ML Bootcamp

Devika Subramanian, Rice University, (c) 2019.

Lab 0

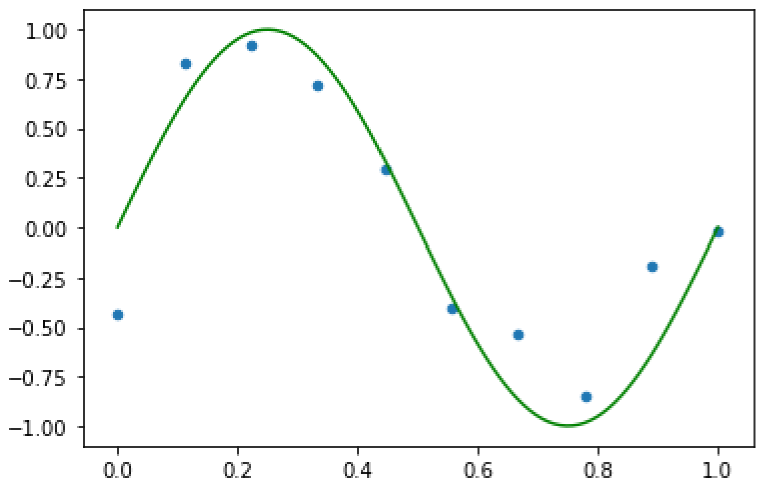
- regression example from intro slide deck
- classification example from intro slide deck

Regression example

- generate 10 points uniformly from [0,1] and calculate y = sin(2πx) + N(0,0.02)
- fit a linear model on these (x,y)
- transform x into [1,x,x^2,..] and fit a linear model on the transformed x,y.
- vary degree of transforming polynomial and observe variation in training error and error on an independent test set.

In [1]:
```
# generate data from a sine curve with added noise
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
%matplotlib inline

x = np.linspace(0.0,1.0,num=100)
y = np.sin(2*np.pi*x)
xsample = np.linspace(0.0,1.0,num=10)
ysample = np.sin(2*np.pi*xsample) + np.random.randn(10)*0.2
plt.plot(x,y,color='g')
plt.scatter(xsample,ysample,s=20,marker='o')
```

Out[1]: <matplotlib.collections.PathCollection at 0x259cdebfef0>

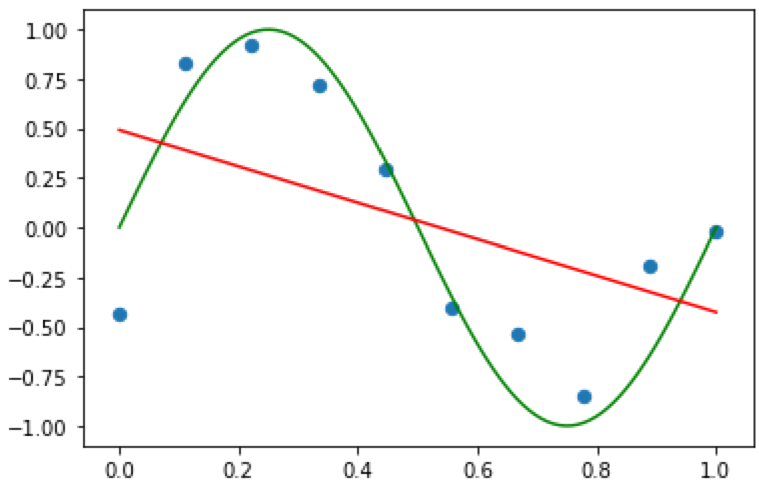


Degree 1 linear regression model (in red)

In [2]:
```
# fit a linear function on xsample, ysample
lr1 = linear_model.LinearRegression()
lr1.fit(xsample.reshape(-1,1),ysample)
ypred = lr1.predict(xsample.reshape(-1,1))
print("score = {0}".format(lr1.score(xsample.reshape(-1,1),ysample)))
print("ypred = \n {0} \n ysample = \n {1} \n xsample = \n {1}".format(ypred, y
sample, xsample))
plt.plot(x,y,color='g')
plt.scatter(xsample,ysample,marker='o',s=40)
plt.plot(xsample,ypred,'r')
```

```
score = 0.2428942438779429
ypred =
 [ 0.49209948  0.39013188  0.28816429  0.18619669  0.08422909 -0.0177385
 -0.1197061  -0.2216737  -0.32364129 -0.42560889]
 ysample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 xsample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x259cdf8edd8>]
```

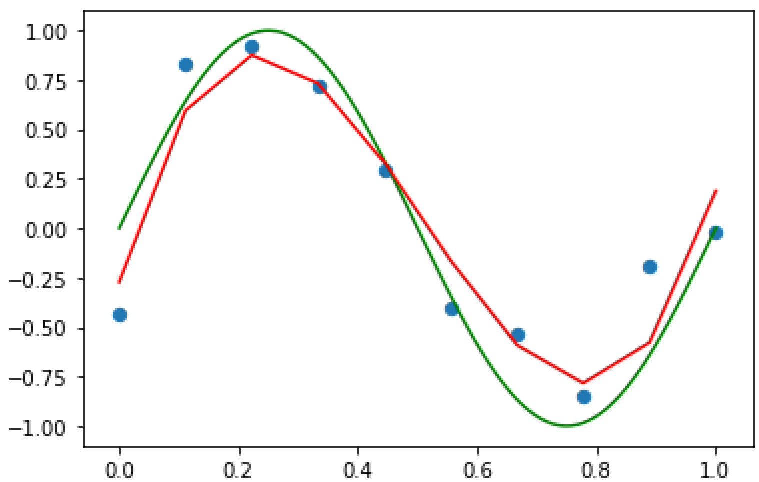


Degree 3 linear model (in red)

In [3]:
```
# fit a third degree polynomial
poly = PolynomialFeatures(3)
xpoly_sample = poly.fit_transform(xsample.reshape(-1,1))
lr3 = linear_model.LinearRegression()
lr3.fit(xpoly_sample,ysample)
ypred3 = lr3.predict(xpoly_sample)
print("score = {0}".format(lr3.score(xpoly_sample,ysample)))
print("ypred3 = \n {0} \n ysample = \n {1} \n xsample = \n {1}".format(ypred3,
ysample, xsample))
plt.plot(x,y,color='g')
plt.scatter(xsample,ysample,marker='o',s=40)
plt.plot(xsample,ypred3,'r')
```

```
score = 0.9029271154925571
ypred3 =
 [-0.27223925  0.59480348  0.87500525  0.73245576  0.33124472 -0.16453817
 -0.5908032  -0.78346067 -0.57842087  0.18840591]
 ysample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 xsample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
```

Out[3]: [<matplotlib.lines.Line2D at 0x259ce009b38>]

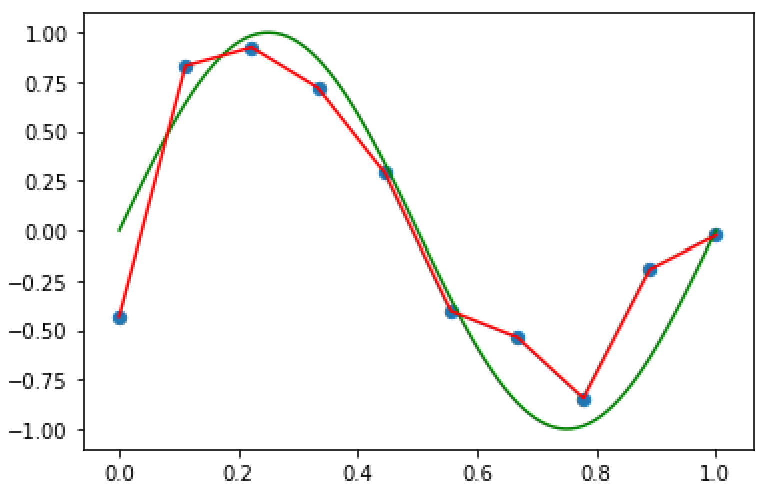


Degree 9 linear model (in red)

In [4]:
```
# fit a ninth degree polynomial
poly = PolynomialFeatures(9)
xpoly_sample = poly.fit_transform(xsample.reshape(-1,1))
lr9 = linear_model.LinearRegression()
lr9.fit(xpoly_sample,ysample)
ypred9 = lr9.predict(xpoly_sample)

print("score = {0}".format(lr9.score(xpoly_sample,ysample)))
print("ypred9 = \n {0} \n ysample = \n {1} \n xsample = \n {1}".format(ypred9,
ysample, xsample))
plt.plot(x,y,color='g')
plt.scatter(xsample,ysample,marker='o',s=40)
plt.plot(xsample,ypred9,'r')
```

```
score = 1.0
ypred9 =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 ysample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 xsample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x259cf0504a8>]
```

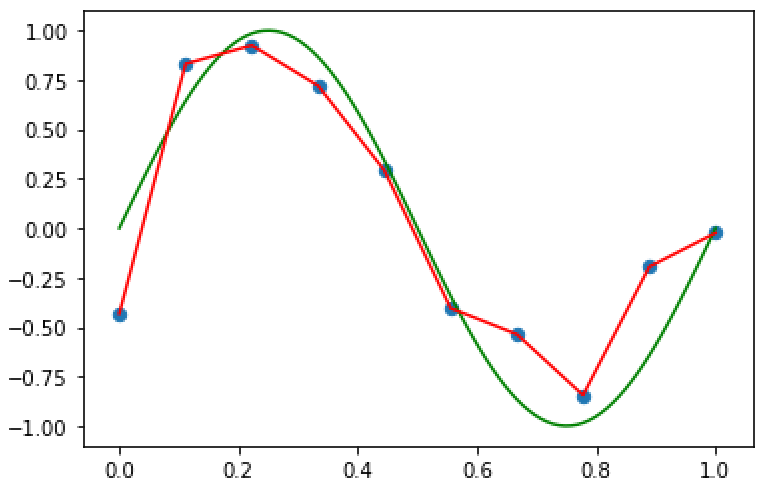


Degree 14 linear model (in red)

In [5]:
```
# fit a 14th degree polynomial
poly = PolynomialFeatures(14)
xpoly_sample = poly.fit_transform(xsample.reshape(-1,1))
lr14 = linear_model.LinearRegression()
lr14.fit(xpoly_sample,ysample)
ypred14 = lr14.predict(xpoly_sample)

print("score = {0}".format(lr14.score(xpoly_sample,ysample)))
print("ypred14 = \n {0} \n ysample = \n {1} \n xsample = \n {1}".format(ypred1
4, ysample, xsample))
plt.plot(x,y,color='g')
plt.scatter(xsample,ysample,marker='o',s=40)
plt.plot(xsample,ypred14,'r')
```

```
score = 1.0
ypred14 =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 ysample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
 xsample =
 [-0.43260986  0.83131974  0.92502673  0.72094489  0.29241997 -0.4046673
 -0.53635233 -0.84469209 -0.19455005 -0.02438674]
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x259cf0b5dd8>]
```

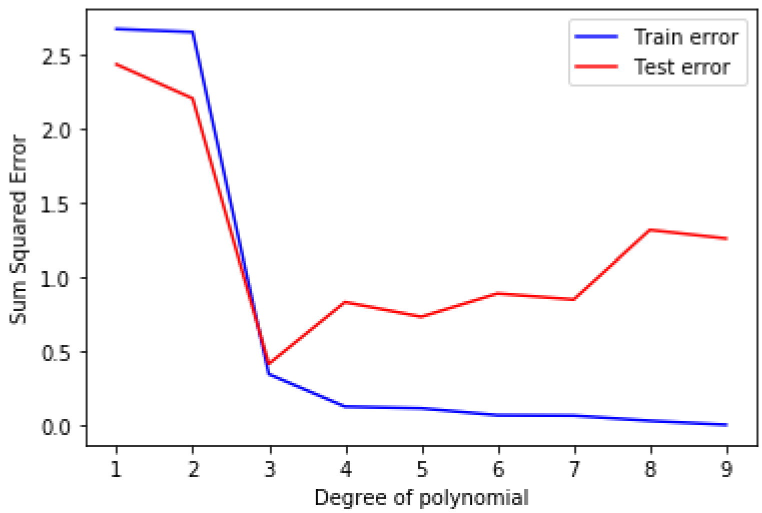


Measuring train and test error as a function of model degree

In [6]:
```
# train/test errors as a function of model complexity
train_error = np.zeros((10,))
test_error = np.zeros((10,))

# make an independent test set
xtest = np.linspace(0.0,1.0,num=30)[:10]
ytest = np.sin(2*np.pi*xtest) + np.random.randn(10)*0.2

for d in range(1,10):
    poly = PolynomialFeatures(d)
    xpoly_sample = poly.fit_transform(xsample.reshape(-1,1))
    lr = linear_model.LinearRegression()
    lr.fit(xpoly_sample,ysample)
    ypred_train = lr.predict(xpoly_sample)
    ypred_test = lr.predict(poly.fit_transform(xtest.reshape(-1,1)))
    tr_err = ysample-ypred_train
    train_error[d] = np.dot(tr_err.T,tr_err)
    te_err = ytest-ypred_test
    test_error[d] = np.dot(te_err.T,te_err)
    print("d = {0} train_error = {1} test_error = {2}".format(d, train_error[d
], test_error[d]))
    print("score = {0}".format(lr.score(xpoly_sample,ysample)))
    # also try lr.coef with higher order model

plt.plot(range(1,10),train_error[1:10],'b',label='Train error')
plt.plot(range(1,10),test_error[1:10],'r',label='Test error')
plt.legend(loc='upper right')
plt.xlabel('Degree of polynomial')
plt.ylabel('Sum Squared Error')
```

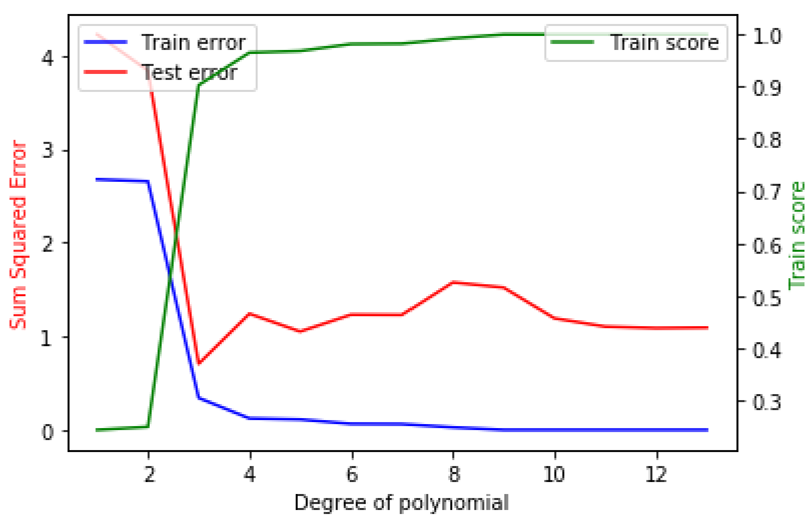
```
d = 1 train_error = 2.6737305772372886 test_error = 2.434505105786301
score = 0.24289424387794278
d = 2 train_error = 2.6530163997871057 test_error = 2.2039461040123993
score = 0.24875976492721474
d = 3 train_error = 0.34281437887566546 test_error = 0.41241639676995867
score = 0.9029271154925571
d = 4 train_error = 0.12310589546024964 test_error = 0.830190156450388
score = 0.9651407726496433
d = 5 train_error = 0.11178946221370376 test_error = 0.7306141996244822
score = 0.9683451855484866
d = 6 train_error = 0.06483981401584299 test_error = 0.8881854207277382
score = 0.9816396622624548
d = 7 train_error = 0.06238489431220379 test_error = 0.8472323454922389
score = 0.9823348085327133
d = 8 train_error = 0.02708517305627619 test_error = 1.3150274750822573
score = 0.9923304387506173
d = 9 train_error = 1.6667078278701455e-21 test_error = 1.2570033773837834
score = 1.0
```

Out[6]: Text(0, 0.5, 'Sum Squared Error')



In [24]:
```
numd = 14
# train/test errors as a function of model complexity - REVISED
train_error = np.zeros((numd,))
test_error = np.zeros((numd,))
scorer = np.zeros((numd,))

# make an independent test set
xtest = np.linspace(0.0,1.0,num=30)[:numd]
ytest = np.sin(2*np.pi*xtest) + np.random.randn(numd)*0.2

for d in range(1,numd):
    poly = PolynomialFeatures(d)
    xpoly_sample = poly.fit_transform(xsample.reshape(-1,1))
    lr = linear_model.LinearRegression()
    lr.fit(xpoly_sample,ysample)
    ypred_train = lr.predict(xpoly_sample)
    ypred_test = lr.predict(poly.fit_transform(xtest.reshape(-1,1)))
    tr_err = ysample-ypred_train
    train_error[d] = np.dot(tr_err.T,tr_err)
    te_err = ytest-ypred_test
    test_error[d] = np.dot(te_err.T,te_err)
    scorer[d] = lr.score(xpoly_sample,ysample)
    print("d = {0:2d}, score = {1:6.3f}, train_error = {2:4.3f}, test_error = 
{3:4.3f} --> {4}".format(d, scorer[d]*100, train_error[d], test_error[d], np.w
here(d < 2, 'start ',np.where(test_error[d] < test_error[d-1], 'test better',
'test not better'))))
    # also try lr.coef with higher order model

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(range(1,numd),train_error[1:numd],'b',label='Train error')
ax1.plot(range(1,numd),test_error[1:numd],'r',label='Test error')
ax1.legend(loc='upper left')
ax2.plot(range(1,numd),scorer[1:numd],'g',label='Train score')
ax2.legend(loc='upper right')
ax1.set_xlabel('Degree of polynomial')
ax1.set_ylabel('Sum Squared Error', color='r')
ax2.set_ylabel('Train score', color='g')
plt.show()
```

```
d =  1, score = 24.289, train_error = 2.674, test_error = 4.224 --> start
d =  2, score = 24.876, train_error = 2.653, test_error = 3.838 --> test bett
er
d =  3, score = 90.293, train_error = 0.343, test_error = 0.707 --> test bett
er
d =  4, score = 96.514, train_error = 0.123, test_error = 1.243 --> test not
better
d =  5, score = 96.835, train_error = 0.112, test_error = 1.052 --> test bett
er
d =  6, score = 98.164, train_error = 0.065, test_error = 1.232 --> test not
better
d =  7, score = 98.233, train_error = 0.062, test_error = 1.230 --> test bett
er
d =  8, score = 99.233, train_error = 0.027, test_error = 1.578 --> test not
better
d =  9, score = 100.000, train_error = 0.000, test_error = 1.522 --> test bet
ter
d = 10, score = 100.000, train_error = 0.000, test_error = 1.193 --> test bet
ter
d = 11, score = 100.000, train_error = 0.000, test_error = 1.105 --> test bet
ter
d = 12, score = 100.000, train_error = 0.000, test_error = 1.089 --> test bet
ter
d = 13, score = 100.000, train_error = 0.000, test_error = 1.094 --> test not
better
```



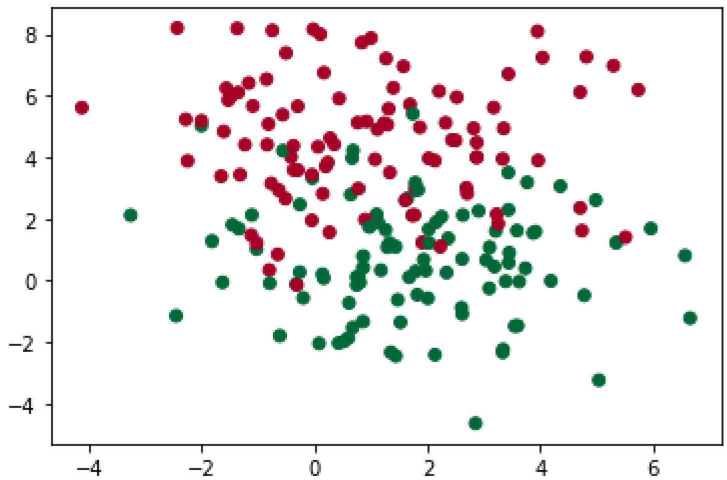
A simple classification problem (2 class)

- generate data from two classes
- build 1-KNN and 15-KNN model
- generate independent test set from the same distribution
- select k by examining variation of test error against k

Generate data

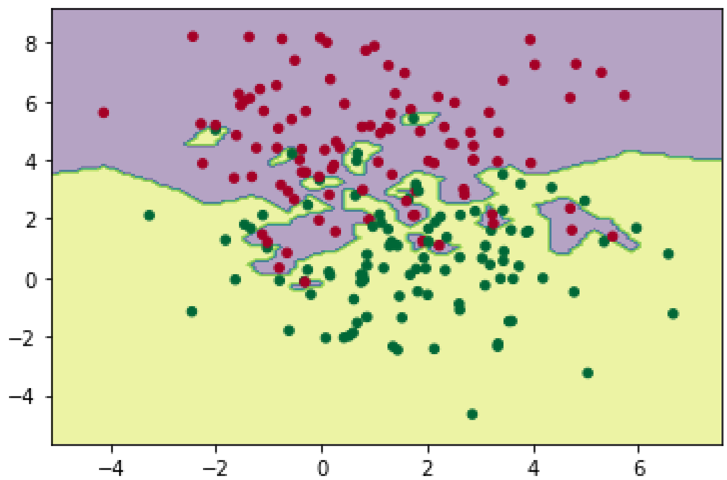
In [8]:
```
# generate data for classification test
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=200, centers=2, n_features=2, cluster_std = 2.0,ra
ndom_state=0)
plt.scatter(X[:,0],X[:,1],c=y,cmap=plt.cm.RdYlGn)
```

Out[8]: <matplotlib.collections.PathCollection at 0x259cf34c048>



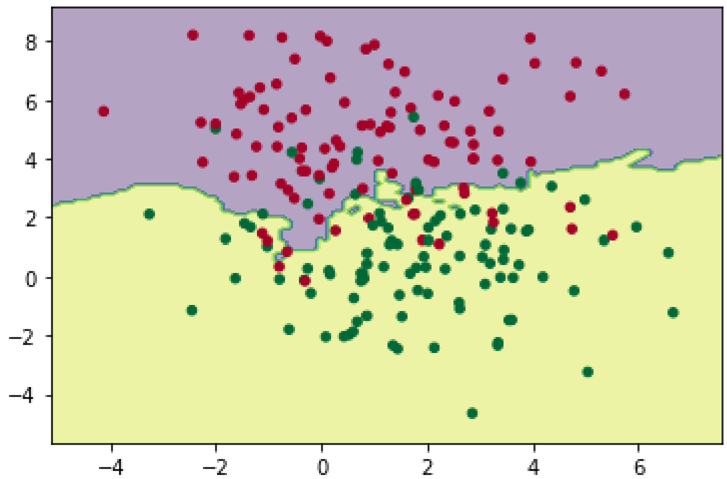
Build KNN model with 1 neighbor

In [9]:
```
# build a 1-KNN model
from sklearn.neighbors import KNeighborsClassifier

# Plotting decision regions
def plot_knn_boundary(X,y,clf):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max,
0.1))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap=plt.cm.RdYlGn)

clf1 = KNeighborsClassifier(n_neighbors=1)
clf1.fit(X,y)
plot_knn_boundary(X,y,clf1)
```

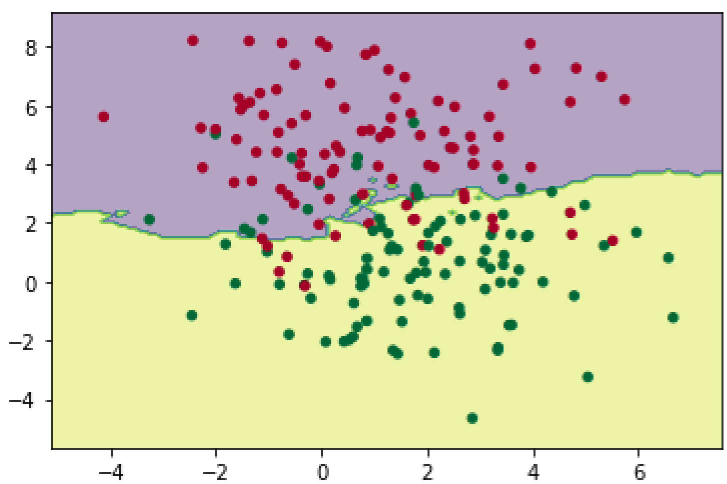


Build KNN model with 9, 15 neighbors

In [10]:
```
clf9 = KNeighborsClassifier(n_neighbors=9)
clf9.fit(X,y)
plot_knn_boundary(X,y,clf9)
```



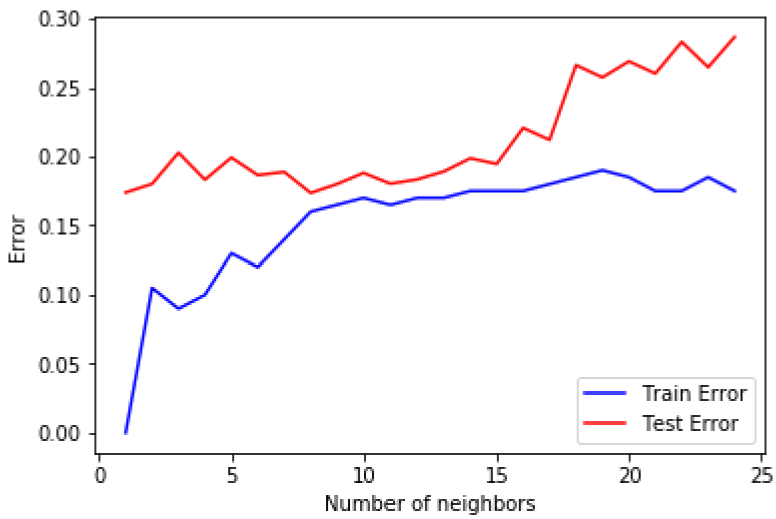
In [11]:
```
clf15 = KNeighborsClassifier(n_neighbors=15)
clf15.fit(X,y)
plot_knn_boundary(X,y,clf15)
```



Variation in train and test accuracy with k

In [19]:
```
numk = 25
train_accuracy =np.zeros((numk,))
test_accuracy = np.zeros((numk,))

Xtest, ytest = make_blobs(n_samples=20000, centers=2, n_features=2, cluster_st
d = 2.0,random_state=3)

for k in range(1,numk):

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)
    train_accuracy[k] = 1.0 - knn.score(X, y)
    test_accuracy[k] = 1.0 - knn.score(Xtest, ytest)
    print("k = {0:3d} train_accuracy = {1:4.3f} test_accuracy = {2:4.3f}".form
at(k, train_accuracy[k], test_accuracy[k]))

plt.plot(range(1,numk),train_accuracy[1:numk],'b',label='Train Error')
plt.plot(range(1,numk),test_accuracy[1:numk],'r',label='Test Error')
plt.legend(loc='lower right')
plt.xlabel('Number of neighbors')
plt.ylabel('Error')
plt.show()
```

```
k =   1 train_accuracy = 0.000 test_accuracy = 0.174
k =   2 train_accuracy = 0.105 test_accuracy = 0.180
k =   3 train_accuracy = 0.090 test_accuracy = 0.203
k =   4 train_accuracy = 0.100 test_accuracy = 0.183
k =   5 train_accuracy = 0.130 test_accuracy = 0.199
k =   6 train_accuracy = 0.120 test_accuracy = 0.186
k =   7 train_accuracy = 0.140 test_accuracy = 0.189
k =   8 train_accuracy = 0.160 test_accuracy = 0.174
k =   9 train_accuracy = 0.165 test_accuracy = 0.180
k =  10 train_accuracy = 0.170 test_accuracy = 0.188
k =  11 train_accuracy = 0.165 test_accuracy = 0.180
k =  12 train_accuracy = 0.170 test_accuracy = 0.183
k =  13 train_accuracy = 0.170 test_accuracy = 0.189
k =  14 train_accuracy = 0.175 test_accuracy = 0.199
k =  15 train_accuracy = 0.175 test_accuracy = 0.195
k =  16 train_accuracy = 0.175 test_accuracy = 0.221
k =  17 train_accuracy = 0.180 test_accuracy = 0.212
k =  18 train_accuracy = 0.185 test_accuracy = 0.266
k =  19 train_accuracy = 0.190 test_accuracy = 0.257
k =  20 train_accuracy = 0.185 test_accuracy = 0.269
k =  21 train_accuracy = 0.175 test_accuracy = 0.260
k =  22 train_accuracy = 0.175 test_accuracy = 0.283
k =  23 train_accuracy = 0.185 test_accuracy = 0.265
k =  24 train_accuracy = 0.175 test_accuracy = 0.287
```



In [0]: