Machine Learning 2 Lab Report: Image & Text classification

1. Task 1 – Image classification using boosting algorithm

1.1 Image Preprocessing

I explored several image preprocessing and features extraction methods, including image flipping, creating sub-windows, image flattening and using HoG features.

After some trial-and-error, I found that using raw data (i.e. not extracting HoG features) with sub-windows created and added to the flattened image vector produced the most improvement, albeit not by a significant amount. For sub-windows, I split each image into twelve grid squares as shown in figure 1 below.

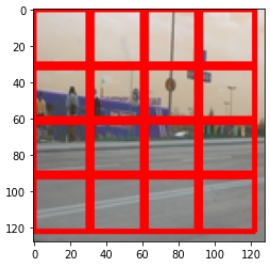

Figure 1: sub-windows for image preprocessing

Below table shows the results on test set with different combinations of preprocessing trained on optimized Adaboost classifier.

PREPROCESSING	ACCURACY
SUB-WINDOWS; NO HOG; FLATTEN	0.43
SUB-WINDOWS; HOG; FLATTEN	0.40
NO SUB-WINDOWS; HOG; FLATTEN	0.37
NO SUB-WINDOWS; HOG; NO FLATTEN	0.37
NO SUB-WINDOWS; HOG; NO FLATTEN;	0.37

Table 1: prediction accuracy with different features extraction settings

1.2 Classifier Specification

I used Adaboost as my boosted classifier. The weaker classifiers are decision stumps. Adaboost evidently improves the performance of decision stumps, regardless of whether features extractions are used. The use of decision stump gives an accuracy of 0.33, whereas the use of Adaboost gives at least 0.37.

1.3 Cross-Validation (CV) Results

For cross-validation, I used grid-search with 5-folds on the train set. I ran results for the following hyper-parameter settings:

- Number of estimators: [10, 50, 100, 150]
- Learning rate :[0.05, 0.25, 0.5, 0.75, 1]
- Algorithm: ['SAMME.R', 'SAMME']

The accuracy results range from 0.42 to 0.58. The best hyper-parameter results consist of three combinations, each producing a mean test accuracy of 0.58 as shown in below table. The following chart provides a visualization of the CV results.

settings	rank_test_score	mean_test_score	std_test_score
SAMME_0.05_100	1	0.575000	0.048591
SAMME_0.25_50	1	0.575000	0.048591
SAMME_0.25_150	1	0.575000	0.061237

Table 2: top ranked cross-validation results; settings: algorithm; learning rate; no. of estimators

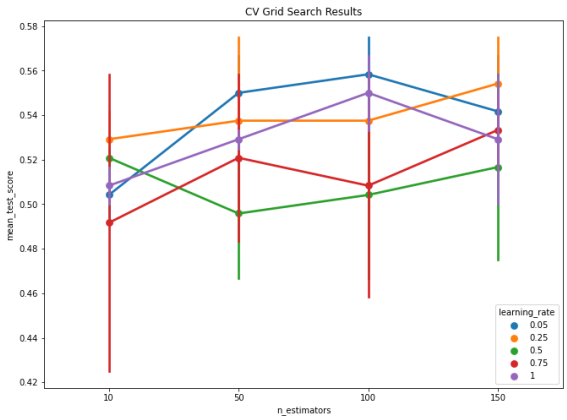

Figure 2: cross-validation results for Adaboost (image)

1.4 Test Set Results

For the test set results, I selected an optimised result from the table above (i.e. algo: SAMME; learning rate: 0.25; number of estimators: 50). The resulting accuracy is **0.493.**

2. Task 2 – Image classification using Support Vector Machine (SVM)

2.1 Image Preprocessing

I used the same image preprocessing and features extraction for this task as task 1.

2.2 Classifier Specification

I used the SVM with a one-vs-rest decision function of shape.

2.3 Kernels

I explored nine kernels in total, including four from scikit-learn (linear, 'rbf', sigmoid and polynomial) and five custom kernels (Cauchy, cosine, multiquadric, inverse-multiquadric and t-student). The five custom kernels are defined as follows:

- Cauchy: $\boldsymbol{(k(x_i, x_j) = 1/(1 + \left(\frac{\|x_i - x_j\|^2}{\sigma}\right))}$
- Cosine: $\boldsymbol{k(x_i, x_j) = x_i . x_j \,/(\|x_i\|. \|x_j\|)}$
- Multiquadric: $\boldsymbol{k(x_i, x_j) = \sqrt{(\|x_i - x_j\|^2 + c^2)}}$
- Inverse-multiquadric: $\boldsymbol{k(x_i, x_j) = 1/\sqrt{(\|x_i - x_j\|^2 + c^2)}}$
- T-student: $\boldsymbol{k(x_i, x_j) = 1/(1 + \|x_i - x_j\|^d)}$

2.4 Cross-Validation (CV) Results

In terms of hyperparameters, SVM has both C and gamma in addition to the kernels. For the kernels, I ran some trial-and-error for those with their own hyperparameters (e.g. d for T-student) and selected the best one.

For cross-validation, I used grid-search with 5-folds on the train set. I ran results for the following hyper-parameter settings:

- C: [0.1, 1, 10, 100]
- Gamma: [1.0,0.1,0.01,0.001]
- Kernels: [linear, 'rbf', poly, sigmoid, cauchy, cosine, multiquadric, inverse-multiquadric, t_student]

The accuracy results range from 0.16 to 0.59. The best hyper-parameter results consist of 5 different combinations, but note that all five settings consist of the **polynomial** kernel. Conversely, the worst performing ones consist of the multiquadric kernel. The mean test accuracy for each of these is **0.59** as shown in below table.

c_gamma_kernel	rank_test_score	mean_test_score	std_test_score
0.1_0.1_poly	1	0.591667	0.096465
100_1.0_poly	1	0.591667	0.096465
0.1_0.01_poly	1	0.591667	0.096465
0.1_0.001_poly	1	0.591667	0.096465
10_0.01_poly	1	0.591667	0.096465

Table 3: top ranked cross-validation results; settings: algorithm; learning rate; no. of estimators

The following charts illustrate the results. An observation is that in most cases the different values of C and gamma had little impact on the accuracy result in this image classification study.

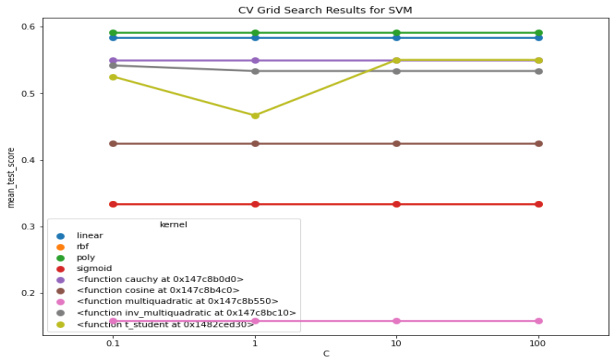

Figure 3: cross-validation results for SVM (hyperparameter c; image)

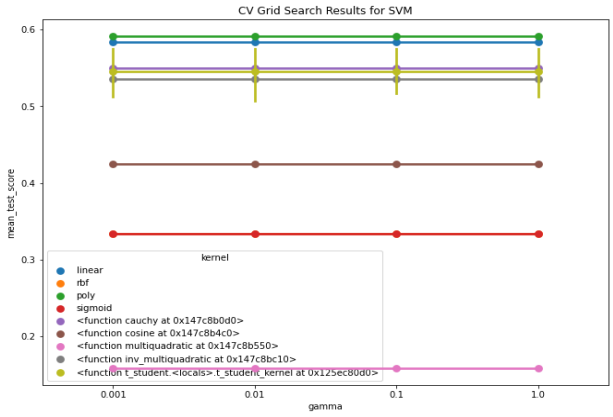

Figure 4: cross-validation results for SVM (hyperparameter gamma; image)

2.5 Test Set Results

For the test set results, I selected one of the optimised results from the table above (i.e. kernel: polynomial; C: 100; Gamma: 1). The resulting accuracy is **0.487**.

2.6 Conclusion for Task 1 & 2

The results from both optimised models are very similar, this may suggest that for images or this particular dataset, pre-processing and features extractions are more important in determining the prediction accuracy. In terms of the cross-validation results, the optimised SVM produced only slightly higher accuracy (0.59 vs 0.58 for Adaboost). As for test set results, the optimised Adaboost model produced slightly better result (0.493 vs 0.487 for SVM)

3. Task 3 – Text classification using boosting algorithm

3.1 Text Data Preprocessing and Bag of Words

First, I pre-processed the data (on the entire corpus) with the following procedures: convert all words to lower case; remove punctuations; tokenise each word; remove stop-words and apply word stemming to each word. After cleaning the data, I split the data into train and test set (80%/20% respectively). Note that this splitting is done prior to applying the bag of words method in order to prevent data leakage.

For the Bag of Words, I have chosen to use the TF-IDF method as opposed to Count Vectorizer. The former is generally better as it not only focuses on the frequency of words present in the corpus but also considers the importance of the words. I have set the maximum features to 20000.

To further limit computation time and improve the accuracy of the model, I used the truncated singular value decomposition (SVD), also known as latent semantic analysis (LSA), to reduce dimensionality of the dataset. This also works for sparse matrices efficiently.

To explore whether the use of SVD improves accuracy, I ran the classifier with the same hyperparameter settings to compare the results. The results in below table shows that it does indeed improve the accuracy significantly by roughly 0.3 (from 0.53 to the range of 0.79 - 0.83).

BAG OF WORDS (TF-IDF) & SVD (LSA)	ACCURACY
TF-IDF ONLY	0.54
TF-IDF + SVD (DIMENSIONS = 10)	0.79
TF-IDF + SVD (DIMENSIONS = 100)	0.83
TF-IDF + SVD (DIMENSIONS = 500)	0.82
TF-IDF + SVD (DIMENSIONS = 1000)	0.81

Table 4: comparing results with different SVD settings

3.2 Classifier Specification

I used Adaboost as my boosted classifier. The weaker classifiers are decision stumps.

3.3 Cross-Validation (CV) Results

For cross-validation, I used grid-search with 5-folds on the train set. I ran results for the following hyper-parameter settings:

- Number of estimators: [10, 50, 100, 150]
- Learning rate :[0.05, 0.25, 0.5, 0.75, 1]
- Algorithm: 'SAMME'

The CV mean test accuracy results ranges from 0.62 and 0.79 and is shown in below table. The best hyper-parameter uses a learning rate of 1 and 150 estimators, producing a mean test accuracy of **0.79**.

settings	rank_test_score	mean_test_score	std_test_score
SAMME_1_150	1	0.79050	0.019631
SAMME_0.75_150	2	0.78225	0.021101
SAMME_1_100	3	0.77975	0.018648
SAMME_0.75_100	4	0.77150	0.014062

Table 5: top 4 ranked cross-validation results; settings: algorithm; learning rate; no. of estimators

The following chart provides a visualization of the CV results. It is observed that in almost all cases, the higher the learning rate and increasing number of estimators results in improved accuracy. Also, there is a steepest increase in performance from 10 estimators to 50 estimators except for the lowest learning rate of 0.05.

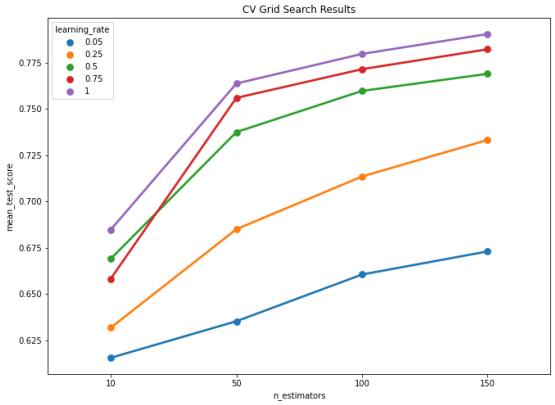

Figure 5: cross-validation results for Adaboost (text)

3.4 Test Set Results

For the test set results, I selected the best optimised result from the table above (i.e. algo: SAMME; learning rate: 1.0; number of estimators: 150). The resulting accuracy is **0.813**.

4. Task 4 – Text classification using Support Vector Machine (SVM)

4.1 Text Data Preprocessing and Bag of Words

I used the same data pre-processing and Bag of Words techniques in this task as in Task 3.

4.2 Classifier Specification

I used the SVM with a one-vs-rest decision function of shape.

4.3 Kernels

In this task, I used the same nine kernels as in task 2 with nine in total, including four from scikit-learn (linear, 'rbf', sigmoid and polynomial) and five custom kernels (Cauchy, cosine, multiquadric, inverse-multiquadric and t-student). The five custom kernels are defined in Task 2.

4.4 Cross-Validation (CV) Results

In terms of hyperparameters, SVM has both C and gamma in addition to the kernels. For the kernels, I ran some trial-and-error for those with their own hyperparameters (e.g. d for T-student) and selected the best one.

For cross-validation, I used grid-search with 5-folds on the train set. I ran results for the following hyper-parameter settings:

- C: [0.1, 1, 10, 100]
- Gamma: [1.0,0.1,0.01,0.001]
- Kernels: [linear, 'rbf', poly, sigmoid, cauchy, cosine, multiquadric, inverse-multiquadric, t_student]

The accuracy results range from 0.25 to **0.86.** The best hyper-parameter results consist of 2 different combinations, both uses the **RBF** kernel. The sigmoid, cosine and linear kernels also produced strong results just very slightly underperforming the RBF kernel. Conversely, the worst performing ones consist of the multiquadric kernel. The mean test accuracy for the top 5 settings are shown in below table.

c_gamma_kernel	rank_test_score	mean_test_score	std_test_score
10_1.0_rbf	1	0.85900	0.019226
100_1.0_rbf	1	0.85900	0.019226
1_1.0_sigmoid	3	0.85750	0.015104
1_1.0_<function cosine at 0x127394b80>	4	0.85725	0.016534
1_0.1_linear	4	0.85725	0.016534

Table 6: top ranked cross-validation results; (text)

The following charts illustrate the results. An interesting observation is that with some of the best performing kernels (e.g. sigmoid and RBF), the higher the C and gamma values the better the performance. In other cases the different values of C and gamma had little impact on the accuracy result in this text classification study.

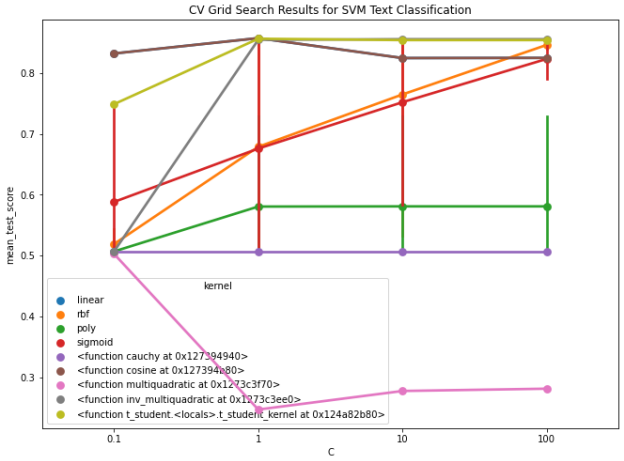

Figure 6: cross-validation results for SVM (hyperparameter c; text)

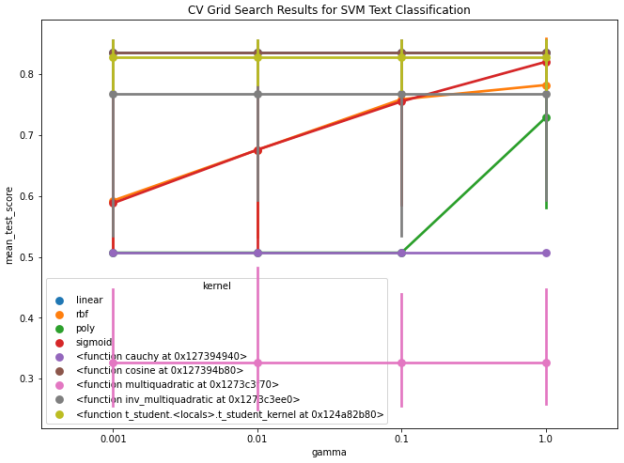

Figure 7: cross-validation results for SVM (hyperparameter gamma; text)

4.5 Test Set Results

For the test set results, I selected one of the optimised results from the table above (i.e. kernel: RBF; C: 10; Gamma: 1). The resulting accuracy is **0.855**.

4.6 Conclusion for Task 3 & 4

Overall, the SVM is shown to outperform the Adaboost model in text classification. Specifically, with the cross-validation results, the optimised SVM gives an accuracy of 0.86 vs. 0.79 with Adaboost. The optimised SVM also produce a higher accuracy in the test set results, giving an accuracy of 0.855 vs 0.813 for Adaboost.

4.7 Conclusion for Image vs. Text Classification

Comparing all four tasks, the main observation is that both models work much better for the text classification than image classification. This can be attributed to the particular dataset, or it could mean that the feature extraction of the images is not optimal. Also, the difference in the classification nature (i.e. multi-class for image task and binary class for text) may have an impact. The results from both optimised models are differs to a larger extend for the text classification as compared to image classification. This further suggests that more data augmentation techniques could be explored for the image classification task.