Machine Learning Coursework - OULAD Analysis

mbtj48

1 Data Gathering and Analysis

Machine learning and data gathering are paramount for modern, cutting edge technologies. Thus we have been tasked to develop two machine learning models to predict final grades from the OULAD.

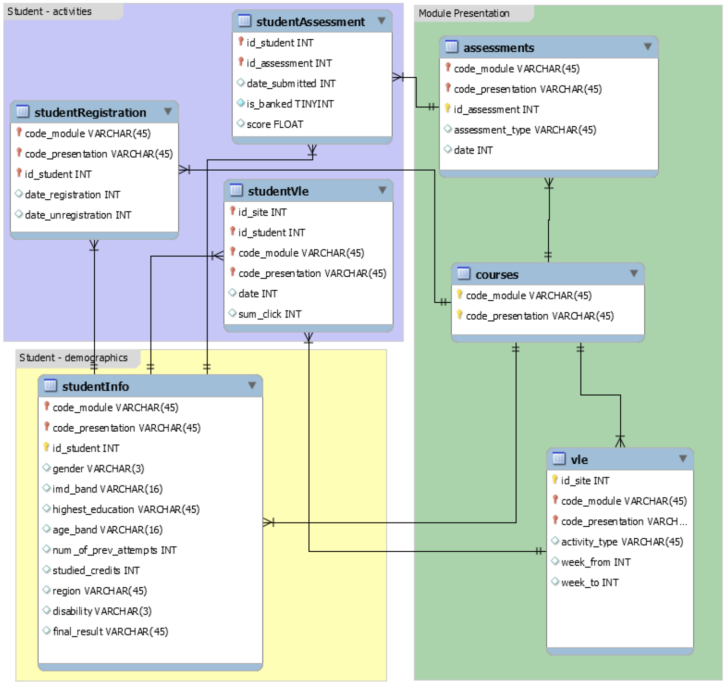


Figure 1: Dataset Schema

Firstly, I noticed useful features such as the score in the studentAssessment table, and sum-click in the studentVle table. Therefore I started by grouping the sum-click and score features, finding the net clicks within the portal for a given student through the year and their average mark. I expected these features to show a positive correlation because higher scores and grades generally correlate with high effort [tab 1]. Then, I plotted the data and noticed that a logistic regression model should perform highly. I added more data to my model; intending to use as much data as possible to aid the model in finding patterns: ie calculating how many days early they submitted coursework and their summative and formative (weight = 0) marks. As the student would be more prepared and committed, I expected a positive correlation from this data.

My interpretation of the data evolved as, I included a wide range of the available data, thus I started to interpret the data differently: including the mean, median, mean absolute deviation, standard deviation and variance for different data in the schema. I then produced a correlation heatmap [fig 11], plus the sorted numerical correlations [tab 1].

Feature	Correlation
daysEarlystdScore	−0.259014
studied-credits	−0.176016
region-Wales	0.008382
age-band	0.068551
score	0.317339
sum-click	0.376107
totalCoursework	0.427175
summativeAgainstCredits	0.490646

Table 1: Correlations

Surprisingly, age-band shows a weak correlation; in theory, you would expect a mild negative correlation. However, this could be due to limited data (3 unique ranges - [fig 3]). To improve this correlation, I would need more spread out ranges or use integers instead of the ranges.

After data gathering, I preprocessed the data with an imputer and scaler. The imputer changes all NA values to the median of that feature. While the scaler, normalises features to be within $0-1$, this prevents feature domination with large ranges and makes the features unit dependent. Further, I exchanged region, code module and code presentation to columned data by one-hot encoding those categories.

2 Model Selection

The following phase involved selecting models. Here, I split the data into train and test sets with a 75/25 split; then tested a variety of models and compared how they performed in cross-validation on the training data. Generally, there was a wide performance range, with classifiers generally doing better than regressors. See table 3 for further information. I decided to pick one regressor and one classifier to explore: Logistic Regression and Random Forest Classifier.

3 Model A - Logistic Regression

Moving on to hyperparameter tuning: For this model, I decided to use a grid search to validate the best combination within the domain. I explored a logarithmic range of the C parameter; until after testing, I reached an optimal range of 950-1100. It also checked tolerance values around the default, ending with 0.0015. The other set of combinations check the same values of C and adjust the solver and penalty used. Finally, I removed the second set of combinations as it did not help to increase performance.

4 Model B - Random Forest Classifier Appendix

Initially, I used Random Search to tune the classifier. This generates parameters to cross-validate the model. I decided to check the number of estimators, the maximum depth, the minimum number of samples and the minimum number of samples required to split an internal node. I moved on to use Bayesian optimisation to optimise this search problem. This uses the previous iterations to strategically select the next best parameters from the search space to minimise the loss function. I defined the loss as $3 - \overline{acc} - \overline{acc_{bal}} - \overline{f1_{weighted}}$; this, therefore, seeks to reduce the primary metrics for a classification problem. Next, I removed the unimportant attributes from the forest's feature importances [fig 6] and tuned the model again. I noticed the number of estimators showed little correlation for the model improving the loss function [fig 9] and this second tune slightly improved the model [figs $7 \rightarrow 10$].

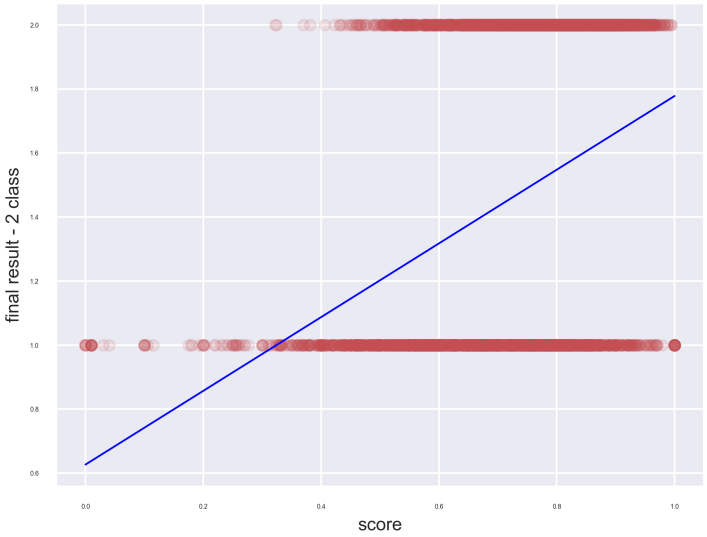


Figure 2: Linear Regression (2 class) against score

5 Conclusion

Model	**Logistic**		**Random Forest**	
Classes	**2**	**4**	**2**	**4**
Explained Var	0.697	0.664	0.823	0.729
RMSE	0.275	0.585	0.213	0.523
MAE	0.076	0.300	0.045	0.263
r2 Score	0.697	0.662	0.819	0.729
f1 Score (weighted) (Recall and Precision)	0.924	0.706	0.955	0.739
Accuracy	0.924	0.721	0.955	0.742

Table 2: Metrics of Final Models

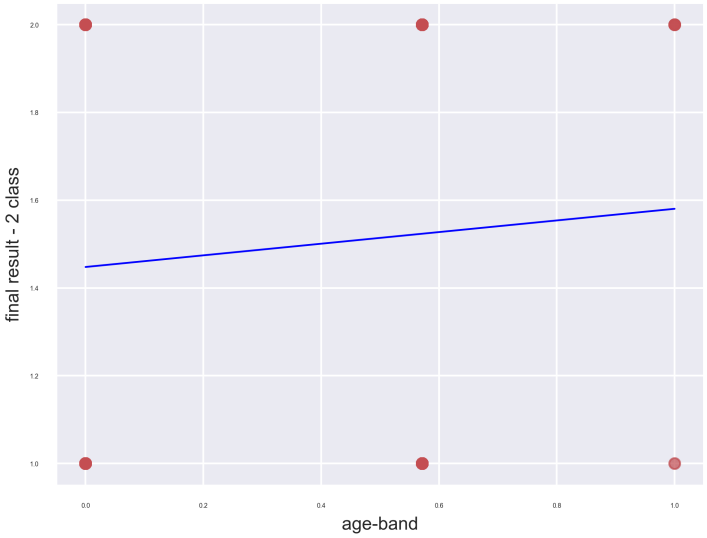


Figure 3: Linear Regression (2 class) against age-band

The logistic regression model finished with an accuracy of 0.721%, while the random forest classifier finished with 0.742%. Although notably, the weighted average of the f1 score for both models are lower; which is likely a better metric to measure, as it reduces metric skew on imbalanced data. Furthermore, upon analysing the classification report, classes with low balance have lower performance due to the data imbalance. The confusion matrices [tables 4,5,6,7] showed that "close" classes such as withdrawn and fail and pass and distinction are easily mistaken. Finally, note the two-class metrics (these merge withdrawn and fail & pass and distinction, as these are subclasses of the primary labels) which are generally high performing. The random forest 2-class model would be good to implement to determine whether a given student is likely to fail, due to the high accuracy and f1 score [2].

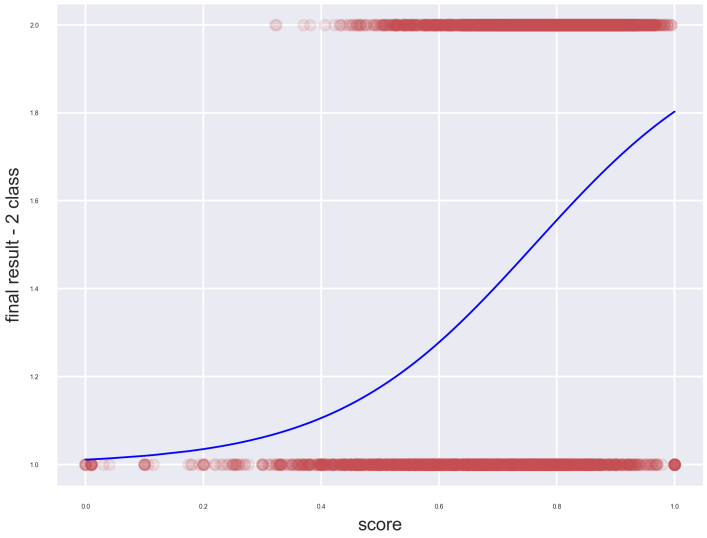


Figure 4: Logistic Regression (2 class) against score

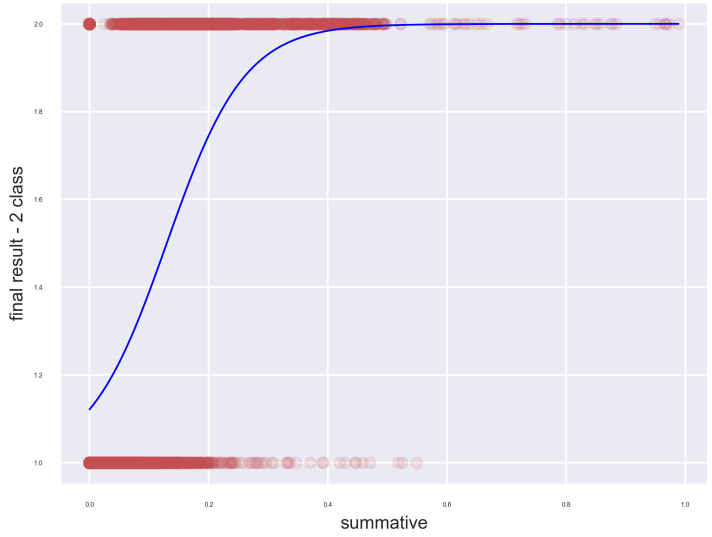


Figure 5: Logistic Regression (2 class) against summative

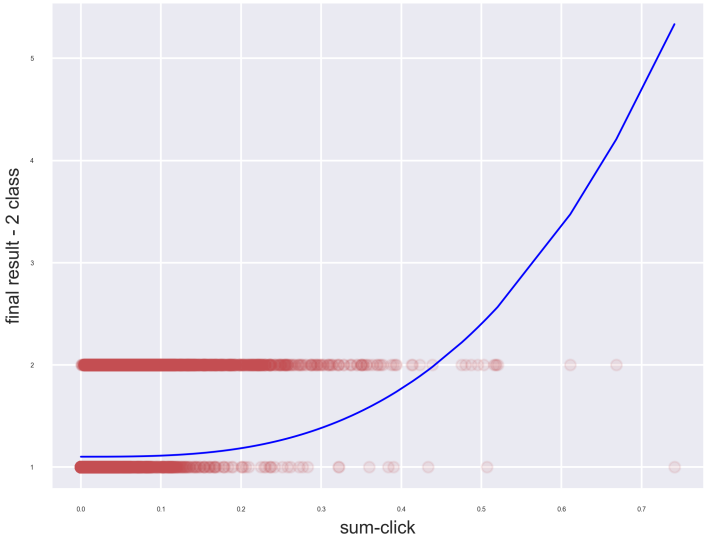


Figure 8: SVR-Poly-Kernel (2 class) against sum-click

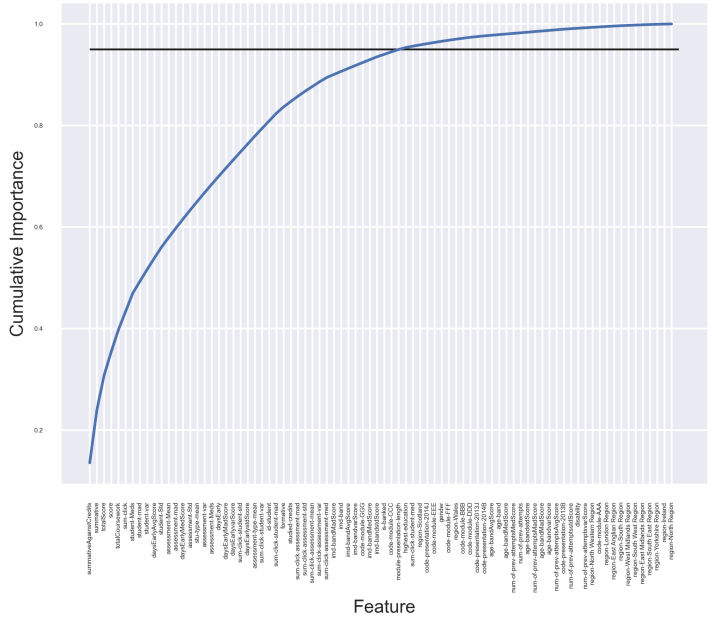


Figure 6: Cumulative Importances of Features

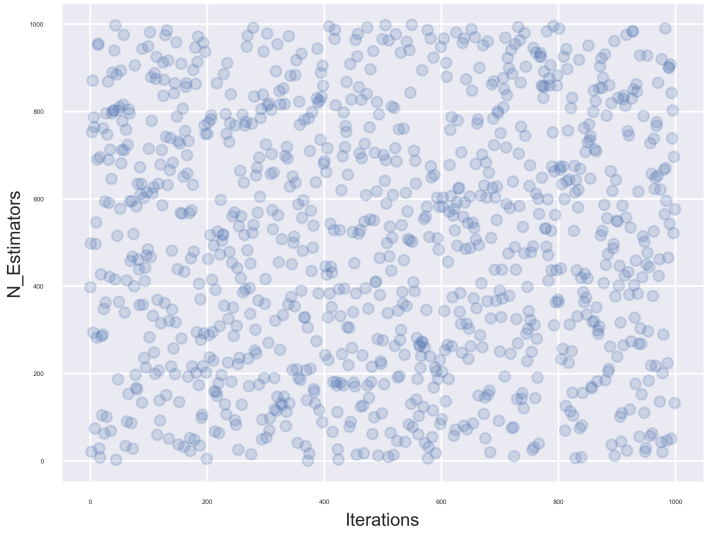


Figure 9: Selected n-Estimators againts iteration

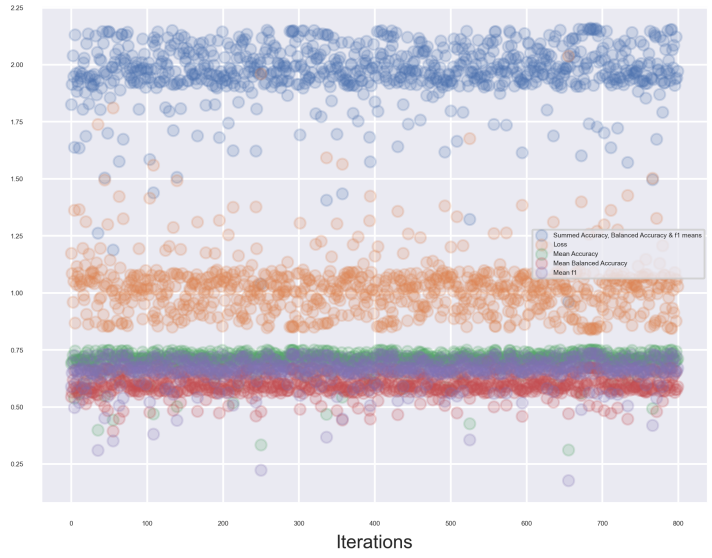


Figure 7: Metrics against iteration 1

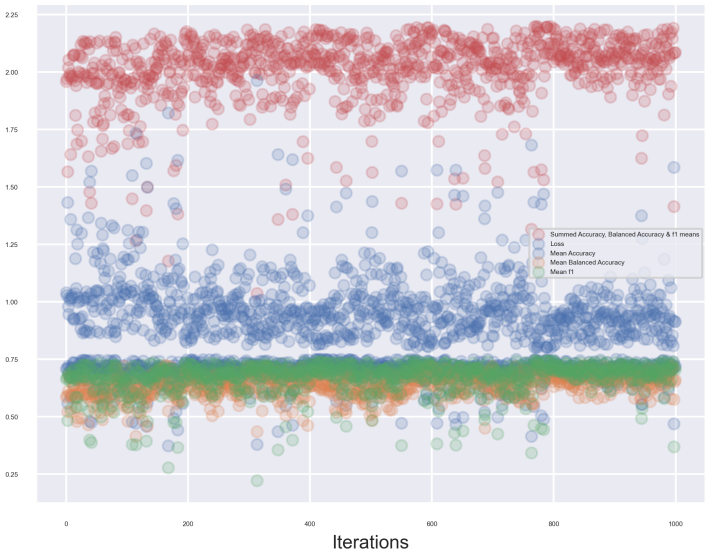


Figure 10: Metrics against iteration 2

Model	Class	Mean	SD
Linear	4	0.604	0.010
Logistic	4	0.702	0.007
-	3	0.767	0.005
-	2	0.915	0.003
SVR Linear	4	0.575	0.013
SVR Poly	4	0.746	0.008
SVR RBF	4	0.723	0.009
SVC	4	0.681	0.007
-	3	0.782	0.005
-	2	0.932	0.003
DT	4	0.674	0.003
-	3	0.755	0.005
-	2	0.915	0.002
RF	4	0.748	0.005
-	3	0.809	0.005
-	2	0.942	0.003

Table 3: Model Selection CV Accuracy Metrics

	With	Fail	Pass	Dist
With	2056	462	22	4
Fail	606	841	251	3
Pass	8	205	2615	293
Dist	0	0	248	535

Table 4: RF 4 Class Confusion Matrix

	Fail	Pass
Fail	3937	308
Pass	60	3844

Table 5: RF 2 Class Confusion Matrix

	With	Fail	Pass	Dist
With	2097	373	72	2
Fail	723	640	335	3
Pass	84	125	2748	164
Dist	1	3	390	389

Table 6: Logistic 4 Class Confusion Matrix

	Fail	Pass
Fail	3889	356
Pass	261	3643

Table 7: Logistic 2 Class Confusion Matrix

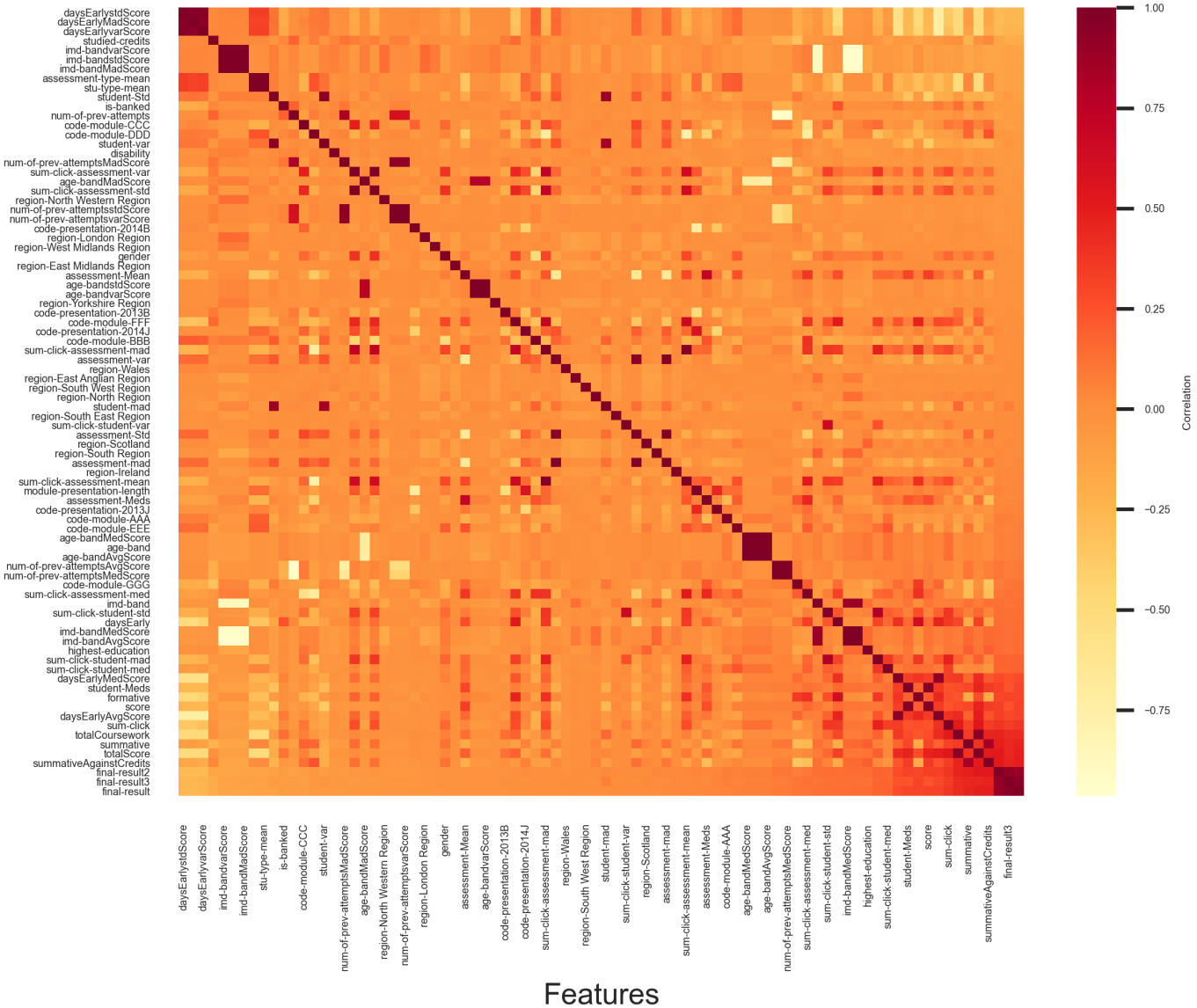


Figure 11: Correlation Heatmap

Image Processing Coursework - Non-local Means Denoising

mbtj48

1 Description

The concept for non-local means denoising uses multiple squared neighbourhoods "similar" to a given neighbourhood of pixels, and then calculates the value of the pixels in that neighbourhood based on the weighted average of these similar neighbourhoods. This leaves two main areas where implementation becomes difficult: finding similar neighbourhoods in the image and how specifically to average them.[6]

Formally, non-local means is defined as:

$$NL[u](p) \;=\; \frac{1}{C(p)} \int_{\Omega} f(p,q)\; u(q) dq \qquad [1,\ 2]$$

Here we have a filter applied to the image u at a pixel p which lies within the image area Ω. Where, $u(q)$ is the original unfiltered pixel value at position q and $C(p)$ is the normalising factor of the function. Finally, we have the function $f(p,q)$, which calculates the weight of the similarity of this pixel to other neighbourhoods, this is commonly the Gaussian weighting function which uses a normal distribution of weightings relating the two pixel values. This typically acts as the Euclidean distance between the two pixels: $d(\,B(p),\,B(q)\,)$. [2] Due to this algorithm's continuous nature we cannot extrapolate a direct implmentation from this definition, so we have to manipulate the algorithm to use discrete pixel values.

2 Implementations

The pixelwise implementation introduced in [1] considers every pixel and its neighbourhood in the image, calculating the weight based on the squared Euclidean distance between two pixels which is the similarity between the colour vectors. These are then entered into an exponential kernel, providing similar pixels with a higher weight and weights smaller than $2\sigma^2$ are set to 1, as they are not similar at all. The values found from this update the center pixel of each neighbourhood. The formula for applying pixelwise to a image $u = (u_1, u_2, u_3)$, at a given pixel p is:

$$u_i(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} u_i(q)\, w(p,q) \qquad [2]$$

This has large similarities to the integral definition at the start of the report. With $C(p)$ acting as the normalising function, $u_i(q)$ as the unfiltered value of another pixel q at the center of another patch. Finally $w(p,q)$ as the weight between the two pixels, defined earlier.

The patchwise implementation introduced in [2] works very similarly to the pixelwise implementation, however after calculating how similar other neighbourhoods are, it denoises all of the neighbourhood of p rather than just p itself. This is done by finding the average of every pixel within a neighbourhood first, and is then repeated across all possible neighbourhoods which then gives N^2 estimates for every pixel, these are then averaged to calculate the final value for every pixel. Formally, the additional formula for patchwise over pixelwise is:

$$u_i(p) = \frac{1}{N^2} \sum_{Q=Q(q,f)|q \in B(q,f)} Q_i(p) \qquad [2]$$

Separable NLM is a highly efficient implmentation, which separates the image into row and column vectors then processes the whole vector at once. This updates the formula for the euclidean distance to

$$d^2_{ij} = \begin{cases} \overline{V}(l_{ii}) + \overline{V}(l_{jj}) - 2\overline{V}(l_{ij}) & i \geq j \\ \overline{V}(l_{ii}) + \overline{V}(l_{jj}) - 2\overline{V}(l_{ji}) & j > i \end{cases} \qquad [3]$$

In order to calculate the output of the image the formula above is used to process rows then columns, while also doing columns then rows, as the process is not commutative, and find the average of the two results. See Figure 3 in [3]. That paper then further talks about finding optimal parameters and proofs of the complexity.

Efficiencies

For the complexities below, the image is of size $N \times N$, search window of size $S \times S$ and neighbourhood of size $K \times K$. The initial pixelwise implementation described above has a completxity of: $\mathcal{O}(N^4K^2)$. However, the common default pixelwise implementation limits the research window for similar pixels, which improves the complexity to: $\mathcal{O}(N^2S^2K^2)$ [3]. Patchwise is similarly efficient with a complexity of: $\mathcal{O}(N^2K^2 + N^2) \rightarrow \mathcal{O}(N^2K^2)$. This is because patchwise performs an extra N^2 calculations to find the weight across every neighbourhood; these are then averaged at the end to give the final result. However, the extra computation for patchwise generally results in a higher PSNR value [2] when compared to the initial pixelwise implementation in [1]. Alternatively, many minor enhancements have been found for these implementations; for example, the separable NLM described above has a complexity of $\mathcal{O}(N^2S)$. Therefore making it linear in calculating the new value for every pixel. Generally, this increase in speed is a success for PSNR performance over similar, more complex algorithms. See the images and performance stats in [3]

3 Parameters

(a) Original (b) Noise

(c) High noise (d) Extreme noise

Figure 1: Original Images

In order to compare the given parameters h (strength of the Gaussian filter), templateWindowSize (size of the local neighbourhood of similar pixels), and searchWindowSize (the proximity for the denoising to search for similar neighbourhoods) [4], I will fix two at their recommended values of 10,7 & 21 respectively, and alter the third value with a low value, a recommended value and a high value for each noisy image. This provided me with 28 images to compare from each original image. See below for the original images:

(a) h = 5 - noise (b) h = 10 - noise

(c) h = 10 - high noise (d) h = 30 - ext noise

Figure 2: Varied h Images

Starting with parameter h. This provides the largest change to 1a. I tested values of 5,10 & 30. Initially, at values of 5, when applied to 1d the changes are minor, this is evident as there is only 0.006 improvement in the PSNR value. However improves when applied to 1c, removing noise from the sky and some of the sunlit buildings, but gives a worse PSNR value by 0.1. Continuing, this h futher removes noise from 1b and retains detail when applied to 1a, as seen in 2a. Increasing this value to 10 starts to cause loss of detail in 1a and 1b. It cannot remove the noise from 1d, however performs highly on 1c, as seen in 2c. Further, this shows one of the largest increases in PSNR moving from 28.60 to 29.65. Using $h = 30$ provide extremes of using $h = 10$; here, lots of detail is lost even on 1c, although it can successfully remove some noise from 1d but the image lacks far too much in quality as seen in 2d.

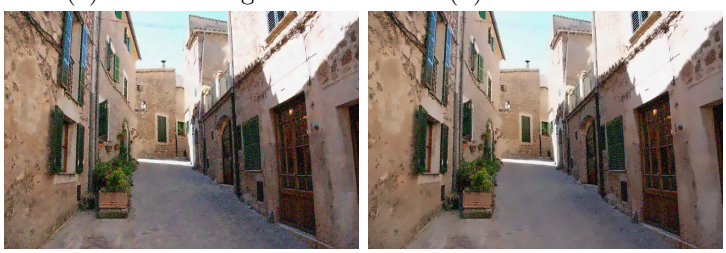
(a) t = 7 - original (b) t = 3 - high noise

(c) t = 7 - high noise (d) t = 15 - high noise

Figure 3: Varied *templateWindow* Images

Now altering the templateWindowSizes through 3,7 & 15. These affect the image quite clearly, with lower values resulting in higher detail within homogeneous regions. See 3a where this removes some minor detail from 1a, which is undesirable. Moving onto image 3c, which looks close to the optimal value for high noise images, especially compared to image 3b which uses a neighbourhood of size 3, and the improvements from the original are unnoticable. However, the PSNR value improves massively from 28.60 to 29.71 so therefore the image has still reduced noise.

(a) s = 45 - original (b) s = 11 - noise



(c) s = 11 - high noise (d) s = 21 - high noise

Figure 4: Varied *searchWindow* Images

I finally iterated the searchWindowSize through 11,21 & 45. (this is typically much smaller than the image size to improve performance) [4]. Generally, the higher value resulted in greater smoothing of surfaces, however, I felt it removed too much detail across all noisy images; for example in image 4a. In image 4b, the noise is blurred however similar areas have a "brushed" effect, which is the affect of using a small search window to find similar neighbourhoods. Image 4d shows how a search window of 21 affects image 1c; clearly it keeps enough detail in rock faces wihout smoothing the floor surfaces, which is almost perfect considering the large σ of noise in 1c.

4 Comparisons

Gaussian Filtering - The image is put through a gaussian filter, which primarily removes noise through blurring. It does this by looking at the neighbourhood of a pixel and assigns the new intensity of that pixel based on the difference of the surrounding pixels intensity, then passed through a gaussian distribution matrix (the filter) [6]. Typically, this does not perform as well as non local means (In [1], they find the mean square error to be just less than double for NLM) but is much more efficient at calculating the output image. [1]

(a) Noise - PSNR 31.61 (b) High noise - PSNR 30.14

Figure 5: A 5×5 gaussian filter applied to 1b & 1c

Above, I applied the cv2.GaussianBlur function to the same alley image from 1a. Clearly, it does perform well at removing some noise however detail is quite heavily lost when the filter size is increased. If I compare the PSNR value for 5a to the NLM, it lies in the average found by the range of parameters I used, with the best being 31.9. For 5b it out performs the NLM PSNR values for the parameters I used, however I feel the images look better after the NLM denoising, for example see image 3c.

Anisotropic Filtering [1] - Anisotropic filtering aims to improve the blurring caused by Gaussian filtering by convolving the image at a point u only in the direction orthogonal to $Du(x)$. This is defined as:

$$AF_h u(x) = \int G_h(t) u(x + t\frac{Du(x)^\perp}{|Du(x)|})dt \quad [1]$$

Generally, for reducing noise in the image, this is more computationally efficient that NLM, however NLM performs higher when improving PSNR. For example, see the mean square error table from [1] which shows that anisotropic filtering performs better than the gaussian filter, slightly, but cannot perform near NLM.

Spacial Median Filtering [6] - The algorithm iterates through every pixel in the image, and exchanges its value to the median value of pixels in its neighbourhood. This is a local filter, and generally it is much more efficient than the NLM variants, but typically it does not perform as well at improving PSNR.

(a) Original - PSNR 32.19 (b) High noise - PSNR 30.14

Figure 6: A 5×5 median filter applied to 1a & 1c

Above, I applied the cv2.medianBlur function to the alley image from 1a. It performs similarly to the Gaussian blur in terms of PSNR values, however looking at 6b, certain edges have been removed as they are extremes in those neighbourhoods. This could be combined with an edge sharpening algorithm such as the laplacian in order to perform better visually and therefore perform better than NLM.

Spacial Mean Filtering [6] - This filter is similar to spacial median filtering, however instead of finding the median of the region of interest, we find the mean. Similar to the other algorithms, this aims to reduce noise through smoothing and blurring, and therefore loses a lot of detail with medium to large size filters.

(a) Original - PSNR 30.37 (b) Ext noise - PSNR 28.46

Figure 7: A 5×5 mean filter applied to 1a & 1d

The images above show how the 5×5 mean filter affects the alley image. Evidently, a lot of detail has been lost through too much blurring, however I feel this is the best evidence for improvements on image 1d; this is further evident when looking at PSNR values which are not rivalled with NLM, but perform much worse on images with less noise.

For consistency, I kept the filter sizes the same for all of the examples above, however in reality the optimal filter sizes are much smaller than 5.

Deep Neural Network for Image Denoising [5] - Recently, progression in the denoising research field has moved towards denoising using unsupervised neural networks. A recent version seen in [5], uses 3 different layers in the neural network. A convolution (the output function to activate certain nodes in the network) and ReLU (the output function to active certain nodes in the network) layer, a convolution, batch normalisation (a method used to improve speed and performance of deep learning neural networks) & ReLU layer, and finally just a convolutional layer. See Figure 1 in [5] for the diagram of this network. This proposed algorithm, DnCNN, uses a residual learning method to train a residual mapping of $\mathcal{R}(y) \approx v$ (from the noise function $y = x + v$), which then gives $x = y - \mathcal{R}(y)$. This then uses a residual mean square error formula between desired residual images and estimated ones from noisy input, in order to train the layers of the neural network. Although there is little material directly comparing this to NLM, it performs much much higher on the 12 commonly used testing images in Figure 3 from [5], however this algorithm requires a long period of time for training before it can perform really well.

5 Modifications

Generally, modificatations are well documented and varied. Firstly, paper [7] introduces probabilistic early termination to elimate patches which are not similar to the current patch. This is done by calculating the partial sum of euclidean distances to determine with x probability whether x exceeds a threshold in order to terminate the current computation. Further, see paper [8], which uses the preclassification of a search window to determine whether the window is in an edge section or a surface/ textured section. From this, the algorithm then decides whether it should use NLM or local mean. This improves the run time and gives a minor increase in PSNR performance. Also, see [9], which uses singular value decomposition to affectively factorise the matrix containing the region of interest, allowing the columns of V to act as a basis for the space of image patches. From this we can also use the SVD property to approximate the patch in order to speed up NLM. The results in [9], show that this method performs higher than the prefiltering modification and the limited range modification.

6 Applications

Non local means denoising has a surprisingly wide range of applications. For example, this report has spoken about NLM in 2 dimensions, however in [10], they remove noise from electrocardiograms using NLM in 1 dimension, by translating the parameters of NLM to relevant parameters for the ECG signal; for example the main filter strength value h becomes the bandwidth λ of the ECG signal. See Figure 2 from that paper. They initialise the ECG signal, then add noise to the signal which is then denoised through the use of 1 dimensional NLM filter. Further applications of NLM have been applied to 3D magnetic resonance imaging [11]. This uses the blockwise NLM method introduced in [2], translated to 3D space through using voxels rather than pixels. Obviously, this will affect the formulas as well as the time complexity, so it is important to optimise to achieve the same results. Therefore [11] uses several optimisation techniques, such as multithreading, and automatic tuning of the smoothing parameter & block selction.

References

[1] Buades, Antoni, Bartomeu Coll, and J-M. Morel. "A non-local algorithm for image denoising." 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Vol. 2. IEEE, 2005.

[2] Buades, Antoni, Bartomeu Coll, and Jean-Michel Morel. "Non-local means denoising." Image Processing On Line 1 (2011): 208-212.

[3] Ghosh, Sanjay & Chaudhury, Kunal. (2016). Fast separable nonlocal means. Journal of Electronic Imaging. 25.

[4] OpenCV - Image denoising https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html

[5] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising" IEEE Transactions on Image Processing 2017: 3142 - 3155

[6] Ioannis Ivrissimtzis, Image Processing, Lecture 4 & 5, Image Noise & Spacial Filtering

[7] Ramanathan Vignesh, Byung Tae Oh and C.-C.Jay Kuo, "Fast Non-Local Means (NLM) Computation With Probabilistic Early Termination"

[8] Hernández-Gutiérrez, I.V., Gallegos-Funes, F.J. & Rosales-Silva, A.J. Improved preclassification non local-means (IPNLM) for filtering of grayscale images degraded with additive white Gaussian noise. J Image Video Proc. 2018, 104 (2018)

[9] Jeff Orchard, Mehran Ebrahimi, Alexander Wong, "Efficient nonlocal-means denoising using the SVD"

[10] Singh, P., Shahnawazuddin, S. & Pradhan, G. Circuits Syst Signal Process (2018) 37: 4527. https://doi.org/10.1007/s00034-018-0777-9

[11] Pierrick Coupé, Pierre Yger, Sylvain Prima, Pierre Hellier, Charles Kervrann, et al.. An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonance images.. IEEE Transactions on Medical Imaging, Institute of Electrical and Electronics Engineers, 2008, 27 (4), pp.425-41.

Appendices

(1a)

(1b)

(1c)

(1d)

(2a)

(2b)

(2c)

(2d)

(3a)

(3b)

(3c)

(3d)

(4a)

(4b)

(4c)

(4d)

(5a)

(5b)

(6a)

(6b)

(7a)

(7b)