code-usage-tutorial

August 17, 2023

This document acts as a tutorial for the reuse of the code in this repository

The tutorial demonstrates how to generate learning curves for the phosphoproteomics data. It can easily be adapted to work with RNA-seq or proteomics data. How this can be done will be signposted in the tutorial.

The tutorial follows the steps below

```
1. Data loading
2. Feature selection
3. Model building
4. Learning curve generation
```

1 Data loading

We read in three omics data types proteomics, ran-sq and phosphoproteomics. We also read in one hot encoded representation of the drugs and ic50 values. These data types are named accordingly.

[ ]:
```
import Data_imports as di_nb
```

[ ]:
```
#input data
prot, rna, phospho_ls, one_hot_cls, one_hot_drugs, ic50_df1 = di_nb.
  ↪read_input_data()
_all_cls = prot.index
_all_drugs = ic50_df1.columns
assert prot.shape[0] == rna.shape[0] == phospho_ls.shape[0]
assert phospho_ls.shape[0]  == one_hot_cls.shape[0]
```

2 Featrue selection (FS) and data createing for each drug

Next, we apply feature selection to the omics data as detailed in the paper.

We then use the create_all_drugs function from the data_preprocessing module to create the data for each drug.

[ ]:
```
from DRP_utils import data_preprocessing as dp_nb
```

2.1 RNA-seq feature selection

[4]:
```
#read in landmark genes for fs and find landmarks that overlap with rna data
landmark_genes = pd.read_csv(
    f'{codebase_path}/downloaded_data_small/landmark_genes_LINCS.txt',
    sep='\t')
landmark_genes.index = landmark_genes['Symbol']

overlapping_landmarks, _ = dp_nb.keep_overlapping(
    pd.DataFrame(landmark_genes['Symbol']), rna.T)

overlapping_landmarks = overlapping_landmarks['Symbol'].values

#create input data for each drug
x_all, x_drug, y_list = dp_nb.create_all_drugs(
    rna[overlapping_landmarks], one_hot_drugs, ic50_df1, _all_cls)

x_all = x_all.astype(np.float32)
x_drug = x_drug.astype(np.float16)

#fmt index to include drug cell line paris
cls_drugs_index = x_all.index + '::' + x_drug.index
x_all.index = cls_drugs_index
x_drug.index = cls_drugs_index
y_list.index = cls_drugs_index

x_all.shape, x_drug.shape, len(y_list)
```

[4]:
```
((11583, 908), (11583, 345), 11583)
```

The above printout shows the shape of the rna-seq omics data, drug representations and truth values (IC50 values). The printout shows that each object has information for the 11583 drug cell line pairs. It shows the rna-seq omics data has 908 features for each drug cell line pair.

2.2 Protomics feature selection

[5]:
```
#use the same landmark genes, that were used for fs for rna datan
#for fs with prot data

#find overlapping landmark genes and prot features
overlapping_landmarks, _ = dp_nb.keep_overlapping(
    pd.DataFrame(landmark_genes['Symbol']), prot.T)

overlapping_landmarks = overlapping_landmarks['Symbol'].values

#create prot data for all drugs
x_all_prot, x_drug, y_list = dp_nb.create_all_drugs(
    prot[overlapping_landmarks], one_hot_drugs, ic50_df1, _all_cls)
```

```
#fmt index to include drug cell line paris
cls_drugs_index = x_all_prot.index + '::' + x_drug.index
x_all_prot.index = cls_drugs_index
y_list.index = cls_drugs_index
x_drug.index = cls_drugs_index

x_all_prot = x_all_prot.astype(np.float32)
x_all_prot.shape
```

[5]:
```
(11583, 721)
```

The above printout shows that the protomics omics data has 721 features for each drug cell line pair.

2.3 Phosphoproteomics feature selection

[14]:
```
landmark_genes = pd.read_csv(
    f'{codebase_path}/downloaded_data_small/landmark_genes_LINCS.txt',sep='\t')
landmark_genes.index = landmark_genes['Symbol']

phos_genes = np.array([c.split('(')[0] for c in phospho_ls.columns])
unique_phos = pd.DataFrame(np.unique(phos_genes))
unique_phos.index = np.unique(phos_genes)
overlapping_landmarks, _ = dp_nb.keep_overlapping(
    pd.DataFrame(landmark_genes['Symbol']), unique_phos)

#read in targets of proteins data
dd_path = '/data/home/wpw035/down_data_small'
enz_sub = pd.read_csv(
    f'{dd_path}/enzsub_relations_phosSite_signor.csv',
    index_col=0)
enz_sub.index = enz_sub['enzyme_genesymbol']

#landmark genes with targets
lands_with_target = [g for g in landmark_genes.index if g in enz_sub.index]
land_targets = set(enz_sub.loc[lands_with_target]['substrate_genesymbol'])

overlapping_land_targets, _ = dp_nb.keep_overlapping(
    pd.DataFrame(land_targets, index=land_targets), unique_phos)

targets_also_landmarks = set(overlapping_land_targets.index).intersection(
    landmark_genes.index)
print(len(targets_also_landmarks))

overlap_inds = []
for gene in targets_also_landmarks:
    overlap_inds.extend(np.where(phos_genes == gene)[0])
print(len(overlap_inds))

kep_cols = phospho_ls.columns[overlap_inds]
x_all_phos, x_drug, y_list = dp_nb.create_all_drugs(
    phospho_ls[kep_cols], one_hot_drugs, ic50_df1, _all_cls)

#fmt index to include drug cell line paris
cls_drugs_index = x_all_phos.index + '::' + x_drug.index
x_all_phos.index = cls_drugs_index
y_list.index = cls_drugs_index
x_drug.index = cls_drugs_index
```

```
186
1064
```

The above printout shows t It shows the protomics omics data has 1064 features for each drug cell line pair.

3 Model building

We build a neural network using the kears API.

[16]:
```
_input_shape=None
def build_cnn_kt(hp):
    if _input_shape == None:
        raise Exception('add input shape dim')
    phos_input = layers.Input(shape=(_input_shape, 1))
    x = layers.Conv1D(
        filters=hp.Int('filts', 8, 32, 8), kernel_size=16,
        activation='relu')(phos_input)
    x = layers.MaxPooling1D(pool_size=2)(x)
    x = layers.Conv1D(
        filters=hp.Int('filts', 8, 32, 8), kernel_size=8, activation='relu')(x)
    x = layers.MaxPooling1D(pool_size=2)(x)
    x = layers.Flatten()(x)
    x = layers.Dense(hp.Int('units', 32, 258, 32), activation='relu')(x)
    x = layers.Dense(hp.Int('units', 32, 258, 32), activation='relu')(x)
    drug_input = layers.Input(shape = (xdrug_train.shape[1]))
    concatenated = layers.concatenate([x, drug_input])
    hidd = layers.Dense(hp.Int('units_hid', 32, 258, 32),␣
 ↪activation='relu')(concatenated)
    hidd = layers.Dense(hp.Int('units_hid', 32, 258, 32),␣
 ↪activation='relu')(hidd)
    output_tensor = layers.Dense(1)(hidd)
    model = tf.keras.Model([phos_input,drug_input], output_tensor)
    opt = tf.keras.optimizers.RMSprop(learning_rate=hp.Choice('lr', [1e-4,␣
↪1e-3]))
    model.compile(
        optimizer=opt,
        loss=tf.keras.metrics.mean_squared_error,
        metrics=['mae'])
    return model
```

[ ]:
```
_train_size  = 0.6 #train size relative to total data set size
lg_space = np.logspace(1, np.log2(64), base=2.0, num=5).astype(int)
lg2 = np.logspace(np.log2(64), np.log2(len(x_all) * _train_size),
                  base=2.0, num=50).astype(int)
lg_space = np.concatenate((lg_space, lg2))
lg_space = np.unique(lg_space)
lg_space
```

4 Learning curve generation

Below we create the learning curves for 30 different test train splits. For each split, we find the test train and validation data.

Then we use the run_lc_ucl function from the Learning_curve module to generate learning curves. This function trains and tests models for each dataset size spesficed for the learning curve, as detailed in the paper.

The parameters taken by this function include the training validation and testing data of the omics type the learning curve is to be run for.

Here this is done for Phosphoproteomics data, simply swap out x_train_phos, x_val_phos and x_test_phos for x_train_rna, x_val_rna and x_test_rna to find the learning curves for rna-seq data. the veriable data_type also needs to be changed from 'Phos_LTL' to 'rna' (this variable controls where the results are saved). The _input_shape verialbe also needs to be changed to x_train_rna.shape[1]

[ ]:
```
import Learning_curve as lc_nb
```

[ ]:
```
#phos
#finds a test train split then finds the learning curve
#for that split. Repeats for mutiple (N) test train splits
N = 30
t1 = time.time()
for run in range(0, N):
    print(f'run {run} of {N}')
    #test train split
    rand_seed = 42 + run
    pairs_with_truth_vals =  y_list.index
    train_pairs, test_pairs, val_pairs = tts_nb.split(
        rand_seed, _all_cls, _all_drugs, pairs_with_truth_vals,
        train_size=_train_size)

    #rna test train selection
    x_train_rna, x_test_rna = x_all.loc[train_pairs], x_all.loc[test_pairs]
    x_val_rna = x_all.loc[val_pairs]
    y_train, y_test = y_list[train_pairs], y_list[test_pairs]
    y_val = y_list[val_pairs]
    xdrug_train, xdrug_test = x_drug.loc[train_pairs], x_drug.loc[test_pairs]
    xdrug_val = x_drug.loc[val_pairs]

    #prot test train selection
    x_train_prot, x_test_prot = x_all_prot.loc[train_pairs], x_all_prot.
↪loc[test_pairs]
    x_val_prot = x_all_prot.loc[val_pairs]


    #phos train test selection
    x_train_phos = x_all_phos.loc[train_pairs]
    x_test_phos = x_all_phos.loc[test_pairs]
    x_val_phos = x_all_phos.loc[val_pairs]

    #consistencey checks

    assert (x_train_prot.index == x_train_rna.index).all()
    assert (x_test_prot.index == x_test_rna.index).all()
    assert (x_val_prot.index == x_val_rna.index).all()

    assert (y_train.index == x_train_rna.index).all()
    assert (y_test.index == x_test_rna.index).all()
    assert (xdrug_test.index == x_test_rna.index).all()

    assert (x_train_phos.index == x_train_rna.index).all()
    assert (x_test_phos.index == x_test_rna.index).all()
    assert (x_val_phos.index == x_val_rna.index).all()

    #inconsistencey checks
    assert x_train_rna.shape[1] != x_train_prot.shape[1]
    assert x_test_rna.shape[1] != x_test_prot.shape[1]
    assert x_val_rna.shape[1] != x_val_prot.shape[1]

    assert x_train_phos.shape[1] != x_train_rna.shape[1]
    assert x_test_phos.shape[1] != x_test_rna.shape[1]
    assert x_val_phos.shape[1] != x_val_rna.shape[1]

    del x_train_rna, x_test_rna, x_val_rna
    del x_train_prot, x_test_prot, x_val_prot
    #---------- Make sure correct dtype -----------------
    data_type = 'Phos_LTL'
    #run the learning curve
    _input_shape = x_train_phos.shape[1]
    mse_r2, bms, bhps = lc_nb.run_lc_ucl(
        build_cnn_kt,
        [x_train_phos, xdrug_train],
        y_train,
        [x_val_phos, xdrug_val],
        y_val,
        [x_test_phos, xdrug_test],
        y_test,
        lg_space,
        num_trails=15,
        epochs=100,
        direc='UCL-del4')

    #save data
    mse_r2.to_csv(f'LC-metric-results/{data_type}/run{run}')

    bhps_df = pd.DataFrame([bhp.values for bhp in bhps])
    bhps_df.to_csv(f'Optimal-hyperparameters/{data_type}/run{run}df')
    with open(f'Optimal-hyperparameters/{data_type}/run{run}.pkl', 'wb') as f:
        pickle.dump(bhps, f)

    scratch_path = '/data/scratch/wpw035/In-use/DRP/rna-phos-intersect-models'
    model_path = f'{scratch_path}/NN-models/{data_type}/
 ↪run{run}model_train_size_'
    for train_size, model in zip(lg_space, bms):
        model.save(model_path + str(train_size))

    np.savetxt(f'train_test_inds/{data_type}/train_inds{run}', y_train.index,
               fmt='%s')
    np.savetxt(f'train_test_inds/{data_type}/test_inds{run}', y_test.index,
               fmt='%s')
    np.savetxt(f'train_test_inds/{data_type}/val_inds{run}', y_val.index,
               fmt='%s')

delt = time.time() - t1
```