In [1]:

```
import numpy as np
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import special
```

In [34]:

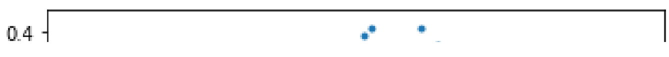
```
t = np.array([-0.49,  0.48, -0.19,  0.24, -0.08,  0.49, -0.21, -0.16, -0.13,
        0.5 ,  0.09,  0.14, -0.27,  0.01,  0.37, -0.42, -0.45, -0.14,
        0.17,  0.14, -0.38,  0.18, -0.45, -0.3 , -0.12,  0.02, -0.11,
        0.38, -0.01,  0.12,  0.44,  0.32,  0.2 , -0.26,  0.04,  0.23,
        0.18,  0.14,  0.15, -0.21, -0.41,  0.06, -0.25, -0.31, -0.42,
       -0.09,  0.34, -0.05,  0.21,  0.26,  0.27,  0.27, -0.28, -0.49,
        0.01,  0.29, -0.15,  0.34,  0.49,  0.03, -0.12,  0.15, -0.02,
       -0.02,  0.24,  0.47, -0.06,  0.34, -0.29, -0.43, -0.09,  0.14,
        0.23,  0.23,  0.08,  0.12, -0.42, -0.12, -0.19,  0.45, -0.36,
       -0.08,  0.13, -0.16,  0.23,  0.44,  0.5 , -0.3 ,  0.45, -0.24,
       -0.42,  0.28,  0.11, -0.47,  0.11, -0.27,  0.41,  0.46, -0.36,
       -0.32, -0.05,  0.45, -0.08, -0.39, -0.34, -0.37,  0.32, -0.24,
       -0.44, -0.03, -0.4 ,  0.47,  0.27, -0.32, -0.2 , -0.37, -0.13,
       -0.07,  0.08, -0.44, -0.44,  0.43, -0.04,  0.09,  0.35, -0.02,
       -0.12,  0.42, -0.39, -0.25, -0.45,  0.44, -0.02,  0.31, -0.48,
        0.23, -0.07, -0.17,  0.4 ,  0.35, -0.2 ,  0.45, -0.1 ,  0.17,
        0.21, -0.11, -0.3 ,  0.48,  0.31,  0.21,  0.2 ,  0.27,  0.47,
       -0.39,  0.46,  0.31,  0.23,  0.5 , -0.43,  0.13, -0.34,  0.02,
       -0.27,  0.3 , -0.18,  0.41,  0.17, -0.15,  0.29, -0.05, -0.39,
        0.25,  0.28,  0.02, -0.21,  0.11,  0.16,  0.47,  0.1 ,  0.03,
        0.22,  0.43,  0.16, -0.21, -0.16,  0.39,  0.25,  0.04, -0.4 ,
        0.22,  0.46,  0.12,  0.15,  0.48, -0.48, -0.15, -0.42,  0.28,
       -0.36,  0.31])

y = np.array([-0.06, -0.09, -0.19,  0.  , -0.03, -0.18, -0.12, -0.31, -0.17,
       -0.18,  0.09,  0.16, -0.28,  0.15, -0.04,  0.04,  0.06, -0.28,
        0.18,  0.11,  0.27,  0.34,  0.11, -0.22, -0.18,  0.39, -0.29,
       -0.06,  0.1 ,  0.08, -0.  , -0.02,  0.06, -0.35,  0.16,  0.28,
       -0.03,  0.17,  0.37, -0.15,  0.19,  0.21, -0.29, -0.33,  0.08,
       -0.24,  0.06,  0.02,  0.26,  0.07,  0.1 ,  0.09, -0.52,  0.03,
        0.3 ,  0.1 , -0.15, -0.26, -0.26,  0.1 , -0.27,  0.11,  0.02,
        0.1 ,  0.09, -0.23, -0.03, -0.21, -0.4 , -0.04, -0.17,  0.07,
        0.08,  0.15,  0.27,  0.41,  0.07, -0.31, -0.29, -0.3 ,  0.06,
       -0.1 ,  0.08, -0.29,  0.07, -0.2 ,  0.06, -0.25, -0.13, -0.34,
        0.05,  0.21,  0.07,  0.13,  0.32, -0.26, -0.01, -0.17, -0.31,
       -0.21,  0.11, -0.1 , -0.01,  0.16, -0.01,  0.1 , -0.1 , -0.28,
        0.08, -0.09,  0.04, -0.07, -0.11, -0.35, -0.23,  0.05, -0.25,
       -0.12,  0.15,  0.01,  0.17,  0.02, -0.01,  0.04, -0.21,  0.03,
       -0.15,  0.08,  0.08, -0.38,  0.19, -0.18,  0.13, -0.03,  0.1 ,
        0.1 , -0.05, -0.29, -0.36,  0.12, -0.42, -0.32, -0.  ,  0.22,
        0.  , -0.36, -0.33,  0.06, -0.04,  0.18, -0.08, -0.22, -0.25,
        0.09, -0.03, -0.17,  0.02, -0.09,  0.13,  0.24, -0.04,  0.15,
       -0.44, -0.02, -0.24,  0.09,  0.17, -0.35, -0.15, -0.1 ,  0.08,
        0.  ,  0.09,  0.22, -0.38,  0.06,  0.14, -0.26,  0.19,  0.41,
        0.13, -0.13,  0.3 , -0.15, -0.02, -0.09, -0.07,  0.31,  0.04,
       -0.02, -0.05,  0.07,  0.16, -0.21,  0.01, -0.2 , -0.09,  0.11,
        0.08, -0.  ])
```

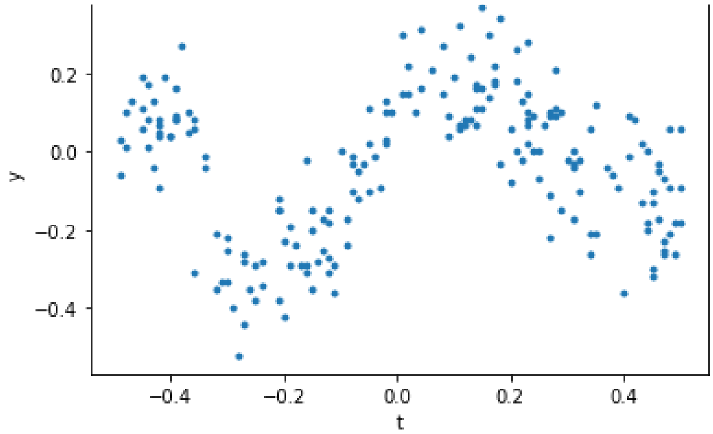
In [3]:

```
plt.plot(t, y, '.')
plt.xlabel('t')
plt.ylabel('y')
```

Out[3]:

```
Text(0, 0.5, 'y')
```





In [4]:

```
def fourier(t, p):
    n = t.size
    return np.stack(
        [np.cos(f * 2*np.pi*t) for f in range(0, (p+1)//2)] +
        [np.sin(f * 2*np.pi*t) for f in range(1, (p+2)//2)])
```

In [5]:

```
def legendre(t, p):
    n = t.size
    return np.stack(
        [special.legendre(d)(t) for d in range(p)])
```

Compute performance measure (cost function - MSE)

Measure average cross-validated MSE.

In [224]:

```
def find_ground_truth(feature,p,n_folds):
    n=200
    mse_val = 0
    mse_val_total = 0

    scores_svm = []
    scores_svm_val =0
    scores_svm_val_total = 0

    arr_mse = [];
    arr_svr = [];

    dict = {};  # Store index of MSE values using dictionary

    for j in range(1,p+1):   #Iterate through Features
        mse_val = 0;
        X = feature(t,j+1)     #Get X for given model
        for train_index, test_index in folds.split(X.T,y):     #Iterate through folds

            X_train, X_test, y_train, y_test = X[:,train_index], X[:,test_index], \
                                                y[train_index], y[test_index]

            theta_hat=np.linalg.inv(X_train @ X_train.T) @ X_train @ y_train   # fit-trai
n model
            y_predict = theta_hat @ X_test
            mse_val += ((y_test - y_predict)**2).mean()   # compute cost function


        mse_val_total = mse_val/n_folds;

        dict.update({mse_val_total:j})
        arr_mse.append(mse_val_total)
```

```
        plt.plot(t,theta_hat @ X, '.',label=str(j))
        plt.legend(fontsize=7, loc="right")
    return arr_mse,dict;
```

Use 10-fold cross-validation to compare models.

In [16]:

```
from sklearn.model_selection import KFold
folds = KFold(n_splits=n_of_splits)
```

In [159]:

```
n_of_splits = 5
n_of_features = 5
```

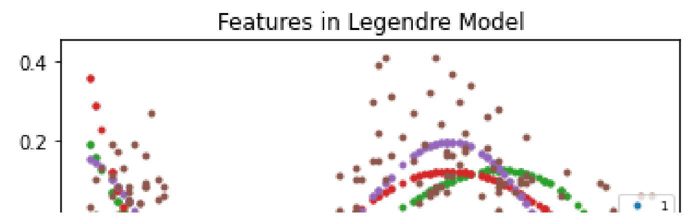
Output: best model and best p.

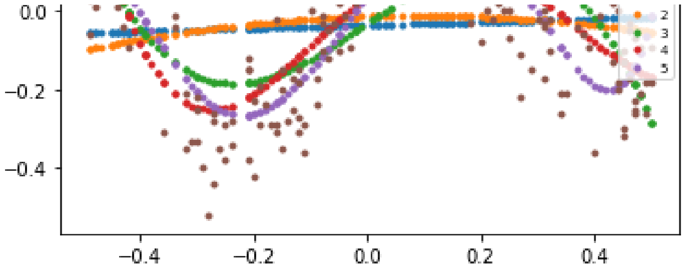
In [153]:

```
fourier_mse,fourier_dict = find_ground_truth(fourier,n_of_features,n_of_splits)
fourier_min_mse=min(fourier_mse);

plt.plot(t, y, '.')
plt.title("Features in Fourier Model")
plt.show()

legendre_mse,legendre_dict = find_ground_truth(legendre,n_of_features,n_of_splits)
legendre_min_mse=min(legendre_mse);

plt.plot(t, y, '.')
plt.title("Features in Legendre Model")
plt.show()


if(fourier_min_mse < legendre_min_mse):
    print('Best Model : Fourier\nNumber of Feature(s): ',fourier_dict.get(fourier_min_ms
e), ' Min MSE:',fourier_min_mse)
else:
    print('Best Model : Legegndre\nNumber of Feature(s): ', legendre_dict.get(legendre_m
in_mse), ' Min MSE:',legendre_min_mse)
```

```
Min Score: 0.014012158251002462
```


```
Min Score: 0.015759456113231
```






```
Best Model : Fourier
Number of Feature(s):  5  Min MSE: 0.014012158251002462
```

Cross Validation Score for different Models

In [291]:

```
from sklearn.model_selection import cross_val_score
```

> **Logistic, RandomForest and SV Models will not be useful since this is not classification problem.**

LinearRegression Model

In [292]:

```
from sklearn.linear_model import LinearRegression
```

In [293]:

```
np.mean(cross_val_score(LinearRegression(), legendre(t,20).T, y, scoring='r2', cv=10))
```

Out[293]:

```
0.625961574846807
```

SVR Model

In [294]:

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

In [295]:

```
np.mean(cross_val_score(SVR(gamma='auto'), fourier(t,20).T, y,cv=10))
```

Out[295]:

```
0.6000705718453085
```

Putting eveything togather

In [267]:

```
folds=6
features=5
```

In [280]:

```
def check_accuracy(features,folds,model):
    arr_svr = [];
    dict = {};
    for j in range(1,p+1):
        if(model== "SVM"):
```

```
            score=np.mean(cross_val_score(SVR(gamma='auto'), features(t,j).T, y,cv=folds
))
        else:
            score=np.mean(cross_val_score(LinearRegression(), legendre(t,20).T, y, scori
ng='r2', cv=10))
        arr_svr.append(score)
        dict.update({score:j})
    return dict,arr_svr;
```

In [289]:

```
def check_models(model):
    dict_fourier,arr_f_score = check_accuracy(fourier,features,model)
    dict_legendre,arr_l_score = check_accuracy(legendre,features,model)

    max_f_score=max(arr_f_score)
    max_l_score=max(arr_l_score)

    if(max_f_score>max_l_score):
        print('\nModel:',model,'\nMethod: Fourier \nAccuracy: ',max_f_score, 'Features:'
,dict_fourier.get(max_f_score))
    else:
        print('\nModel:',model,'\nMethod: Legendre \nAccuracy: ',max_l_score, 'Features:
',dict_fourier.get(max_l_score))
    return;
```

In [290]:

```
check_models("Linear Regression")
check_models("SVM")
```

```
Model: Linear Regression
Method: Legendre
Accuracy:  0.625961574846807 Features: 15

Model: SVM
Method: Fourier
Accuracy:  0.6675567648992817 Features: 6
```

Hyperparameter Tuning Using Grid Search Cross-Validation

A common use of cross-validation is for tuning hyperparameters of a model. The most common technique is what is called grid search cross-validation

In [232]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
```

In [233]:

```
# create a cross-validation scheme
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

# specify range of hyperparameters to tune
hyper_params = [{'n_features_to_select': list(range(1, 6))}]
```

In [234]:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE

X_train, X_test, y_train, y_test = train_test_split(X.T, y,
    test_size=0.3,random_state =1)
model = linear_model.LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, Fa
lse]}
grid = GridSearchCV(model,parameters, cv=None)
```

```
grid.fit(X_train, y_train)
print ('r2 / variance : ', grid.best_score_)
print("Residual sum of squares: %.2f"
              % np.mean((grid.predict(X_test) - y_test) ** 2))
```

```
r2 / variance :  0.6145568248069132
Residual sum of squares: 0.02
```

```
C:\Users\cinni\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978: Future
Warning: The default value of cv will change from 3 to 5 in version 0.22. Specify it expl
icitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

The Recursive Feature Elimination (RFE) method works by recursively removing attributes and building a model on those attributes that remain. It uses accuracy metric to rank the feature according to their importance. The RFE method takes the model to be used and the number of required features as input.

In [195]:

```
from sklearn.feature_selection import RFE
```

In [365]:

```
# perform grid search
# 3.1 specify model
lm = LinearRegression()
lm.fit(X_train, y_train)
rfe = RFE(lm)
```

In [370]:

```
# call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train, y_train)

# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    0.1s finished
```

Out[370]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	split0_tes
0	0.006249	0.007654	0.000000	0.000000	1	{'n_features_to_select': 1}	(
1	0.010683	0.009038	0.000000	0.000000	2	{'n_features_to_select': 2}	(
2	0.006241	0.007643	0.000000	0.000000	3	{'n_features_to_select': 3}	(
3	0.006248	0.007652	0.000000	0.000000	4	{'n_features_to_select': 4}	(
4	0.003125	0.006250	0.001305	0.002609	5	{'n_features_to_select': 5}	(

5 rows × 21 columns

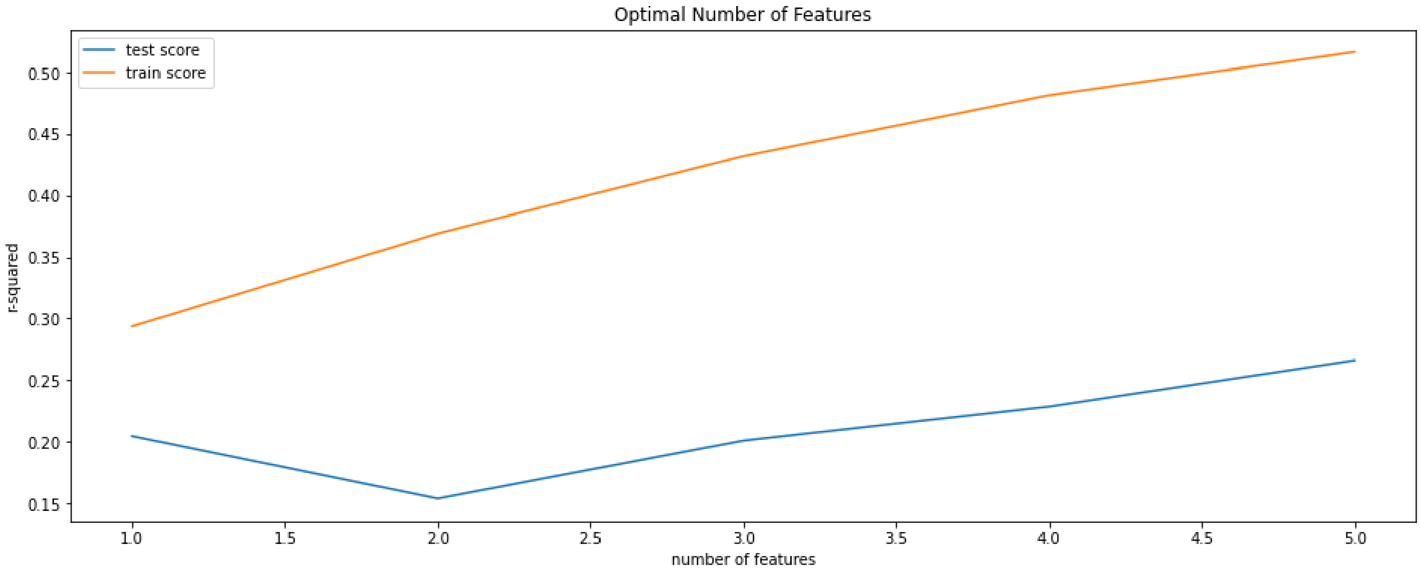
Plotting CV results

In [371]:

```
# plotting cv results
plt.figure(figsize=(16,6))

plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

Out[371]:

```
<matplotlib.legend.Legend at 0x1e37c35cf88>
```



In [284]:

```
model_cv.best_params_
```

Out[284]:

```
{'n_features_to_select': 5}
```

Fit with a different model and a different p.

MODEL - SVR P = 15

In [296]:

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
svm = SVR(gamma='auto')
```

In [297]:

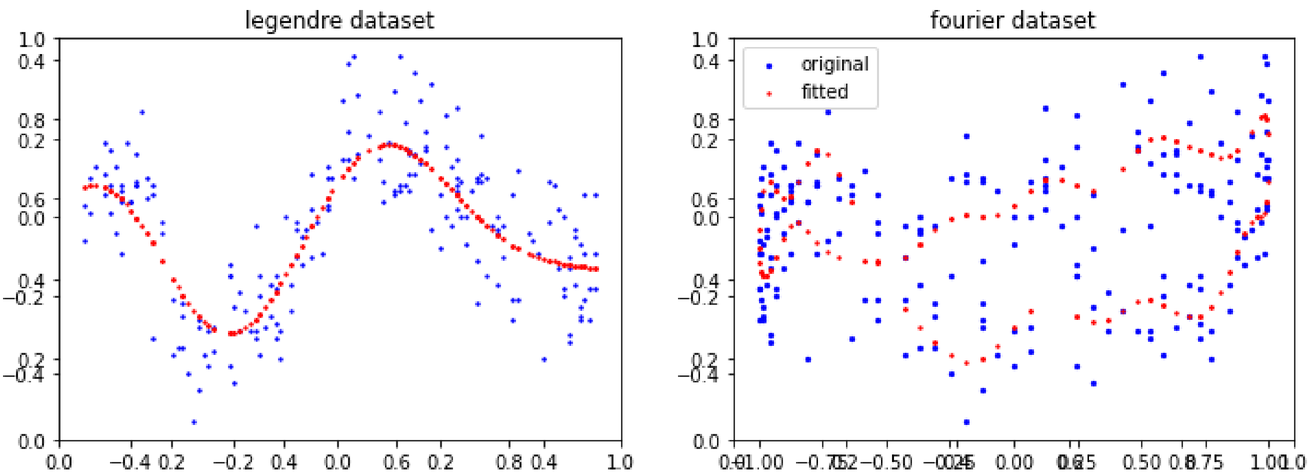
```
p = 15
```

In [158]:

```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

X = legendre(t,p)    # LEGENDRE
svr = svm.fit(X.T, y)
```

```python
yfit = svr.predict(X.T)

ax1.scatter(X.T[:,1], y, s=2, color="blue", label="original")
ax1.scatter(X.T[:,1], yfit, s=2, color="red", label="fitted")
ax1.set_title('legendre dataset')
ax1.set_frame_on(False)

score = svr.score(X.T,y)
print("LEGENDRE R-squared:", score,"MSE:", mean_squared_error(y, yfit))

X = fourier(t,p)    # FOURIER
svr = svm.fit(X.T, y)
yfit = svr.predict(X.T)

ax2.scatter(X.T[:,1], y, s=5, color="blue", label="original")
ax2.scatter(X.T[:,1], yfit, s=2, color="red", label="fitted")
ax2.set_title('fourier dataset')
ax2.legend()
ax2.set_frame_on(False)

score = svr.score(X.T,y)
print("FOURIER  R-squared:", score,"MSE:", mean_squared_error(y, yfit))
```

```
LEGENDRE R-squared: 0.6512863777266 MSE: 0.012613445968155168
FOURIER  R-squared: 0.7259009617437868 MSE: 0.009914534988841926
```



In [ ]: