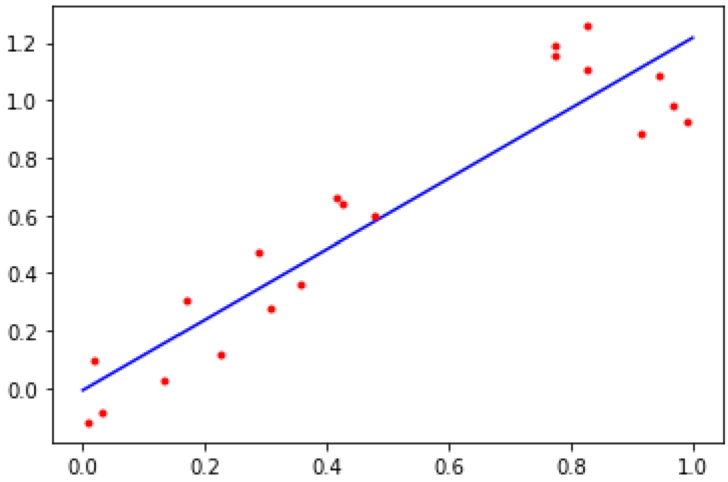
In [10]:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas

# Function that creates the X matrix as defined for fitting our model
def create_X(x,deg):
    X = np.ones((len(x),deg+1))
    for i in range(1,deg+1):
        X[:,i] = x**i
    return X

# Function for predicting the response
def predict_y(x,beta):
    return np.dot(create_X(x,len(beta)-1),beta)

# Function for fitting the model
def fit_beta(df,deg):
    return np.linalg.lstsq(create_X(df.x,deg),df.y,rcond=None)[0]

# Function for computing the MSE
def mse(y,yPred):
    return np.mean((y-yPred)**2)

# Loading training, validation and test data
dfTrain = pandas.read_csv('Data_Train.csv')
dfVal = pandas.read_csv('Data_Val.csv')
dfTest = pandas.read_csv('Data_Test.csv')

############ TRAINING A MODEL

# Fitting model
deg = 1
X = create_X(dfTrain.x,deg)
beta = fit_beta(dfTrain,deg)

# Computing training error
yPredTrain = predict_y(dfTrain.x,beta)
err = mse(dfTrain.y,yPredTrain)
print('Training Error = {:2.3}'.format(err))

# Computing test error
yPredTest = predict_y(dfTest.x,beta)
err = mse(dfTest.y,yPredTest)
print('Test Error = {:2.3}'.format(err))
```

```
Training Error = 0.0258
Test Error = 0.0154
```

In [11]:

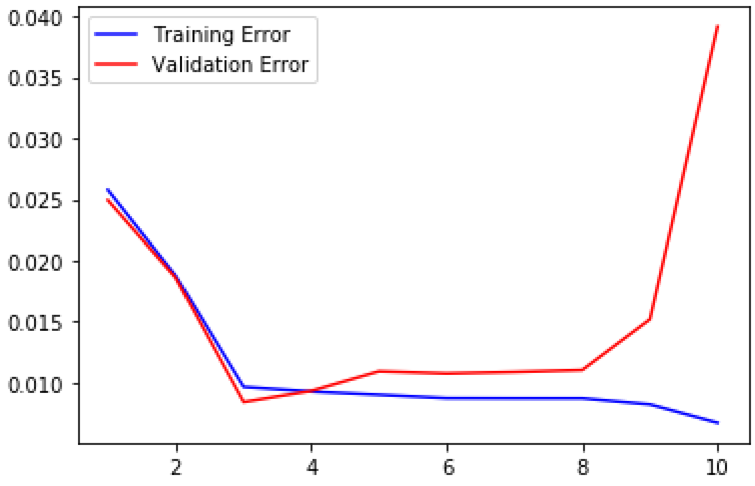
```
############ PLOTTING FITTED MODEL
x = np.linspace(0,1,100)
y = predict_y(x,beta)

plt.plot(x,y,'b-',dfTrain.x,dfTrain.y,'r.')
plt.show()
```



In [12]:

```
############ HYPER-PARAMETER TUNING

# Initializing range of degree values to be tested and errors
degRange = list(range(1,11))
errTrain = np.zeros(len(degRange))
errVal = np.zeros(len(degRange))

# Computing error as a function of degree
# Fitting model
for i in degRange:
    deg = i
    X = create_X(dfTrain.x,deg)
    beta = fit_beta(dfTrain,deg)
    print('Printing errors for degree',i)
# Computing training error
    yPredTrain = predict_y(dfTrain.x,beta)
    err1 = mse(dfTrain.y,yPredTrain)
    errTrain[i-1]=err1
    print('Training Error = {:2.3}'.format(err1))
# Computing test error
    yPredTest = predict_y(dfVal.x,beta)
    err2 = mse(dfVal.y,yPredTest)
    errVal[i-1] = err2
    print('Test Error = {:2.3}'.format(err2))

# Plotting training and validation errors
plt.plot(degRange,errTrain,'b-',degRange,errVal,'r-')
plt.legend(('Training Error','Validation Error'))
plt.show()
```

```
Printing errors for degree 1
Training Error = 0.0258
Test Error = 0.0249
Printing errors for degree 2
Training Error = 0.0188
Test Error = 0.0186
Printing errors for degree 3
Training Error = 0.00967
Test Error = 0.00843
Printing errors for degree 4
Training Error = 0.00929
Test Error = 0.00934
Printing errors for degree 5
Training Error = 0.00902
Test Error = 0.0109
Printing errors for degree 6
Training Error = 0.00874
Test Error = 0.0108
Printing errors for degree 7
Training Error = 0.00873
Test Error = 0.0109
Printing errors for degree 8
Training Error = 0.00873
Test Error = 0.0111
Printing errors for degree 9
Training Error = 0.00823
Test Error = 0.0152
Printing errors for degree 10
Training Error = 0.00673
Test Error = 0.0392
```



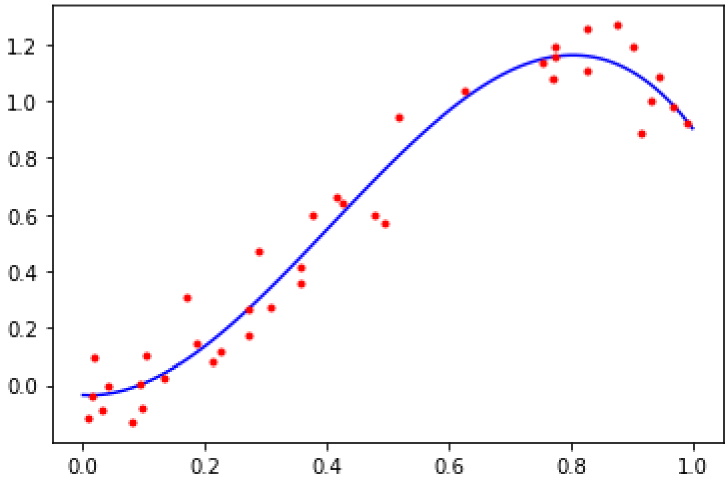
In [13]:

```
############ TRAINING SELECTED MODEL

# Concatenating data training and validation data frames
df = pandas.concat([dfTrain,dfVal])
# Fit model using the optimal degree found in the previous cell

# we see from the above obatined values of errors for all degrees- the test error is le
ast for degree 3,
# so we choose the optimum degree that best fits the data to be '3'
degOpt = 3
X = create_X(dfTrain.x,degOpt)
betaOpt = fit_beta(df,degOpt)


# Compute and print training and test errors
# # Computing training error
yPredTrain = predict_y(df.x,betaOpt)
err = mse(df.y,yPredTrain)
print('Training Error = {:2.3}'.format(err))
# Computing test error
yPredTest = predict_y(dfTest.x,betaOpt)
err = mse(dfTest.y,yPredTest)
print('Test Error = {:2.3}'.format(err))
```

```
Training Error = 0.0087
Test Error = 0.0108
```

In [14]:

```
############ PLOTTING FITTED MODEL
# Plot the fitted model as in the second cell
# PLOTTING FITTED MODEL
x = np.linspace(0,1,100)
y = predict_y(x,betaOpt)
plt.plot(x,y,'b-',df.x,df.y,'r.')
plt.show()
```


1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0