hyperparameterized_model

May 28, 2020

[1]:

```
import itertools
import numpy as np
import pandas as pd
# for data scaling and splitting
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# for neural net
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# for evaluation
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:

```
data = pd.read_csv("data/combined_expression.csv")
data.head()
```

[2]:

```
   CELL_LINE_NAME  cluster    TSPAN6      TNMD       DPM1     SCYL3  C1orf112  \
0         1240123        2  8.319417  3.111183   9.643558  4.757258  3.919757
1         1240131        1  7.611268  2.704739  10.276079  3.650299  3.481567
2         1240132        1  7.678658  2.845781  10.180954  3.573048  3.431235
3         1240134        1  3.265063  3.063746  10.490285  3.340791  3.676912
4         1240140        1  7.090138  2.988043  10.264692  4.119555  3.432585

        FGR       CFH     FUCA2  …   C6orf10   TMEM225    NOTCH4      PBX2  \
0  3.602185  3.329644  9.076950  …  3.085394  3.462811  3.339030  4.614897
1  3.145538  3.565127  7.861068  …  2.801456  2.985889  3.180068  5.415729
2  3.090781  4.116643  8.121190  …  2.934962  2.952937  3.164655  5.707506
3  3.512821  3.873922  8.790851  …  3.041839  3.398847  3.106710  5.773963
4  3.308033  3.318371  6.927761  …  3.028787  3.225982  3.275820  5.334283

       AGER      RNF5    AGPAT1    DFNB59     PRRT1     FKBPL
0  3.395845  3.419193  3.971646  3.729310  3.320022  6.447316
1  3.299858  3.028414  3.877889  3.911516  3.379405  4.729557
2  3.434295  2.961345  4.272194  3.085696  3.002557  5.653588
3  3.412641  3.136110  4.422262  3.522122  3.509437  5.953242
4  3.864678  3.259242  3.840581  5.809553  3.674587  5.577503
```

```
[5 rows x 16384 columns]
```

[3]:
```
data.shape
```

[3]:
```
(541, 16384)
```

[4]:
```
selected_genes = pd.read_csv('cleaned/boruta.csv')
selected_genes = selected_genes.values.tolist()
selected_genes = list(itertools.chain(*selected_genes))
```

[5]:
```
# retrieving proper columns
X = data.loc[:, selected_genes]
y = data['cluster'].values

# scaling the data
scalar = MinMaxScaler()
x_scaled = scalar.fit_transform(X)

# splitting data (20% test, 80% train)
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2,␣
↪random_state=0)
```

1 Gridsearch for Input and Output Layer

[8]:
```
def create_model(optimizer='adam', init='normal', dropout=0.3):
    model = Sequential()
    # adding layers and adding droplayers to avoid overfitting
    hidden_layers = len(selected_genes)

    # first hidden layer
    model.add(Dense(hidden_layers, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(dropout))
    # second hidden layer
    model.add(Dense((hidden_layers*1.5), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(dropout))
    # third hidden layer
    model.add(Dense((hidden_layers), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(dropout))
    # fourth hidden layer
    model.add(Dense((hidden_layers*0.25), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(dropout))
```

```
    model.add(Dense(1, activation='sigmoid'))
    # compiling
    model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
↪metrics=['accuracy'])
    return model
```

[9]:
```
model = KerasClassifier(build_fn=create_model)
# parameters
epochs = [25, 50, 75]
batches = [16, 32, 64]
optimizers = ['sgd', 'adagrad', 'adam']
init = ['normal', 'uniform']
# grid search
param_grid = dict(epochs=epochs, batch_size=batches, optimizer=optimizers,␣
↪init=init)
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=1,␣
↪n_jobs=-1)
grid_result = grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 162 out of 162 | elapsed: 12.0min finished
```

```
Train on 432 samples
Epoch 1/50
432/432 [==============================] - 5s 10ms/sample - loss: 0.5902 -
accuracy: 0.7060
Epoch 2/50
432/432 [==============================] - 0s 724us/sample - loss: 0.4992 -
accuracy: 0.7986
Epoch 3/50
432/432 [==============================] - 0s 721us/sample - loss: 0.4154 -
accuracy: 0.8287
Epoch 4/50
432/432 [==============================] - 0s 714us/sample - loss: 0.4150 -
accuracy: 0.8403
Epoch 5/50
432/432 [==============================] - 0s 733us/sample - loss: 0.3976 -
accuracy: 0.8333
Epoch 6/50
432/432 [==============================] - 0s 718us/sample - loss: 0.3561 -
accuracy: 0.8449
Epoch 7/50
432/432 [==============================] - 0s 719us/sample - loss: 0.3740 -
accuracy: 0.8495
```

```
Epoch 8/50
432/432 [==============================] - 0s 725us/sample - loss: 0.3642 -
accuracy: 0.8542
Epoch 9/50
432/432 [==============================] - 0s 745us/sample - loss: 0.3584 -
accuracy: 0.8472
Epoch 10/50
432/432 [==============================] - 0s 731us/sample - loss: 0.3800 -
accuracy: 0.8403
Epoch 11/50
432/432 [==============================] - 0s 736us/sample - loss: 0.3470 -
accuracy: 0.8727
Epoch 12/50
432/432 [==============================] - 0s 752us/sample - loss: 0.3610 -
accuracy: 0.8542
Epoch 13/50
432/432 [==============================] - 0s 760us/sample - loss: 0.2887 -
accuracy: 0.8981
Epoch 14/50
432/432 [==============================] - 0s 766us/sample - loss: 0.3572 -
accuracy: 0.8681
Epoch 15/50
432/432 [==============================] - 0s 754us/sample - loss: 0.3068 -
accuracy: 0.8588
Epoch 16/50
432/432 [==============================] - 0s 776us/sample - loss: 0.3170 -
accuracy: 0.8727
Epoch 17/50
432/432 [==============================] - 0s 777us/sample - loss: 0.3144 -
accuracy: 0.8843
Epoch 18/50
432/432 [==============================] - 0s 783us/sample - loss: 0.2931 -
accuracy: 0.8819
Epoch 19/50
432/432 [==============================] - 0s 784us/sample - loss: 0.2967 -
accuracy: 0.8958
Epoch 20/50
432/432 [==============================] - 0s 786us/sample - loss: 0.2830 -
accuracy: 0.8912
Epoch 21/50
432/432 [==============================] - 0s 783us/sample - loss: 0.2809 -
accuracy: 0.8843
Epoch 22/50
432/432 [==============================] - 0s 813us/sample - loss: 0.2871 -
accuracy: 0.8889
Epoch 23/50
432/432 [==============================] - 0s 787us/sample - loss: 0.2936 -
accuracy: 0.8889
```

```
Epoch 24/50
432/432 [==============================] - 0s 800us/sample - loss: 0.2943 -
accuracy: 0.8843
Epoch 25/50
432/432 [==============================] - 0s 799us/sample - loss: 0.2786 -
accuracy: 0.8981
Epoch 26/50
432/432 [==============================] - 0s 829us/sample - loss: 0.3033 -
accuracy: 0.8819
Epoch 27/50
432/432 [==============================] - 0s 847us/sample - loss: 0.2969 -
accuracy: 0.8912
Epoch 28/50
432/432 [==============================] - 0s 826us/sample - loss: 0.2265 -
accuracy: 0.9259
Epoch 29/50
432/432 [==============================] - 0s 832us/sample - loss: 0.2443 -
accuracy: 0.9144
Epoch 30/50
432/432 [==============================] - 0s 824us/sample - loss: 0.2362 -
accuracy: 0.9167
Epoch 31/50
432/432 [==============================] - 0s 836us/sample - loss: 0.2563 -
accuracy: 0.8889
Epoch 32/50
432/432 [==============================] - 0s 834us/sample - loss: 0.2560 -
accuracy: 0.8981
Epoch 33/50
432/432 [==============================] - 0s 824us/sample - loss: 0.2386 -
accuracy: 0.9097
Epoch 34/50
432/432 [==============================] - 0s 835us/sample - loss: 0.2865 -
accuracy: 0.8935
Epoch 35/50
432/432 [==============================] - 0s 856us/sample - loss: 0.2594 -
accuracy: 0.9028
Epoch 36/50
432/432 [==============================] - 0s 837us/sample - loss: 0.2298 -
accuracy: 0.9190
Epoch 37/50
432/432 [==============================] - 0s 821us/sample - loss: 0.2972 -
accuracy: 0.8819
Epoch 38/50
432/432 [==============================] - 0s 828us/sample - loss: 0.2410 -
accuracy: 0.9120
Epoch 39/50
432/432 [==============================] - 0s 825us/sample - loss: 0.2615 -
accuracy: 0.9051
```

```
Epoch 40/50
432/432 [==============================] - 0s 857us/sample - loss: 0.2973 -
accuracy: 0.9074
Epoch 41/50
432/432 [==============================] - 0s 868us/sample - loss: 0.2127 -
accuracy: 0.9120
Epoch 42/50
432/432 [==============================] - 0s 853us/sample - loss: 0.2529 -
accuracy: 0.9028
Epoch 43/50
432/432 [==============================] - 0s 858us/sample - loss: 0.2314 -
accuracy: 0.8981
Epoch 44/50
432/432 [==============================] - 0s 877us/sample - loss: 0.2178 -
accuracy: 0.9259
Epoch 45/50
432/432 [==============================] - 0s 847us/sample - loss: 0.2356 -
accuracy: 0.9120
Epoch 46/50
432/432 [==============================] - 0s 853us/sample - loss: 0.2399 -
accuracy: 0.9074
Epoch 47/50
432/432 [==============================] - 0s 848us/sample - loss: 0.2350 -
accuracy: 0.9282
Epoch 48/50
432/432 [==============================] - 0s 849us/sample - loss: 0.2432 -
accuracy: 0.9144
Epoch 49/50
432/432 [==============================] - 0s 854us/sample - loss: 0.2227 -
accuracy: 0.9259
Epoch 50/50
432/432 [==============================] - 0s 858us/sample - loss: 0.2009 -
accuracy: 0.9167
```

[10]:
```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.881944 using {'batch_size': 32, 'epochs': 50, 'init': 'normal',
'optimizer': 'adagrad'}
```

[ ]:
```
```