cs109a_hw7-submit

November 8, 2018

1 CS109A Introduction to Data Science

1.1 Homework 7: Classification with Logistic Regression, LDA/QDA, and Trees

Harvard University Fall 2018 Instructors: Pavlos Protopapas, Kevin Rader

```
In [1]: #RUN THIS CELL
        import requests
        from IPython.core.display import HTML
        styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/maste
        HTML(styles)
```

```
Out[1]: <IPython.core.display.HTML object>
```

1.1.1 INSTRUCTIONS

- To submit your assignment follow the instructions given in Canvas.
- If needed, clarifications will be posted on Piazza.
- This homework can be submitted in pairs.
- If you submit individually but you have worked with someone, please include the name of your **one** partner below.

Name of the person you have worked with goes here:
Avriel Epps, Erin Williams

```
In [2]: %matplotlib inline
        import numpy as np
        import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set()

        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 train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.pipeline import make_pipeline
from sklearn.datasets import make_blobs
```

Question 1 [20 pts]: Overview of Multiclass Thyroid Classification

In this problem set 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_hw7.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; for this assignment we chose two predictors so we can visualize the decision boundaries.

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

For most of this problem set, we'll measure overall classification accuracy as the fraction of observations classified correctly.

1.1 Load the data and examine its structure. How many instances of each class are there in our dataset? In particular, what is the ratio of the number of observations in class 2 (hyperthyroidism) to the number of observations in class 3 (hypothyroidism)? We'll refer to this as the *hyper-to-hypo ratio*.

1.2: We're going to split this data into a 50% training set and a 50% test set. But since our dataset is small, we need to make sure we do it correctly. Let's see what happens when we *don't* split correctly: for each of 100 different random splits of the data into 50% train and 50% test, compute the hyper-to-hypo for the observations end up in the training set. Plot the distribution of the hyper-to-hypo ratio; on your plot, also mark the hyper-to-hypo ratio that you found in the full dataset. Discuss how representative the training and test sets are likely to be if we were to have selected one of these random splits.

1.3 Now, we'll use the `stratify` option to split the data in such a way that the relative class frequencies are preserved (the code is provided). Make a table showing how many observations of each class ended up in your training and test sets. Verify that the hyper-hypo ratio is roughly the same in both sets.

1.4 Provide the scatterplot of the predictors in the (training) data in a way that clearly indicates which class each observation belongs to.

1.5: When we first start working with a dataset or algorithm, it's typically a good idea to figure out what *baselines* we might compare our results to. For regression, we always compared against a baseline of predicting the mean (in computing R^2). For classification, a simple baseline is always predicting the *most common class*. What "baseline" accuracy can we achieve on the thyroid classification problem by always predicting the most common class? Assign the result to `baseline_accuracy` so we can use it later. (**note: don't look at the test set until instructed**)

1.6 Make a decision function to separate these samples using no library functions; just write out your logic by hand. Your manual classifier doesn't need to be well-tuned (we'll be exploring algorithms to do that!); it only needs to (1) predict each class at least once, and (2) achieve an accuracy at least 10% greater accurate than predicting the most likely class. Use the `overlay_decision_boundaries` function provided above to overlay the decision boundaries of your function on the training set. (Note that the function modifies an existing plot, so call it after plotting your points.)

Based on your exploration, do you think a linear classifier (i.e., a classifier where all decision boundaries are line segments) could achieve above 85% accuracy on this dataset? Could a non-linear classifier do better? What characteristics of the data lead you to these conclusions?

1.1

```
In [3]: # your code here
        data = pd.read_csv('dataset_HW7.csv')
        display(data.head())
        display(data.describe())
        display(data.info())
```

	Biomarker 1	Biomarker 2	Diagnosis
0	0.262372	0.875473	1
1	0.693152	0.262372	1
2	0.262372	0.405472	1
3	-0.105349	1.064714	1
4	0.000010	1.131405	1

	Biomarker 1	Biomarker 2	Diagnosis
count	215.000000	215.000000	215.000000
mean	0.414441	0.303155	1.441860
std	0.888106	2.174369	0.726737
min	-2.302485	-11.512925	1.000000
25%	0.000010	-0.510809	1.000000
50%	0.262372	0.693152	1.000000
75%	0.530634	1.410989	2.000000
max	4.032469	4.030695	3.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 3 columns):
Biomarker 1    215 non-null float64
Biomarker 2    215 non-null float64
Diagnosis      215 non-null int64
```

```
dtypes: float64(2), int64(1)
memory usage: 5.1 KB


None
```

```
In [4]: # your code here
        (data['Diagnosis']).value_counts()
```

```
Out[4]: 1    150
        2     35
        3     30
        Name: Diagnosis, dtype: int64
```

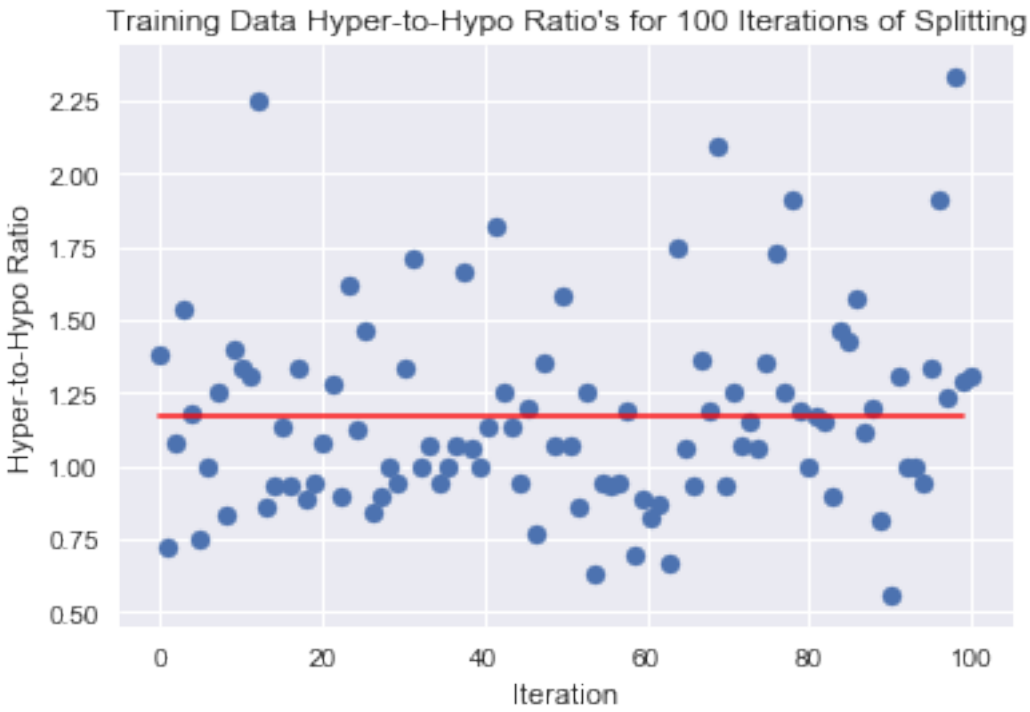
```
In [5]: data['Diagnosis'].value_counts(normalize=True)
```

```
Out[5]: 1    0.697674
        2    0.162791
        3    0.139535
        Name: Diagnosis, dtype: float64
```

Answer: There are 150 instances of normal thyroid levels, 35 instances of hyperthroidism, and 30 cases of hypothyroidism. The hyper-to-hypo ratio is 7:6.

1.2 We're going to split this data into a 50% training set and a 50% test set. But since our dataset is small, we need to make sure we do it correctly. Let's see what happens when we *don't* split correctly: for each of 100 different random splits of the data into 50% train and 50% test, compute the hyper-to-hypo for the observations end up in the training set. Plot the distribution of the hyper-to-hypo ratio; on your plot, also mark the hyper-to-hypo ratio that you found in the full dataset. Discuss how representative the training and test sets are likely to be if we were to have selected one of these random splits.

```
In [6]: ratio_list = []
        for i in range (0, 100):
            data_train, data_test = train_test_split(data, test_size=.5)
            counts_train = (data_train['Diagnosis']).value_counts()
            train_hyper = counts_train[2]
            train_hypo = counts_train[3]
            ratio = train_hyper/train_hypo
            ratio_list.append(ratio)

        x=np.linspace(0, 100, 100)
        y=0*x + (7/6)

        len(ratio_list)

        plt.scatter(x,ratio_list)
        plt.plot(y, color = 'r', alpha = 0.8)
        plt.title("Training Data Hyper-to-Hypo Ratio's for 100 Iterations of Splitting")
        plt.xlabel("Iteration")
        plt.ylabel("Hyper-to-Hypo Ratio");
```



Answer: Because we've failed to stratify the data and there are so few instances of hyper-and hypo-thyroidism, we see that the hyper-to-hypo ratio varies from about 0.6 up to 2.3 (vs. the value of 1.6 calculated from the original data). These training and test sets are unlikely to give a good represenataion of the data if we split this way.

1.3 Now, we'll use the `stratify` option to split the data in such a way that the relative class frequencies are preserved (the code is provided). Make a table showing how many observations of each class ended up in your training and test sets. Verify that the hyper-hypo ratio is roughly the same in both sets.

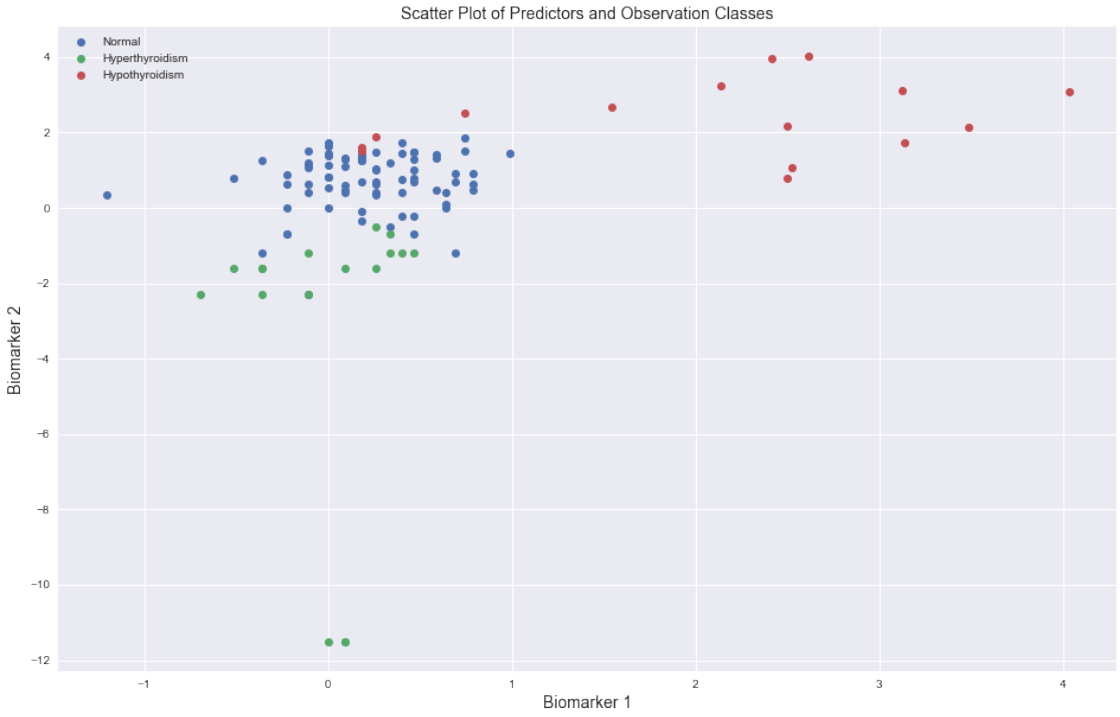
```
In [7]: data_train, data_test = train_test_split(data, test_size=.5, stratify=data.Diagnosis, r

In [8]: len(data_train), len(data_test)

Out[8]: (107, 108)

In [9]: #Still need to put this into a new table.
        display(data_test.groupby('Diagnosis').count())
        display(data_train.groupby('Diagnosis').count())
```

Diagnosis	Biomarker 1	Biomarker 2
1	75	75
2	18	18
3	15	15

```
           Biomarker 1  Biomarker 2
Diagnosis
1                   75           75
2                   17           17
3                   15           15
```

1.4 Provide the scatterplot of the predictors in the (training) data in a way that clearly indicates which class each observation belongs to.

```
In [10]: def scatter(df):
             fig, ax = plt.subplots(figsize=(16,10));
             groups = df.groupby('Diagnosis');
             names = {'1':'Normal', '2':'Hyperthyroidism', '3':'Hypothyroidism'};
             for name, group in groups:
                 ax.plot(group['Biomarker 1'], group['Biomarker 2'], marker = 'o', linestyle =
             ax.legend();
             ax.set_xlabel('Biomarker 1', size=14);
             ax.set_ylabel('Biomarker 2', size=14);
             ax.set_title('Scatter Plot of Predictors and Observation Classes', size=14);
             return (fig,ax)

         scatter(data_train);
```



1.5: When we first start working with a dataset or algorithm, it's typically a good idea to figure out what *baselines* we might compare our results to. For regression, we always compared

against a baseline of predicting the mean (in computing R^2). For classification, a simple baseline is always predicting the *most common class*. What "baseline" accuracy can we achieve on the thyroid classification problem by always predicting the most common class? Assign the result to `baseline_accuracy` so we can use it later. (**note: don't look at the test set until instructed**)

```
In [11]: y_train = np.array(data_train['Diagnosis'])
         X_train = np.array(data_train.drop(['Diagnosis'], inplace=False, axis =1))
         y_test = np.array(data_test['Diagnosis'])
         X_test = np.array(data_test.drop(['Diagnosis'], inplace=False, axis =1))

         df_value = {'Test': data_test['Diagnosis'].value_counts(), 'Train': data_train['Diagno
         df_value = pd.DataFrame(data=df_value)
         baseline_accuracy = float(df_value.Test.values[0])/float(df_value.Test.values[0] + df_
         baseline_accuracy
```

```
Out[11]: 0.6944444444444444
```

1.6 Make a decision function to separate these samples using no library functions; just write out your logic by hand. Your manual classifier doesn't need to be well-tuned (we'll be exploring algorithms to do that!); it only needs to (1) predict each class at least once, and (2) achieve an accuracy at least 10% greater accurate than predicting the most likely class. Use the `overlay_decision_boundaries` function provided above to overlay the decision boundaries of your function on the training set. (Note that the function modifies an existing plot, so call it after plotting your points.)

Based on your exploration, do you think a linear classifier (i.e., a classifier where all decision boundaries are line segments) could achieve above 85% accuracy on this dataset? Could a non-linear classifier do better? What characteristics of the data lead you to these conclusions?

```
In [12]: def overlay_decision_boundary(ax, model, colors=None, nx=200, ny=200, desaturate=.5):
             """
             A function that visualizes the decision boundaries of a classifier.

             ax: Matplotlib Axes to plot on
             model: Classifier (has a `.predict` method)
             X: feature vectors
             y: ground-truth classes
             colors: list of colors to use. Use color colors[i] for class i.
             nx, ny: number of mesh points to evaluated the classifier on
             desaturate: how much to desaturate each of the colors (for better contrast with t
             """
             # Create mesh
             xmin, xmax = ax.get_xlim()
             ymin, ymax = ax.get_ylim()
             xx, yy = np.meshgrid(
                 np.linspace(xmin, xmax, nx),
                 np.linspace(ymin, ymax, ny))
             X = np.c_[xx.flatten(), yy.flatten()]

             # Predict on mesh of points
```

```
            if hasattr(model, 'predict'):
                model = model.predict
            y = model(X)
            y = y.reshape((nx, ny))

            # Generate colormap.
            if colors is None:
                colors = sns.utils.get_color_cycle()
                y -= y.min() # If first class is not 0, shift.
            assert np.max(y) <= len(colors)
            colors = [sns.utils.desaturate(color, desaturate) for color in colors]
            cmap = matplotlib.colors.ListedColormap(colors)

            # Plot decision surface
            ax.pcolormesh(xx, yy, y, zorder=-2, cmap=cmap, norm=matplotlib.colors.NoNorm(), vm
            xx = xx.reshape(nx, ny)
            yy = yy.reshape(nx, ny)
        #     ax.contourf(xx, yy, y, cmap=cmap, vmin=0, vmax=3)
            ax.contour(xx, yy, y, colors="black", linewidths=1, zorder=-1)
```

In [13]:
```
# Update the following function:
def predict_manual_one_sample(x):
    if float(x[0]) > 1.35:
        return 3
    elif float(x[1]) > -.5:
        return 1
    else:
        return 2
    return
```

In [14]:
```
def predict_manual(X):
    return np.array([predict_manual_one_sample(x) for x in X])
```

In [15]:
```
manual_predictions_train = predict_manual(X_train)
manual_predictions_test = predict_manual(X_test)

accuracy_train = accuracy_score(y_train, manual_predictions_train)
accuracy_test = accuracy_score(y_test, manual_predictions_test)

print("Accuracy Train:", accuracy_train)
print("Accuracy Test:", accuracy_test)
```

```
Accuracy Train: 0.897196261682243
Accuracy Test: 0.8611111111111112
```

In [16]:
```
assert accuracy_train >= (baseline_accuracy * 1.10), "Accuracy too low"
assert all(np.sum(manual_predictions_train == i) > 0 for i in [1, 2, 3]), "Should pred
```

Your answer here

Yes, I think a linear classifier (i.e., a classifier where all decision boundaries are line segments) could achieve above 85% accuracy on this dataset, because we achieved that. I do think a non-linear classifier would do better, because when looking at the plotted data, the classes are group in elipses type shapes, so it seems the boundaries between them are non-linear.

Question 2 [20 pts]: Multiclass Logistic Regression

2.1 Fit two one-vs-rest logistic regression models using sklearn. For the first model, use the train dataset as-is (so the decision boundaries will be linear); for the second model, also include quadratic and interaction terms. For both models, use L_2 regularization, tuning the regularization parameter using 5-fold cross-validation.

For each model, make a plot of the training data with the decision boundaries overlayed.

2.2 Interpret the decision boundaries: - Do these decision boundaries make sense? - What does adding quadratic and interaction features do to the shape of the decision boundaries? Why? - How do the different models treat regions where there are few samples? How do they classify such samples?

2.3 Compare the performance of the two logistic regression models above using 5-fold cross-validation. Which model performs best? How confident are you about this conclusion? Does the inclusion of the polynomial terms in logistic regression yield better accuracy compared to the model with only linear terms? Why do you suspect it is better or worse?

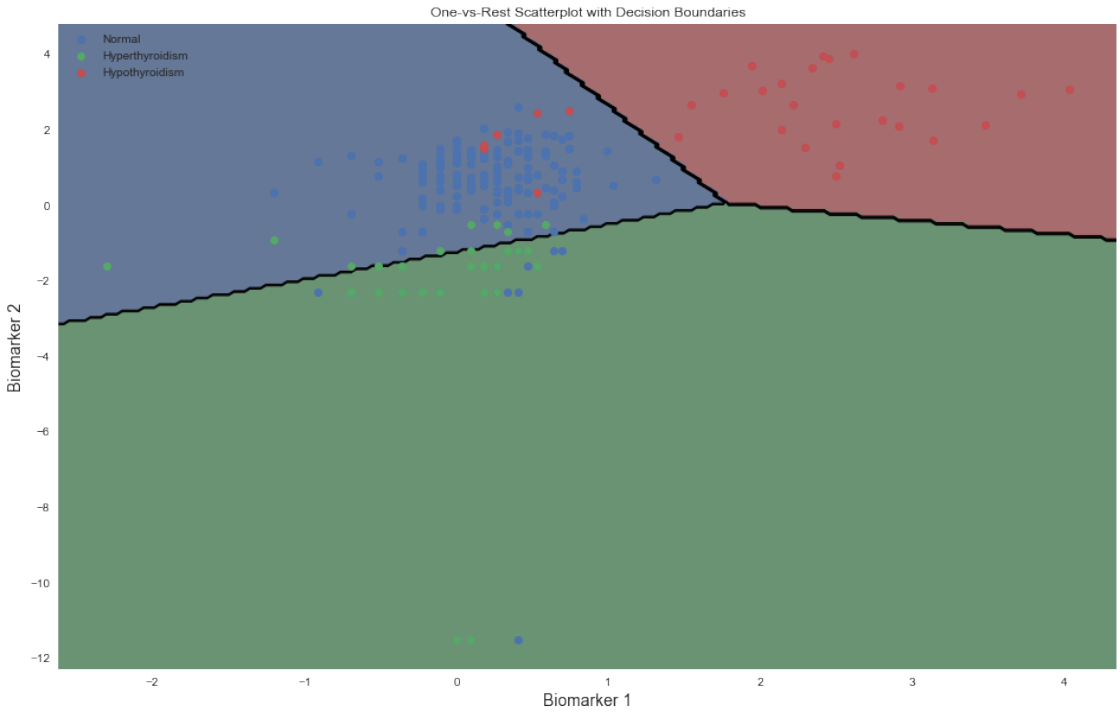
Hint: You may use the `cross_val_score` function for cross-validation.

2.1 Fit two one-vs-rest logistic regression models using sklearn. For the first model, use the train dataset as-is (so the decision boundaries will be linear); for the second model, also include quadratic and interaction terms. For both models, use L_2 regularization, tuning the regularization parameter using 5-fold cross-validation.

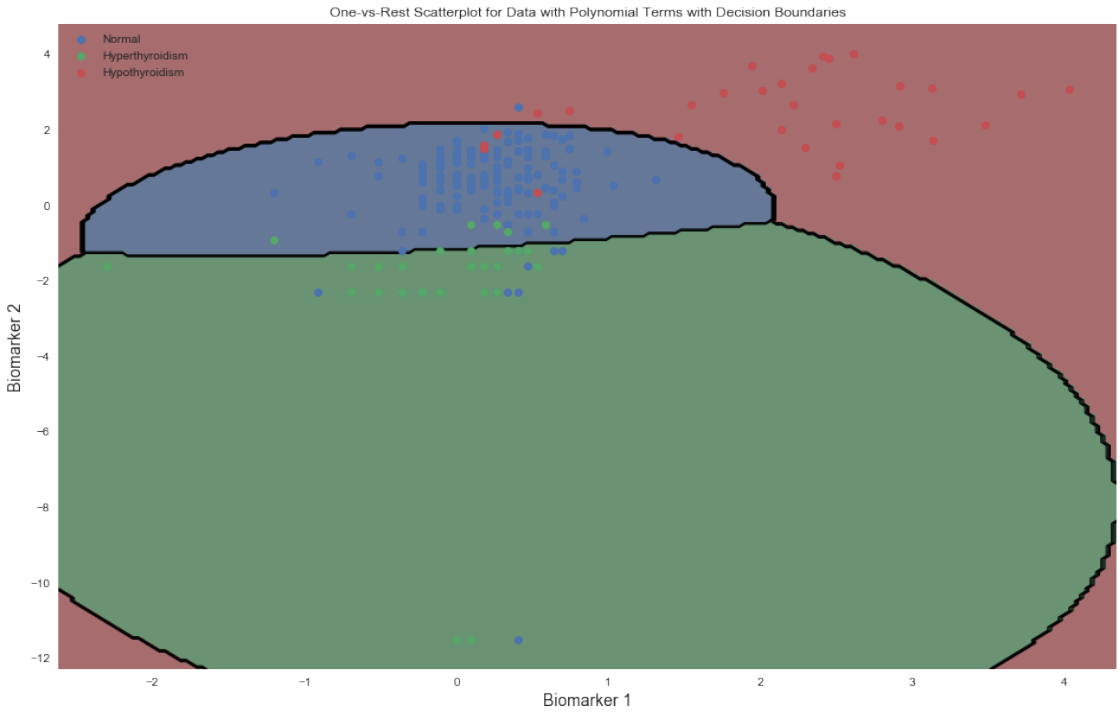
For each model, make a plot of the training data with the decision boundaries overlayed.

Hint: You should use `LogisticRegressionCV`. For the model with quadratic and interaction terms, use the following Pipeline:

```
In [17]: polynomial_logreg_estimator = make_pipeline(
             PolynomialFeatures(degree=2, include_bias=False),
             LogisticRegressionCV(multi_class = 'ovr',penalty = 'l2',cv = 5, max_iter = 10000)

         # Note that you can access the logistic regression classifier itself by
         # polynomial_logreg_estimator.named_steps['logisticregressioncv']

In [18]: LogRegOvR = LogisticRegressionCV(multi_class = 'ovr',penalty = 'l2',cv = 5, max_iter =

         fix, ax = scatter(data)
         ax.set_title("One-vs-Rest Scatterplot with Decision Boundaries")
         overlay_decision_boundary(ax, LogRegOvR)
```



In [19]:
```
LogRegOvR_poly = polynomial_logreg_estimator.fit(X_train, y_train)

fix, ax = scatter(data)
ax.set_title("One-vs-Rest Scatterplot for Data with Polynomial Terms with Decision Bou
overlay_decision_boundary(ax, LogRegOvR_poly)
```



2.2

Your answer here

The first model plot looks linear and the second looks curved. However, The boundary between Normal and hyperthyroidism basically looks linear. What is most concerning about the second plot is that the "background" is all hypothyroidism, meaning that anything that wouldn't fit in the other two classes would be considered hypothyroidism, which seems wrong at lower levels of Biomarker 2 and extreme levels of Biomarker 1.

2.3

```
In [20]: # your code here
         print(LogRegOvR.score(X_train, y_train),LogRegOvR.score(X_test,y_test))

         print(LogRegOvR_poly.score(X_train, y_train),LogRegOvR_poly.score(X_test, y_test))
```

```
0.9158878504672897 0.8425925925925926
0.9345794392523364 0.8611111111111112
```

Your answer here The polynomial model fits best because it's got a higher score for both train and test. As mentioned in 1.6, we assumed a curvilinear model would be better, because when looking at the data they seem to group in the shape of ellipses.

Question 3 [20 pts]: Discriminant Analysis

3.1 Consider the following synthetic dataset with two classes. A green star marks a test observation; which class do you think it belongs to? How would LDA classify that observation? How would QDA? Explain your reasoning.

3.2 Now let's return to the thyroid dataset. Make a table of the total variance of each class for each biomarker.

3.3 Fit LDA and QDA on the thyroid data, and plot the decision boundaries. Comment on how the decision boundaries differ. How does the difference in decision boundaries relate to characteristics of the data, such as the table you computed above?

3.1

```
In [21]: X_blobs, y_blobs = make_blobs(centers=[[0., 0.], [1., 0.]], cluster_std=[[.4, .1], [.
         plt.scatter(X_blobs[y_blobs==0][:,0], X_blobs[y_blobs==0][:,1], label="Class 0")
         plt.scatter(X_blobs[y_blobs==1][:,0], X_blobs[y_blobs==1][:,1], label="Class 1")
         plt.scatter([.75], [0.], color="green", marker="*", s=150, label="Test observation")
         plt.legend();
```

Answer It seems to us that the test observation would be in Class 0. LDA would almost certainly classify it as zero, while QDA might classify it as one. Although there are data in Class 1 that share the same x value as the test observation, there are none that share the same y value (or come close); however, the test observation seems to be very close to existing x and y values for Class 0, which is why I would classify it in that group.

3.2 Now let's return to the thyroid dataset. Make a table of the total variance of each class for each biomarker.

```
In [22]: data.groupby('Diagnosis').var()
```

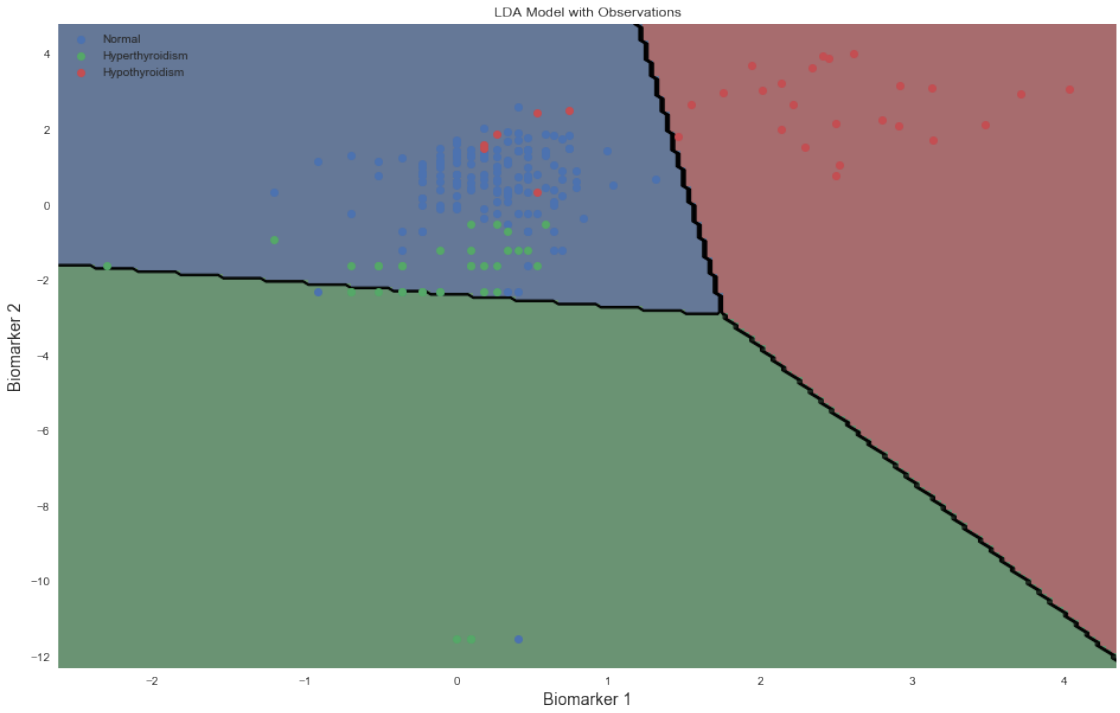
```
Out[22]:            Biomarker 1  Biomarker 2
         Diagnosis
         1             0.151141     1.932455
         2             0.308598     8.202238
         3             1.092134     0.901756
```

3.3 Fit LDA and QDA on the thyroid data, and plot the decision boundaries. Comment on how the decision boundaries differ. How does the difference in decision boundaries relate to characteristics of the data, such as the table you computed above?

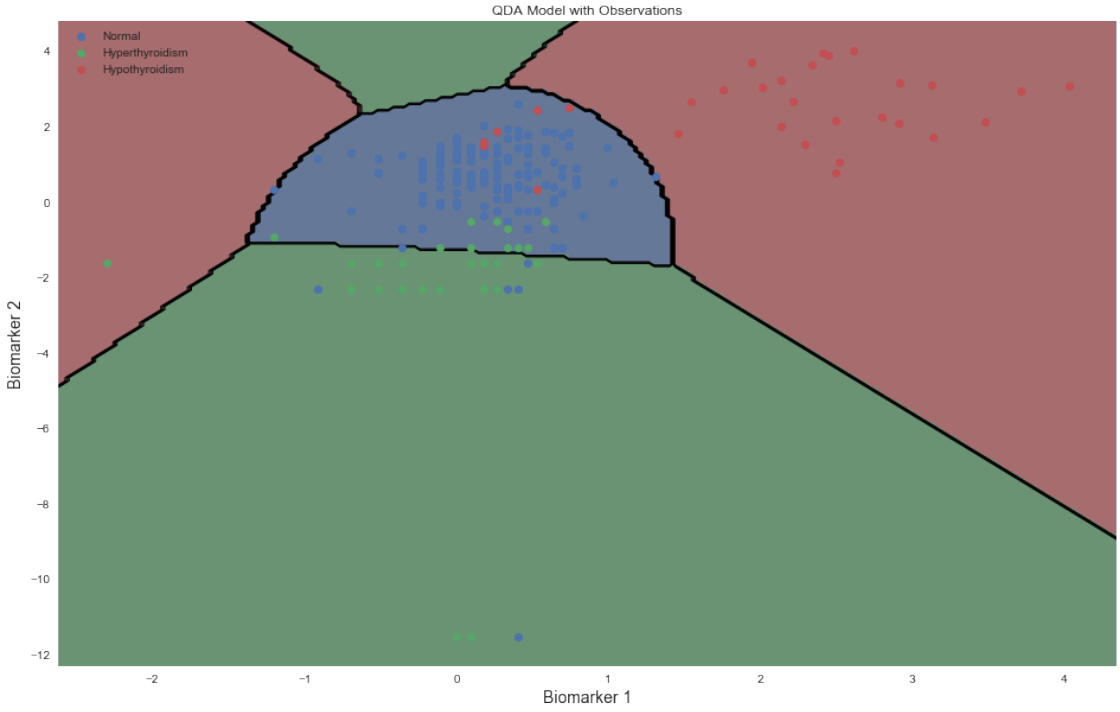
```
In [23]: lda = LinearDiscriminantAnalysis(store_covariance=True)#, priors=[1,1,1])
         qda = QuadraticDiscriminantAnalysis(store_covariance=True)#, priors=[1,1,1])

         lda.fit(X_train, y_train)

         qda.fit(X_train, y_train)

         lda.predict(X_test)

         qda.predict(X_test)


         print('LDA accuracy train={:.2%}, test: {:.2%}'.format(
             lda.score(X_train, y_train), lda.score(X_test, y_test)))
         print('QDA accuracy train={:.2%}, test: {:.2%}'.format(
             qda.score(X_train, y_train), qda.score(X_test, y_test)))
```

```
LDA accuracy train=85.05%, test: 81.48%
QDA accuracy train=87.85%, test: 85.19%
```

```
In [24]: fix, ax = scatter(data)
         ax.set_title("LDA Model with Observations")
         overlay_decision_boundary(ax, lda)
```



```
In [25]: fix, ax = scatter(data)
         ax.set_title("QDA Model with Observations")
         overlay_decision_boundary(ax, qda)
```



The decision boundaries in the LDA model are similarly sized, while the boundaries in the QDA model vary quite a bit in size. This makes sense, because LDA assumes that everything is normally distributed and that the covariance for each class for each predictor is constant. Given the table we created, we can see that the covariance is not constant, so we would assume QDA would be a better model for this data.

Question 4 [20 pts]: Fit Decision Trees

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

Hint: You should use the `DecisionTreeClassifier` class to fit a decision tree classifier and the `max_depth` attribute to set the tree depth.

4.1. Fit a decision tree model to the thyroid data set with (maximum) tree depths 2, 3, ..., 10. Make plots of the accuracy as a function of the maximum tree depth, on the training set and the mean score on the validation sets for 5-fold CV. 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 how this tree will generalize? Which hyperparameter setting gives the best cross-validation performance?

4.2: Visualize the decision boundaries of the best decision tree you just fit. How are the shapes of the decision boundaries for this model different from the other methods we have seen so far? Given an explanation for your observation.

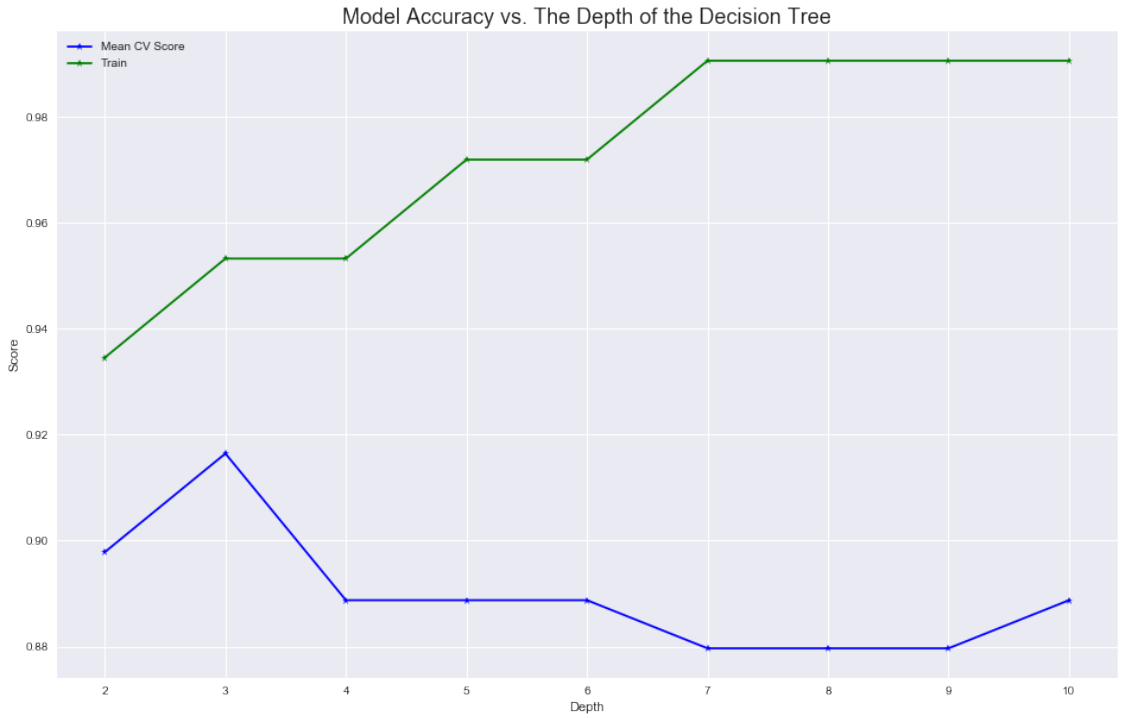
4.3 Explain *in words* how the best fitted model diagnoses 'hypothyroidism' for a new patient. You can use the code below to examine the structure of the best decision tree.

4.1

```
In [26]: for i in range(2,11):
             cls = DecisionTreeClassifier(max_depth = i)
             score = cross_val_score(cls, X_train, y_train, cv=5)
             mean = (score[0]+score[1]+score[2]+score[3]+score[4])/5
             words = "The mean is: "
             print(i, score, words, mean)
```

```
2 [0.86363636 0.81818182 0.9047619  0.95238095 0.95238095] The mean is:  0.8982683982683983
3 [0.90909091 0.86363636 0.9047619  0.95238095 0.95238095] The mean is:  0.9164502164502165
4 [0.86363636 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8796536796536796
5 [0.90909091 0.81818182 0.9047619  0.9047619  0.95238095] The mean is:  0.8978354978354979
6 [0.90909091 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8887445887445887
7 [0.86363636 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8796536796536796
8 [0.90909091 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8887445887445887
9 [0.86363636 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8796536796536796
10 [0.90909091 0.77272727 0.9047619  0.9047619  0.95238095] The mean is:  0.8887445887445887
```

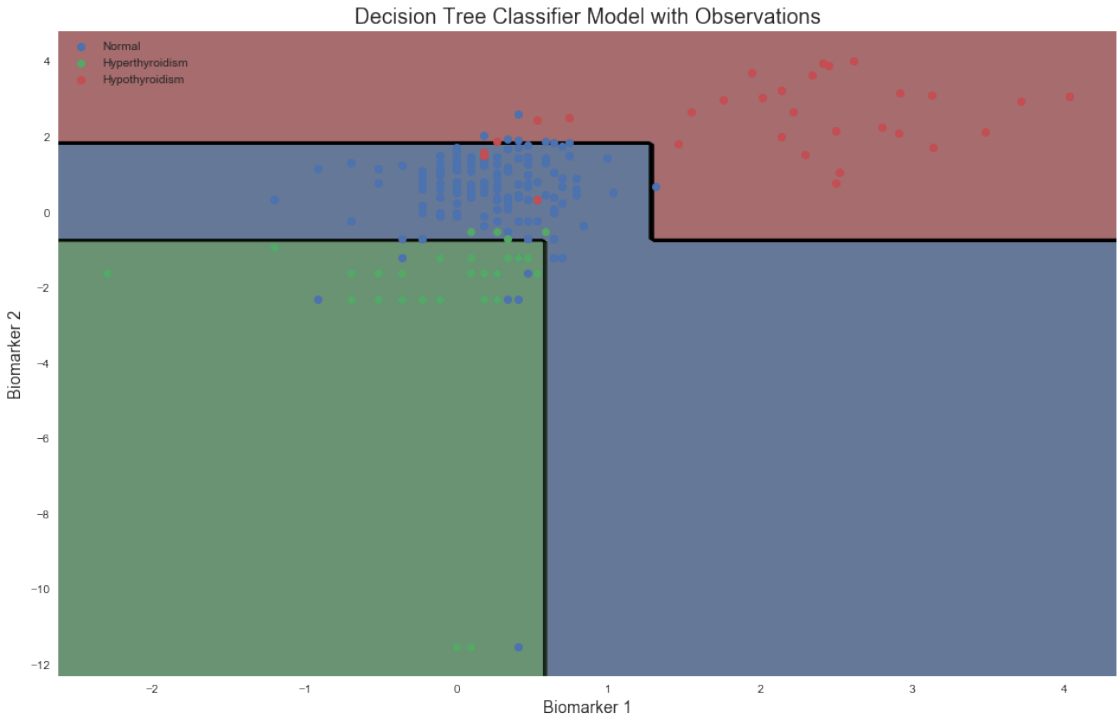
```
In [27]: depths = [2,3,4,5,6,7,8,9,10]
         scores = []
         scores_train = []
         mean_CV_score = []
         for depth in depths:
             dt = tree.DecisionTreeClassifier(max_depth = depth)
             dt.fit(X_train, y_train)
             scores.append(dt.score(X_test, y_test))
             scores_train.append(dt.score(X_train, y_train))
             mean_CV_score.append(np.mean(cross_val_score(DecisionTreeClassifier(criterion='gi


         fig, ax = plt.subplots(figsize=(16,10))
         plt.plot(depths, mean_CV_score, 'b*-', label = 'Mean CV Score')
         plt.plot(depths, scores_train, 'g*-', label = 'Train')
         plt.xlabel('Depth')
         plt.ylabel('Score')
         plt.title('Model Accuracy vs. The Depth of the Decision Tree', size = 18)
         plt.legend();
```



At classification depth of 7, the model achieves near perfect accuracy of the training set. However, given the mean CV scores, we can see that a depth of 7 is overfitting to the training set, and a max depth of 3 has the best performance for the validation set and the training set.

4.2: Visualize the decision boundaries of the best decision tree you just fit. How are the shapes of the decision boundaries for this model different from the other methods we have seen so far? Given an explanation for your observation.

```
In [28]: dt_model = DecisionTreeClassifier(criterion='gini',max_depth=3).fit(X_train, y_train)


         fix, ax = scatter(data)
         ax.set_title("Decision Tree Classifier Model with Observations", size = 18)
         overlay_decision_boundary(ax, dt_model)
```



Your answer here

These classifier boundaries are now all vertical and horizontal rather than just linear. They seem to classify the points well, and better than the LDA model; however it is unclear from eyeballing if they are better at classifying than the QDA boundaries.

4.3 Explain *in words* how the best fitted model diagnoses 'hypothyroidism' for a new patient. You can use the code below to examine the structure of the best decision tree.

Entirely optional note: You can also generate a visual representation using the `export_graphviz`. However, viewing the generated GraphViz file requires additional steps. One approach is to paste the generated graphviz file in the text box at http://www.webgraphviz.com/. Alternatively, you can run GraphViz on your own computer, but you may need to install some additional software. Refer to the Decision Tree section of the sklearn user guide for more information.

```
In [29]: # This code is adapted from
         # http://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html
         def show_tree_structure(clf):
             tree = clf.tree_

             n_nodes = tree.node_count
             children_left = tree.children_left
             children_right = tree.children_right
             feature = tree.feature
             threshold = tree.threshold

             # The tree structure can be traversed to compute various properties such
             # as the depth of each node and whether or not it is a leaf.
```

```
    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent depth
    while len(stack) > 0:
        node_id, parent_depth = stack.pop()
        node_depth[node_id] = parent_depth + 1

        # If we have a test node
        if (children_left[node_id] != children_right[node_id]):
            stack.append((children_left[node_id], parent_depth + 1))
            stack.append((children_right[node_id], parent_depth + 1))
        else:
            is_leaves[node_id] = True

    print(f"The binary tree structure has {n_nodes} nodes:\n")

    for i in range(n_nodes):
        indent = node_depth[i] * "  "
        if is_leaves[i]:
            prediction = clf.classes_[np.argmax(tree.value[i])]
            print(f"{indent}node {i}: predict class {prediction}")
        else:
            print("{}node {}: if X[:, {}] <= {:.3f} then go to node {}, else go to noc
                indent, i, feature[i], threshold[i], children_left[i], children_right
```

```
In [30]: show_tree_structure(dt_model)
```

```
The binary tree structure has 11 nodes:

node 0: if X[:, 1] <= -0.693 then go to node 1, else go to node 6
  node 1: if X[:, 0] <= 0.582 then go to node 2, else go to node 5
    node 2: if X[:, 0] <= -0.053 then go to node 3, else go to node 4
      node 3: predict class 2
      node 4: predict class 2
    node 5: predict class 1
  node 6: if X[:, 0] <= 1.270 then go to node 7, else go to node 10
    node 7: if X[:, 1] <= 1.879 then go to node 8, else go to node 9
      node 8: predict class 1
      node 9: predict class 3
    node 10: predict class 3
```

Your answer here

If Biomarker 2 is >= -0.693 and Biomarker 1 is >= 1.270, then this model will predict Hypothyroidism.

Similarly, if Biomarker 2 is >= to -0.693, Biomarker 1 is <= 1.270, and Biomarker 2 is >= 1.879, then this model will predict Hypothyroidism.

Question 5 [18 pts]: k-NN and Model comparison

We have now seen six different ways of fitting a classification model: **linear logistic regression**, **logistic regression with polynomial terms**, **LDA**, **QDA**, **decision trees**, and in this problem we'll add **k-NN**. Which of these methods should we use in practice for this problem? To answer this question, we now compare and contrast these methods.

5.1 Fit a k-NN classifier with uniform weighting to the training set. Use 5-fold CV to pick the best k.

Hint: Use `KNeighborsClassifier` *and* `cross_val_score`.

5.2 Plot the decision boundaries for each of the following models that you fit above. For models with hyperparameters, use the values you chose using cross-validation. - Logistic Regression (linear) - Logistic Regression (polynomial) - Linear Discriminant Analysis - Quadratic Discriminant Analysis - Decision Tree - k-NN

Comment on the difference in the decision boundaries between the following pairs of models. Why does this difference make sense given how the model works? - Linear logistic regression; LDA - Quadratic logistic regression; QDA. - k-NN and whichever other model has the most complex decision boundaries

5.3 Describe how each model classifies an observation from the test set in one short sentence for each (assume that the model is already fit). For example, for the linear regression classifier you critiqued in hw5, you might write: "It classifies the observation as class 1 if the dot product of the feature vector with the the model coefficients (with constant added) exceeds 0.5."

- Logistic Regression (One-vs-Rest)
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- k-Nearest-Neighbors Classifier
- Decision Tree

5.4 Estimate the validation accuracy for each of the models. Summarize your results in a graph or table. (Note: for some models you have already run these computations; it's ok to redo them here if it makes your code cleaner.)

5.5 Based on everything you've found in this question so far, which model would you expect to perform best on our test data?

Now evaluate each fitted model's performance on the test set. Also, plot the same decision boundaries as above, but now showing the test set. How did the overall performance compare with your performance estimates above? Which model actually performed best? Why do you think this is the case?

5.6. 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 - Interpretability

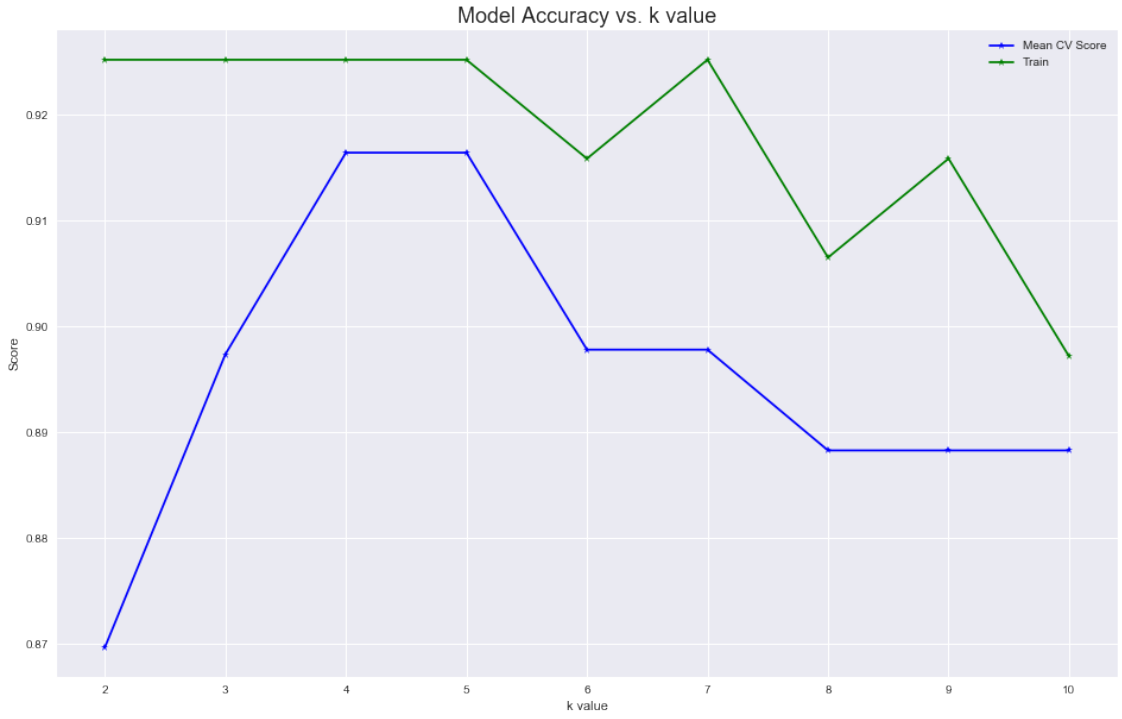
If you were a clinician who had to use the classifier to diagnose thyroid disorders in patients, which among the six methods would you be most comfortable in using? Justify your choice in terms of at least 3 different aspects.

5.1

```
In [31]: scores = []
         scores_train = []
         mean_CV_score = []
         ks=[2,3,4,5,6,7,8,9,10]
         for k in ks:
```

```
        knn = KNeighborsClassifier(k)
        knn.fit(X_train, y_train)
        scores.append(knn.score(X_test, y_test))
        scores_train.append(knn.score(X_train, y_train))
        mean_CV_score.append(np.mean(cross_val_score(KNeighborsClassifier(k),X_train, y_t
```

In [32]:
```
fig, ax = plt.subplots(figsize=(16,10))
plt.plot(ks, mean_CV_score, 'b*-', label = 'Mean CV Score')
plt.plot(ks, scores_train, 'g*-', label = 'Train')
plt.xlabel('k value')
plt.ylabel('Score')
plt.title('Model Accuracy vs. k value', size = 18)
plt.legend();
```



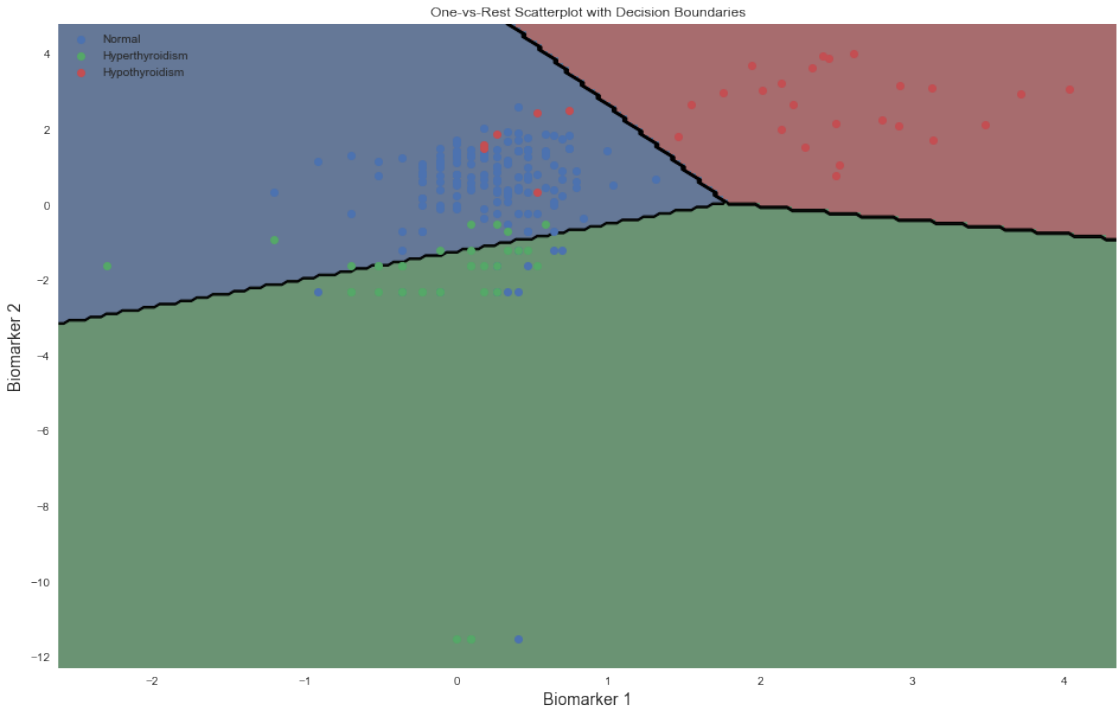
It looks like k=4 or k=5 is the best model.

In [33]:
```
knn = KNeighborsClassifier(4)
knn_model= knn.fit(X_train, y_train)
score = knn.score(X_test, y_test)
score_train = knn.score(X_train, y_train)

print("Train Score:", score_train)
print("Test Score:", score)
```

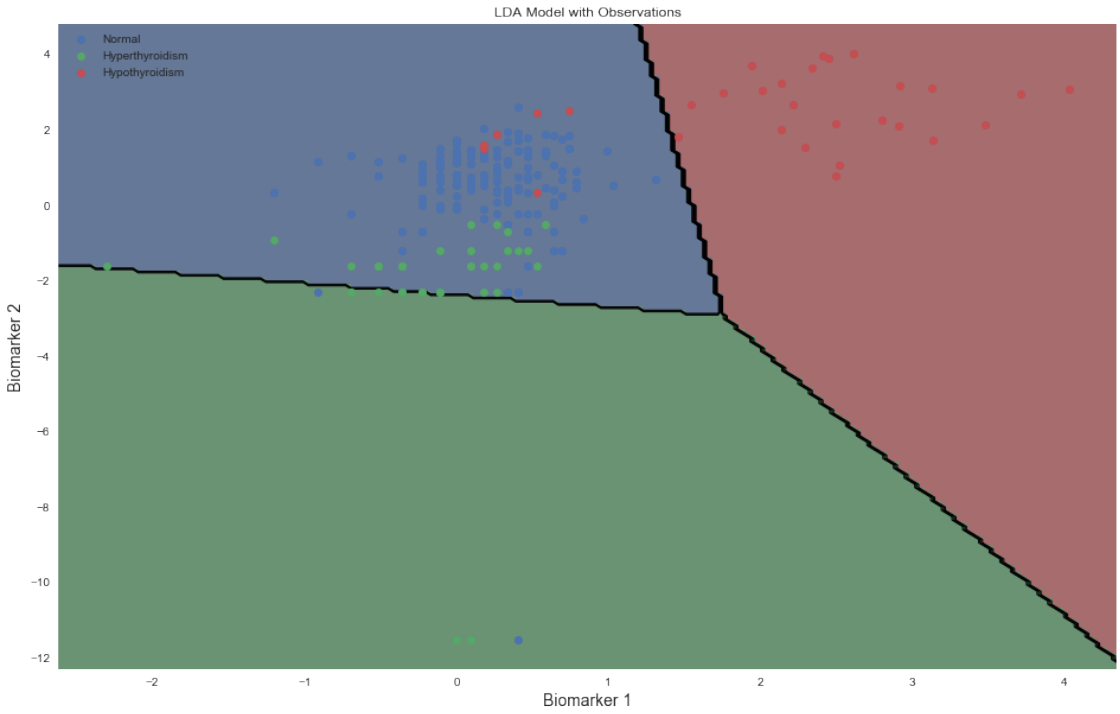
```
Train Score: 0.9252336448598131
Test Score: 0.8518518518518519
```

5.2

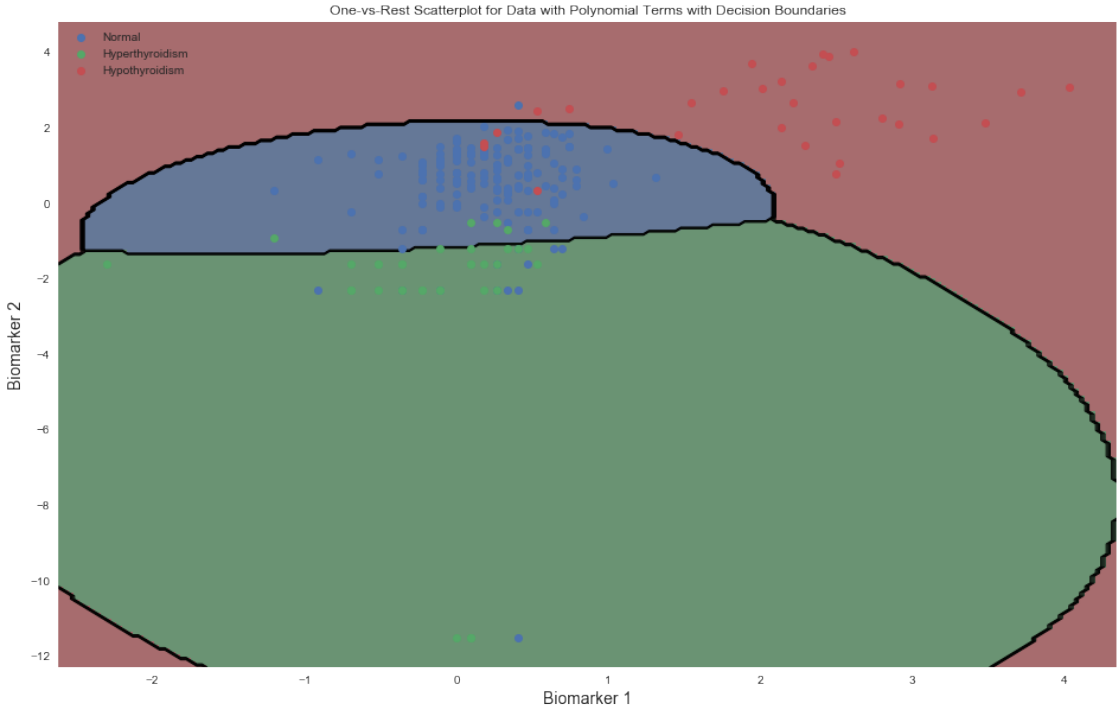
```
In [34]: fix, ax = scatter(data)
         ax.set_title("One-vs-Rest Scatterplot with Decision Boundaries")
         overlay_decision_boundary(ax, LogRegOvR)
```



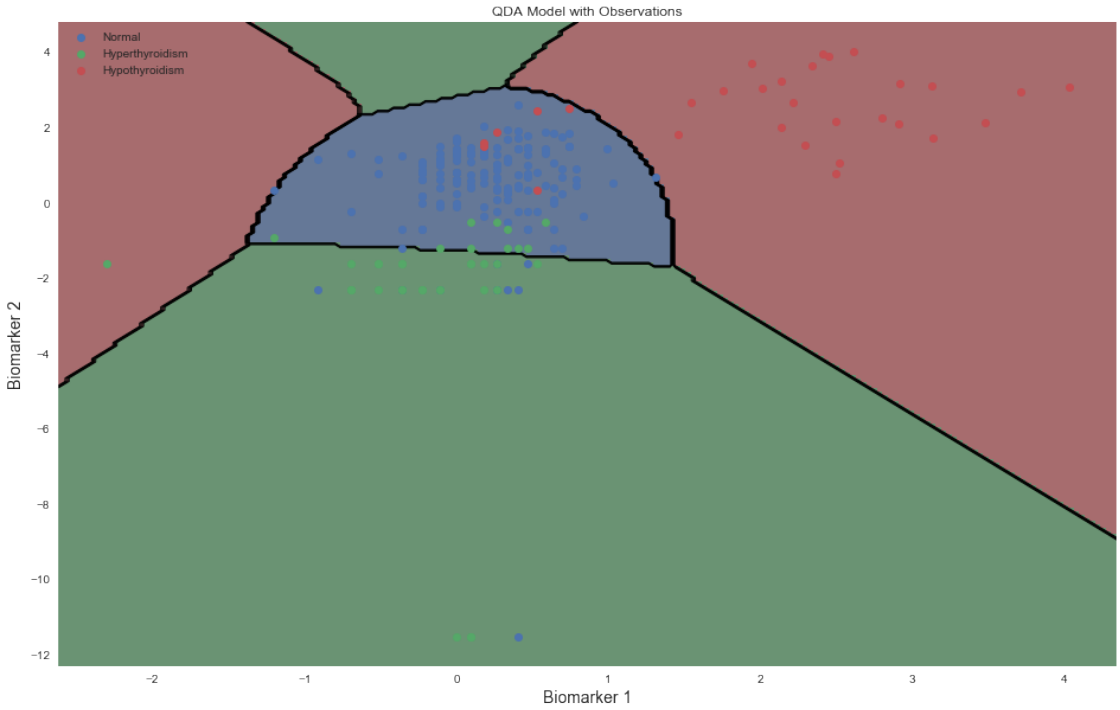

```
In [35]: fix, ax = scatter(data)
         ax.set_title("LDA Model with Observations")
         overlay_decision_boundary(ax, lda)
```



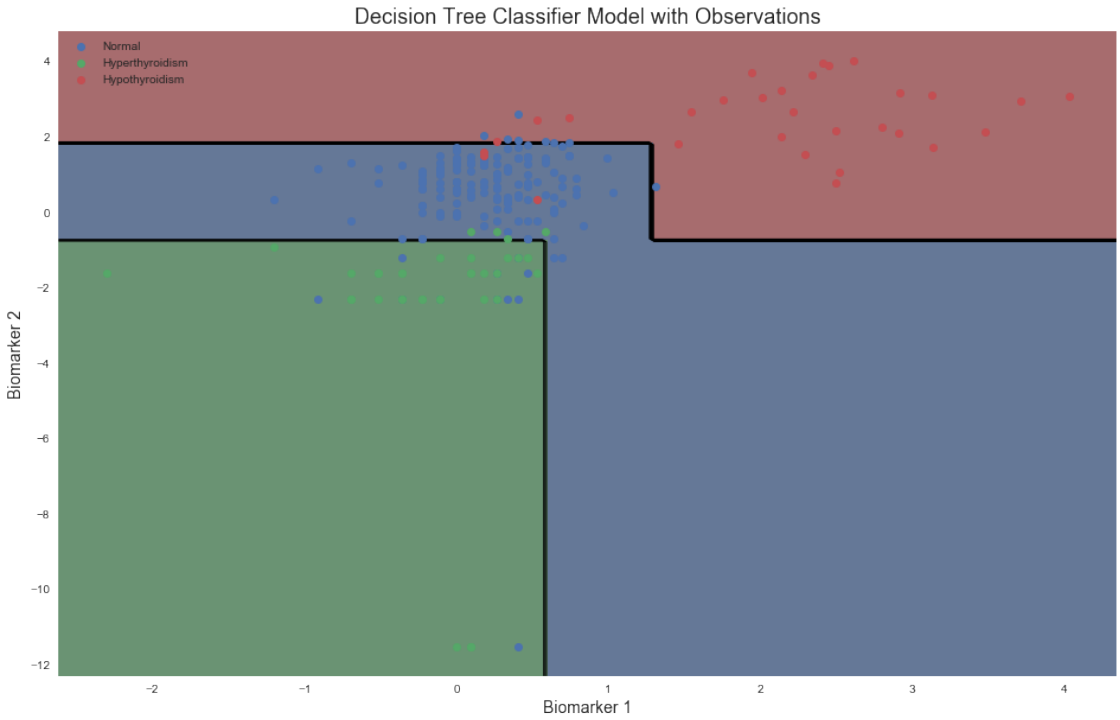
In [36]:
```
fix, ax = scatter(data)
ax.set_title("One-vs-Rest Scatterplot for Data with Polynomial Terms with Decision Bou
overlay_decision_boundary(ax, LogRegOvR_poly)
```



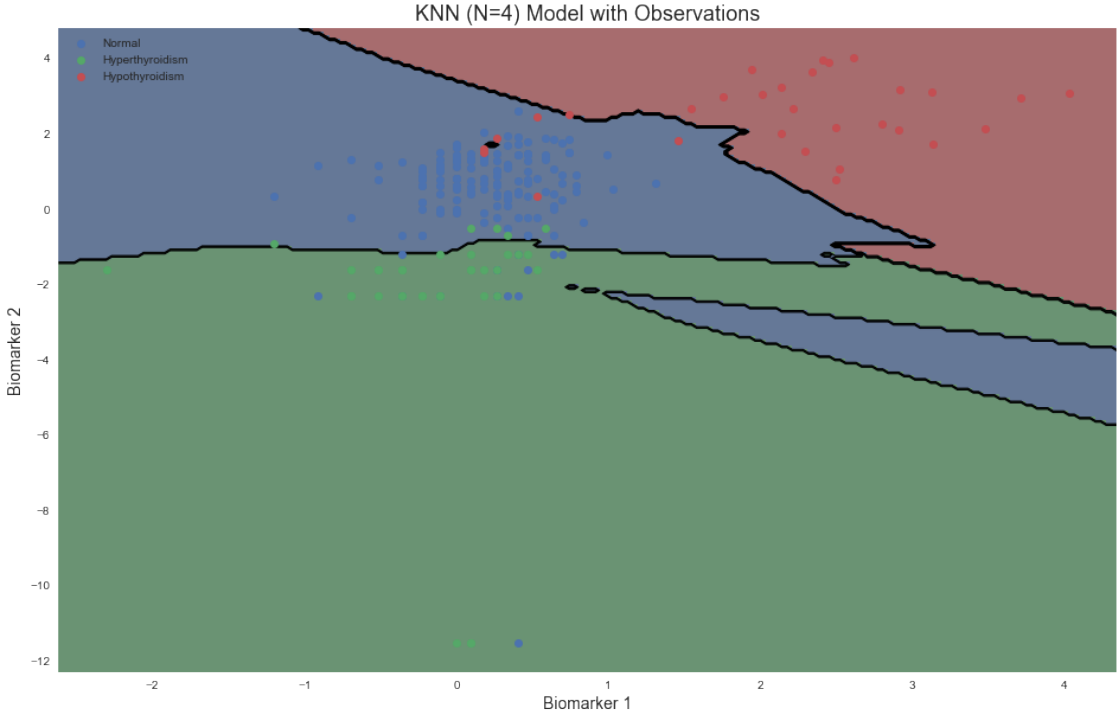
In [37]:
```
fix, ax = scatter(data)
ax.set_title("QDA Model with Observations")
overlay_decision_boundary(ax, qda)
```



In [38]:
```
fix, ax = scatter(data)
ax.set_title("Decision Tree Classifier Model with Observations", size = 18)
overlay_decision_boundary(ax, dt_model)
```



In [39]:
```
# Your code here
fix, ax = scatter(data)
ax.set_title("KNN (N=4) Model with Observations", size = 18)
overlay_decision_boundary(ax, knn_model)
```



Linear Logistic v. LDA: The polynomial logistic regression and the LDA have very similar decision boundaries. Overall, they appear to provide the same accuracy predicting normal thyroid vs. hypothyroid instances. However, the logistic provides decision boundaries that appear slightly more accurate when preducting hyer vs. hypo. Both provide only linear boundaries, which makes sense. Both regressions are linear, they just use slightly different methods to calculate the linear boundaries.

Polynomial Logistic v. QDA: The polynomial logistic appears to have two large ovals as decision boundaries containing normal and hyper, while the extra space outisde those ovals is reserved for instances of hypo. QDA has one "oval" for normal outcomes, then two areas where hyper and hypo are bound.

KNN vs. QDA: Because of how the KNN regression works in selected nearest neighbors, it's boundaries do not appear to be smooth or to follow a logical pattern. The regression with the next most complicated boundaries, the QDA, has a very smooth collection/flow of boundaries.

5.3

Logistic Regression (One-vs-Rest): Logistic regression classifies the observation as class 1 if the dot product of the feature vector with the the model coefficients (with constant added) exceeds a certain value. One-vs-rest is used when there are >2 classification outcomes, and it prepares a classifier for each possible pair (here, 1 v. 2, 2 v. 3, 1 v. 3).

Linear Discriminant Analysis: LDA minimizes the disperion between samples of the same class and maximizes the dispersion between samples of different classes. It passes in the feature vector, then gives the probability of the reponse category given the value of the feature vector's dot product with the model coefficients.

Quadratic Discriminant Analysis: Similar to LDA, but with quadratic terms. The probability is calculated under the assumption that each class has its own covariance matrix, resulting in more specific probabilities.

k-Nearest-Neighbors Classifier: Takes a deature vector, calculates the dot product with model coefficients, and determines where to place the value on the kNN line.

Decision Tree: Takes the feature vector and applies decision nodes -- if the vector as a certain feature, it progresses to the next node. There is one feature per node, and the vector works its way down the tree until it reaches the most appropriate endpoint.

5.4

```
In [40]: # your code here
         index = ["Baseline","Linear Log Reg", "Poly Log Reg", "LDA", "QDA",'Decision Tree', "
         scores = [baseline_accuracy,LogRegOvR.score(X_train, y_train), LogRegOvR_poly.score(X_
                 lda.score(X_train, y_train), qda.score(X_train, y_train),
                 dt_model.score(X_train, y_train), knn.score(X_train, y_train)]
         columns = ['Score']
         score_df =  pd.DataFrame(data = scores, columns = columns, index = index)
         score_df
```

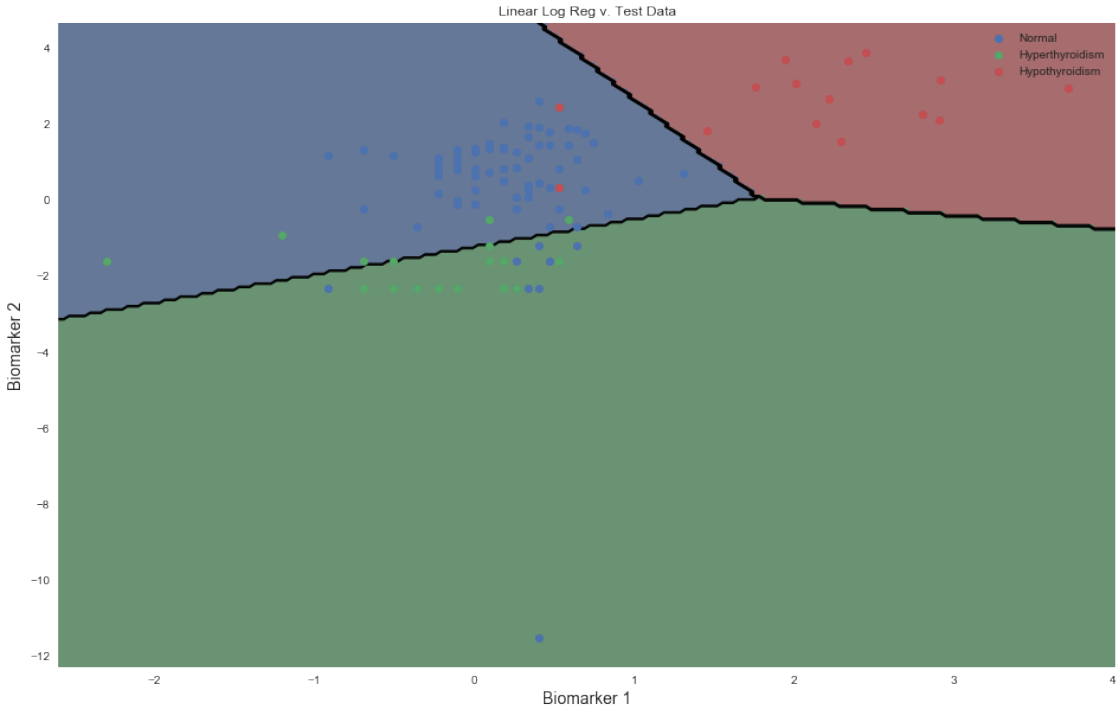
```
Out[40]:                    Score
         Baseline        0.694444
         Linear Log Reg  0.915888
         Poly Log Reg    0.934579
         LDA             0.850467
```

```
QDA              0.878505
Decision Tree    0.953271
kNN              0.925234
```

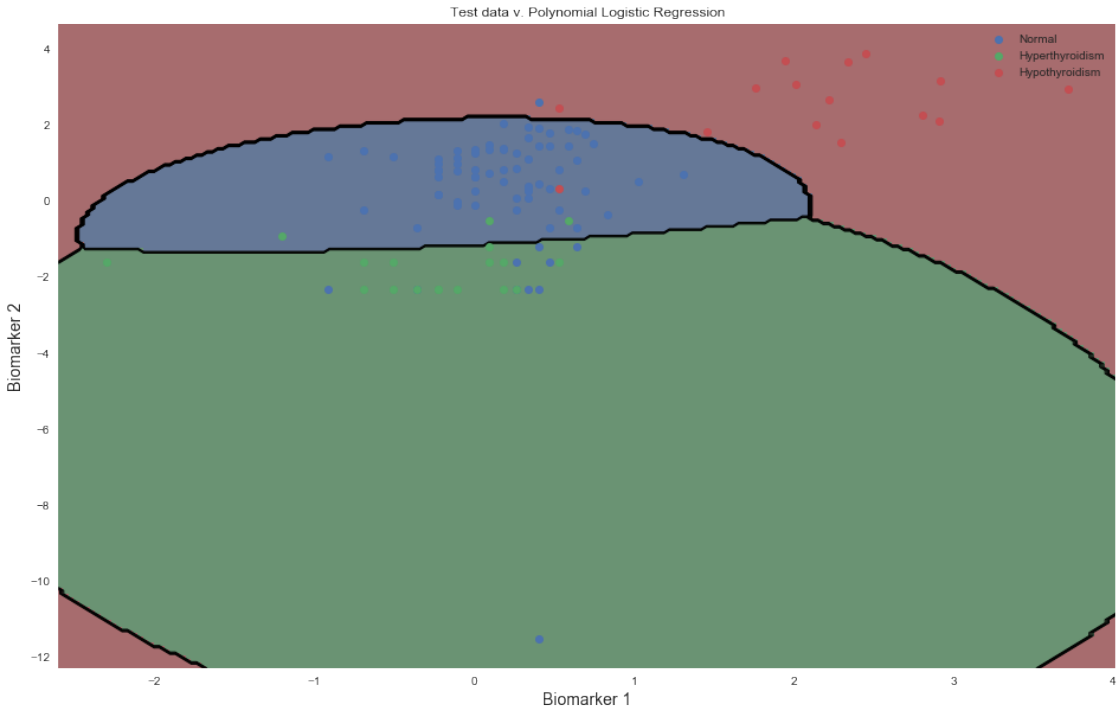
Answer: We would expect the Decision Tree to perform best based on the accuracy score.

5.5

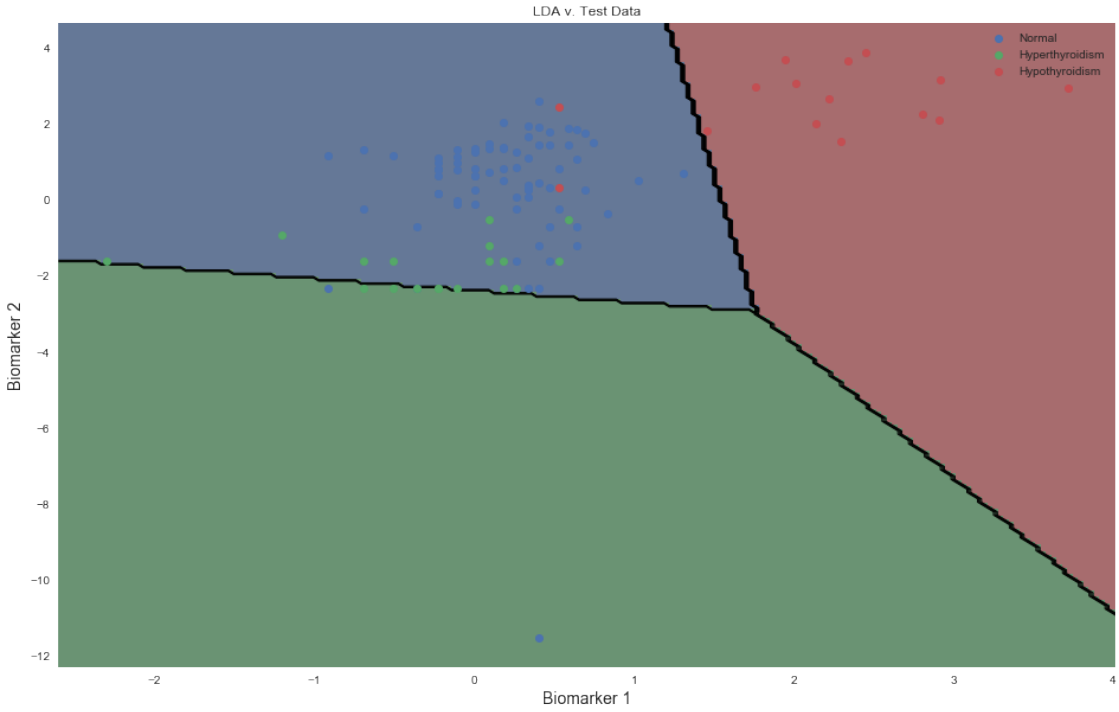
```
In [41]: fix, ax = scatter(data_test)
         ax.set_title("Linear Log Reg v. Test Data")
         overlay_decision_boundary(ax, LogRegOvR)
```



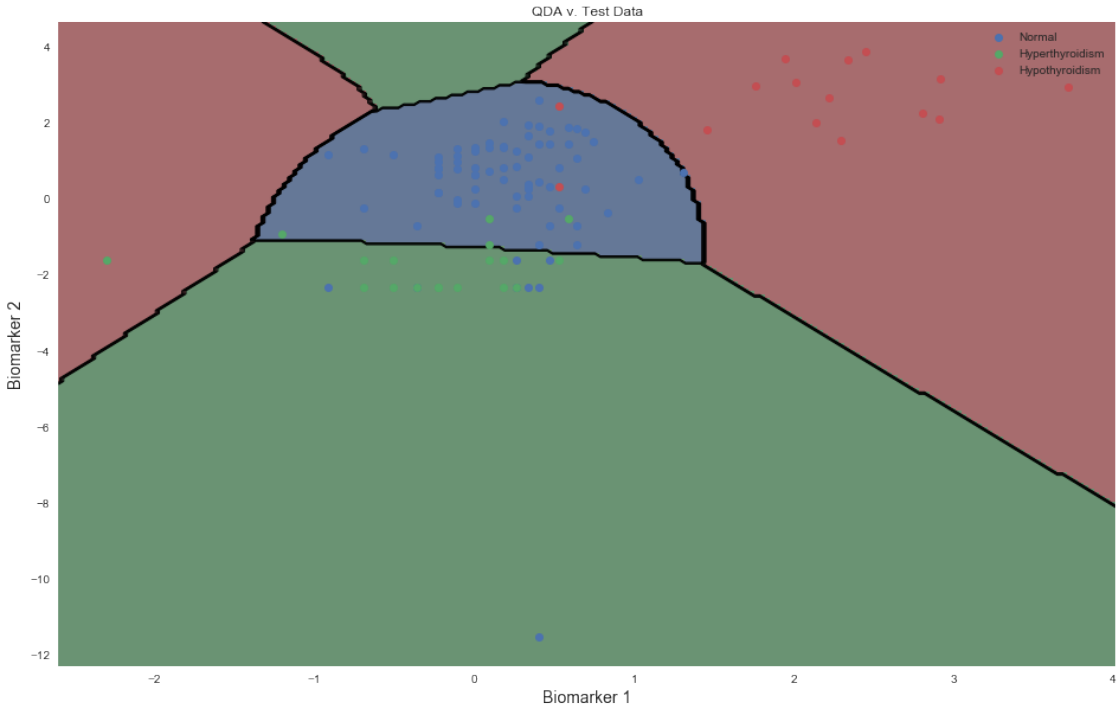

```
In [42]: fix, ax = scatter(data_test)
         ax.set_title("Test data v. Polynomial Logistic Regression")
         overlay_decision_boundary(ax, LogRegOvR_poly)
```



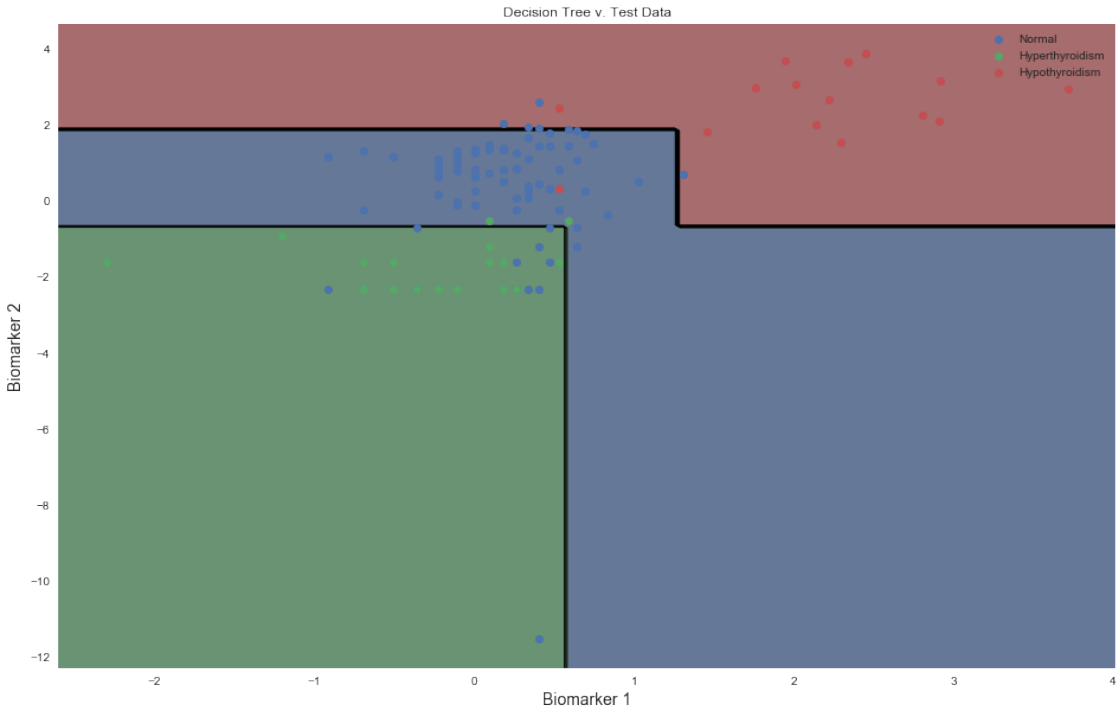

```
In [43]: fix, ax = scatter(data_test)
         ax.set_title("LDA v. Test Data")
         overlay_decision_boundary(ax, lda)
```



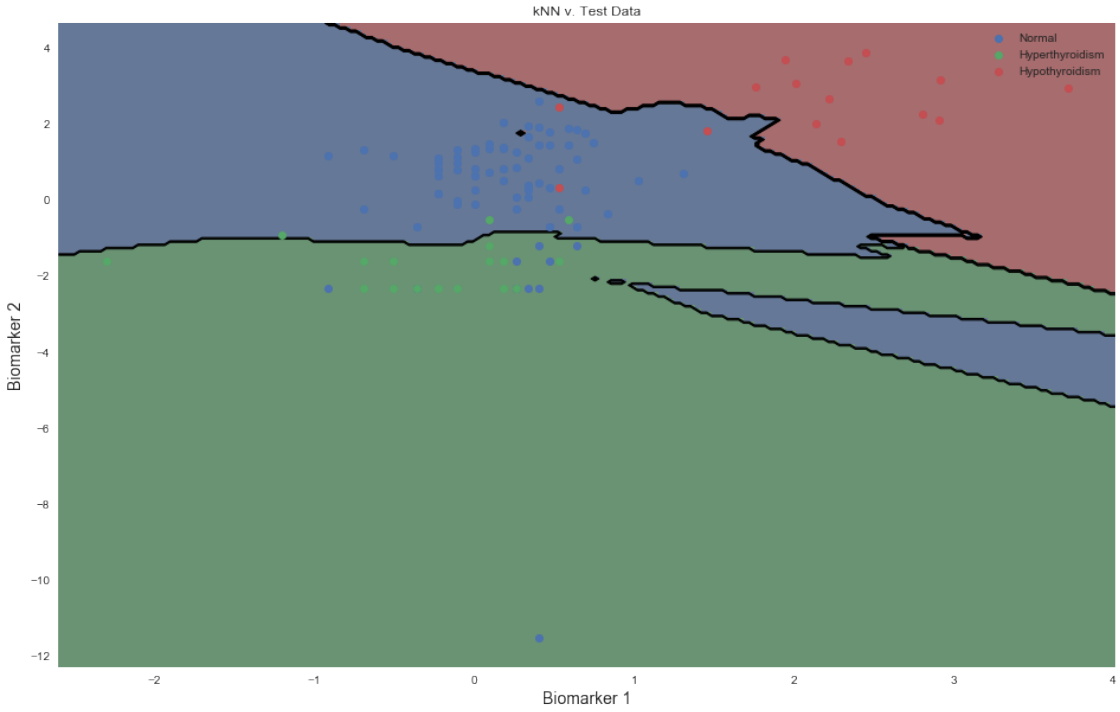
In [44]:
```
fix, ax = scatter(data_test)
ax.set_title("QDA v. Test Data")
overlay_decision_boundary(ax, qda)
```



In [45]:
```
fix, ax = scatter(data_test)
ax.set_title("Decision Tree v. Test Data")
overlay_decision_boundary(ax, dt_model)
```



In [46]:
```
fix, ax = scatter(data_test)
ax.set_title("kNN v. Test Data")
overlay_decision_boundary(ax, knn)
```



```
In [47]: # your code here
         index = ["Baseline","Linear Log Reg", "Poly Log Reg", "LDA", "QDA",'Decision Tree', "
         scores = [baseline_accuracy,LogRegOvR.score(X_test, y_test), LogRegOvR_poly.score(X_te
                 lda.score(X_test, y_test), qda.score(X_test, y_test),
                 dt_model.score(X_test, y_test), knn.score(X_test, y_test)]
         columns = ['Test Score']
         test_score_df =  pd.DataFrame(data = scores, columns = columns, index = index)
         dfs= (score_df, test_score_df)
         final_score_df = pd.concat(dfs, axis=1, sort = False)
         final_score_df
```

Out[47]:

	Score	Test Score
Baseline	0.694444	0.694444
Linear Log Reg	0.915888	0.842593
Poly Log Reg	0.934579	0.861111
LDA	0.850467	0.814815
QDA	0.878505	0.851852
Decision Tree	0.953271	0.833333
kNN	0.925234	0.851852

Answer: We expected the decision tree to perform best, but the Polynomial Logistic Regression actually performed best on the test set. Every single model's score dropped from the training set values when running the test set. Notably, the linear log reg, LDA, and Decision Tree scores all fell below the .85 threshold. We believe that the Polynomial Logistic Regression performed well because of it's polynomial and interaction terms. With only 215 observations, the data set was relatively small. Using interactions terms enhanced the predictive power, and because it only increased the predictors from 2 to 5, overfitting was not an issue.

5.6. 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 - Interpretability

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? Justify your choice in terms of at least 3 different aspects.

```
In [48]: import sys
         # sys.getsizeof(LogRegOvR)
         models = [LogRegOvR, LogRegOvR_poly, lda, qda, dt_model,knn]
         size = []
         for item in models:
             s = sys.getsizeof(item)
             size.append(s)
         score = []
         for item in models:
             sc = item.score(X_test, y_test)
             score.append(sc)
         rows = ['LogRegOvR', 'LogRegOvR_poly', 'lda', 'qda', 'dt_model','knn']
```

```
data = [score, size]

df = pd.DataFrame(data, index = ("Performance", "Memory Storage"), columns = rows)
df
```

Out[48]:

	LogRegOvR	LogRegOvR_poly	lda	qda	dt_model	knn
Performance	0.842593	0.861111	0.814815	0.851852	0.833333	0.851852
Memory Storage	56.000000	56.000000	56.000000	56.000000	56.000000	56.000000

Answer: Based on accuracy alone, we would not consider the linear logistic, LDA, and decision tree as their classification performace returns values below 0.85 (an implied goal in this pset). All six models return the same memory storage value (we did confirm this outside our loop!), so that's not a factor. While knn performs well, the decision boundaries are complex and hard to interpret. This leaves us with the polynomial logistic regression and the QDA model. Both have a similar level of interpretability and complexity, so we would select the polynomial logistic regression based on classification performace.

Question 6: [2 pts] Including an 'abstain' option

Note this question is only worth 2 pts.

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 could one design a thyroid classification model with an abstain option, such that the cost to the hospital is minimized?

Hint: Think of ways to build on top of the logistic regression model and have it abstain on patients who are difficult to classify.

6.1 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 (without quadratic or interaction terms) from Question 2? Note that this needs to be evaluated on the patients in the test set.

6.2 Design a classification strategy (into the 3 groups plus the *abstain* group) that has as low cost as possible per patient (certainly lower cost per patient than the logistic regression model). Give a justification for your approach.

6.1

Your answer here

```
In [49]: # your code here
```

```
In [50]: # your code here
```

6.2

Your answer here