In [17]:

```
import pandas as pd
import numpy as np
#*********************************************#
#
#    Data Preprocessing
#
#*********************************************#

df = pd.read_excel('index_201101-201511_36.xlsx')
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71842 entries, 0 to 71841
Data columns (total 8 columns):
数据时间        71842 non-null datetime64[ns]
关键词         71842 non-null object
数据类型        71842 non-null object
地区          71842 non-null object
指数          71842 non-null float64
导入时间        71842 non-null datetime64[ns]
数据周期        71842 non-null int64
_AutoID_    71842 non-null int64
dtypes: datetime64[ns](2), float64(1), int64(2), object(3)
memory usage: 4.4+ MB
```

In [2]:

```
#check stock data
def check_colinfo(df_index,col_name):
    '''accept a dataframe of index collection with at least firm tickers and timestamp;
       print the anomalous column value information.
    '''
    #if there is null value in column
    if df_index[col_name].isnull().any():
        #get the array of the index of null value
        idx = np.where(df_index[col_name].isnull())
        #get the tickers of firms with null value
        null_tic = list(df_index.ix[idx[0]]['关键词'].unique())
        if len(null_tic) <=10:
            for i in null_tic:
                print(col_name,'of',i,'have/has NULL value, please CHECK!')
        else:
            print(len(null_tic),'indexes have NULL value, please CHECK!')

    #if there is negative value
    if (df_index[col_name]<0).any():
        neg_tic = list(df_index.ix[df_index[col_name]<0,'关键词'].unique())
        if len(neg_tic) <=10:
            for i in neg_tic:
                print(col_name,'of',i,'have/has negative value(s), please CHECK!')
        else:
            print(len(neg_tic),'indexes have negative value(s), please CHECK!')

    #if there is repetitive record on the same day
    #for convenience we assume date as one seperate float column
    if df_index.duplicated(['关键词','数据时间']).any():
        dup_tic = list(df_index.ix[np.where(df_index.duplicated(['关键词','数据时间']))[0],'关键词']
        if len(dup_tic) <=10:
            for i in dup_tic:
                print(col_name,'of',i,'have/has repetitive value(s), please CHECK!')
        else:
            print(len(dup_tic),'indexes have repetitive value(s), please CHECK!')

check_colinfo(df,'指数')
```

In [18]:

```
# since the way we collect data may mistake zero to some negative values.
df.ix[df['指数']<0,'指数'] = 0
# drop duplicates and choose the first value of duplicated items
df = df.drop_duplicates(subset=['数据时间', '关键词'], keep='first')
df.to_csv('index_201101-201511_36_cleaned.csv')
```

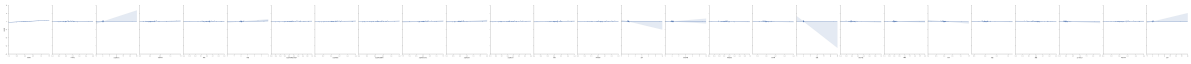
In [1]:

```
#*****************************************#
#
#    Data preprocessing
#
#*****************************************#

########################################
#>>> 1>  Read file and Data preprocessing

# read excel file into dataframe
import pandas as pd
import numpy as np
df = pd.read_excel('final_index.xlsx')

#check data
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 28 columns):
年月            60 non-null datetime64[ns]
发病数           60 non-null int64
HIV阳性         60 non-null float64
艾滋病论坛         60 non-null float64
弓形体病          60 non-null float64
淋病            60 non-null float64
梅毒            60 non-null float64
艾滋病初期症状图片     60 non-null float64
艾滋病检查         60 non-null float64
艾滋病检查费用       60 non-null float64
艾滋病能活多久       60 non-null float64
艾滋病症状         60 non-null float64
艾滋病治疗         60 non-null float64
抽搐            60 non-null float64
带状疱疹          60 non-null float64
盗汗            60 non-null float64
恶性肿瘤          60 non-null float64
呼吸困难          60 non-null float64
肌肉痛           60 non-null float64
咳嗽            60 non-null float64
食欲不振          60 non-null float64
嗜睡            60 non-null float64
头痛            60 non-null float64
消瘦            60 non-null float64
胸痛            60 non-null float64
血小板减少         60 non-null float64
阿昔洛韦          60 non-null float64
丘疹            60 non-null float64
dtypes: datetime64[ns](1), float64(26), int64(1)
memory usage: 13.2 KB
None
```

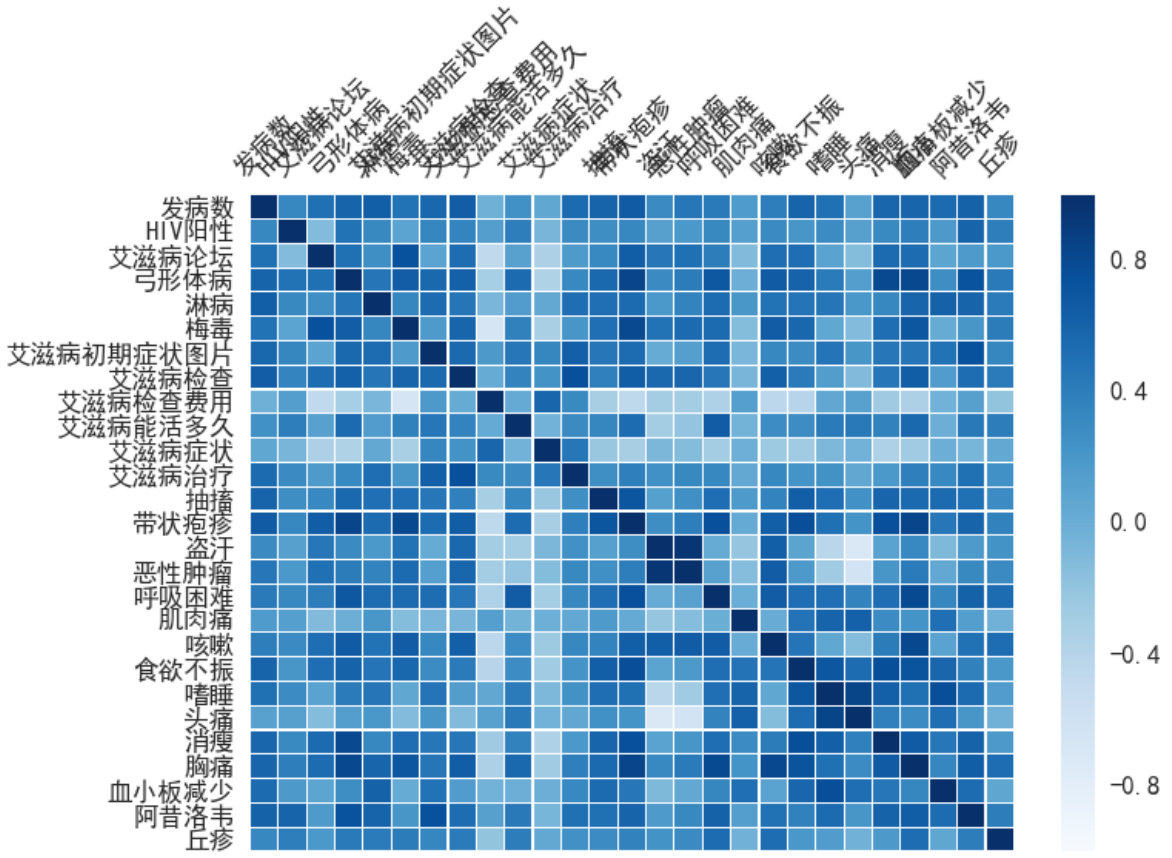
In [17]:

```
########################################
#>>> 2> Visualize

import seaborn as sns # pip install seaborn
import matplotlib.pyplot as plt
sns.set(
        style='ticks',
        font='SimHei',  # fix Chinese output
        rc={'axes.unicode_minus':False} # fix negative notation output
       )

sns.pairplot(df.ix[1:,:], x_vars=column_names[1:], size=7, aspect=0.8,y_vars='发病数',kind='reg',)
plt.show()
```



In [48]:

```
########################################
#>>> 0> # Function to visualize correlation in heatmap
def plot_corr(data):

    # set appropriate font and dpi
    sns.set(
            font='SimHei',  # fix Chinese output
            font_scale=1.2,
            rc={'axes.unicode_minus':False} )# fix negative notation output)
    sns.set_style({"savefig.dpi": 100})
    # plot it out
    ax = sns.heatmap(data, cmap=plt.cm.Blues, linewidths=.1)
    # set the x-axis labels on the top
    ax.xaxis.tick_top()
    # rotate the x-axis and y-axis labels
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)

    # get figure (usually obtained via "fig,ax=plt.subplots()" with matplotlib)
    fig = ax.get_figure()
    # specify dimensions and save


# Get correlation
data = df.ix[:,1:].corr(method='pearson')
plot_corr(data)
plt.show()
```

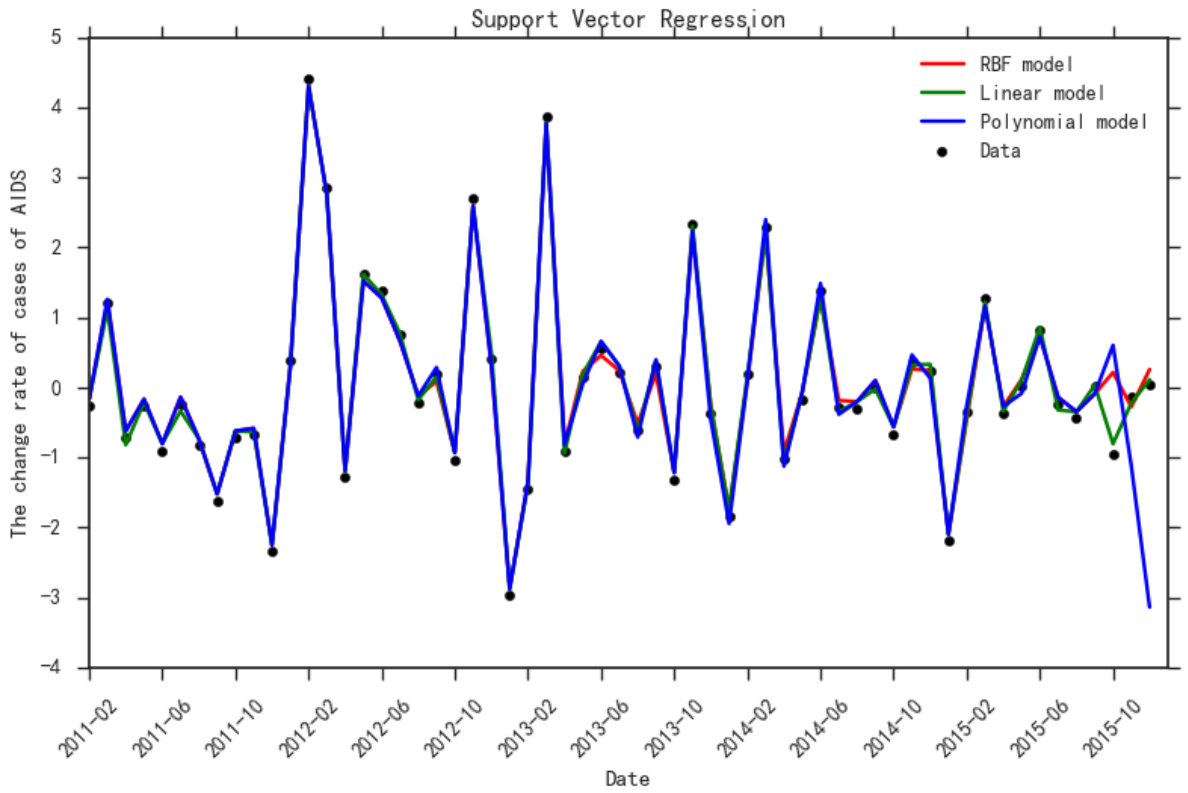


In [3]:

```
########################################
#>>> 0>  Function for Splitting dataset into trainset and testset

def train_test_split(X,y,test_size=0.05):

    test_num = int(np.ceil(test_size * len(X)))
    X_train = X[:-test_num]
    X_test = X[-test_num:]
    y_train = y[:-test_num]
    y_test =  y[-test_num:]
    return X_train,X_test,y_train,y_test

########################################
#>>> 3>  Prepare datasets

from sklearn import preprocessing

# Get the percent change
column_names = df.columns
for name in column_names[1:]:
    df[name] = df[name].pct_change()

# Drop the first row  without value
df.drop(0,axis=0,inplace=True)

# Standardize data
df_matrix = df.ix[:,1:].as_matrix()
df_matrix = preprocessing.scale(df_matrix.T).T

# Split dataset
X=df_matrix[:,1:]
y=df_matrix[:,0]
X_train,X_test,y_train,y_test=train_test_split(X,y,0.05)

# Convert format of date
date_tick = df.ix[:,0].dt.strftime('%Y-%m')
```

In [4]:

```
#*****************************************#
#
#    SVM
#
#*****************************************#

from sklearn.svm import SVR
from sklearn import cross_validation
from matplotlib.ticker import FuncFormatter, MaxNLocator
from sklearn.metrics import mean_squared_error


########################################
#>>> 1>  Function for Using SVM to predict

def svr_predict_aids(X_train,y_train,dates,X,y,x):
    """
    Visualize applying SVM
    """
    # Build model
    svr_rbf = SVR(kernel= 'rbf', C= 1e3, gamma= 0.1)
    svr_rbf.fit(X_train,y_train)
    svr_lin = SVR(kernel= 'linear', C= 1e3)
    svr_lin.fit(X_train,y_train)
    svr_poly = SVR(kernel= 'poly', C= 1e3, degree= 2)
    svr_poly.fit(X_train,y_train)

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_xlim(0,len(date_tick))

    ax.scatter(np.reshape(list(range(len(date_tick))),(len(date_tick), 1)), y, color= 'black', label
    ax.plot(range(len(X)),svr_rbf.predict(X), color= 'red', label= 'RBF model') # plotting the line
    ax.plot(range(len(X)),svr_lin.predict(X), color= 'green', label= 'Linear model') # plotting the
    ax.plot(range(len(X)),svr_poly.predict(X), color= 'blue', label= 'Polynomial model') # plotting
    ax.set_xlabel('Date')
    ax.set_ylabel('The change rate of cases of AIDS')
    ax.set_title('Support Vector Regression')
    ax.xaxis.set_major_locator(MaxNLocator(nbins=15, integer=True))

    dates = list(date_tick)
    dates.append(dates[-1])
    dates = np.array(dates)
    ax.set_xticklabels(dates.reshape(int(len(dates)/4),4)[:,0])

    plt.legend(loc='best')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    score_rbf = mean_squared_error(svr_rbf.predict(X_test),y_test)
    score_lin = mean_squared_error(svr_lin.predict(X_test),y_test)
    score_poly = mean_squared_error(svr_poly.predict(X_test),y_test)
    return score_rbf, score_lin, score_poly

# SVM to predict
score_rbf, score_lin, score_poly = svr_predict_aids(X_train,y_train,date_tick,X,y,len(y)-1)
print("MSE of rbf :",score_rbf,'\n',"MSE of linear :",score_lin,'\n',"MSE of poly :",score_rbf)
```



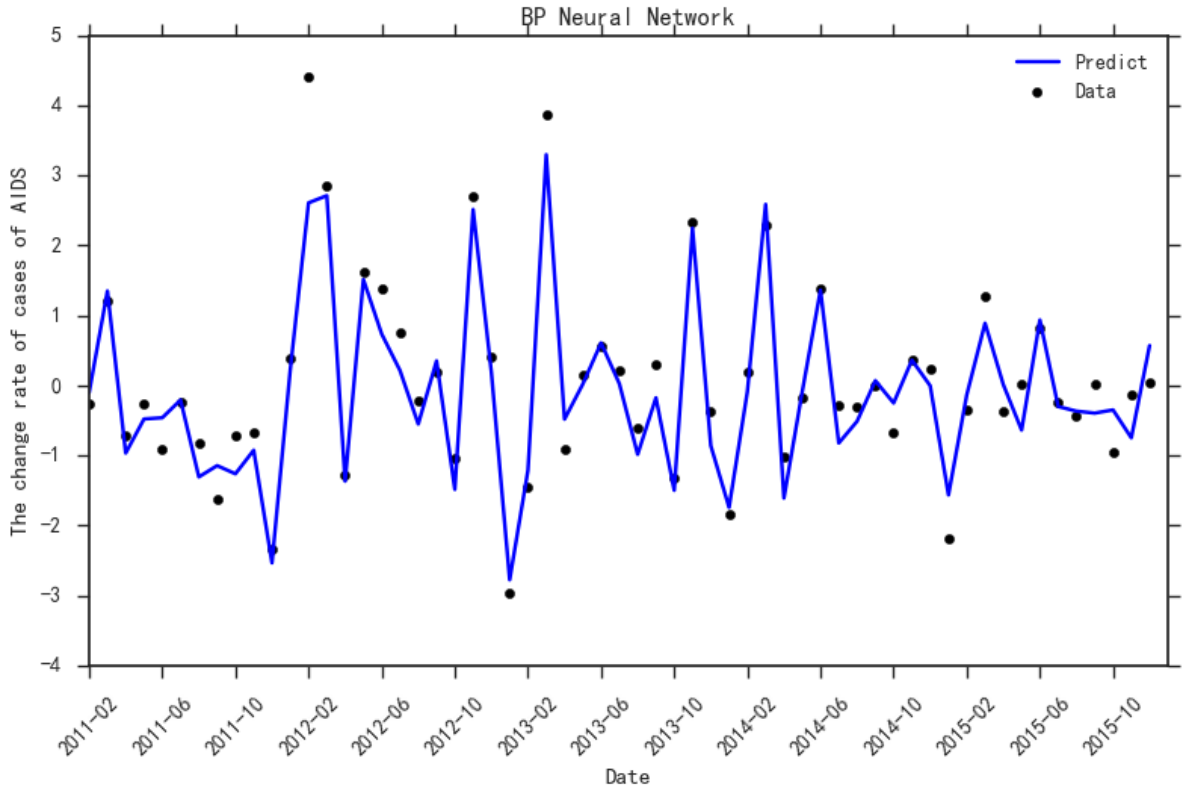

```
MSE of rbf : 0.486988051034
 MSE of linear : 0.0131702659883
 MSE of poly : 0.486988051034
```

In [10]:

```
#*****************************************#
#
#    Back propagation neural network
#
#*****************************************#


from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.grid_search import GridSearchCV
from keras.optimizers import SGD


model = Sequential()
# Create model
model.add(Dense(30,input_dim=26,activation='relu'))
#model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1,activation='linear'))

# Complie
optimizer = SGD(lr=0.01, momentum=0.9)
model.compile(loss='mean_squared_error', optimizer=optimizer)

# Fit
results = model.fit(X_train,y_train,epochs = 10, batch_size=10, verbose=0)
score_bp = model.evaluate(X_test,y_test)
print("MSE of testset IN BP: ",score_bp)
print(y_test)

# Predict
y_pred = model.predict(X)

# plot prediction data
fig_2 = plt.figure()
ax = fig_2.add_subplot(111)
ax.set_xlim(0,len(date_tick))
ax.scatter(range(len(y_pred)), y, color= 'black', label= 'Data')
ax.plot(range(len(y_pred)),y_pred,'b',label='Predict')

ax.set_xlabel('Date')
ax.set_ylabel('The change rate of cases of AIDS')
ax.set_title('BP Neural Network')
ax.xaxis.set_major_locator(MaxNLocator(nbins=15, integer=True))

dates = list(date_tick)
dates.append(dates[-1])
dates = np.array(dates)
ax.set_xticklabels(dates.reshape(int(len(dates)/4),4)[:,0])

plt.legend(loc='best')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

```
3/3 [==============================] - 0s
MSE of testset IN BP:  0.349272966385
```

MSE of testset in BP: 0.349272960385
[-0.96306257 -0.12848555 0.03439329]



In [12]:

```
########################################
#>>> 0> # Function to view results
def show_best_score(grid_result):
    """
    print best choice for keras regression
    """
    mse_arr = np.array([])
    for params, mean_score, scores in grid_result.grid_scores_:

        print("%f (%f) with: %r" % (scores.mean(), scores.std(), params))
        mse_arr = np.concatenate((mse_arr,np.array([scores.mean(),params])))

    mse_arr = mse_arr.reshape((int(len(mse_arr)/2),2)) # reshape to 2 dimensions for calculating mse
    print("Best: %f using %s" % (mse_arr[mse_arr[:,0]==mse_arr[:,0].min()][0][0],mse_arr[mse_arr[:,0



########################################
#>>> 1> # Function to create model for tuning activation
def create_model_1(activation = 'sigmoid'):

    # create model
    model = Sequential()
    model.add(Dense(30,input_dim=26,activation='relu'))
    model.add(Activation('relu'))
    model.add(Dense(1,activation=activation))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='SGD')
    return model

# Find optimal activation
model = KerasRegressor(build_fn=create_model_1, epochs = 10, batch_size=10, verbose=0)
# define the grid search parameters
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear'
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)

show_best_score(grid_result)
```

```
2.847612 (0.711022) with: {'activation': 'softmax'}
2.131128 (0.797848) with: {'activation': 'softplus'}
1.991538 (0.976736) with: {'activation': 'softsign'}
2.027911 (0.789575) with: {'activation': 'relu'}
1.915652 (0.760310) with: {'activation': 'tanh'}
2.021431 (0.806095) with: {'activation': 'sigmoid'}
2.082010 (0.816970) with: {'activation': 'hard_sigmoid'}
2.131582 (0.930664) with: {'activation': 'linear'}
Best: 1.915652 using {'activation': 'tanh'}
```

In [16]:

```
########################################
#>>> 2> # Function to create model for tuning optimal batchsize and training epochs
def create_model_2(batch_size = 30,input_dim = 26):

    # create model
    model = Sequential()
    model.add(Dense(batch_size,input_dim=input_dim,activation='relu'))
    model.add(Activation('relu'))
    model.add(Dense(1,activation='linear'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='SGD')
    return model


model = KerasRegressor(build_fn=create_model_2, verbose=0)
batch_size = [10,20,30]
epochs = [10,50]

param_grid = dict(batch_size=batch_size,nb_epoch=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)
show_best_score(grid_result)
```

```
2.131330 (0.813639) with: {'nb_epoch': 10, 'batch_size': 10}
2.046298 (1.029935) with: {'nb_epoch': 50, 'batch_size': 10}
2.580343 (1.162664) with: {'nb_epoch': 10, 'batch_size': 20}
2.222788 (0.926217) with: {'nb_epoch': 50, 'batch_size': 20}
1.812820 (0.855541) with: {'nb_epoch': 10, 'batch_size': 30}
1.993234 (0.840102) with: {'nb_epoch': 50, 'batch_size': 30}
Best: 1.812820 using {'nb_epoch': 10, 'batch_size': 30}
```

In [18]:

```
########################################
#>>> 3> # Function to create model for tuning optimizer
def create_model_3(optimizer = 'SGD'):

    # create model
    model = Sequential()
    model.add(Dense(30,input_dim=26,activation='relu'))
    model.add(Activation('relu'))
    model.add(Dense(1,activation='linear'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model

# Find optimaizer
model = KerasRegressor(build_fn=create_model_3, epochs = 10, batch_size=10, verbose=0)
# define the grid search parameters
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)
show_best_score(grid_result)
```

```
1.876631 (0.959333) with: {'optimizer': 'SGD'}
2.059060 (1.042237) with: {'optimizer': 'RMSprop'}
1.884204 (0.733410) with: {'optimizer': 'Adagrad'}
1.824416 (0.792038) with: {'optimizer': 'Adadelta'}
1.846100 (0.716938) with: {'optimizer': 'Adam'}
2.383727 (0.977617) with: {'optimizer': 'Adamax'}
1.947258 (0.973459) with: {'optimizer': 'Nadam'}
Best: 1.824416 using {'optimizer': 'Adadelta'}
```

In [19]:

```
########################################
#>>> 4> # Function to create model for tuning learning rate and momentum

def create_model_4(learn_rate=0.01, momentum=0.9):

    # create model
    model = Sequential()
    model.add(Dense(30,input_dim=26,activation='relu'))
    model.add(Activation('relu'))
    model.add(Dense(1,activation='linear'))

    # Compile model
    optimizer = SGD(lr=learn_rate, momentum=momentum)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model


# Find optimal learning rate and momentum
model = KerasRegressor(build_fn=create_model_4, epochs = 10, batch_size=10, verbose=0)
# define the grid search parameters
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
param_grid = dict(learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)

show_best_score(grid_result)
```

```
2.407559 (0.934983) with: {'momentum': 0.0, 'learn_rate': 0.001}
2.108527 (0.787898) with: {'momentum': 0.2, 'learn_rate': 0.001}
2.366757 (0.852796) with: {'momentum': 0.4, 'learn_rate': 0.001}
2.075800 (0.708892) with: {'momentum': 0.6, 'learn_rate': 0.001}
2.064825 (0.953951) with: {'momentum': 0.8, 'learn_rate': 0.001}
1.889727 (0.806629) with: {'momentum': 0.9, 'learn_rate': 0.001}
2.394459 (1.200493) with: {'momentum': 0.0, 'learn_rate': 0.01}
2.028620 (0.982316) with: {'momentum': 0.2, 'learn_rate': 0.01}
1.848259 (0.886741) with: {'momentum': 0.4, 'learn_rate': 0.01}
2.250743 (0.679329) with: {'momentum': 0.6, 'learn_rate': 0.01}
1.986153 (0.722411) with: {'momentum': 0.8, 'learn_rate': 0.01}
2.412243 (0.328795) with: {'momentum': 0.9, 'learn_rate': 0.01}
2.288509 (0.680591) with: {'momentum': 0.0, 'learn_rate': 0.1}
2.121893 (0.390046) with: {'momentum': 0.2, 'learn_rate': 0.1}
2.502005 (0.290244) with: {'momentum': 0.4, 'learn_rate': 0.1}
2.429795 (0.835475) with: {'momentum': 0.6, 'learn_rate': 0.1}
nan (nan) with: {'momentum': 0.8, 'learn_rate': 0.1}
nan (nan) with: {'momentum': 0.9, 'learn_rate': 0.1}
nan (nan) with: {'momentum': 0.0, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.2, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.4, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.6, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.8, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.9, 'learn_rate': 0.2}
nan (nan) with: {'momentum': 0.0, 'learn_rate': 0.3}
nan (nan) with: {'momentum': 0.2, 'learn_rate': 0.3}
nan (nan) with: {'momentum': 0.4, 'learn_rate': 0.3}
nan (nan) with: {'momentum': 0.6, 'learn_rate': 0.3}
nan (nan) with: {'momentum': 0.8, 'learn_rate': 0.3}
nan (nan) with: {'momentum': 0.9, 'learn_rate': 0.3}
```

```
exError                                 Traceback (most recent call last)
<ipython-input-19-61f7839b2539> in <module>()
     25 grid_result = grid.fit(X_train, y_train)
     26
---> 27 show_best_score(grid_result)

<ipython-input-12-445e99176453> in show_best_score(grid_result)
     12
     13     mse_arr = mse_arr.reshape((int(len(mse_arr)/2),2)) # reshape to 2 dimens
ions for calculating mse
---> 14     print("Best: %f using %s" % (mse_arr[mse_arr[:,0]==mse_arr[:,0].min()]
[0][0],mse_arr[mse_arr[:,0] ==mse_arr[:,0].min()][0][1]))
     15
     16

IndexError: index 0 is out of bounds for axis 0 with size 0
```

In [ ]:

```
##########################################
#>>> 5> # Function to create model for tuning network weight initialization
# it is not necessary to run this step and neither is dropout because in our project the sample is t

def create_model_5(init_mode='uniform'):

    # create model
    model = Sequential()
    model.add(Dense(30,input_dim=26,kernel_initializer=init_mode,activation='relu'))
    model.add(Activation('relu'))
    model.add(Dense(1,activation='linear'))

    # Compile model
    optimizer = SGD(lr=0.01, momentum=0.9)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [ ]:

```

```