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):

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

import statsmodels.api as sm
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import LogisticRegression
import graphviz
import pydot
from sklearn.externals.six import StringIO
```

```
/Users/yijunshen/anaconda3/lib/python3.6/site-packages/statsmodels/comp
at/pandas.py:56: FutureWarning: The pandas.core.datetools module is dep
recated and will be removed in a future version. Please use the pandas.
tseries module instead.
  from pandas.core import datetools
```

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')
df = df.rename(columns={'Biomarker 1': 'Biomarker_1', 'Biomarker 2': 'Bi
omarker_2'})
msk = np.random.rand(len(df)) < 0.5
data_train = df[msk]
data_test = df[~msk]
```

In [3]:
```
print(data_train.shape, data_test.shape)
data_train.head()
```

```
(102, 3) (113, 3)
```

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]:
```
data_train.describe()
```

Out[4]:

	Biomarker_1	Biomarker_2	Diagnosis
count	102.000000	102.000000	102.000000
mean	0.344213	0.136095	1.392157
std	0.830087	2.201512	0.677164
min	-2.302485	-11.512925	1.000000
25%	-0.105349	-1.076236	1.000000
50%	0.222351	0.641859	1.000000
75%	0.470010	1.273894	2.000000
max	4.032469	3.970292	3.000000

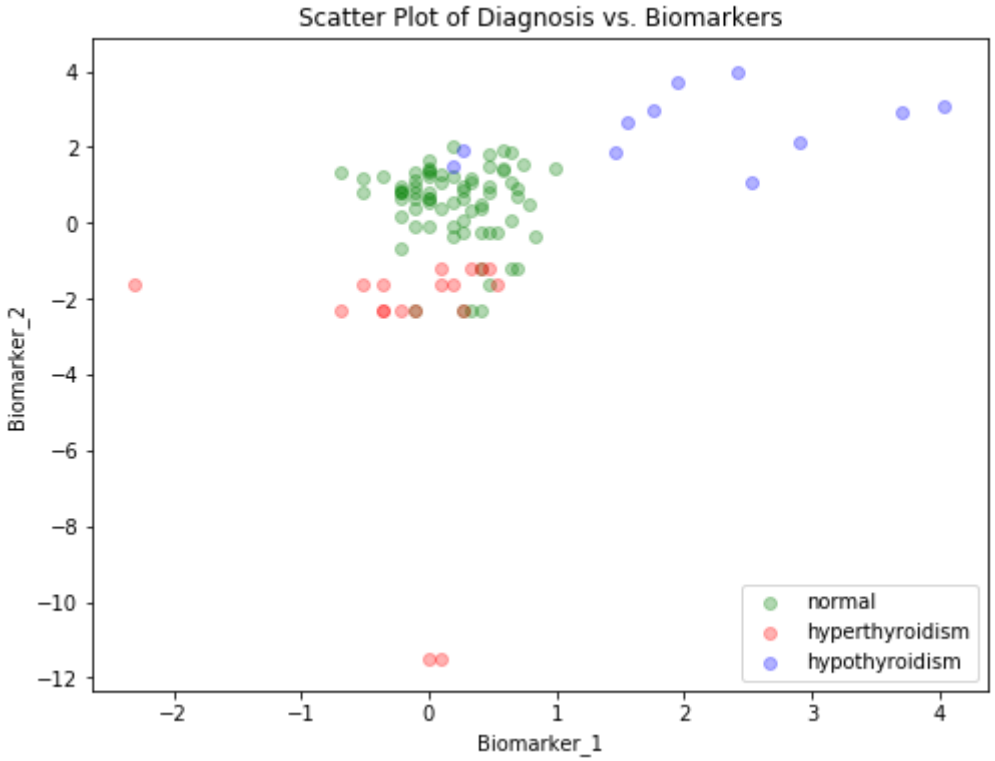
Question 1: Fit Classification Models

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:

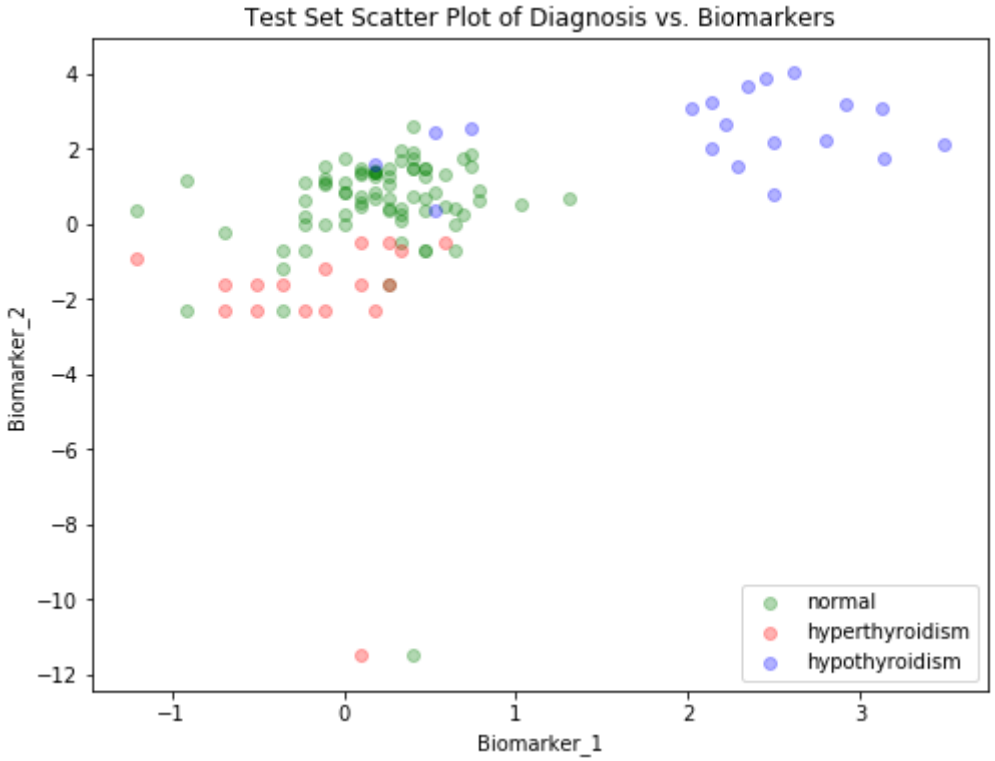
    - Multiclass Logistic Regression with quadratic terms
    - Linear Discriminant Analysis
    - Quadratic Discriminant Analysis
    - k-Nearest Neighbors
      Note: you may use either the OvR or multinomial variant for the multiclass logistic regression (with L_2 regularization). Do not forget to use 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 [5]:
```
data_train_1 = data_train.loc[data_train['Diagnosis'] == 1.0]
data_train_2 = data_train.loc[data_train['Diagnosis'] == 2.0]
data_train_3 = data_train.loc[data_train['Diagnosis'] == 3.0]

fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.scatter(data_train_1.Biomarker_1, data_train_1.Biomarker_2, label="no
rmal", color="green", alpha=0.3)
ax.scatter(data_train_2.Biomarker_1, data_train_2.Biomarker_2, label="hy
perthyroidism", color="red", alpha=0.3)
ax.scatter(data_train_3.Biomarker_1, data_train_3.Biomarker_2, label="hy
pothyroidism", color="blue", alpha=0.3)
ax.set_xlabel("Biomarker_1")
ax.set_ylabel("Biomarker_2")
ax.set_title("Scatter Plot of Diagnosis vs. Biomarkers")
ax.legend(loc=4)
plt.show()
```



In [6]:
```
data_test_1 = data_test.loc[data_test['Diagnosis'] == 1.0]
data_test_2 = data_test.loc[data_test['Diagnosis'] == 2.0]
data_test_3 = data_test.loc[data_test['Diagnosis'] == 3.0]

fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.scatter(data_test_1.Biomarker_1, data_test_1.Biomarker_2, label="norm
al", color="green", alpha=0.3)
ax.scatter(data_test_2.Biomarker_1, data_test_2.Biomarker_2, label="hype
rthyroidism", color="red", alpha=0.3)
ax.scatter(data_test_3.Biomarker_1, data_test_3.Biomarker_2, label="hypo
thyroidism", color="blue", alpha=0.3)
ax.set_xlabel("Biomarker_1")
ax.set_ylabel("Biomarker_2")
ax.set_title("Test Set Scatter Plot of Diagnosis vs. Biomarkers")
ax.legend(loc=4)
plt.show()
```



Answer:

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?

- Possibly yes. We can draw a nealy horizontal line to divide green and red dots, though this may cause some green dots classified into red ones. We can also draw a line to divide blue and green dots, with only a few blue dots misclassified into green.
- Generally speaking, the data points can be separated well by a linear clasifier.

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).

- Multinomial has the same setup as in logistic regression, and the only difference is that now the dependent variable is catogorical, which has k outcomes instead of binary outcomes.
- One-vs-rest (OvR) trains a single classfier per class, viewing the samples of that class as positive and all the rest of the samples from other classes as negative.

In [7]:
```
X_train = data_train[["Biomarker_1", "Biomarker_2"]]
y_train = data_train["Diagnosis"]
X_test = data_test[["Biomarker_1", "Biomarker_2"]]
y_test = data_test["Diagnosis"]
```

Multinomial Logistic Regression

In [8]:
```
# Multinomial Logistic Regression
X_train_mul = sm.add_constant(X_train)
X_test_mul = sm.add_constant(X_test)
reg_Cs = np.hstack((10.**np.arange(-5, 0), 10**np.arange(0, 6)))
log_multi = LogisticRegressionCV(reg_Cs, cv=5, penalty='l2', multi_class
="multinomial")
log_multi.fit(X_train_mul, y_train)
print("Multinomial Logistic regression classifier on train set has accur
acy:", log_multi.score(X_train_mul, y_train))
print("Multinomial Logistic regression classifier on test set has accura
cy:", log_multi.score(X_test_mul, y_test))
```

```
Multinomial Logistic regression classifier on train set has accuracy:
0.892156862745
Multinomial Logistic regression classifier on test set has accuracy: 0.
884955752212
```

In [9]:
```
log_multi.C_
```

Out[9]:
```
array([ 10.,  10.,  10.])
```

One-vs-rest (OvR) Logistic Regression

In [10]:
```
# One-vs-rest (OvR) Logistic Regression
log_ovr = LogisticRegressionCV(reg_Cs, cv=5, penalty='l2', multi_class=
"ovr")
log_ovr.fit(X_train_mul, y_train)
print("OVR Logistic regression classifier on train set has accuracy:", l
og_ovr.score(X_train_mul, y_train))
print("OVR Logistic regression classifier on test set has accuracy:", lo
g_ovr.score(X_test_mul, y_test))
```

```
OVR Logistic regression classifier on train set has accuracy: 0.8431372
54902
OVR Logistic regression classifier on test set has accuracy: 0.84070796
4602
```

In [11]:
```
log_ovr.C_
```

Out[11]:
```
array([  0.01,   1.  ,  10.  ])
```

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?

- One obvious rsult is that accuracy score on train set is always higher than accuracy score on test set.
- Overall classification accuracy of the multinomial method is higher than the one of the OvR method. I think this is because for the current dataset, the three classes are exclusive, and therefore multinomial logistic regression is a better choice.

Multiclass Logistic Regression with quadratic terms

In [12]:
```
# Multiclass Logistic Regression with quadratic terms
X_train2 = X_train.copy()
X_test2 = X_test.copy()
X_train2["B1_sq"] = X_train2["Biomarker_1"]**2
X_train2["B2_sq"] = X_train2["Biomarker_2"]**2
X_test2["B1_sq"] = X_test2["Biomarker_1"]**2
X_test2["B2_sq"] = X_test2["Biomarker_2"]**2
# poly2 = PolynomialFeatures(2)
# X_train_mul2 = poly2.fit_transform(X_train2)
# X_test_mul2 = poly2.fit_transform(X_test2)

X_train_mul2 = sm.add_constant(X_train2)
X_test_mul2 = sm.add_constant(X_test2)
log_multi2 = LogisticRegressionCV(reg_Cs, cv=5, penalty='l2', multi_clas
s="multinomial")
log_multi2.fit(X_train_mul2, y_train)
print("Multinomial Logistic regression with queadratic terms on train se
t has accuracy:",\
      log_multi2.score(X_train_mul2, y_train))
print("Multinomial Logistic regression with queadratic terms on test set
 has accuracy:",\
      log_multi2.score(X_test_mul2, y_test))
```

```
Multinomial Logistic regression with queadratic terms on train set has
accuracy: 0.892156862745
Multinomial Logistic regression with queadratic terms on test set has a
ccuracy: 0.884955752212
```

In [13]:
```
log_multi2.C_
```

Out[13]:
```
array([ 100.,  100.,  100.])
```

Linear Discriminant Analysis

In [14]:
```
# Linear Discriminant Analysis
# cross validation
LDA = LinearDiscriminantAnalysis()
model_LDA = LDA.fit(X_train, y_train)
print("LDA on train set has accuracy:", model_LDA.score(X_train, y_train
))
print("LDA on test set has accuracy:", model_LDA.score(X_test, y_test))
```

```
LDA on train set has accuracy: 0.872549019608
LDA on test set has accuracy: 0.83185840708
```

Quadratic Discriminant Analysis

In [15]:
```
# Quadratic Discriminant Analysis
# cross validation
QDA = QuadraticDiscriminantAnalysis()
model_QDA = QDA.fit(X_train, y_train)
print("QDA on train set has accuracy:", model_QDA.score(X_train, y_train
))
print("QDA on test set has accuracy:", model_QDA.score(X_test, y_test))
```

```
QDA on train set has accuracy: 0.872549019608
QDA on test set has accuracy: 0.849557522124
```

k-Nearest Neighbors

In [16]:
```
# k-Nearest Neighbors
train_scores = []
valid_scores = []

for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9001)
.split(X_train.index):
    X_train_cv = X_train.iloc[itrain,:]
    y_train_cv = y_train.iloc[itrain]
    X_valid_cv = X_train.iloc[ivalid,:]
    y_valid_cv = y_train.iloc[ivalid]
    train_score = []
    valid_score = []

    for i in range(1,50):
        knn_i = KNeighborsClassifier(i)
        knn_i.fit(X_train_cv, y_train_cv)
        train_score.append(knn_i.score(X_train_cv, y_train_cv))
        valid_score.append(knn_i.score(X_valid_cv, y_valid_cv))
    train_scores.append(train_score)
    valid_scores.append(valid_score)

train_scores = np.array(train_scores)
print("train score with k from 1 to 50:")
print(np.mean(train_scores, axis=0))
valid_scores = np.array(valid_scores)
print("valid score with k from 1 to 50:")
print(np.mean(valid_scores, axis=0))
best_k = np.argmax(np.mean(valid_scores, axis=0)) + 1
print("best choice of k is:", best_k)
```

```
train score with k from 1 to 50:
[ 0.97548931  0.9092442   0.92890696  0.9166215   0.92152966  0.8846732
9
  0.88958145  0.87976513  0.88964167  0.88476363  0.87979524  0.8773562
2
  0.88229449  0.85284553  0.85284553  0.82351701  0.83574225  0.8161397
2
  0.82107799  0.79891599  0.79891599  0.79400783  0.79153869  0.7939777
2
  0.79644685  0.74504667  0.74504667  0.73041253  0.73041253  0.7205962
1
  0.72059621  0.71571816  0.71571816  0.71571816  0.71571816  0.7157181
6
  0.71571816  0.71571816  0.71571816  0.71571816  0.71571816  0.7157181
6
  0.71571816  0.71571816  0.71571816  0.71571816  0.71571816  0.7157181
6
  0.71571816]
valid score with k from 1 to 50:
[ 0.86238095  0.84238095  0.91095238  0.87190476  0.86142857  0.8123809
5
  0.84142857  0.83190476  0.88095238  0.80333333  0.83238095  0.8133333
3
  0.82333333  0.80333333  0.82285714  0.80285714  0.81285714  0.8028571
4
  0.81285714  0.78380952  0.78380952  0.7647619   0.7647619   0.7257142
9
  0.72571429  0.71619048  0.71619048  0.71619048  0.71619048  0.7161904
8
  0.71619048  0.71619048  0.71619048  0.71619048  0.71619048  0.7161904
8
  0.71619048  0.71619048  0.71619048  0.71619048  0.71619048  0.7161904
8
  0.71619048  0.71619048  0.71619048  0.71619048  0.71619048  0.7161904
8
  0.71619048]
best choice of k is: 3
```

In [17]:
```
knn3 = KNeighborsClassifier(3)
knn3.fit(X_train, y_train)
print("knn with k=3 on train set has accuracy:", knn3.score(X_train, y_t
rain))
print("knn with k=3 on test set has accuracy:", knn3.score(X_test, y_tes
t))
```

```
knn with k=3 on train set has accuracy: 0.93137254902
knn with k=3 on test set has accuracy: 0.867256637168
```

Answer:

4. Compare the training and test accuracies of these models with these classification methods:

	Classification performance (train set)	Classification performance (test set)
Linear Logistic Regression (multinomial)	0.892	0.885
Linear Logistic Regression (OvR)	0.843	0.841
Logistic Regression with Polynomial Terms	0.892	0.885
LDA	0.873	0.832
QDA	0.873	0.850
k-NN with k=3	0.931	0.867

- For the training set, we can see that all models have relatively high accuracy score, where the linear logistic (OvR) regression has the lowest accuracy of 0.84 and the k-NN model with k=3 has the highest accuracy of 0.93.
- For the test set, the accuracy scores are a little lower than those of the training set, but still high. The highest test accuracy score is given by the multinomial logistic regression model, and the lowest is given by the LDA model. The k-NN (k=3) model, which has the highest training score, now drops the most in the test set, from train score of 0.93 to test score of 0.87.

Answer:

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

- Not really. With the inclusin of the polynomial terms in logistic regression, the accuracy scores of both train and test are still the same as the model with only linear terms.

Question 2: Visualize Decision Boundaries

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

In [18]:
```
#--------  plot_decision_boundary
# A function that visualizes the data and the decision boundaries
# Input:
#      x (predictors)
#      y (labels)
```

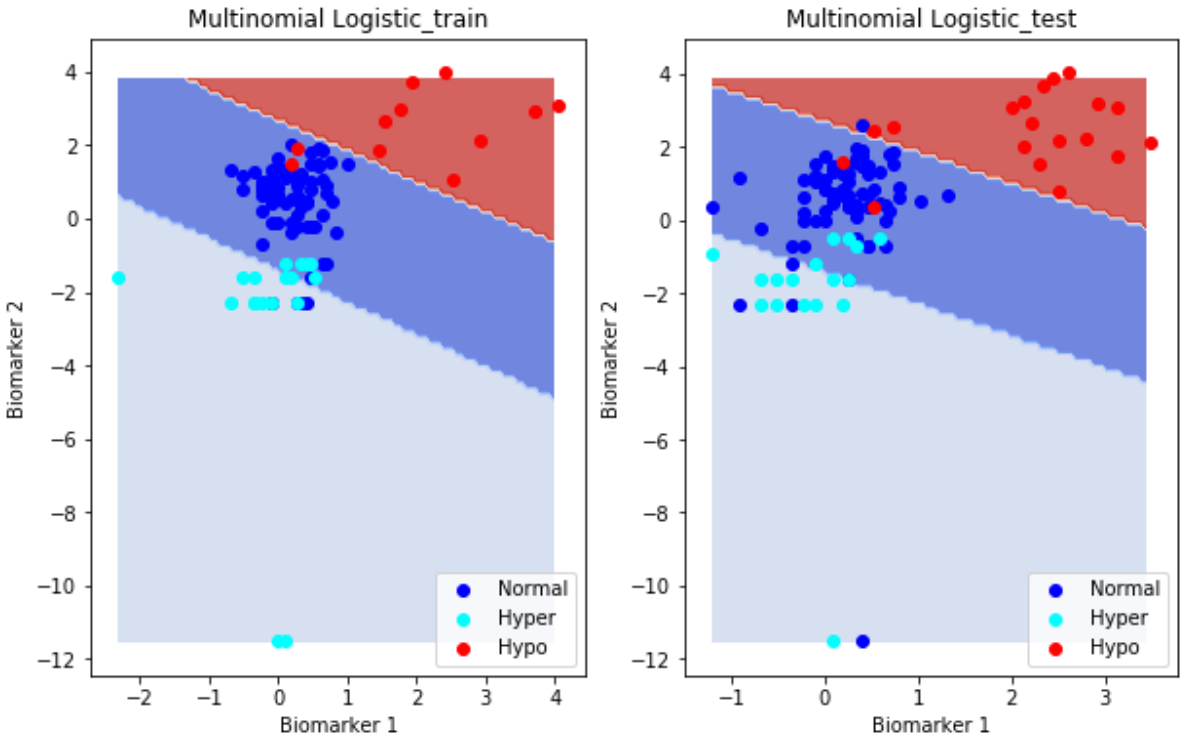
```
#      model (the classifier you want to visualize)
#      title (title for plot)
#      ax (a set of axes to plot on)
#      poly_degree (highest degree of polynomial terms included in the m
odel; None by default)

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')
    ax.legend(loc=4)
```

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.

Decision Boundaries for Multinomial Logistic Regression

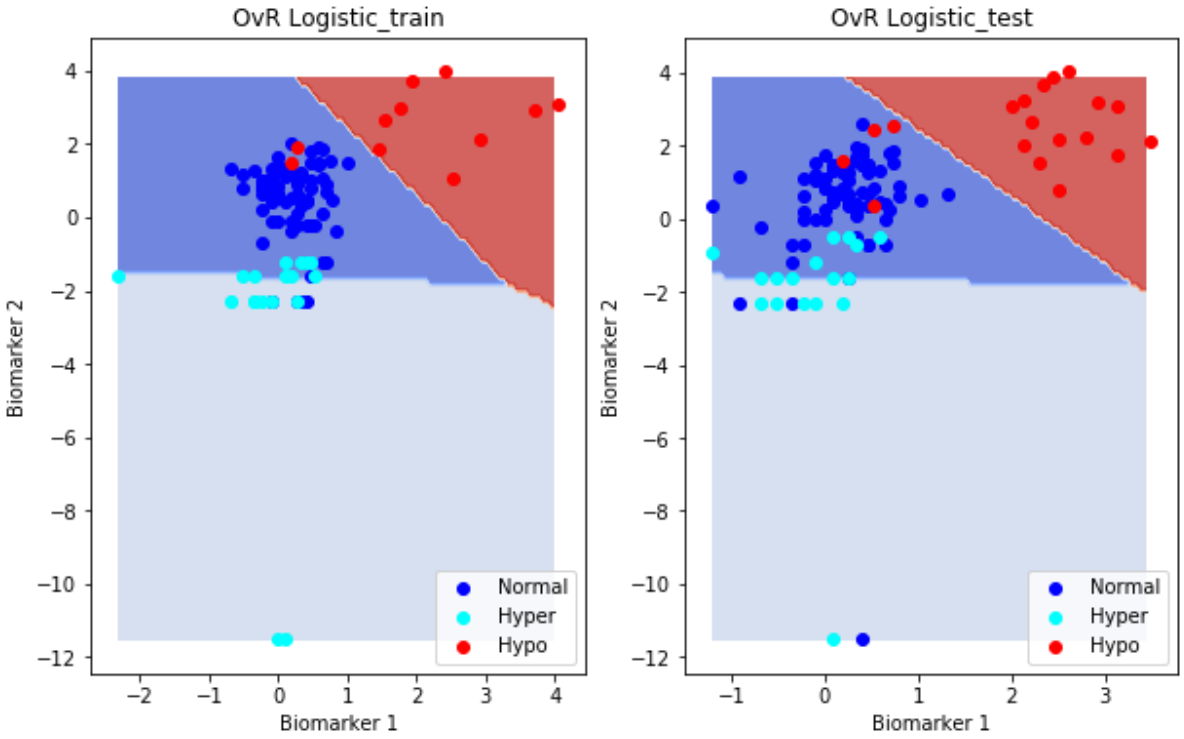
In [19]:
```
log_mul = LogisticRegression(penalty="l2", solver="lbfgs", C=10, fit_int
ercept=True, multi_class="multinomial")
log_mul.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), log_mul
, "Multinomial Logistic_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), log_mul,
"Multinomial Logistic_test", ax2)
```



Decision Boundaries for One-vs-rest (OvR) Logistic Regression

In [20]:
```
log_ovr = LogisticRegression(penalty="l2", solver="lbfgs", C=10, fit_int
ercept=True, multi_class="ovr")
log_ovr.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), log_ovr
, "OvR Logistic_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), log_ovr,
"OvR Logistic_test", ax2)
print(log_ovr.score(X_train, y_train), log_ovr.score(X_test, y_test))
```

```
0.872549019608 0.867256637168
```

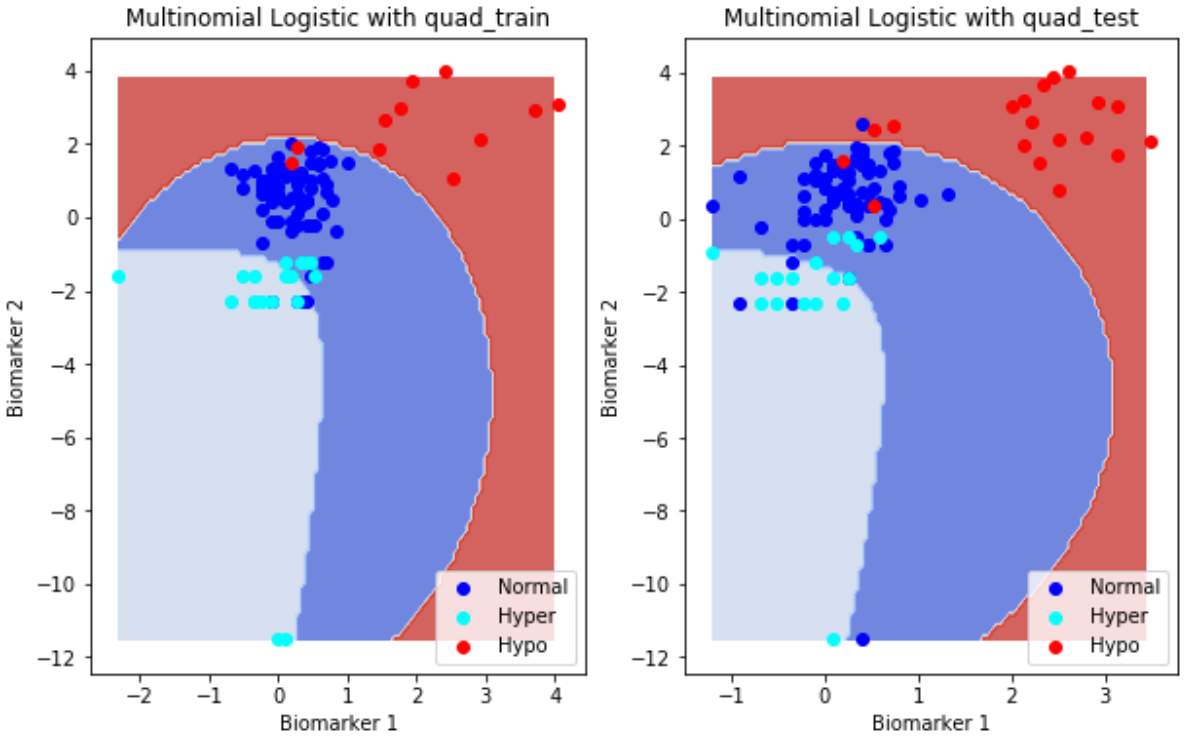


Decision Boundaries for Multiclass Logistic Regression with quadratic terms

In [21]:
```
poly = PolynomialFeatures(2)
X_train3 = poly.fit_transform(X_train)
X_test3 = poly.fit_transform(X_test)
# X_train3.shape
```

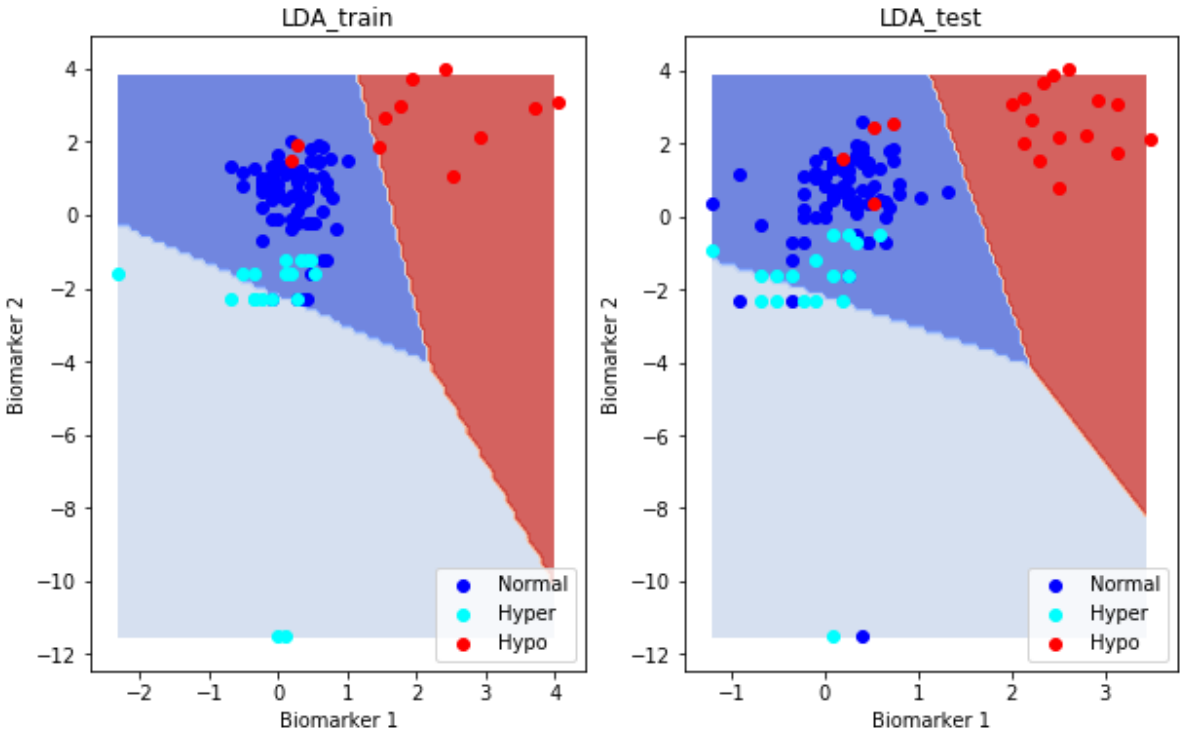
In [22]:
```
log_mul_poly = LogisticRegression(penalty="l2", solver="lbfgs", C=100, f
it_intercept=True, multi_class="multinomial")
log_mul_poly.fit(X_train3, y_train)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), log_mul
_poly, "Multinomial Logistic with quad_train", ax1, poly_degree=2)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), log_mul_p
oly, "Multinomial Logistic with quad_test", ax2, poly_degree=2)
log_mul_poly.score(X_test3, y_test)
```

Out[22]: 0.89380530973451322



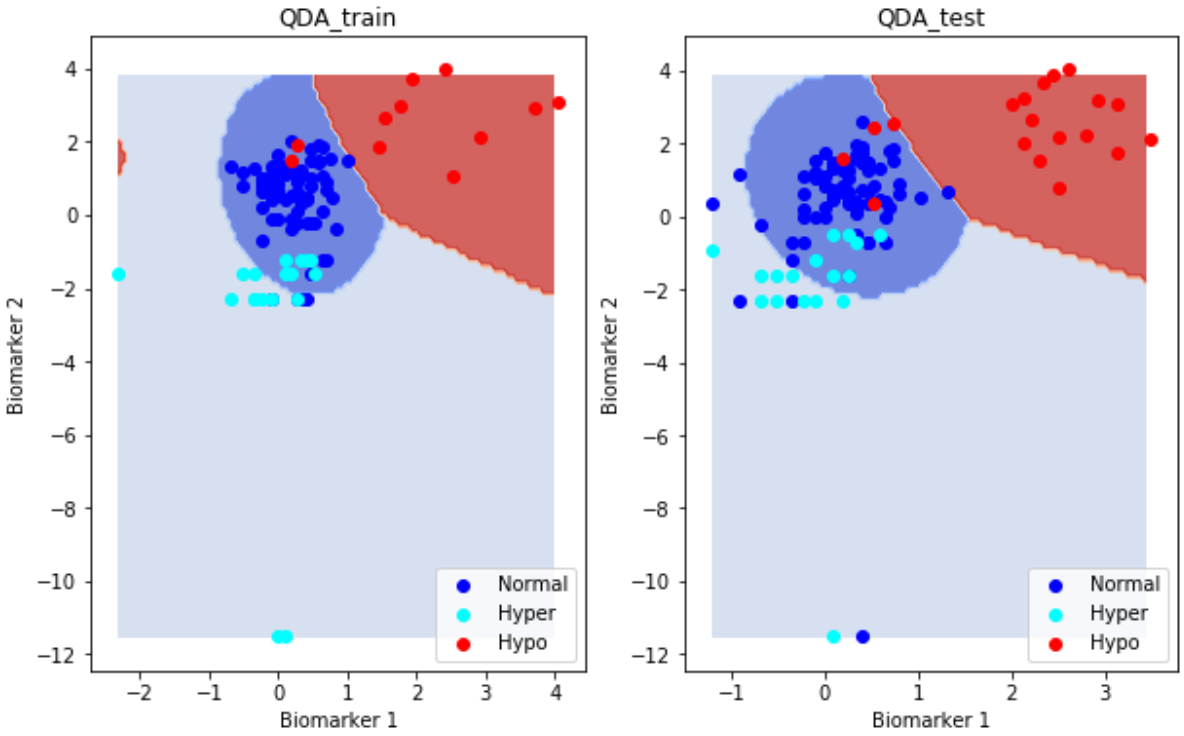
Decision Boundaries for LDA

In [23]:
```
LDA = LinearDiscriminantAnalysis()
model_LDA = LDA.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), model_LDA, "LDA_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), model_LDA, "LDA_test", ax2)
```



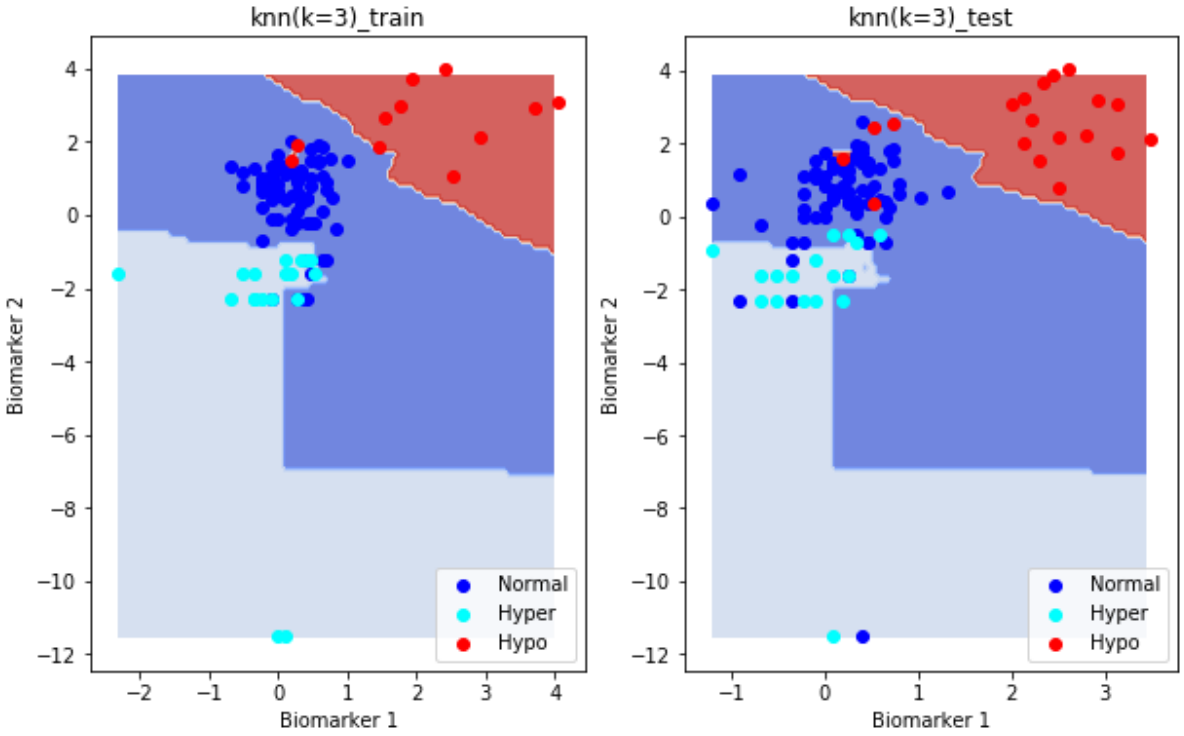
Decision Boundaries for QDA

In [24]:
```
QDA = QuadraticDiscriminantAnalysis()
model_QDA = QDA.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), model_QDA, "QDA_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), model_QDA, "QDA_test", ax2)
```



Decision Boundaries for k-NN

In [25]:
```
knn3 = KNeighborsClassifier(3)
knn3.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), knn3,
"knn(k=3)_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), knn3, "kn
n(k=3)_test", ax2)
```



Answer:

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

- OvR and multinomial logistic regression both have linear boundaries, but still look different. The multinomial boundaries are almost parallel, whereas the OvR boundaries are not. A possible explanation is because OvR fits a binary problem for each label.
- Linear logistic regression and LDA both have linear boundaries. The reason that the linear boundaries are different is because the linear coefficients are estimated differently by the two models. LDA makes more restrictive Guassian assumptions.
- The quadratic logistic regression and QDA both have decision boundaries as curves. But the boundaries of quadratic logistic divide the three classes into ring-shape areas, and boundaries of QDA divide the three classes into three clusters. An explanation for this is QDA assume the distributions of the three classes are Gaussian.

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.

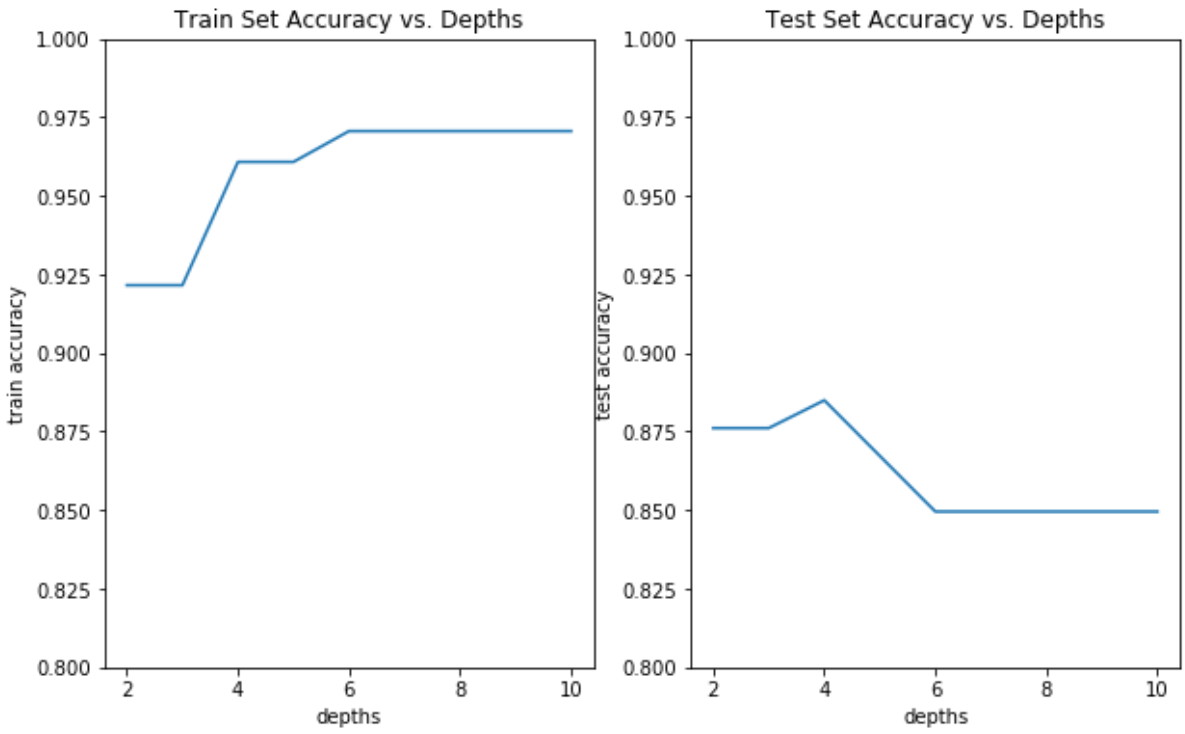
You can get a graphic for this visualization by pasting the generated graphviz file in the text box at http://www.webgraphviz.com/ (http://www.webgraphviz.com/) , or you can do it on your own computer.

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 [26]:
```
depths = range(2,11)
dt_train_scores = []
dt_test_scores = []
for depth in depths:
    dt = DecisionTreeClassifier(max_depth = depth, criterion='gini')
    dt.fit(X_train, y_train)
    dt_train_scores.append(dt.score(X_train, y_train))
    dt_test_scores.append(dt.score(X_test, y_test))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))
ax1.plot(depths, dt_train_scores)
ax1.set_xlabel("depths")
ax1.set_ylabel("train accuracy")
ax1.set_ylim(0.8,1)
ax1.set_title("Train Set Accuracy vs. Depths")
ax2.plot(depths, dt_test_scores)
ax2.set_xlabel("depths")
ax2.set_ylabel("test accuracy")
ax2.set_ylim(0.8,1)
ax2.set_title("Test Set Accuracy vs. Depths")
plt.show()
```



Answer:

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?

- Starting from maximum tree depth of 7, the fitted decision tree model achieves accuracy on the training set of 0.97, which is a near-perfect classification. This could indicate potential overfitting, and it is possible that the test accuracy of this model will get lower.
- As we can see from the second plot which is depths versus test set score, we can see that the accuracy score is not as high as before when depth gets bigger.

In [27]:
```
train_scores_mean = []
valid_scores_mean = []

for depth in depths:
    for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9
001).split(X_train.index):
        X_train_cv = X_train.iloc[itrain,:]
        y_train_cv = y_train.iloc[itrain]
        X_valid_cv = X_train.iloc[ivalid,:]
        y_valid_cv = y_train.iloc[ivalid]

        train_scores = []
        valid_scores = []
        dt = DecisionTreeClassifier(max_depth = depth, criterion='gini')
        dt.fit(X_train_cv, y_train_cv)

        train_scores.append(dt.score(X_train_cv, y_train_cv))
        valid_scores.append(dt.score(X_valid_cv, y_valid_cv))

    train_scores_mean.append(np.mean(train_scores))
    valid_scores_mean.append(np.mean(valid_scores))

print("cv train scores:", train_scores_mean)
print("cv validation scores:", valid_scores_mean)
best_depth = valid_scores_mean.index(max(valid_scores_mean)) + 2 # range
(2,11) where 2 has index 0
best_depth
```

```
cv train scores: [0.87804878048780488, 0.91463414634146345, 0.926829268
29268297, 0.95121951219512191, 0.96341463414634143, 0.9634146341463414
3, 0.96341463414634143, 0.96341463414634143, 0.96341463414634143]
cv validation scores: [1.0, 0.90000000000000002, 0.90000000000000002,
0.84999999999999998, 0.90000000000000002, 0.75, 0.75, 0.75, 0.849999999
99999998]
```

Out[27]: 2

In [28]:
```
dt2 = DecisionTreeClassifier(max_depth = 2, criterion='gini')
dt2.fit(X_train, y_train)
print("decision tree with depth=2 on train set has accuracy:", dt2.score
(X_train, y_train))
print("decision tree with depth=2 on test set has accuracy:", dt2.score(
X_test, y_test))
```

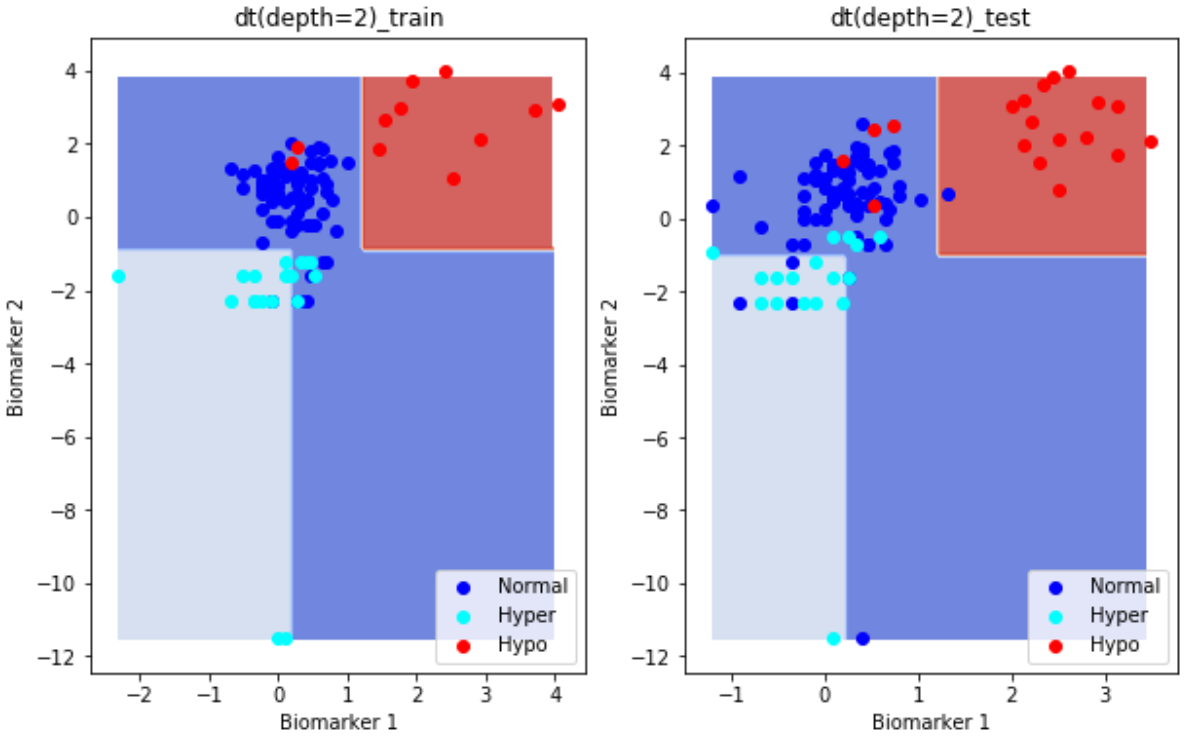
```
decision tree with depth=2 on train set has accuracy: 0.921568627451
decision tree with depth=2 on test set has accuracy: 0.87610619469
```

Answer:

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)?

- The decision tree with a depth of 2 does a good job on classification, with train accuracy of 0.92 and test accuracy of 0.87.
- Compared to the models fitted in part 2, this decision tree with depth of 2 did almost as good as k-NN with k=3. It did better on the training set than multinomial logistic regression with or without quadratic terms, but a little bit worse on the test than those two models. The decision tree model does better than all the rest of the previous models, including OvR Logistic Regression, LDA and QDA.

In [29]:
```
dt2 = DecisionTreeClassifier(max_depth = 2, criterion='gini')
dt2.fit(X_train, y_train)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
plot_decision_boundary(X_train.as_matrix(), y_train.as_matrix(), dt2, "d
t(depth=2)_train", ax1)
plot_decision_boundary(X_test.as_matrix(), y_test.as_matrix(), dt2, "dt
(depth=2)_test", ax2)
```



Answer:

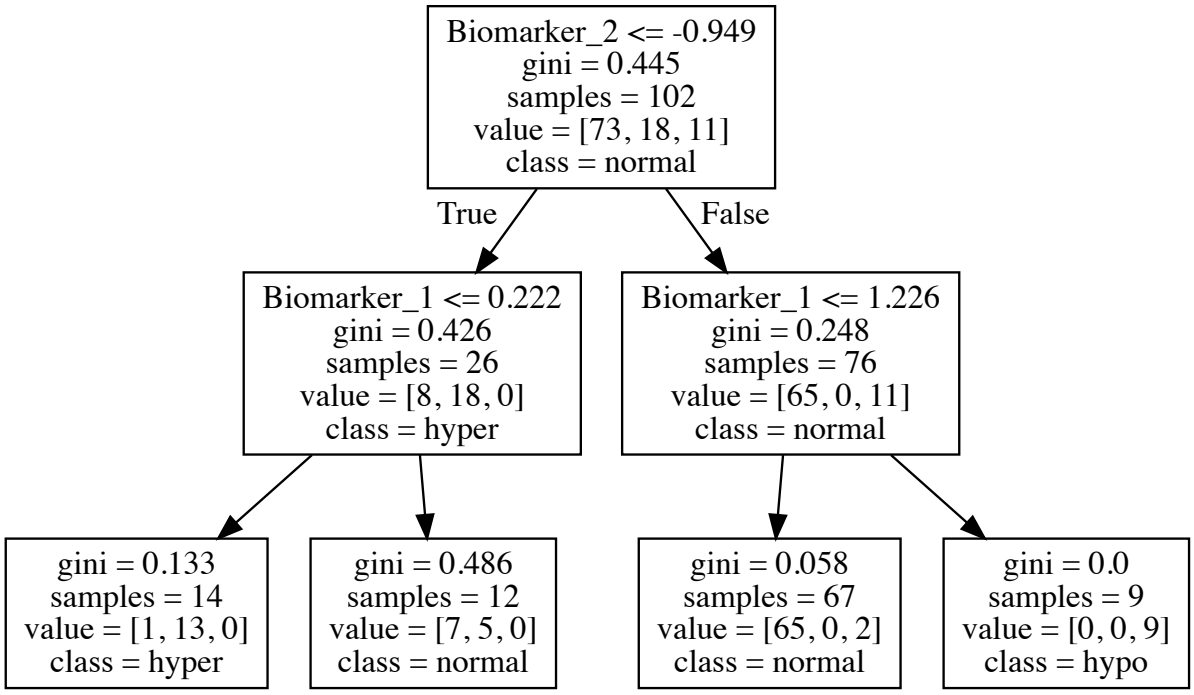
3. Use the code provided in Part 2(c) to visualize the decision boundary of the fitted decision tree. How is the decision boundary of the decision tree model different from the other methods? Given an explanation for your observation.

- The decision boundary of the decision tree model has straight lines, either purely horizontal or purely vertical, instead of inclined lines or curves as in the previous models.
- This is because a decision tree model compares the values of one predicotr against a threshold value each time, which gives a binary outcome of true or false each time. Because each time it only looks at one predictor, for the current dataset with two predictors, the dicision boundary can only be horizontal or vertical lines.

In [30]:
```
export_graphviz(dt2, out_file="tree.dot",\
                feature_names=["Biomarker_1", "Biomarker_2"], class_name
s=["normal", "hyper", "hypo"])
```

In [31]:
```
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[31]:



Answer:

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.

- The model first checks whether Biomarker_2 is less or equal to -0.949.
- If the above criterion is false, which means Biomarker_2 > -0.949, the model will then check whether Biomarker_1 is less or equal to 1.226.
- If the above criterion is false again, which means Biomarker_1 > 1.226, the model will classify that patient to the class of hypothyroidism.

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?

	Classification performance (test set)	**Complexity of decision boundary**	**Memory storage**	**Ease of interpretability**
Linear Logistic Regression (multinomial)	0.885	3	3	3
Linear Logistic Regression (OvR)	0.841	3	3	3
Logistic Regression with Polynomial Terms	0.885	4	3	4
LDA	0.832	3	2	4
QDA	0.850	4	2	5
k-NN with k=3	0.867	5	5	2
Decision Tree with depth=2	0.876	1	5	1

- Complexity of decision boundary: a scale from 1 to 5, where 1 means not complex and 5 means very complex. Non-linear decision boundaries are more complex than linear ones.
- Memory storage: a scale from 1 to 5, where 1 means using few memory storage and 5 means using a lot of memory storage. Parametric models usually store a set of estimated parameters, which is much smaller than the size of the dataset it self. Therefore, non-parametric models such as k-NN and decision tree will require a bigger memory storage than parametric models.
- Ease of interpretability: a scale from 1 to 5, where 1 means easy to interpret and 5 means hard to interpret. Parametric models involve a lot of estimated parameters and model assumptions, so they are generally more complex and hard to interpret than non-parametric models.

Answer:

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?

- I would recommend using a decision tree model with depth=2, because it not only gives a high accuracy score, it is also very easy to interpret since the tree can be visualiezed. Moreover, the cost of using the decision tree to predict data is logarithmic in the number of data points used to train the tree, as indicated in the scikit-learn introduction. It also gives high accuracy score, which is critical in medical prediction.
- For a medical classification problem like this one, it is quite common to use decision tree in real life. A medical report such as a physical exam report usually lists every indicator and compares them to a threshold, which is the basic idea of the decision tree. So using the decision tree in this multiclass thyroid classification, we compare two biomarkers to threshold values, which I think is very easy to understand both for doctors and patients.

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 [32]:
```
log_mn = LogisticRegression(penalty="l2", solver="lbfgs", C=10, fit_inte
rcept=True, multi_class="multinomial")
log_mn.fit(X_train, y_train)
max_proba = []
for i in range(X_test.shape[0]):
    max_proba.append(max(log_mn.predict_proba(X_test)[i]))
unsure = []
for index, m in enumerate(max_proba):
    if m < 0.6:
        unsure.append(index)

prediction = log_mn.predict(X_test)
unsure
```

Out[32]:
```
[29, 59, 77, 90, 99]
```

In [33]:
```
prediction_df = data_test.copy()
prediction_df = prediction_df.reset_index(drop=True)
prediction_df["predition"] = prediction
prediction_df["probability"] = max_proba
prediction_df["mistake"] = prediction_df["Diagnosis"] - prediction_df["p
redition"]
prediction_df["abstain"] = 0
for n in unsure:
    prediction_df.set_value(n,'abstain',1.0)
# prediction_df
```

In [34]:
```
mistakes = (prediction_df['mistake'] != 0).sum()
print("number of total misclassifications:", mistakes)
abstains = (prediction_df['abstain'] != 0).sum()
print("number of total abstains:", abstains)
prediction_df["success_abstain"] = prediction_df['mistake'] * prediction
_df['abstain']
success_abstain = (prediction_df['success_abstain'] != 0).sum()
print("number of successful abstains:", success_abstain)
patients = mistakes + abstains - success_abstain
print("number of patients misclassified or abstained:", patients)
total_cost = 5000*(mistakes - success_abstain) + 1000*abstains
print("total cost:", total_cost)
print("average cost per patient:", total_cost / X_test.shape[0])
```

```
number of total misclassifications: 13
number of total abstains: 5
number of successful abstains: 3
number of patients misclassified or abstained: 15
total cost: 55000
average cost per patient: 486.725663717
```

the naive approach result (the OvR losigstic regression model with never abstaining)

In [35]:
```
log_ovr = LogisticRegression(penalty="l2", solver="lbfgs", C=10, fit_int
ercept=True, multi_class="ovr")
log_ovr.fit(X_train, y_train)

prediction2 = log_ovr.predict(X_test)

prediction_df2 = data_test.copy()
prediction_df2 = prediction_df2.reset_index(drop=True)
prediction_df2["predition"] = prediction2
prediction_df2["mistake"] = prediction_df2["Diagnosis"] - prediction_df2
["predition"]
mistakes2 = (prediction_df2['mistake'] != 0).sum()
print("number of total misclassifications:", mistakes2)
total_cost2 = 5000* mistakes2
print("total cost:", total_cost2)
print("average cost per patient:", total_cost2 / X_test.shape[0])
```

```
number of total misclassifications: 15
total cost: 75000
average cost per patient: 663.716814159
```

Answer:

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.

- Among all the classfication models above, we can see that multinomial logistic regression has the best test set performance. Therefore, I used multinomial logistic regression as my base model.
- My approach is that after applying the multinomial logistic regression model to the test set, I group all predictions with a classification probability less than 0.6 to the group of abstain. Applying the method to the current test set, we can see that it abstains 5 patient, where 3 of them are successful abstains from misclassification, which is above 50%.
- This abstaining method has a total cost of 55000, which is much lower than the total cost of 75000 from the naive OvR logistic regression without abstaining.