Python For Applied Machine Learning - Practical 5

Michael Halstead

May 15, 2024

1 Introduction

Today we will be covering:

- Multilayer perception (neural network) classification with HOG+BoVW and LBP in one feature vector.
- Further code abstraction for your assignment.

2 Neural Network using Sklearn

In this section we will be using *from sklearn.neural_network import MLPClassifier* as our neural network. We will also be using the HOG, BoVW, and LBP feature extraction techniques from previous practicals.

Your argparse function should look something like this:

```
def parse_args():
  parser = argparse.ArgumentParser( description='Extracting command line arguments', add_help=True )
  # parser.add_argument( '--exr', action='store', required=True )
  parser.add_argument( '--dataset', action='store', default='data/texture_snippets/' )
  parser.add_argument( '--classes', '-c', nargs="+", default=[] )
  parser.add_argument( '--split', action='store', type=float, default=0.3 )
  parser.add_argument( '--orient', action='store', type=int, default=8 )
  parser.add_argument( '--ppc', action='store', type=int, default=8 )
  parser.add_argument( '--cpb', action='store', type=int, default=1 )
  parser.add_argument( '--nclusters', action='store', type=int, default=32 )
  parser.add_argument( '--C', action='store', type=float, default=1.0 )
  parser.add_argument( '--radius', action='store', type=int, default=1 )
```

```
  parser.add_argument( '--npoints', action='store', type=int,
    default=8 )
  parser.add_argument( '--nbins', action='store', type=int, default
    =64 )
  parser.add_argument( '--method', action='store', default='default
    ' )
  parser.add_argument( '--hidden', action='store', nargs="+",
    default=[32] )
  parser.add_argument( '--epochs', action='store', type=int,
    default=10 )
  return parser.parse_args()
```

As this is an example of your assignment we will be further abstracting our code to put things in functions and call them in something like the following manner:

```
if __name__ == "__main__":
  flags = parse_args()
  # get the training and evaluation features and labels
  train_feats, train_labels, eval_feats, eval_labels = extract_data
    (flags)
  # set up the mlp
  model = create_mlp(flags, len( np.unique(train_labels )))
  # train the mlp
  model.fit(train_feats, train_labels)
  # evaluate the model
  cls = model.predict(eval_feats)
  probs = model.predict_proba(eval_feats)
  mlp_metrics(eval_labels, cls, probs[:,1])
```

You should notice that we have three functions that we need to create: *extract_data*, *create_mlp*, and *mlp_metrics*. The other functions are based on the output of *create_mlp*. I will now step through the functions you need to create, keep in mind these should be in module files that make sense.

For extracting the data you will use the following pseudo-code:

```
def extract_data(flags):
  split the texture snippets dataset into train and evaluation
    files make sure to return the labels associated with the images
    .
  train the bovw feature extractor
  extract the bovw features for both the training and evaluation
    sets
  extract the lbp features for both the training and evaluation
    sets

  horizontally concatenate the bovw and lbp features
  Normalise the data if you'd like.
  return your training data, training labels, evaluation data,
    evaluation labels
```

You will need to make sure that the features are image specific (i.e. you want the samples to relate to an image), so make sure you sort your files correctly. Next we will create the function to make the neural network.

```
def create_mlp(flags, n_classes):
  get the hidden layers where the last layer is n_classes
```

```
  return the mlp(hidden_layer_size=<your hidden layers>,
                 activation='relu',
                 solver='adam', random_state=1,
                 max_iter=flags.epochs, warm_start=True)
```

This will create your neural network that we will use to classify the images. The final function we will create is the *mlp_metrics*.

```
def mlp_metrics(labels, preds, probs):
 compute and display the accuracy score
 compute and display the confusion matrix
 comute and display the confusion matrix mean of the diagonals
 if there are two classes:
   compute and display the precision recall curve and f1 score
```

So we have further abstracted our code so that only our function calls are in our python script. This makes it much easier to follow code and makes your script neater, this is what's expected of you in the assignment. Run this with two classes and see what you get. Run it with more than two classes and see what you get. Run it with all the classes and see what you get.