Python For Applied Machine Learning - Practical 5

Michael Halstead

May 14, 2024

1 Introduction

Today we will be covering:

- Histogram of oriented gradients with bag of visual words using support vector machines for classification; and
- Local binary patterns with support vector machines with precision recall curves.

In this practical I will be assuming that you have the module files I created in the previous practicals as we will not be reproducing them here.

2 HOG+BoVW using SVMs

In the previous practical we used the HOG feature descriptor and created a bag of visual worlds clustering technique. We looked at the output of this to see if we could discern anything from them. In this practical we will use these BoVW features with support vector machines (SVMs) to classify our texture images.

I will now cover some information about SVMs. Support vector machines are a seminal part of traditional machine learning, we will use *sklearn.svm.SVC*. The basic concept is that they draw a hyper-plane around the data using various hyper-parameters and support vectors. They also employ the kernel trick outlined in the lecture and if you would like to understand SVM's from first principles I recommend:
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Let's assume that we are going to classify the data in Figure 1, two class Gaussian distributions We have seen this type of data throughout the practicals, and what we see here can be extended into multiple classes (as we will see in the practical). But for explanation I will stick with the two classes.

We will be covering two types of kernels to use in our SVMs: linear, and radial basis function (RBF) (polynomial also exists and I encourage you to play

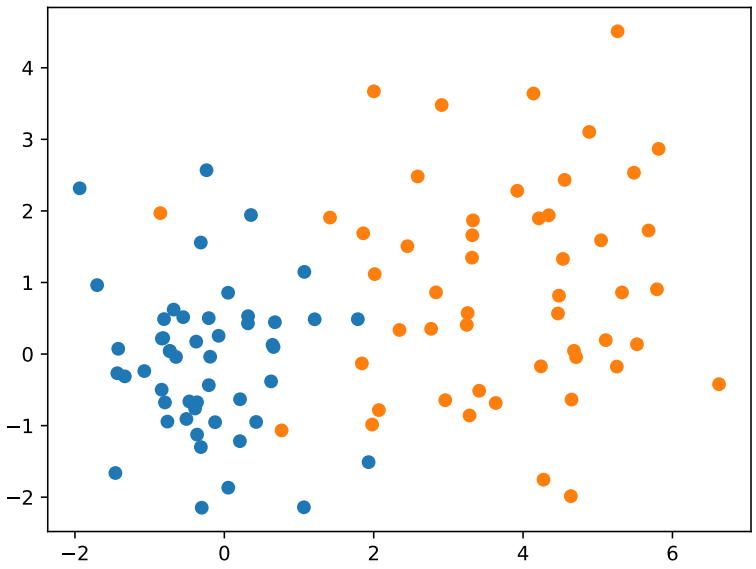

Figure 1: The input data being used to train an SVM.

with this in your own time). The first step after choosing a kernel are the hyper-parameters. For the linear kernel we have a single hyper-parameter C, for RBF we have two, the same C and *gamma*. In all our experiments we will be using the default values, which are $C = 1.0$ and $gamma =$ "$scale$", you can read more about them here:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

If you choose to use SVMs in your assignment you will need to perform a parameter sweep of these hyper-parameters to determine which one is the best. In other words, don't just use the default values, you can see the lecture on parameter sweeps for inspiration.

Essentially, you can somewhat think of C as reflecting how precisely we model our hyper-plane on the training data. In SVMs the hyper-plane is the classification "line", but also the variance away from that classification line. These are illustrated in Figure 2 where the classification line is the single black line and the hyper-planes are the dotted lines. While this isn't strictly the case, we can consider data points inside the hyper-plane as being somewhat uncertain, not an exact description but good enough for this tutorial.

For the C hyper-parameter higher values have two problems, it takes longer to train the hyper-plane and it can significantly over-fit to the training data. Figure 2 outlines two values of C (1.0 and 0.001) you should notice that the left

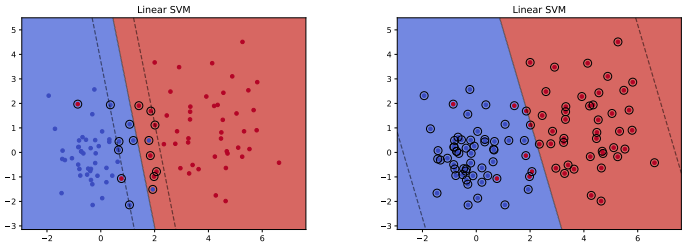

Figure 2: The result of the linear SVM including the hyper-plane and support vectors. The left hand side is using a C of 1.0 and the right hand side a C of 0.001. Note the hyper-plane (dotted lines) and the support vectors (encircled data points). Overall the left had side has an accuracy of 0.96 and the right an accuracy of 0.92.

hand figure fits the data much more precisely, and has a much smaller hyper-plane. This precision, while good for the training data, can be detrimental to our evaluation data as we may not exactly fit the same distribution.

The RBF kernel is a Gaussian kernel that attempts to represent features in an easier dimension for classification. The second hyper-parameter used by RBF, *gamma*, outlines the influence of the support vectors on the shape of the hyper-plane. Large values of *gamma* result in a curvature of the hyper-plane while lower values more represent the linear kernel. Curvature in this case represents once again how accurately we represent the data, where low curvature can result in under-fitting and high curvature can result in over-fitting.

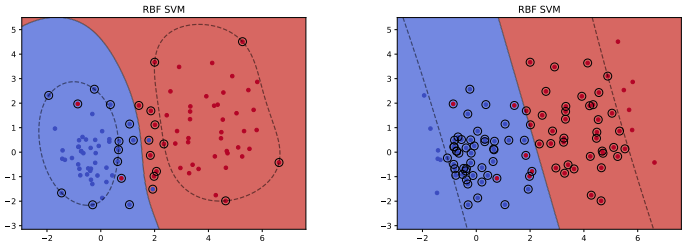

Figure 3: The result of the RBF kernel for SVM training, with hyper-plane and support vectors. The left hand side uses the "scale" *gamma* and the right hand side uses 0.001. Respectively they achieve and accuracy score of 0.96 and 0.94.

Now for the practical itself. You will first split the texture snippets dataset into training and evaluation filenames, where we need the label for each filename (see previous practical). You will then extract HOG features for the training set and fit/train a BoVW clustering object. We then need to extract the BoVW features for each image in both the training and evaluation set, to do this the

following pseudo-code can be used.

```
def get_bovw_features(files, bovw_object, orientations,
    pixels_per_cell, cells_per_block)
  for each file in the files
    extract the HOG features into a N,orientations matrix
    use bovw_object to predict the histogram based on your HOG
    matrix
    store a single feature vector for each image in a matrix (
    vstack).

get the training features
get the evaluation features
```

At this point you can consider normalising your datasets based on the training mean and standard deviation. Try it without this normalisation first and then insert the normalisation and see if it impacts your results.

Next we need to create the SVM objects, to do this we will use *from sklearn.svm import SVC*. You will create an object for both the linear and RBF kernels and fit using the bovw training features. To do the linear SVM we would use *obj=SVC(kernel='linear', C=flags.C* where C is an input to your argparse with a default value of 1.0. You can look up how to perform the same operation for the RBF kernel but for the gamma parameter we will use 'scale' as the default input.

Now we can classify our evaluation dataset: *preds = obj.predict(bovw evaluation data)*. Do this for both the linear and RBF kernels and output both the accuracy score and the confusion matrix for each. Which one performs better? How do you go about testing for different C values? Why can't we use the precision recall curve for this?

3 LBP with SVMs for Precision Recall

In this exercise we will create SVM models for both the linear and RBF kernels, but we will produce decision boundary scores so we can use a precision recall curve, like what's expected in the assignment. First, read in the files and split into training and evaluation sets you can use the previous function here.

Next we will extract the feature vector of LBP

```
def extract_lbp_feature_vector(files, radius=1, npoints=8, method='
    uniform', nbins=10)
  for every file in files
    read in a grayscale image
    perform lbp using the radius, npoints, and method
    extract the histogram of the lbp features
    store vector of features into a matrix.

get training set lbp features
get evaluation set lbp features
```

You can now normalise both your training and evaluation feature vectors based on the mean and standard deviation of the training set. Follow the same structure for the SVMs listed in the previous section to create a linear and RBF

SVM object. Output the accuracy score, the confusion matrix, and the precision recall curve if we only have 2 classes.