 0.902654867257

3) Linear Logistic Regression score: 0.884955752212

4) K-NN score with k = 3: 0.8672566371681416

5) LDA score: 0.83185840708

6) QDA score: 0.849557522124

Compare all 6 models based on complexity of decision boundary

In our opinion, here are how our models rank based on boundary complexity

1) Most Complex: Decision tree at Depth 10

2) k-NN

3) Logistic Regression with Polynomial

4) QDA

5) LDA

6) Least Complex: Linear Logistic Regression

Compare all 6 models based on memory storage

1) Least storage required: k-NN, which is non-parametric so does not store data. The only issue is that we had to run this many times with cross validation in order to find the optimal k.

2) Decision tree: all of the testing of various depths likely require time like k-NN, but it is non-paramentric, so stores very little.

3) Linear Logistic Regression

4) Logistic Regression with Polynomial

5) LDA

6) Most Storage Required: QDA

Compare all 6 models based on ease of interpretability

Again, this is relatively opinion based

1) Most intepretable: Linear Logistic Regression- linear with two coefficients

2) LDA- still linear, but coefficients are harder to visualize simply

3) Logistic Regression with Polynomial- non-linear and more coefficients that MLR, but it is clear how they make the model that they do.

4) QDA- like LDA, but also has polynomial factors, so there are more coefficients

5) k-NN: easy to understand the algorithm, but no clear pattern and no coefficients to summarize the model easily.

6) Least interpretable: Decision Tree- approaching black-box-y algorithms that are tough to interpret

If you were a clinician who had to use the classifier to diagnose thyroid disorders in patients, which among the six methods would you be most comfortable in using?

I would choose Logistic Regression with Polynomial terms. Decision trees clearly perform the best, and are efficient, but with decisions as important as diagnoses, I would want to make sure that I understand the reason for a particular diagnosis and I don't think decision trees clearly show what is happening, especially with the classification boundaries. Logistic regression with quadratic terms doesn't perform the absolute best in terms of accuracy and space, but it performs well enough and its accuracy is just a touch behind Decision trees. Additionally, the boundary is complex, but still understandable, so I think this is the best combination of diagnosis accuracy and transparency, so that the process does not become a black-box.

Question 5: Including an 'abstain' option

One of the reasons a hospital might be hesitant to use your thyroid classification model is that a misdiagnosis by the model on a patient can sometimes prove to be very costly (e.g. if the patient were to file a law suit seeking a compensation for damages). One way to mitigate this concern is to allow the model to 'abstain' from making a prediction, whenever it is uncertain about the diagnosis for a patient. However, when the model abstains from making a prediction, the hospital will have to forward the patient to a thyroid specialist (i.e. an endocrinologist), which would incur additional cost. How does one design a thyroid classification model with an abstain option, such that the cost to the hospital is minimized?

1. More specifically, suppose the cost incurred by a hospital when a model mis-predicts on a patient is \$5000, and the cost incurred when the model abstains from making a prediction is \$1000. What is the average cost per patient for the OvR logistic regression model from Question 1, Part 3? Note that this needs to be evaluated on the patients in the test set. Your task is to design a classification strategy (into the 3 groups plus the *abstain* group) that has as low cost as possible per patient. Give a justification for your approach.
2. **Presentation:** Prepare a set of 5 slides explaining your approach to the hospital management. Your presentation must be accessible to the lay man. Explain in particular how your approach would be robust to changes in the costs of using the abstain option.

Hint: think of a way to use the estimated probabilities from the logistic regression model to decide who to classify as *abstain*.

In [17]:
```
# part 1, cost for mis-predict is 5000, cost for no-predict is 1000. Fin
d cost/patient
```

```
# for OvR logistic regression for patients in test set. Design a classif
ication strategy
# with lowest possible cost/patient

# CAUTION: TAKES SOME TIME TO RUN

# train ovr model, 5 fold CV

mis_cost = 5000
abst_cost = 1000
logreg_ovr = LogisticRegressionCV(cv=5, penalty='l2', multi_class='ovr')
logreg_ovr.fit(X_train, Y_train)
score = logreg_ovr.score(X_test, Y_test)
cost_ovr = (1-score)*mis_cost
print("Cost per patient of OvR method (it doesn't have any abstaining):
 ", cost_ovr)
print("Now let's do better!")

# rather than using a default classification strategy, we will manipulat
e
# the predicted probabilities, and classify only if we are very confiden
t
# we will use cross validation on the training set to determine the best
# threshold

# we will use MLR with polynomial features, use the score calculated ear
lier
# get polynomial terms
poly = PolynomialFeatures(2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.fit_transform(X_test)
logreg_mul = LogisticRegressionCV(cv=5, penalty='l2', multi_class='multi
nomial')
logreg_mul.fit(X_train_poly, Y_train)

# use CV, test for many possible thresholds
from sklearn.cross_validation import KFold
n = X_train_poly.shape[0]
scores = []
thresholds = np.arange(0,1,.05)
costs = []

# iterate through all classification thresholds
for p in thresholds:
    # 5 fold cross validation!
    ave_cost = 0
    for f_train, f_test in KFold(n, n_folds=5, shuffle=False, random_sta
te=11):

        # train, valid split
        train_x = X_train_poly[f_train]
        valid_x = X_train_poly[f_test]
        train_y = Y_train[f_train]
        valid_y = Y_train[f_test]

        # train MLR with poly terms model and predict
        logreg_mul = LogisticRegressionCV(cv=5, penalty='l2', multi_clas
```