Practical 2: Learning Visual Attributes

Overview:

For this practical we were given an open-ended research problem. The problem was taken from a dataset related to objects identified in images. The bits containing by these objects are extrapolated into many different values, such as size, colour histograms, orientated gradient histograms, etc. Using these values, we are to determine the colour and texture of the identified objects.

Approach

Given that there are two outputs (colour and texture) I decided to train two separate instances of the same model for either category. These separate instances have the same parameters.

Pre-Processing

The pre-processing stage is crucial to producing a good model. The first task was to remove some of the attributes. Logically texture and colour are removed and stored as the set targets. Some of the other attributes are also suitable to be removed. The columns “Image” and “ID” were removed as they relate to the image the object was taken from and the ID of the object themselves. These attributes were removed as they do not contain any value for the classifier as they are identification numbers. If they were left in, they would most likely reduce performance, any increase in performance would be superficial and the model would not generalise as well. I explored removing other fields also. I initially removed attributes “X” and “Y” as they related to the position of the object, which should have no relevance to the colour or texture of the object, however at the end of the development process I added them back in to see if it changed the performance and it in fact increased the performance of my model against the provided test set. A possible explanation for this could be that certain objects tend to be in certain parts of images, a trivial example would blue clouds at the top of images. Alternatively, the performance increase could be due to a lack of diversity in the provided sets. I also explored removing other features however the performance difference was negligible I found. I use the training data in as diverse a way as possible.

To test and evaluate my model I split the training data into 4 folds for K-fold cross-validation. After this I then create a new instance of the chosen model and using the same hyperparameters train the model on the whole training set to maximise training. This is then used to predict the training set.

I performed different scaling for each of the categories. For the colour category I used the sklearn Standard Scaler. I chose to use the standard scaler for this task due to my underlying model. The standard scaler scales using the mean and variance of each independent feature. This helps the features become more normally distributed. Having normally distributed features is crucial for my model as I use the radial basis function kernel and L2 regulariser which assume features are centred around 0 and that the variance of each feature is in the same order of magnitude.

The Algorithm

The specific model I have chosen to solve this problem is a Support Vector Machine (SVM). SVMs are an effective algorithm that try to maximise the margin of decision functions to improve the performance of classification. An SVM is suitable for this task as they are good at solving classification problems in high-dimensional space (with kernels) as well as being efficient in classifying multi-class classification problems. I decided to pick this method due to the high-dimensionality of the problem as well as the number of data points being more suitable for SVM than K-Nearest Neighbours. Specifically, I used the Support Vector Classifier (SVC) provided by sklearn with an RBF kernel.

The parameters of the model (and hyperparameters) of the model were chosen with the aid of a parameter finding method that I created (parameter_finder.py). I originally used the sklearn Grid Search Cross Validation Class however I decided to create a similarly purposed script myself to better explore how the results generalised. This script uses arrays for each of the possible parameters I chose to explore (C, Gamma, Class Weight and Decision Function Shape) and creates a model for each possible combination of parameters. The script then outputs the first model and its performance, and then outputs any model that are better than the current best score until every combination has been tried. The performance of each model is evaluated against unseen data using the balanced accuracy as a metric. I chose to use the balanced accuracy as this is essentially the metric used to evaluate the performance of the model on the leader board server. Using this I ended up with model parameters:

Class Weight = Balanced

Decision Function Shape = One Versus One

γ (Gamma) = 1e-05 (Hyperparameter)

C = 100 (Hyperparameter)

Kernel = RBF

The class weight parameter proved to consistently increase the performance of the model comparatively to model with no class balancing. This an expected behaviour as having balanced classes is very likely to increase the performance for balanced accuracy as it is the average recall on each class. Therefore, by forcing the model to be more balanced this will increase this average. The decision function shape relates to how a multi-classification problem can be broken down into several binary classification problems. There are two approaches to this, 'One vs One' (OVO) and 'One vs Rest' (OVR). OVO relates to turning a multiclass problem into $n(n - 1) / 2$ binary classification problems where n is the number of classes. In the context of the colour problem this would look like:

Blue vs Pink

Blue vs Red

Red vs Pink

The other decision function shape, OVR, turns a multiclass problem into n classification problems, where n is the number of classes. In the context of colour this would look like

Red vs. [Blue, Pink, Green, Black, White, etc]

Blue vs. [Red, Pink, Green, Black, White, etc]

The chosen prediction for any given input vector is therefore given by the number of votes cast to each class from each binary classification problem. When using the parameter finding script it was observable that OVO performed better than OVR. Once again this is an expected behaviour as OVO splits the problem into more sub problems. By having more sub-problems the scale of each problem is smaller which can be beneficial for kernel-based algorithms as their performance does not always scale proportionally.

The SVC model provided by sklearn uses a squared L2 penalty for regularization. L2 (ridge) works on means rather than medians (Lasso). This is the only available regularisation for this model. The L2 regularization has paramater C which determines the trade-off between the classification and the margin maximisation. For large C, a small margin is traded off against better training classification and for low C a larger margin is traded off against worse training classification. The best regularisation parameter I found was C=100, however this is, importantly, the best parameter when the SVM's kernels γ hyperparameter is very small.

RBF

The radial basis function is a useful kernel given by:

$$K < X_{1,} X_2 > = e^{-\gamma ||X_1 - X_2||^2}$$

The kernel trick is a highly efficient method for separating data by using high dimensional space, where separability becomes more probable. The RBF kernel works by selecting marker points in the existing feature space and then calculating similarity features that represent the Euclidean distance between these marker points and the existing data points to increase dimensionality. What is incredible about this process is that after the calculation between the features and a point are done, the original feature is no longer needed which is a great advantage of the RBF kernel. A disadvantage of the RBF kernel (and other kernels) is that it does not always scale well to large test sets or a large number of features. The RBF kernel can create decision boundaries that are highly non-linear, and though the output hyperplanes, support vectors and margins are difficult to interpret it can create very useful decision boundaries. As such with a complex high-dimensional problem the Radial Basis Function could produce the best boundaries, which is why I choose it. The linear kernel uses a linear classification boundary and therefore was not suitable for this problem. A polynomial kernel increases dimensionality by using polynomial combinations of existing features up to a specified degree. I chose to use the RBF kernel over polynomial as it can create more non-linear boundaries.

The parameter γ determines the decision boundary of the classifier. The smaller the value of γ the smoother the boundary, preventing overfitting. Conversely, the higher the value of γ the less smooth it becomes, and the model tends toward overfitting the data. The value of γ for my model is 1e-05, which was found to be a good value through use of the parameter finding script. This is a very small value and as such suggests that the resulting decision boundary is very smooth to avoid overfitting and maximise balanced accuracy. It should be noted that this is only a good value when the value of C for L2 is 100. This is because the two are closely related and provide trade-off. Therefore, examining both hyperparameters, the small γ suggests I have a smoother decision boundary which avoids overfitting, and the relatively high value of C means that the margin can be smaller to increase classification accuracy.

Performance Evaluation and Prediction

The evaluation metrics I chose to use are the accuracy score, balanced accuracy, confusion matrices, precision, and recall. The accuracy score is the percentage of correct classifications the model makes from a sample input. This is a useful measure that summarises the accuracy of the model however it can be misleading in the case where the input sample is unvaried or if the sample has a different distribution of classes from the real distribution. Therefore, I have also used the balanced accuracy which is the average of recall across each of the classes. This metric is useful to compare against the normal accuracy measure as it accounts for class imbalance and comparison of the two can be more informative than each metric separately. I also decided to show the report the full confusion matrix from each model. The confusion matrix for a multiclass classification models can be very large with many classes however the number of classes is small enough to report. This is an extremely useful matrix that shows where misclassifications are common which may help the refining process. Finally, I also reported the precision and recall of each class. These metrics have the following formula for each class:

$$Precsision = \frac{true\ positives}{(true\ positives + false\ positives)}$$

$$Recall = \frac{true\ positives}{(true\ positives + false\ negatives)}$$

As shown above the precision is how well the model does not classify a negative sample as positive. Oppositely the recall is how many of the samples the model misses for a class.

The performance reported is the result of a pipeline that creates models for 4-fold cross validation. Pipelines are a useful tool that performs a repeatable, reliable set of steps for model training that can then be cross validated. Cross validation ensures that the resulting evaluation metrics are not a statistical fluke by splitting the data into folds, *k-1* folds are used to train a model, and the kth fold becomes the validation set. This is repeated k times with different validation folds and the results are averaged in order reduce the chances of the reported metrics being a statistical fluke. The pipeline takes the SVM model with my specified parameters and the Standard Scaler as its parameters. Then when the cross-validation function is called on this pipeline, the pipeline performs the scaling of the input and the training of each model for the cross validation.

Colour Metrics

Accuracy Score: 0.41

Balanced Accuracy: 0.2

Balanced Accuracy (data_test.csv): 0.2

Confusion Matrix:

Table 1 - A table showing the confusion matrix of colours for the SVM model.

	Y True											
Colour	**Beige**	**Black**	**Blue**	**Brown**	**Gray**	**Green**	**Orange**	**Pink**	**Purple**	**Red**	**White**	**Yellow**
Beige	0	1	0	2	0	0	0	0	0	1	1	0
Black	2	32	7	28	8	17	0	1	2	3	16	2
Blue	0	10	30	25	18	45	2	2	2	0	54	1
Brown	1	14	7	44	14	24	6	4	2	3	41	3
Gray	1	4	7	17	19	30	2	3	0	2	40	1
Green	1	8	18	67	21	147	3	1	0	2	37	7
Orange	0	1	0	6	0	6	1	1	0	0	5	0
Pink	0	0	1	1	4	3	0	0	0	1	9	0
Purple	0	1	0	0	0	0	0	0	0	0	0	0
Red	0	1	0	6	1	2	0	0	0	1	4	0
White	1	1	17	19	16	20	4	4	0	3	267	6
Yellow	0	0	0	3	0	2	0	1	0	0	2	0

Precision and Recall:

Table 2 - A table of precision and recall values for colour labels.

Colour	**Beige**	**Black**	**Blue**	**Brown**	**Gray**	**Green**	**Orange**	**Pink**	**Purple**	**Red**	**White**	**Yellow**	**Average**
Sample count	6	73	87	218	101	297	18	17	6	16	476	20	
Precision	0	0.27	0.16	0.27	0.15	0.47	0.05	0	0	0.07	0.74	0	0.18
Recall	0	0.43	0.34	0.2	0.19	0.5	0.06	0	0	0.06	0.56	0	0.2

The evaluation metrics for colour illuminate many aspects of the model's performance and its drawbacks. As shown the accuracy score appears to be okay however there is disparity between the accuracy and balanced accuracy. This especially interesting as the model parameters are optimised for balanced accuracy. As shown in table 2 there are large class imbalances and therefore the balanced accuracy can be expected to be poor due to lack of samples for under sampled classes such as beige and black. The balanced accuracy reported and the balanced accuracy for the provided test set are very similar (the report values are rounded) which suggests that the value I report is reliable. The confusion matrix and table 2 show the models ability to identify each class and where the model makes misclassifications. These tables show that some classes are much harder to identify than others. For example, the brown class has many samples however the model has poor precision and recall for this class. As shown in the confusion matrix, brown samples are more commonly misclassified as green than they are brown. Comparatively, white has much higher precision and recall. Furthermore 4 of the classes (beige, pink, purple and yellow) are ever correctly classified, these are the most under sampled classes however which explains why they are hard to classify as there have relatively limited data points. An explanation for the difference in performance between classes such as white and brown is likely be to do with the RGB data. White has RGB value (255,255,255) which makes it very easily identifiable as it has high values for each of these which translates into the features. Conversely brown colours can be achieved with various combinations of red, green, and blue making it much less distinguished. The balanced accuracy reported for the training set is low however above average relative to the class.

Texture Metrics

Accuracy Score: 0.45

Balanced Accuracy: 0.30

Balanced Accuracy (data_test.csv): 0.35

Confusion Matrix:

Table 3 - A table showing the confusion matrix of Textures for the SVM model

	Y True									
Colour	**Barbed**	**Blurry**	**Coarse**	**Crusty**	**Fluffy**	**Fuzzy**	**Grassy**	**Gravel**	**Rippled**	**Smooth**
Barbed	0	3	1	0	1	0	3	1	0	5
Blurry	1	62	1	0	41	14	2	0	2	18
Coarse	0	0	0	0	0	0	0	0	0	1
Crusty	0	4	0	1	2	4	0	0	0	3
Fluffy	1	26	1	2	255	10	8	1	5	25
Fuzzy	0	29	0	3	71	36	12	3	3	21
Grassy	4	18	1	3	58	20	178	10	17	34
Gravel	0	0	0	0	7	3	20	10	1	10
Rippled	0	14	1	0	36	8	41	5	34	19
Smooth	1	14	0	1	39	11	8	1	0	25

Precision and Recall:

Table 4 - A table of precision and recall values for each texture labels.

Colour	**Barbed**	**Blurry**	**Coarse**	**Crusty**	**Fluffy**	**Fuzzy**	**Grassy**	**Gravel**	**Rippled**	**Smooth**	**Average**
Sample count	7	171	5	10	510	106	272	31	62	161	
Precision	0	0.44	0	0.07	0.76	0.2	0.51	0.20	0.22	0.25	0.27
Recall	0	0.36	0	0.1	0.5	0.33	0.65	0.32	0.54	0.16	0.33

By looking at the results of the texture classification the model performs better for texture than model. The accuracy score and balanced accuracy scores are higher and the distance between the two score is smaller for texture than for colour. This shows that texture is easier to classify. The balanced accuracy is much higher for this classification problem. This could be due to having less classes as the data appear to be distributed in a similarly uneven manner to colour. The balanced accuracy reported for the test set is higher than the value reported from cross-validation This could be due to differences in the distribution of classes in the test set. Tables 4 shows that the precision and recall for some textures is passable (fluffy and grassy). This model performs better for small classes than colour however this may be because there are less classes. Tables 3 and 4 shows that again some classes are harder to identify than others. Highly sampled classes such as fluffy and fuzzy have very different performance. A possible explanation for this could be due to the gradient histograms being similar. The more complex the texture the more varied the gradients could be and therefore distinguishing between them could become very difficult. This theory would suggest that the only reason fluffy performs better is because it is highly sampled.

Overall, the algorithm performs rather poorly. The balanced accuracy of both classification problems is very low, if optimised for accuracy instead of balanced accuracy the model can achieve better accuracy however this further reduces the balanced accuracy. Although the model performs poorly relative to other models in the class my model performance is average for colour and above average for texture.

Comparison Against Logistic Regression

For comparison I created two additionally scripts containing simple logistic regression models. I gave these models balanced class weights (the same as SVM). The metrics for the logistic regression

Colour

Accuracy: 0.35682239125352894

Balanced Accuracy: 0.14767062051891774

Precision: 0.14423897764849616

Recall: 0.1475547711449892

Texture

Accuracy: 0.41454478430526633

Balanced Accuracy: 0.22470824434192258

Precision: 0.2202070461516216

Recall: 0.22507844169449517

Comparing these results and the results of the tuned SVM model it is clear to see that the SVM model performs better in every metric. The margin is not large, however. One reason for this model performing worse is that its parameters have not been optimised for the problem in the same way the SVM has been. Logistic regression performs worse because its decision boundary is linear and therefore if the data is inseparable in the dimension it is presented in then logistic regression cannot separate the data well. SVM on the other hand can use highly non-linear decision boundaries that allow for better classification. The accuracy of the model is not completely dissimilar from SVM, this is because, due to class imbalances, the model can get passable performance by favouring the highly sampled classes. This in turn reduces the balanced accuracy. The average precision and average recall are also worse. This is likely because by favouring specific classes the macro average of the model parameter is punished as it doesn't account for class imbalance.

Usage

The scripts can be run using the normal command:

```
Python <file>.py
```

Note that the data_prep.py script is required by each of the other scripts.