Task 1 – Image Classification via boosting

1.1 Algorithm & Adaptions

In this task, I implemented an AdaBoost classifier with decision trees to predict the 3 image classes. Traditionally, however, AdaBoost is a 2-class predictor, necessitating some adaptions to the algorithm to accommodate for the multi-class predictions. I decided to follow a variant of the SAMME algorithm, adding a log(K-1) term to model weights – where K is the number of classes – preventing them from potentially turning negative. The method for combining model predictions was also adapted slightly for the 3-class case, using a weighted mode style calculation for each image.

1.2 Cross Validation

To reduce overfitting, all hyper-parameters and data feature engineering techniques were optimised through 5-fold cross validation (CV) on the provided training set, with an 80-20 train-validation split at each fold. The external test set could then be used (via the auto-marker) to assess the final accuracy of the model.

1.3 Feature Processing & Engineering

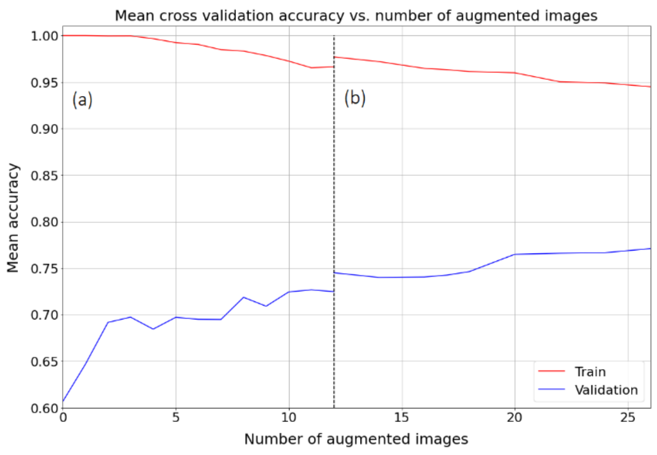
To maximise the information gained from the image dataset, I decided to experiment with different combinations of data extraction and feature processing, comparing their effectiveness via 5-fold cross validation on the training set. Initially, I started off using HoG feature extraction, however, this alone resulted in a low mean cross-validation accuracy (see *Table1*), so I experimented further with grey-scaled pixel features and combined HoG and greyscale features for each image – the latter of which gave the best mean cross-validation results. I believe the reason for this stems from the ability of the decision trees to ignore unnecessary features when choosing splits - so by combining HoG and greyscale features the learners can make use of the extra information and choose better splits.

Feature Extraction	Validation	Train
HoG	54%	100%
Greyscale	51.3%	100%
HoG + Greyscale	57.3%	100%

Table 1. Comparison of mean validation & train accuracies after different feature extraction techniques (each using optimal hyper-parameters), via 5-fold cross validation.

For all the above, features were also standardised to reduce any algorithmic bias - fit on the training set only to prevent data leakage.

As shown in *Table1*, the validation & train set CV accuracies - although slightly closer for combined HoG and greyscale features - are significantly far apart, indicating overfitting in the model. To reduce this, I decided to artificially increase the training set size through image augmentation, starting off by flipping the training images. This provided the model with more varied representations of each object and should, in theory, reduce the extent of overfitting. However, this only resulted in an increase validation accuracy (to 61%) and no decrease in train accuracy, so I decided to further augment images via random variations in brightness, shift and rotation, and test their effect on overfitting as the number of augmented images increased.

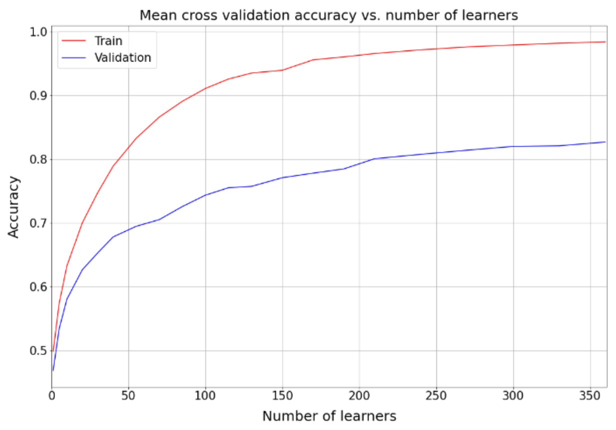


Graph 1. Dependence of mean validation & train accuracies on the number of augmented training images per original. (a) Without max pooling, using 100 learners. (b) With max pooling, using 150 learners. This shows that, of those tested, 26 augmented images act to reduce overfitting the most.

Graph1(a) shows that overfitting is reduced, and validation accuracy improved, as more augmented images are included. However, increasing the training set size requires greater computation time, so to experiment with greater numbers of images, max pooling was used as a form of feature reduction - reducing the images to (128,64) pixels. This allowed for more augmented images to be generated and included in the model, as well as removing redundant pixels. *Graph1(b)* shows how the extent of overfitting was further reduced as more images were added after max pooling.

1.4 Hyper-Parameter Optimisation

Using 26 augmented images with max pooling and combined greyscale and HoG features, the number of learners and maximum tree depth hyper-parameters were then optimised via cross validation on the training set. This was done through an intial random grid search of both hyper-parameters, followed by a finer linear search of each.



Graph 2. Dependence of mean validation & train accuracies on the number of learners (hyper-parameter).

The optimal max. tree depth was found to be 2 – showing that Adaboost prefered weak learners. *Graph2* depicts the mean validation/train accuracies of the linear search across the number of learners - with the best value found at 350.

1.5 Conclusion

The model's performance with optimal hyper-parameters was tested on the test set (via the auto-marker), providing a test accuracy of 53%. However, given the large gap between validation/train accuracies in *Graph2*, I believe the model is still overfitting on the data. Given more time, I would have liked to implement latent variables and sliding windows to identify the location of objects in each image, before proceeding with feature reduction and augmentation. This should help to remove most of the 'background' in the images, which currently acts as noise, and greatly reduce overfitting.

Task 2 – Image Classification via SVMs

2.1 Kernels

For this task, I coded several SVM kernels, optimising their hyper-parameters and comparing their performance via 5-fold CV on the training set. The kernels compared and their formulas are listed below in *Table2*.

Kernel	Formula
Linear	$k(x,y) = x^T y + c$
RBF	$k(x,y) = \exp(-\|x-y\|^2/\sigma^2)$
Sigmoid	$k(x,y) = \tanh(\alpha x^T y + c)$
Cauchy	$k(x,y) = 1/(1+\|x-y\|^2/\sigma^2)$
Histogram Intersection	$k(x,y) = \sum_i \min(x_i, y_i)$
Student's T	$k(x,y) = 1/(1+\|x-y\|^d)$
Log	$k(x,y) = -\log(1+\|x-y\|^d)$

Table 2. Kernels used and their respective covariance matrix calculations.

2.2 Feature Processing & Engineering

The same feature processing techniques and data augmentation as implemented in Task 1 were used here. Although, through CV the optimal number of augmented images when using SVMs proved to be 3.

2.3 Kernel Hyper-Parameter Optimisation

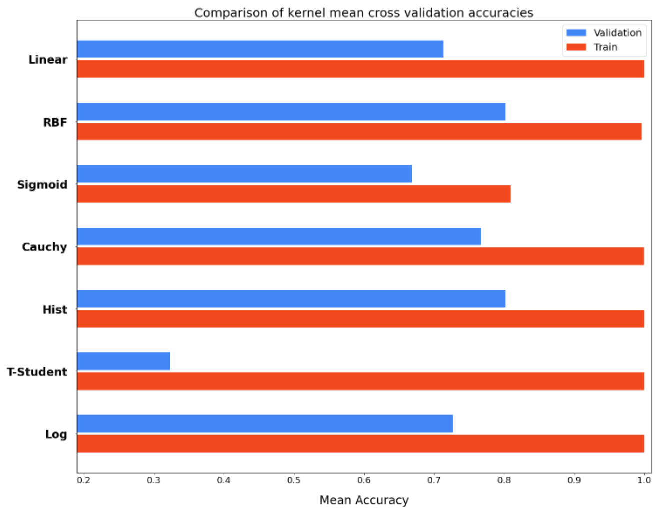
Hyper-parameters for each of the kernels in *Table2* were tuned via linear and grid searches, with these being performed in log-space for RBF, Sigmoid and Cauchy kernels, and integer space for Student's T and Log kernels. These searches were validated via 5-fold CV to reduce overfitting and find the best global values, the results of which are listed in *Table3.*

Table 3. The optimal hyper-parameters found for each kernel (some kernels without hyper-parameters are excluded).

Kernel	Optimal Hyper-Parameters
Linear	$c = 0$
RBF	$\sigma = 159$
Sigmoid	$\sigma = 170, c = 0$
Cauchy	$\sigma = 148$
Student's T	$d = 1$
Log	$d = 2$

2.4 Kernel Performances

The Histogram Intersection kernel provided the highest mean validation accuracy (80.2%), closely followed by the RBF and Cauchy kernels. The Student's T kernel was by far the worst of the tested kernels, strongly overfitting to the training data with validation/train accuracies of 32%/100% respectively. All kernels presented a large difference between validation and training scores, indicating a large degree of overfitting (despite the inclusion of augmented images and max pooling). This suggests that the efforts to reduce overfitting were not effective enough, potentially due to the large degree of background noise within each image. Of all kernels tested, the Sigmoid showed most promising reduction in overfitting, and was the only kernel to provide a training accuracy less than 99% (80.8%). Although, this was at the expense of the mean validation score (66.8%).



Graph 3. Bar chart showing the mean validation & train CV accuracies for each kernel, using their optimised hyper-parameters.

2.5 Conclusion

The model's performance with a Histogram intersection kernel was tested on the test set (via the auto-marker), providing a test accuracy of 59%. Given its lack of hyper-parameters, I had imagined the Histogram kernel would be at a disadvantage to the others, and potentially not perform as well. However, it seems that, due to its simple min(x,y) operation, the Histogram kernel is particularly suited to pixel data.

With more time, I would like to have tried further kernels, such as the Weibull and Wavelet kernels, as well as a more generalised version of the Histogram kernel, where the powers of x_i & y_i can be varied as hyper-parameters. I would also like to have incorporated latent variables (as described in section 1.5) to identify object locations and reduce data noise, which would help to solve the overfitting problem faced by all kernels deployed.

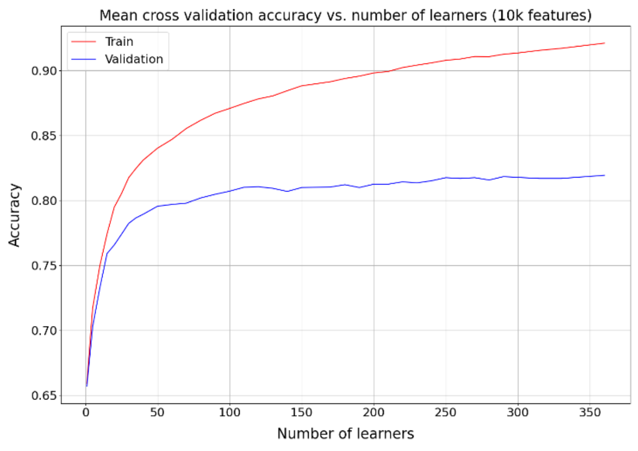
Task 3 – Review Classification via boosting

1.1 Feature Processing & Engineering

To determine the sentiment of each review, I first tokenised each review string, separating out each word as an individual feature. Each review contained a lot of redundant information that is unlikely to affect the sentiment - such as punctuation and words like 'and' or 'the'. Therefore, so I decided to remove all punctuation and common 'stop-words', including words common to movie reviews (e.g. 'cinema') to reduce the noise. In addition, I realised that the feature set could be condensed by grouping together words with different tenses but the same meaning – such as 'shock', 'shocked' and 'shocking'. This was done by passing all tokens through the Lancaster word stemmer. To increase the efficiency of the predictions, I initially vectorised each of the movie reviews, converting the words to indices alongside their respective frequencies. However, I soon went further by converting all review tokens to a TF-IDF format, which effectively weights each token according to both its frequency within each review and number of reviews it appeared in. This should help to reduce the weighting of unimportant tokens that appear in many documents – effectively under-weighting any additional stop-words not previously removed - whilst also increasing the weight of tokens that occur multiple times in each review (since they are more liking to represent the review's sentiment).

1.3 Hyper-Parameter Optimisation

Using the above feature processing, the number of learners and maximum tree depth hyper-parameters were then optimised via cross validation on the training set. This was done through an intial random grid search of both hyper-parameters, followed by a finer linear search of each. The optimal max. tree depth was found to be 2.



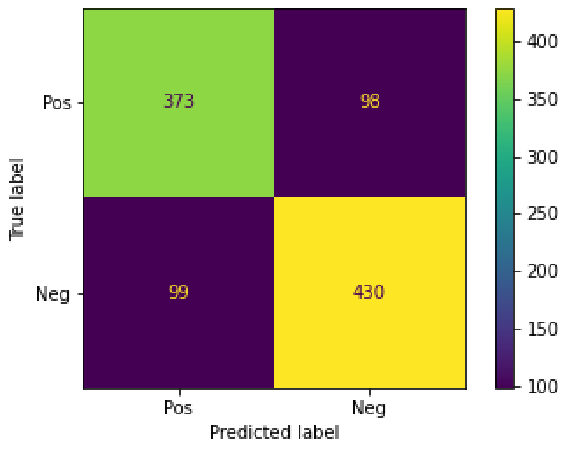
Graph 4. Dependence of mean validation & train accuracies on the number of learners (hyper-parameter).

Graph4 depicts the mean validation/train accuracies of the linear search across the number of learners. Up to around 30 learners the validation accuracy closely follows the training accuracy, and both remain low – indicating underfitting. From around 50 learners, the two scores began at an increasing rate – indicating the onset of overfitting. Taking both under and over fitting into consideration, I determined the optimal number of learners to be 100, since it provides close to the best validation score whilst also not overfitting too much.

1.4 Conclusion

Graph5 shows the confusion matrix for the model's predictions using optimal hyper-parameters. The false-positive and false-negative rates are very similar – indicating little bias across review sentiments.

Graph 5. Confusion matrix showing the mean validation predictions and true values using the RBF kernel.



Task 4 – Review Classification via SVMs

2.1 Kernels

For this task, I coded several SVM kernels, optimising their hyper-parameters and comparing their performance via 5-fold CV on the training set. The kernels compared and their formulas are listed below in *Table4*.

Kernel	Formula
Linear	$k(x,y) = x^T y + c$
RBF	$k(x,y) = \exp\left(-\lVert x-y\rVert^2/\sigma^2\right)$
Sigmoid	$k(x,y) = \tanh(\alpha x^T y + c)$
Cauchy	$k(x,y) = 1/(1+\lVert x-y\rVert^2/\sigma^2)$
Histogram Intersection	$k(x,y) = \sum_i \min(x_i, y_i)$
Spline	$k(x,y) = \prod_i 1 + \mathrm{xy} + \mathrm{xy}\min(x_i,y_i) - \frac{1}{2}(x+y)\min(x_i,y_i)^2 + \frac{1}{3}\min(x_i,y_i)^3$

Table 4. Kernels used and their respective covariance matrix calculations.

2.2 Feature Processing & Engineering

The same feature processing techniques as implemented in Task 3 (tokenisation, stop-word removal, vectorisation and conversion to TF-IDF form) were used here. Although, in addition, to since the kernel calculations required here were comparatively slow to the boosting in task 3, the feature set size was also reduced by taking only the 10k features with the highest TF-IDF scores.

2.3 Kernel Hyper-Parameter Optimisation

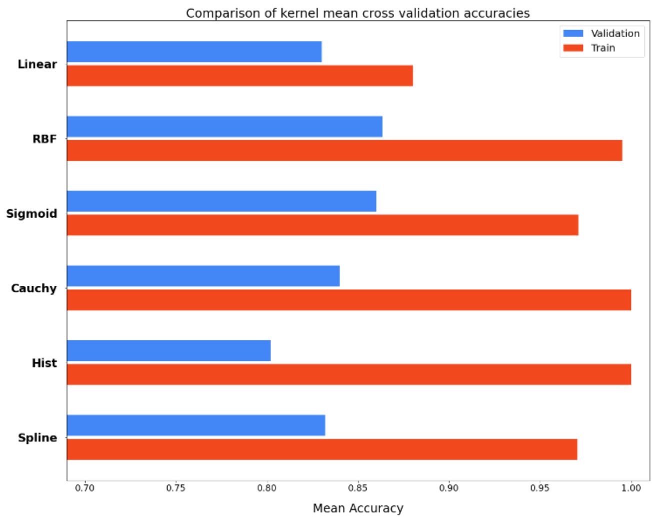
Hyper-parameters for each of the kernels in *Table4* were tuned via linear and grid searches, with these being performed in log-space for RBF, Sigmoid and Cauchy kernels. These searches were validated via 5-fold CV to reduce overfitting and find the best global values, the results of which are listed in *Table5.*

Kernel	Optimal Hyper-Parameters
Linear	$c = 0$
RBF	$\sigma = 0.675$
Sigmoid	$\sigma = 0.68\,, c = 0$
Cauchy	$\sigma = 0.675$

Table 5. The optimal hyper-parameters found for each kernel (some kernels without hyper-parameters are excluded).

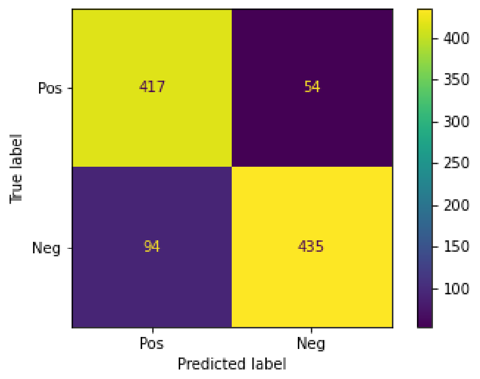
2.4 Kernel Performances

The RBF and Sigmoid kernels provided the highest mean validation accuracies, with the RBF slightly ahead at 86.3%. the Spline and Histogram kernels. When cross-validating the Spline and Histogram kernels, the feature set had to be further reduced to the top 300 features (in terms of TD-IDF scores), as these kernels took longer to run. Due to this, I was surprised by how well the two kernels performed relative to the others – the RBF kernel, for instance experienced a 5% drop in accuracy with 300 features, so in reality, is probably on par with the Spline kernel if runtime is ignored. The Sigmoid kernel showed the least overfitting out of all kernels (giving the closer train and validation scores), and would therefore probably perform well on unseen data.



Graph 6. Bar chart showing the mean validation & train CV accuracies for each kernel, using their optimised hyper-parameters.

4.5 Conclusion



Graph 7. Confusion matrix showing the mean validation predictions and true values using the RBF kernel.

As shown in *Graph7,* the with its optimal parameters, the model tended to produce a false-positive rate around double that of the false-negatives, implying that it found negative reviews more difficult to predict than positive ones. I believe one of the reasons for this could be that some negative reviews contain tokens which are, on the whole indicators of positive sentiment, such as 'great'. In negative reviews these can have their sentiment reversed when combined into bi-grams such as 'not great', which must be categorised as an independent feature. Despite incorporating bi-gras into my token vectorisation, when performing feature reduction, some of these bi-grams may have been dropped. One way to improve this mis-classification could be to weight such bi-grams more strongly in the TF-IDF stage.