FINAL PDF VERSION_BOSTON HOUSING PRICE DATASET_LAAVANYA GANESH_UIN-654324917

September 23, 2017

```
In [1]: # Loading required libraries

In [2]: import numpy
        from keras import optimizers
        from sklearn.model_selection import GridSearchCV
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.wrappers.scikit_learn import KerasRegressor
```

```
Using TensorFlow backend.
```

```
In [3]: # Question 2a

In [4]: # Defining model for Optimizer function

In [5]: def model_optimizer(optimizer='adam'):
            # create model
            model = Sequential()
            model.add(Dense(13, input_dim=13,kernel_initializer='normal', activation='relu'))
            model.add(Dense(1, kernel_initializer='normal'))
            # Compile model
            model.compile(loss='mean_squared_error', optimizer='adam')
            return model

In [6]: # Question 1

In [7]: from sklearn.datasets import load_boston

        # Loading Boston dataset

        X, Y = load_boston(return_X_y=True)
        Y=Y.reshape(506,1)
        X = X.astype(float)
        X=X[1:100,:]
        Y=Y[1:100,:]
        print (X.shape)
        print (Y.shape)
```

```
(99, 13)
(99, 1)
```

```
In [8]: # Creating regression model using KerasRegressor and optimizer function

        estimator_optimizer = KerasRegressor(build_fn=model_optimizer, epochs=200,batch_size=5
                                             verbose=0)
```

```
In [9]: # Fix random seed for reproducibility
        seed = 7
```

```
In [10]: numpy.random.seed(seed)

         # Define the grid search parameters for optimizer function
         # Used the scoring parameter to get the model with highest negative score


         optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
         param_grid_optimizer = dict(optimizer=optimizer)
         grid_optimizer = GridSearchCV(estimator=estimator_optimizer,
                                       param_grid=param_grid_optimizer,
                                       n_jobs=1,scoring='neg_mean_squared_error')
         grid_result_optimizer = grid_optimizer.fit(X, Y)
```

```
In [11]: # Summarize results for optimizer funtion

         print("Best: %f using %s" % (grid_result_optimizer.best_score_,
                                      grid_result_optimizer.best_params_))
         means_optimizer = grid_result_optimizer.cv_results_['mean_test_score']
         stds_optimizer = grid_result_optimizer.cv_results_['std_test_score']
         params_optimizer = grid_result_optimizer.cv_results_['params']
         for mean, stdev, param in zip(means_optimizer, stds_optimizer, params_optimizer):
             print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -16.713254 using {'optimizer': 'Adamax'}
-22.094965 (0.852109) with: {'optimizer': 'SGD'}
-17.516175 (1.853895) with: {'optimizer': 'RMSprop'}
-19.462024 (3.235954) with: {'optimizer': 'Adagrad'}
-20.204300 (3.769810) with: {'optimizer': 'Adadelta'}
-19.039592 (4.702837) with: {'optimizer': 'Adam'}
-16.713254 (1.047244) with: {'optimizer': 'Adamax'}
-18.426794 (2.084573) with: {'optimizer': 'Nadam'}
```

```
In [13]: # Loading required libraries
```

```
In [12]: import numpy
         from keras import optimizers
```

```
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from keras.optimizers import Adamax
```

In [14]:
```
# Question 2b
```

In [15]:
```
# Defining a model for the tuning Learning Rate and Momentum function
# I received Adamax as my best optimizer in Question 2a
# (Adamax optimizer does not have a momentum argument)
```

In [16]:
```
def model_learning(learn_rate=0.01):
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13,kernel_initializer='normal', activation='relu'))
    model.add(Dense(1,kernel_initializer='normal'))
    # Compile model
    optimizer = Adamax(lr=learn_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [17]:
```
from sklearn.datasets import load_boston

# Loading Boston dataset

X, Y = load_boston(return_X_y=True)
Y=Y.reshape(506,1)
X = X.astype(float)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(99, 13)
(99, 1)
```

In [18]:
```
# Creating regression model using KerasRegressor and learning rate function

estimator_learning = KerasRegressor(build_fn=model_learning, epochs=200,batch_size=5,
                                    verbose=0)
```

In [19]:
```
# fix random seed for reproducibility
seed = 7
```

In [20]:
```
numpy.random.seed(seed)

# Define the grid search parameters for learning rate function
```

```
# Used the scoring parameter to get the model with highest negative score

learn_rate = [0.1, 0.2, 0.3]
param_grid_learning = dict(learn_rate=learn_rate)
grid_learning = GridSearchCV(estimator=estimator_learning,
                             param_grid=param_grid_learning,
                             n_jobs=1,scoring='neg_mean_squared_error')
grid_result_learning = grid_learning.fit(X, Y)
```

In [21]:
```
# Summarize results for learning rate and momentum function

print("Best: %f using %s" % (grid_result_learning.best_score_,
                             grid_result_learning.best_params_))
means_learning = grid_result_learning.cv_results_['mean_test_score']
stds_learning = grid_result_learning.cv_results_['std_test_score']
params_learning = grid_result_learning.cv_results_['params']
for mean, stdev, param in zip(means_learning, stds_learning, params_learning):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -18.468527 using {'learn_rate': 0.1}
-18.468527 (5.433792) with: {'learn_rate': 0.1}
-25.607932 (18.975855) with: {'learn_rate': 0.2}
-40.233141 (9.163053) with: {'learn_rate': 0.3}
```

In [22]:
```
# Loading required libraries
```

In [23]:
```
import numpy
from keras import optimizers
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from keras.optimizers import Adamax
```

In [24]:
```
# Question 2c
```

In [25]:
```
# Defining a model for the tuning Activation function
# I received 0.1 as my best learning rate in Question 2b
```

In [26]:
```
def model_activation(activation='relu'):
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13,kernel_initializer='normal',
                    activation=activation))
    model.add(Dense(1,kernel_initializer='normal'))
    # Compile model
    optimizer = Adamax(lr=0.1)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [27]:
```
from sklearn.datasets import load_boston

#loading Boston dataset

X, Y = load_boston(return_X_y=True)
Y=Y.reshape(506,1)
X = X.astype(float)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(99, 13)
(99, 1)
```

In [28]:
```
# Creating regression model using KerasRegressor and activation function

estimator_activation = KerasRegressor(build_fn=model_activation, epochs=200,
                                      batch_size=5,
                                      verbose=0)
```

In [29]:
```
# fix random seed for reproducibility

seed = 7
```

In [30]:
```
numpy.random.seed(seed)

# Define the grid search parameters for activation function
# Used the scoring parameter to get the model with highest negative score

activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid',
              'hard_sigmoid', 'linear']
param_grid_activation = dict(activation=activation)
grid_activation = GridSearchCV(estimator=estimator_activation,
                               param_grid=param_grid_activation,
                               n_jobs=1,scoring='neg_mean_squared_error')
grid_result_activation = grid_activation.fit(X, Y)
```

In [31]:
```
# Summarize results for activation function

print("Best: %f using %s" % (grid_result_activation.best_score_,
                             grid_result_activation.best_params_))
means_activation = grid_result_activation.cv_results_['mean_test_score']
stds_activation = grid_result_activation.cv_results_['std_test_score']
params_activation = grid_result_activation.cv_results_['params']
for mean, stdev, param in zip(means_activation, stds_activation,
                              params_activation):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -19.054416 using {'activation': 'softsign'}
-40.585768 (9.735321) with: {'activation': 'softmax'}
-30.688856 (10.132538) with: {'activation': 'softplus'}
-19.054416 (5.783269) with: {'activation': 'softsign'}
-25.844418 (8.497787) with: {'activation': 'relu'}
-40.066287 (8.215580) with: {'activation': 'tanh'}
-35.695969 (12.672931) with: {'activation': 'sigmoid'}
-38.794644 (9.289991) with: {'activation': 'hard_sigmoid'}
-30.088417 (9.170786) with: {'activation': 'linear'}
```

```
In [32]: # Loading required libraries
```

```
In [33]: import numpy
         from keras import optimizers
         from sklearn.model_selection import GridSearchCV
         from keras.models import Sequential
         from keras.layers import Dense,Dropout
         from keras.wrappers.scikit_learn import KerasRegressor
         from keras.optimizers import Adamax
         from keras.constraints import maxnorm
```

```
In [34]: # Question 2d
```

```
In [35]: # Defining a model for the Dropout (Weight regularization) function
         # Used best parameters from Question 2a, 2b and 2c
```

```
In [36]: def model_dropout(dropout_rate=0.0, weight_constraint=0):
             # create model
             model = Sequential()
             model.add(Dense(13, input_dim=13, kernel_initializer='normal',
                             activation='softsign',
                             kernel_constraint=maxnorm(weight_constraint)))
             model.add(Dropout(dropout_rate))
             model.add(Dense(1, kernel_initializer='normal'))
             # Compile model
             optimizer = Adamax(lr=0.1)
             model.compile(loss='mean_squared_error', optimizer=optimizer)
             return model
```

```
In [37]: from sklearn.datasets import load_boston

         # Loading Boston dataset

         X, Y = load_boston(return_X_y=True)
         Y=Y.reshape(506,1)
         X = X.astype(float)
         X=X[1:100,:]
         Y=Y[1:100,:]
```

```
print (X.shape)
print (Y.shape)
```

```
(99, 13)
(99, 1)
```

In [38]:
```
# Creating regression model using KerasRegressor and Dropout function

estimator_dropout = KerasRegressor(build_fn=model_dropout, epochs=200,batch_size=5,
                                   verbose=0)
```

In [39]:
```
# fix random seed for reproducibility
seed = 7
```

In [40]:
```
numpy.random.seed(seed)

# Define the grid search parameters for dropout function (weight regularization)
# Used the scoring parameter to get the model with highest negative score

weight_constraint = [1, 2, 3, 4, 5]
dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
param_grid_dropout = dict(dropout_rate=dropout_rate,
                          weight_constraint=weight_constraint)
grid_dropout = GridSearchCV(estimator=estimator_dropout,
                            param_grid=param_grid_dropout, n_jobs=1,
                            scoring='neg_mean_squared_error')
grid_result_dropout = grid_dropout.fit(X, Y)
```

In [41]:
```
# Summarize results for dropout function

print("Best: %f using %s" % (grid_result_dropout.best_score_,
                             grid_result_dropout.best_params_))
means_dropout = grid_result_dropout.cv_results_['mean_test_score']
stds_dropout = grid_result_dropout.cv_results_['std_test_score']
params_dropout = grid_result_dropout.cv_results_['params']
for mean, stdev, param in zip(means_dropout, stds_dropout, params_dropout):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -17.392723 using {'dropout_rate': 0.1, 'weight_constraint': 5}
-32.966154 (15.636505) with: {'dropout_rate': 0.0, 'weight_constraint': 1}
-31.043575 (5.903125) with: {'dropout_rate': 0.0, 'weight_constraint': 2}
-28.095099 (7.262361) with: {'dropout_rate': 0.0, 'weight_constraint': 3}
-25.866802 (3.072653) with: {'dropout_rate': 0.0, 'weight_constraint': 4}
-21.920476 (4.024113) with: {'dropout_rate': 0.0, 'weight_constraint': 5}
-30.576785 (12.908142) with: {'dropout_rate': 0.1, 'weight_constraint': 1}
-35.031940 (11.948357) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
-35.773558 (12.155848) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
-23.830563 (1.139947) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
```

```
-17.392723 (8.410557) with: {'dropout_rate': 0.1, 'weight_constraint': 5}
-36.605834 (7.835836) with: {'dropout_rate': 0.2, 'weight_constraint': 1}
-28.885011 (12.319658) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
-32.836470 (6.899735) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
-26.041841 (4.616170) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
-29.103014 (8.324877) with: {'dropout_rate': 0.2, 'weight_constraint': 5}
-38.951418 (8.470975) with: {'dropout_rate': 0.3, 'weight_constraint': 1}
-25.997867 (5.665262) with: {'dropout_rate': 0.3, 'weight_constraint': 2}
-23.519096 (6.208654) with: {'dropout_rate': 0.3, 'weight_constraint': 3}
-28.069157 (4.785519) with: {'dropout_rate': 0.3, 'weight_constraint': 4}
-26.807717 (7.976013) with: {'dropout_rate': 0.3, 'weight_constraint': 5}
-37.637311 (6.062809) with: {'dropout_rate': 0.4, 'weight_constraint': 1}
-32.813395 (6.569560) with: {'dropout_rate': 0.4, 'weight_constraint': 2}
-29.623846 (8.422546) with: {'dropout_rate': 0.4, 'weight_constraint': 3}
-27.491937 (7.418057) with: {'dropout_rate': 0.4, 'weight_constraint': 4}
-26.035676 (4.728212) with: {'dropout_rate': 0.4, 'weight_constraint': 5}
-41.587198 (8.409141) with: {'dropout_rate': 0.5, 'weight_constraint': 1}
-30.550123 (6.452710) with: {'dropout_rate': 0.5, 'weight_constraint': 2}
-27.557603 (6.673912) with: {'dropout_rate': 0.5, 'weight_constraint': 3}
-27.569119 (3.325789) with: {'dropout_rate': 0.5, 'weight_constraint': 4}
-25.104352 (4.515486) with: {'dropout_rate': 0.5, 'weight_constraint': 5}
-38.432861 (8.306065) with: {'dropout_rate': 0.6, 'weight_constraint': 1}
-36.564111 (6.255338) with: {'dropout_rate': 0.6, 'weight_constraint': 2}
-30.048439 (5.208390) with: {'dropout_rate': 0.6, 'weight_constraint': 3}
-28.545987 (1.726999) with: {'dropout_rate': 0.6, 'weight_constraint': 4}
-33.787047 (7.019074) with: {'dropout_rate': 0.6, 'weight_constraint': 5}
-37.550358 (7.509365) with: {'dropout_rate': 0.7, 'weight_constraint': 1}
-34.942625 (6.780524) with: {'dropout_rate': 0.7, 'weight_constraint': 2}
-35.225021 (5.315773) with: {'dropout_rate': 0.7, 'weight_constraint': 3}
-33.550401 (10.068285) with: {'dropout_rate': 0.7, 'weight_constraint': 4}
-30.780016 (8.609422) with: {'dropout_rate': 0.7, 'weight_constraint': 5}
-39.224273 (9.284902) with: {'dropout_rate': 0.8, 'weight_constraint': 1}
-39.265954 (10.354145) with: {'dropout_rate': 0.8, 'weight_constraint': 2}
-36.924890 (9.222685) with: {'dropout_rate': 0.8, 'weight_constraint': 3}
-37.478236 (13.299884) with: {'dropout_rate': 0.8, 'weight_constraint': 4}
-37.298563 (9.109105) with: {'dropout_rate': 0.8, 'weight_constraint': 5}
-38.871007 (8.817487) with: {'dropout_rate': 0.9, 'weight_constraint': 1}
-39.140755 (9.617464) with: {'dropout_rate': 0.9, 'weight_constraint': 2}
-40.147926 (7.933324) with: {'dropout_rate': 0.9, 'weight_constraint': 3}
-37.152418 (7.899911) with: {'dropout_rate': 0.9, 'weight_constraint': 4}
-36.750629 (8.146644) with: {'dropout_rate': 0.9, 'weight_constraint': 5}
```

In [42]:
```
# Loading required libraries
```

In [43]:
```
import numpy
import pandas
from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [44]:
```
from sklearn.datasets import load_boston

# Loading Boston dataset

X, Y = load_boston(return_X_y=True)
Y=Y.reshape(506,1)
X = X.astype(float)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(99, 13)
(99, 1)
```

In [45]:
```
# Question 3a
# Defining Base model
# Best parameters from Question 2 are as follows:
    # a) Optimizer - Adamax
    # b) Learning Rate - 0.1
    # c) Activation - softsign
    # d) Dropout - 0.1
    # e) Weight Constraint - 5

def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, kernel_initializer='normal',
                    activation='softsign',kernel_constraint=maxnorm(5)))
    model.add(Dropout(0.1))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    optimizer = Adamax(lr=0.1)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [46]:
```
# fix random seed for reproducibility
seed = 7
```

In [47]:
```
# Creating regression model using KerasRegressor for baseline
```

```
estimator = KerasRegressor(build_fn=baseline_model, nb_epoch=200, batch_size=5,
                           verbose=0)
```

In [48]:
```
# Evaluating baseline mdoel using 10-fold cross validation
```

In [49]:
```
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(estimator, X, Y, cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Results: 38.89 (31.78) MSE
```

In [50]:
```
from sklearn.datasets import load_boston

# Loading Boston dataset

X, Y = load_boston(return_X_y=True)
Y=Y.reshape(506,1)
X = X.astype(float)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)

# Standardizing

scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
(99, 13)
(99, 1)
```

In [51]:
```
# Evaluate model with standardized dataset

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=baseline_model,
                                         epochs=200, batch_size=5,
                                         verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Standardized: 12.93 (8.78) MSE
```

In [52]:
```
# Question 3b
# Define the deeper model
# Increasing depth of model by adding two more hidden layers

def larger_model():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, kernel_initializer='normal',
                    activation='softsign',
                    kernel_constraint=maxnorm(5)))
    model.add(Dropout(0.1))
    model.add(Dense(6, kernel_initializer='normal', activation='softsign',
                    kernel_constraint=maxnorm(5)))
    model.add(Dropout(0.1))
    model.add(Dense(6, kernel_initializer='normal', activation='softsign',
                    kernel_constraint=maxnorm(5)))
    model.add(Dropout(0.1))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    optimizer = Adamax(lr=0.1)
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [53]:
```
# Evaluate deeper network

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=larger_model,
                                         epochs=200, batch_size=5,
                                         verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Larger: 9.69 (8.61) MSE
```

In [54]:
```
# Question 3c
# Define wider model (Increasing width of network by increasing number of neurons)

def wider_model():
    # create model
    model = Sequential()
    model.add(Dense(25, input_dim=13, kernel_initializer='normal',
                    activation='softsign',
                    kernel_constraint=maxnorm(5)))
```

```
        model.add(Dropout(0.1))
        model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
        optimizer = Adamax(lr=0.1)
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [55]:
```
#Evaluate wider network

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model,
                                         epochs=200, batch_size=5,
                                         verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Wider: 8.94 (5.80) MSE
```

In [56]:
```
# Answer 3a: Running the example provides an improved performance
#            over the baseline model without standardized data,
#            dropping the error (MSE) from 38.89 to 12.93.

# Answer 3b: Running the deeper network model which had two more hidden layers
#             does show a further improvement in performance,
#             by dropping the error (MSE) from 12.93 to 9.69.

# Answer 3c: Running the wider network with had an increased number of
#             neurons does show a further improvement in performance,
#             by dropping the error (MSE) from 9.69 to 8.94.
```

In [ ]: