In [ ]:
```
#PART 1
```

In [1]:
```
import numpy
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
```

```
Using TensorFlow backend.
```

In [2]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [3]:
```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
```

In [4]:
```
#Define baseline model
def baseline_model():
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='relu'))
        model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
        model.compile(loss='mean_squared_error', optimizer='adam')
        return model
```

In [5]:
```
# regressor model
est = KerasRegressor(build_fn=baseline_model, nb_epoch=100, batch_size=5,verb
ose=0)
```

In [6]:
```
#Cross-Validation
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(est, X, Y, cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Results: 57.77 (42.26) MSE
```

In [ ]:

```
#PART 2 - Grid Search Technique to find best activation function
```

In [1]:

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
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 [2]:

```
#Define baseline model
def baseline_model(activation='relu'):
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion=activation))
        model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
        model.compile(loss='mean_squared_error', optimizer='adam')
        return model
```

In [3]:

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [4]:
```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
(99, 13)
(99, 1)
```

In [5]:
```
# create model
model = KerasRegressor(build_fn=baseline_model, epochs=100, batch_size=5, ver
bose=0)
```

In [6]:
```
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', '
hard_sigmoid', 'linear']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X, Y)
```

In [7]:
```
# summarize results
#print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
415.863981 (70.121119) with: {'activation': 'softmax'}
19.527457 (3.958270) with: {'activation': 'softplus'}
83.766124 (24.456973) with: {'activation': 'softsign'}
22.323467 (2.538007) with: {'activation': 'relu'}
85.373221 (25.201355) with: {'activation': 'tanh'}
137.666736 (26.116214) with: {'activation': 'sigmoid'}
193.424570 (92.923065) with: {'activation': 'hard_sigmoid'}
22.507993 (3.867221) with: {'activation': 'linear'}
```

In [ ]:
```
#From the above MSE values it can be seen that 'softplus' function is the best.
```

In [ ]:
```
#PART 2 - Grid Search Technique to find best optimizer
```

In [8]:
```
# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
```

In [9]:
```
#Define baseline model
def baseline_model(optimizer='adam'):
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='softplus'))
        model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [10]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [11]:

```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
(99, 13)
(99, 1)
```

In [12]:

```
# create model
model = KerasRegressor(build_fn=baseline_model, epochs=100, batch_size=5, ver
bose=0)
```

In [13]:

```
# define the grid search parameters
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nada
m']
param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X, Y)
```

In [14]:
```
# summarize results
# print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_para
ms_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
40.416685 (9.559488) with: {'optimizer': 'SGD'}
22.124502 (1.747780) with: {'optimizer': 'RMSprop'}
22.753078 (5.316167) with: {'optimizer': 'Adagrad'}
22.673347 (2.796030) with: {'optimizer': 'Adadelta'}
22.251958 (3.891592) with: {'optimizer': 'Adam'}
19.284509 (4.030259) with: {'optimizer': 'Adamax'}
23.695522 (7.980124) with: {'optimizer': 'Nadam'}
```

In [ ]:
```
#From the above values, it can be noticed that 'Adamax' is the best optimizer
.
```

In [ ]:
```
#PART 2 - Grid Search Technique to find optimal learning rate hyperparameter
#Since Adamax was identified as the best optimizer, we require only best lear
ning rate parameter
```

In [29]:
```
# Use scikit-learn to grid search the batch size and epochs
import numpy
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 [30]:
```
#Define baseline model
def baseline_model(learn_rate=0.01):
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='softplus'))
        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 [31]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [32]:
```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
(99, 13)
(99, 1)
```

In [33]:
```
# create model
model = KerasRegressor(build_fn=baseline_model, epochs=100, batch_size=5, ver
bose=0)
```

In [34]:
```
# define the grid search parameters
learn_rate = [0.1, 0.2, 0.3,0.01,0.02,0.03,0.05,0.06,0.07,0.08,0.09]
#momentum = [0.6, 0.8, 0.9]
#param_grid = dict(learn_rate=learn_rate,momentum=momentum)
#grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
param_grid = dict(learn_rate=learn_rate)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X, Y)
```

In [35]:
```
# summarize results
# print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_para
ms_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
31.589111 (9.194897) with: {'learn_rate': 0.1}
44.498794 (9.792447) with: {'learn_rate': 0.2}
26.142535 (10.741754) with: {'learn_rate': 0.3}
22.742258 (2.888129) with: {'learn_rate': 0.01}
22.329863 (6.765204) with: {'learn_rate': 0.02}
19.416912 (6.155138) with: {'learn_rate': 0.03}
19.808712 (4.104037) with: {'learn_rate': 0.05}
23.570049 (5.805230) with: {'learn_rate': 0.06}
30.887598 (15.601536) with: {'learn_rate': 0.07}
19.235231 (3.554508) with: {'learn_rate': 0.08}
28.944604 (5.923522) with: {'learn_rate': 0.09}
```

In [ ]:
```
#Learning rate of 0.08 gives best result.
```

In [ ]:

```
#PART 2 - Grid Search Technique to find dropout regularization and weight con
straints
```

In [34]:

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
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.constraints import maxnorm
from keras.layers import Dropout
from keras.optimizers import Adamax
```

In [35]:

```
#Define baseline model
def baseline_model(dropout_rate=0.0, weight_constraint=0):
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='relu', kernel_constraint=maxnorm(weight_constraint)))
        model.add(Dropout(dropout_rate))
        model.add(Dense(1, kernel_initializer='normal'))
        # Compile model
        optimizer = Adamax(lr=0.08)
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [36]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [37]:
```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
X=X[1:100,:]
Y=Y[1:100,:]
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
(99, 13)
(99, 1)
```

In [38]:
```
# create model
model = KerasRegressor(build_fn=baseline_model, epochs=100, batch_size=5, ver
bose=0)
```

In [39]:
```
weight_constraint = [1, 2, 3, 4, 5]
dropout_rate = [0.0, 0.1, 0.2]
param_grid = dict(dropout_rate=dropout_rate, weight_constraint=weight_constra
int)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X, Y)
```

In [40]:
```
# summarize results
# print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_para
ms_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
27.234118 (8.877480) with: {'dropout_rate': 0.0, 'weight_constraint': 1}
30.881047 (8.905744) with: {'dropout_rate': 0.0, 'weight_constraint': 2}
21.515676 (5.532413) with: {'dropout_rate': 0.0, 'weight_constraint': 3}
34.859391 (19.812894) with: {'dropout_rate': 0.0, 'weight_constraint': 4}
21.071005 (6.398491) with: {'dropout_rate': 0.0, 'weight_constraint': 5}
28.336629 (6.837541) with: {'dropout_rate': 0.1, 'weight_constraint': 1}
21.163500 (4.560569) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
18.421852 (3.554014) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
20.813800 (5.306052) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
19.119837 (4.824558) with: {'dropout_rate': 0.1, 'weight_constraint': 5}
20.695756 (4.552547) with: {'dropout_rate': 0.2, 'weight_constraint': 1}
24.144506 (6.363927) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
20.041272 (6.059283) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
26.856402 (5.486063) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
30.282504 (15.833308) with: {'dropout_rate': 0.2, 'weight_constraint': 5}
```

In [ ]:
```
#Weight Constraint = 3 and Dropout rate = 0.1 gives best results.
```

In [ ]:
```
#PART 3
```

In [4]:
```
import numpy
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 keras.optimizers import Adamax
from keras.constraints import maxnorm
from keras.layers import Dropout
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
Using TensorFlow backend.
```

In [5]:
```
#loading dataset
from sklearn.datasets import load_boston
data,target = load_boston(return_X_y = True)
Y = target.reshape(506,1)
X = data.astype(float)
print (X.shape)
print (Y.shape)
```

```
(506, 13)
(506, 1)
```

In [ ]:
```
#The model is being defined by using the combination of best hyperparameters
thats were identified as a part of PART2
#Activation function - softplus
#Optimizer - Adamax
#Learning Rate = 0.08
#Dropout Rate = 0.1
#Weight Constraint = 3
```

In [3]:
```
#Define baseline model
def baseline_model():
    # create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='softplus', kernel_constraint=maxnorm(3)))
        model.add(Dropout(0.1))
        model.add(Dense(1, kernel_initializer='normal', activation='softplus'
))
        # Compile model
        optimizer = Adamax(lr=0.08)
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [6]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [5]:
```
# regressor
est = KerasRegressor(build_fn=baseline_model, nb_epoch=100, batch_size=5,verb
ose=0)
```

In [6]:
```
#Cross-Validation - model evaluation of non-standardized dataset but with bes
t hyperparameter combination
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(est, X, Y, cv=kfold)
print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Results: 52.23 (41.18) MSE
```

In [7]:
```
# PART 3A model evaluation using standardized dataset and with best hyperpara
meter combination
numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=baseline_model, epochs=100,
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: 20.46 (25.91) MSE
```

In [8]:
```
#PART 3B Increasing depth of the Network by 2 layers
def model_depth():
        #create model
        model = Sequential()
        model.add(Dense(13, input_dim=13, kernel_initializer='normal', activa
tion='softplus', kernel_constraint=maxnorm(3)))
        model.add(Dropout(0.1))
        model.add(Dense(10, kernel_initializer='normal', activation='softplus
'))
        model.add(Dense(6, kernel_initializer='normal', activation='softplus'
))
        model.add(Dense(1, kernel_initializer='normal', activation='softplus'
))
        # Compile model
        optimizer = Adamax(lr=0.08)
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [9]:
```
#Evaluate model with more depth in the network
numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=model_depth, epochs=100, ba
tch_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("model_depth: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
model_depth: 23.46 (23.55) MSE
```

In [7]:
```
# PART 3C Increasing width of the Network
def model_wider():
        #create model
        model = Sequential()
        model.add(Dense(20, input_dim=13, kernel_initializer='normal', activa
tion='softplus', kernel_constraint=maxnorm(3)))
        model.add(Dropout(0.1))
        model.add(Dense(1, kernel_initializer='normal', activation='softplus'
))
        # Compile model
        optimizer = Adamax(lr=0.08)
        model.compile(loss='mean_squared_error', optimizer=optimizer)
        return model
```

In [8]:
```
#Evaluate model with more width in the network
numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=model_wider, epochs=100, ba
tch_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("model_wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
model_wider: 20.70 (23.62) MSE
```

In [ ]:
```
#Error of 52.23 for the model that did not use standardized dataset but best parameter combination, this is
#better than the 57.77 error we got from the initial model that was defined
#PART 3A - On using standardized datset the error reduced by more than half giving 20.46 MSE
#PART 3B - On increasing the depth of the network by increasing number of hidden layers, a slight increase in error
#can be noticed, it becomes 23.46
#PART 3C - On widening the network the error again becomes 20.70 which is almost same as the model which had just
#one hidden layer and width same as before.
```