In [1]:
```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  7 16:10:24 2019
@author: Glenn

Wicaksa Munajat
CST383 Summer 2022
June 12, 2022
"""
```

Out[1]:
```
'\nCreated on Thu Nov  7 16:10:24 2019\n@author: Glenn\n\nWicaksa Munajat\nCS
T383 Summer 2022\nJune 12, 2022\n'
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import zscore
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

read the data

In [3]:
```
df = pd.read_csv("https://raw.githubusercontent.com/grbruns/cst383/master/ger
bad_loan = df['good.loan'] - 1
```

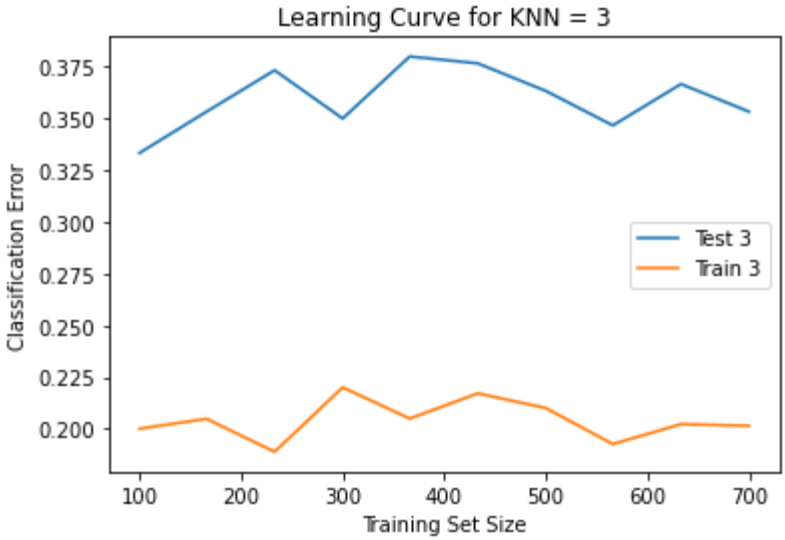
use only numeric data, and scale it

In [4]:
```
df = df[["duration.in.months", "amount", "percentage.of.disposable.income", "
             "age.in.years", "num.credits.at.bank"]]
X = df.apply(zscore).values
y = bad_loan.values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, ran
```

see how knn classifier works as training size changes

In [5]:

```
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
te_errs = []
tr_errs = []
tr_sizes = np.linspace(100, X_train.shape[0], 10).astype(int)

def learning_curve(te_errs, tr_errs, tr_sizes, k):

    for tr_size in tr_sizes:
      X_train1 = X_train[:tr_size,:]
      y_train1 = y_train[:tr_size]

      # train model on a subset of the training data
      knn.fit(X_train1, y_train1)
      # error on subset of training data
      tr_predicted = knn.predict(X_train1)
      err = (tr_predicted != y_train1).mean()
      tr_errs.append(err)

      # error on all test data
      te_predicted = knn.predict(X_test)
      err = (te_predicted != y_test).mean()
      te_errs.append(err)
def plot_curve(tr_sizes, te_errs, tr_errs, k):
    plt.plot(tr_sizes, te_errs, label=f'Test {k}')
    plt.plot(tr_sizes, tr_errs, label=f'Train {k}')
    plt.xlabel('Training Set Size')
    plt.ylabel('Classification Error')
    plt.title(f'Learning Curve for KNN = {k}')
    plt.legend()
    plt.show

learning_curve(te_errs, tr_errs, tr_sizes, k)
plot_curve(tr_sizes, te_errs, tr_errs, k)
```

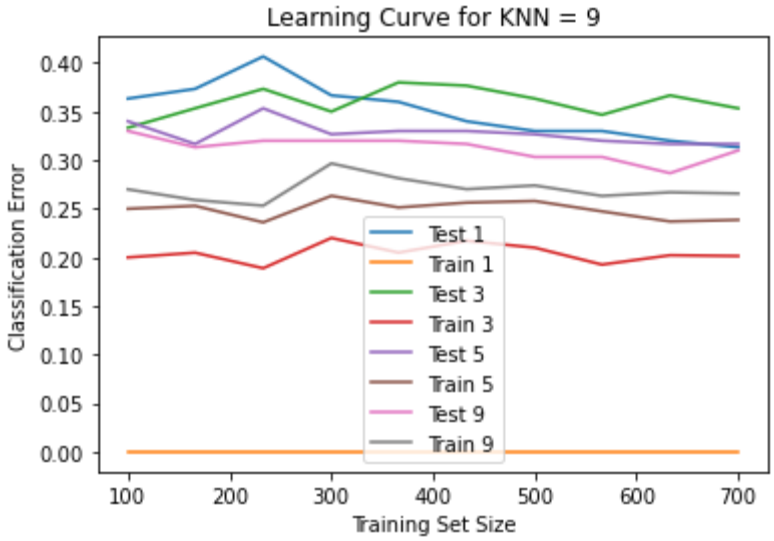


plot the learning curve here

Extend the code by adding a loop so that you produce learning curves for k = 1, 3, 5, and 9.

Put all the plots on one page. Put the value of k in the title of the learning curve plot.

In [6]:
```
kv = [1, 3, 5, 9]

for k in kv:

    knn = KNeighborsClassifier(n_neighbors=k)
    te_errs = []
    tr_errs = []
    tr_sizes = np.linspace(100, X_train.shape[0], 10).astype(int)

    learning_curve(te_errs, tr_errs, tr_sizes, k)
    plot_curve(tr_sizes, te_errs, tr_errs, k)
```



When k=1, we are choosing the closest training sample to the test sample. Since the test sample is in the training dataset, it will choose itself as the closest thus not making any errors. This is why we get a very low error value near 0. As you get higher k values, the training and test errors get close to each other.

On high variance situations, we are over fitting the data. To combat this, we need more data, use fewer features, and control with hyper parameters.

On high bias, we are underfitting. We need to get more features, derive better features from the data, and control with hyper parameters.

A good strategy is to start with a simple algorithm. Use a learning curve for diagnosis. Then examine errors manually to see what to improve.

Learning curves is a good way to diagnose bias and variance.