Image Sentiment Polarity

Michael McKay (Student no: 32270208)

November 2021

Table of Contents

1 Introduction

A folder containing approximately 12,000 images have been provided for analysis. Each photo within the set has been given a classification by a group of people. The classes the images were assigned are ''Very Negative'', ''Negative'', ''Neutral'', ''Positive'' and ''Very Positive". In addition, each photo has also been assigned a "Label confidence", which states how consistent the rating was amongst the test group.

The goal of this assignment is to produce a script within either Python or R which can process a folder of images, extract significant features and then use this information alongside the classifications provided above to train a model to be able to identify if an image fits in one of the above categories.

2 Pre-processing and features used

Once images were loading into Python, the following features were extracted:

1. Hu Moments
2. Haralick Texture
3. Colour Histogram
4. Colourfulness.

Hu moments are a weighted average of image pixel intensities. They can be extracted from the image using the function "HuMoments" once the image has been converted to grayscale (Shape Matching using Hu Moments, 2021). They are also referred to as "Rotation Invariants". This function will return seven different values, which are defined as below:

$$
\begin{aligned}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
\end{aligned}
$$

Image 1: Hu Moments from Wikipedia (Image moments, 2021).

The first moment I_1, refers to the "moment of inertia" around the image's centroid, where the pixels' intensities are analogous to physical density (Image moments, 2021). The first six (I_1 through to I_6) are symmetric reflections, as they are unchanged if the image is converted to a mirror image. The last moment, I_7, is reflection asymmetric and would allow us to distinguish mirror images of otherwise identical images (Image moments, 2021). This

set of features won't tell us anything about the colour profile of our image but will give us information regarding scaling and rotation (Huang, Z., & Leng, J. 2010).

The subroutine will then get Haralick texture values. These are also known as a co-occurrence matrix. It is used as an approach to texture analysis with various applications, especially in medical image analysis. These features will give us information regarding the texture of the image. Texture analysis has become increasingly popular within the medical field for diagnosis, classification, and treatment response assessment of cancerous disease (Bynolfsson. P., et al. (2017). These features could potentially give us useful information for classifying our images. This function returns 13 features.

The next set of features been extracted is the "Colour Histogram". Below is a picture of a colour histogram and the picture it is derived from. Both have been taken from Colour Histogram, 2021.

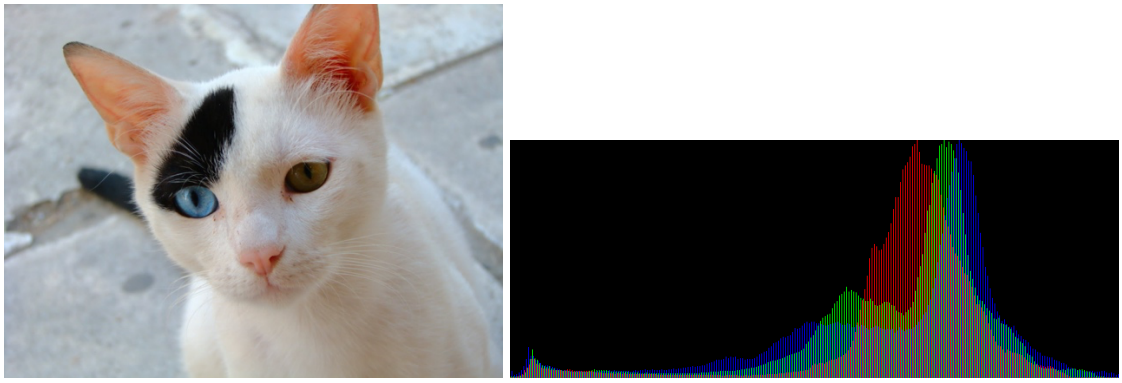


Figure 1 and 2: Image of a cat and its colour histogram (Colour Histrogram, 2021).

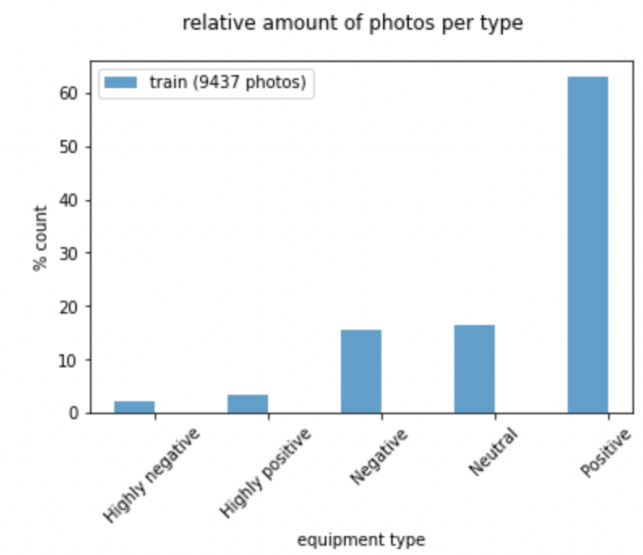
The colour histogram returns 512 features and indicates how much of each colour is present in the image. We can use this feature set to determine how colourful the image has been analysed is.

The last feature set is one implemented by me based on the colourfulness of the image. It returns seven features. These are below:

- rgMean – Average amount of red and blue.
- rgStd – Standard Deviation of the amount of red and blue.
- ybMean – Average amount of yellow and blue.
- ybStd – Standard Deviation of the amount of yellow and blue.
- stdRoot – the square root of the sum of rgStd and ybStd squared individually.
- meanRoot - the square root of the sum of rgMean and ybMean squared individually.
- Colourfulness – StdRoot + 0.3 x meanRoot.

The above features were taken from "Computing image colourfulness with OpenCV and Python 2021". It is believed that the amount of colour would be a strong indicator of how an image was classified, so I introduced more colour related features to try and improve the classification.

In total, we have 539 features that should give us information regarding how colourful the image is and how many corners or edges are present. We also have confidence in our training data. Unfortunately, this feature is not available in our testing set, so we can't use it to train our model. However, we could use it during the EDA process. After loading the data, we'll first check to see how evenly distributed our five classes are:



Label	
Highly negative	2.0
Highly positive	3.0
Negative	15.0
Neutral	16.0
Positive	63.0

Figure 3: Distribution of classes within training data.

The above graph indicates that the classes are very unevenly distributed. Around 63% of cases are “Positive”. Only about 2% are “Highly negative”, and 3% are “Highly positive”. This distribution will make it more difficult to train our models, as it will tend to over predict the majority class.

Our training data does contain a confidence column, which is a number between 0 and 1. We could use this value as a sample weight, which means the samples we are more confident about will be given more priority than the others. Therefore, we should have a look at this column in more detail.

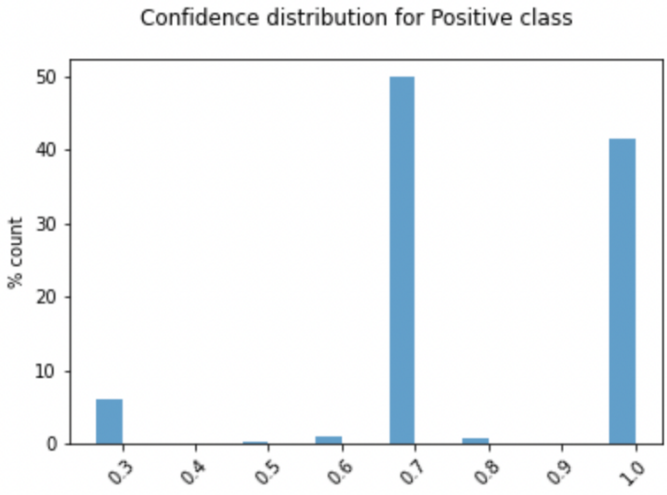


Figure 4: Distribution of Label confidence amongst the Positive class.

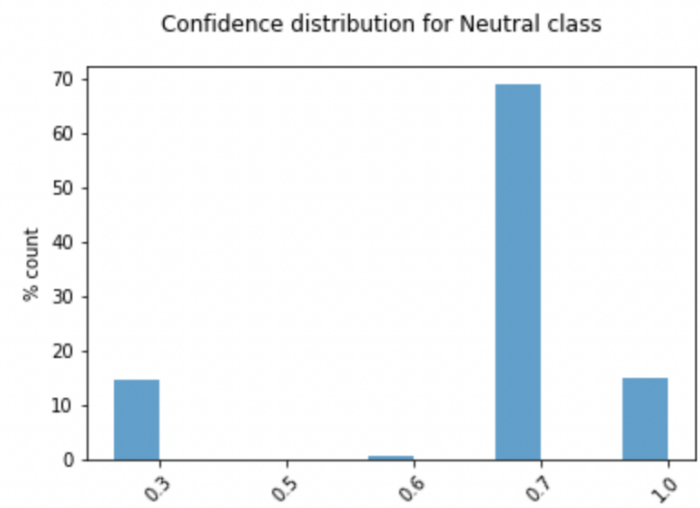


Figure 5: Distribution of Label confidence amongst Neutral class.

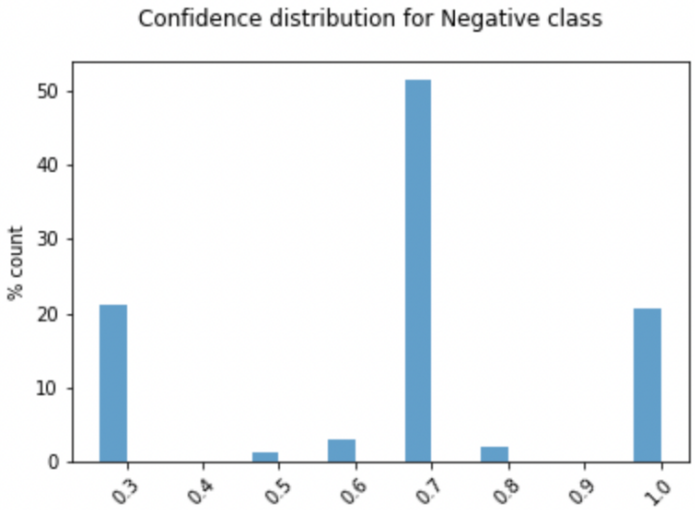


Figure 6: Distribution of Label confidence amongst the Negative class.

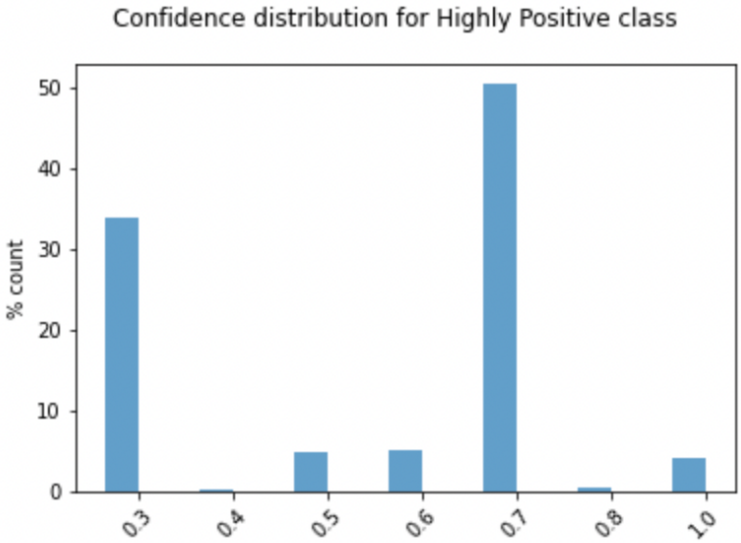


Figure 7: Distribution of Label confidence amongst the Highly Positive class.

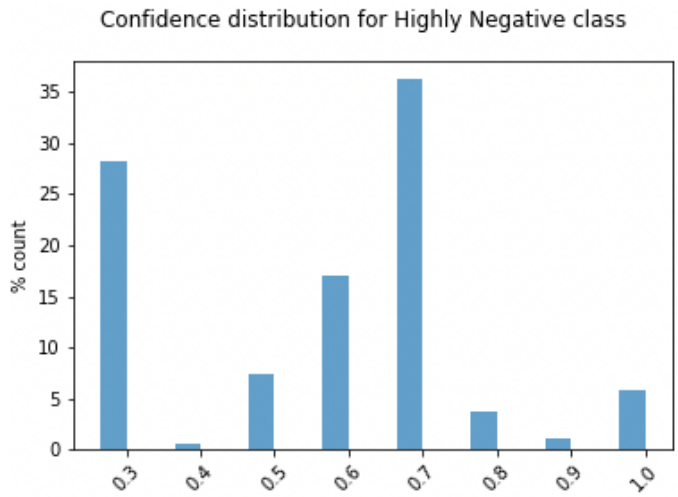


Figure 8: Distribution of Label confidence amongst the Highly negative class.

The above histograms are ordered from highest to lowest in proportion within the training data. As the proportion goes down, the confidence seems to decrease. The mean and std deviation for our five classes has been calculated below:

	Stat	Positive	Neutral	Negative	Highly Negative	Highly Positive
0	Count	5945.000000	1543.000000	1458.000000	188.000000	303.000000
1	Mean	0.786028	0.667348	0.665975	0.578648	0.558217
2	Std Dev	0.197685	0.182089	0.216491	0.183690	0.177621

Figure 9: Count, mean and std dev of label confidence for each class.

As the abundance of a class goes down, we can see that confidence in the label also goes down. We may be able to use this when fitting our model.

There are 528 feature sets to explore within the training set. Each value has been normalised to be within 0 and 1. The output for the describe function on all 538 features is included in an appendix at the end of the report. But from our output, we know we have 23 variables with a Standard Deviation above 0.1, which could contain some useful information. We also know that we have 128 variables with a mean, std dev, min and max of 0. These features have come from the 'Colour Histogram' function and don't contain any information.

We could potentially do a Step Forward to determine which of our features contain the most valuable information. We could do a PCA to reduce the number of features down. But for this assignment, I'll fit the data with a Random Forest model, a GBM model and an SVC model. In theory, performing feature selection for these won't models won't influence their performance. Therefore, I'll pass on all available features to each model. I attempted to remove these features from both the training and the testing dataset, but it decreased the model's accuracy. This could be because the number of observations we have in our dataset far exceeds the number of features we have, which minimises the impact of having too many features.

Also, by the looks of our random selection of images that appeared during EDA, we could assume a relationship between how positive an image is and how colourful that image

is. It seems that positive images contain large amounts of red, blue, and green. Our colour histogram should account for this.

3 Models

3.1 Random Forest

The first model that I will attempt to fit the data is the Random Forest model. This is an ensemble model most often used for classification (but can also be used for regression). This model operates by constructing many decision trees. Each decision tree has access to different features and different sets of bootstrapped samples. This means that any correlation within the features won't affect the model is each tree will contain different feature sets. Also only giving individual trees access to a limited number of features and samples will hopefully reduce overfitting to training data, which can be an issue for decision trees. For classification, the output is the class selected by the most trees. For regression tasks, the mean of the individual trees is returned (Deng, 2018). Ultimately the idea is that a large amount of less knowledgeable, uncorrelated trees operating as a committee will be able to make a more informed decision then a single tree with access to all the data.

These are popularly applied to both data science competitions and practical problems as they are often accurate and need little hyperparameter tuning (Deng, 2018).

When tuning the hyperparameters, I focused on the ones below:

- Number of trees (n_estimators) – the number of trees in the forest. The higher this figure, the more attempts to fit the data and more votes will be given. This will result in an increase in accuracy up to a point but will also increase in processing time.
- Criterion – this could be either “gini'' or “entropy”. This is the function to measure the quality of a split. A split can be based on “gini impurity'' or “entropy” for the information gain.
- BootStrap – whether bootstrap samples are used when building the trees. If set to false, the entire dataset will be used to construct each tree.
- Max Depth – the maximum depth of the tree. If none, nodes are expanded until all leaves are pure or contain less than min_samples_split samples.
- Min Sample Split – the minimum number of samples required to split an internal node.
- Min Sample Leaf – the minimum number of samples required to be a leaf node.

The above definitions were taken from Sklearn.ensemble.RandomForestClassifier, accessed in November 2021.

An advantage of this model is that it can output a list of features ranked by importance.

3.2 Gradient Boosting

The following kind of model I will be attempting to fit the data will be a Gradient Boosting algorithm. Boosting is also a decision tree modelling technique that works by converting weak learners into strong learners by boosting each new tree as a fit on a modified version of the original data set (Singh, H., 2021). The model begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, the weights are increased for those observations which are challenging to classify and lowered for those that are easy to classify. The second tree is then grown on this weighted data (Singh, H., 2021).

The objective is to improve upon the predictions made by the first tree. Our new model is, therefore, the first tree in addition to the second. The classification error is then computed from this new two tree ensemble, and a third tree is grown to predict the revised residuals. This process is then repeated for a specified number of iterations. Following trees help us to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is the weighted sum of the predictions made by the earlier models. Gradient Boosting trains numerous models in a gradual, additive, and sequential manner (Singh, H., 2021).

Hyperparameter tuning is essential for GBM modelling since they are prone to overfitting. The process of tuning the number of iterations for the Gradient Boosting model (and Random Forest as well) is called "Early Stopping". Early stopping performs model optimisation by monitoring the model's performance on a separate test data set. The training procedure is stopped once the performance on the test data set no longer improves beyond a given number of iterations (Singh, H., 2021).

It avoids overfitting by attempting to automatically select the inflection point where performance on the test dataset starts to decrease whilst the performance on the training dataset starts to increase due to overfitting. Early stopping can be based on either an out of bag sample set or cross-validation.

When tuning the hyperparameters, I focused on the number of estimators to prevent overfitting, as mentioned above.

3.3 Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression problems. It can be used for both linear separable data and non-linear separable data. It follows a technique called the ''kernel trick' to transform the data. Based on these transformations, it will find an optimal boundary between the possible outputs. The objective is to identify the optimal separating hyperplane, which maximizes the margin of the training data (Simple Tutorial on SVM and Parameter Tuning in Python and R.).

The hyperparameters of concern with this fitting method are as follows:

- C – the regularization parameter of the error term.
- Kernel – specifies the kernel type to be used in the algorithm. It can be “linear”, “poly”, “rbf”, “sigmoid”, “precomputed”, or something callable.
- Degree – It is the degree of the polynomial kernel function and is ignored by all other kernels. The default is 3.
- Gamma – It is the kernel coefficient for “rbf”, “poly”, “and “sigmoid”. If gamma is set to auto, then 1/n_features will be used instead.

The SVC can be tuned by changing the parameters C, Gamma and the kernel function. For this assignment, a basic grid-search will be done to try and find the ideal settings for these parameters.

3.4 Discussion of model difference(s)

The main difference between the Random Forest and the Gradient Boosting model is that Random Forest builds each decision tree individually while Gradient Boosting builds one tree at a time. This additive model works well in a forward stage-wise manner, introducing a weak learner to improve the shortcomings of existing weak learners. Random Forests combine the results for each decision tree at the end of the process (by majority rules for classification), while gradient boosting combines results along the way.

If the hyperparameters are tuned carefully, gradient boosting can result in better performance than Random Forests. Gradient boosting may not be an ideal choice if the data is very noisy, which can result in overfitting. They also tend to be a lot harder to tune than random forests (Glen, S. 2019). Random forests perform better for multi-class object detection, which tends to have a lot of statistical noise. Gradient boosting performs better when you have unbalanced data (Glen, S. 2019).

The way the Support Vector Machine modelling technique works differs fundamentally from the other two models. This is not a decision tree method and instead works by constructing a hyperplane (or a set of hyperplanes) in a high dimensional space which can be used to separate data points into classes.

4 Experiment setups

Firstly, the training set was loaded into Python. Next, an EDA was performed to look for significant features within the data set. The models picked for this experiment were Random Forest, GBM and SVC. These were picked as there are a lot of features to go through, and these models have built-in methods for feature selection. The use of an option such as PCA was considered but ultimately not used due to concerns that the code within the Jupyter notebook could not be adequately tested in time for assignment submission.

During EDA was determined that the “Label confidence” parameter may be used to weigh each sample during fitting. Therefore, for testing and development purposes, the

training set was split into a train and validation set. This was done because sample weight could not be considered when doing K-Fold cross-validation.

Models were optimised using the train and validation set. A "GridSearch" like optimisation was performed using the Random Forest Classifier were the ideal settings for criterion, bootstrap, number of trees, maximum depth, min sample split, min sample leaf was found. The model was then compared to the default parameters to see if there was any noticeable benefit.

Next, the Gradient Boost model was optimised using the train and validation set. Due to the outcome of the previous optimisation of Random Forest, it was decided to keep this optimisation simpler. The main concern with the Gradient Boost was the number of trees that would be fitted to the data. The more trees fitted, the more likely the model would be overfitted.

Lastly, the Support Vector Classifier model was optimised. The hyperparameters optimised for this were the C value, the kernel, and gamma.

Once hyperparameter optimisation was completed, our tuned models were compared to models with the models fitted using the default settings to see if our tuning had made a significant difference. Confusion matrices were generated for both the tuned and untuned models. Accuracy was then calculated for each as below:

$$Accuracy = \frac{number\ of\ correct\ predictions}{number\ of\ all\ predictions}$$

Lastly, the test data was fitted against our chosen model and results for the model with the highest accuracy were submitted.

5 Experimental results

Firstly, the dataset was imported into Python. Then, head and Shape functions were performed on the imported data frame. We know the import file contains 9437 observations and three features. These been the image ID, label, and confidence of the label. Next, we have a look at the label and label confidence in more detail.

We know from above that our balance is very imbalanced, favouring the "Positive" class greatly.

Label	
Highly negative	2.0
Highly positive	3.0
Negative	15.0
Neutral	16.0
Positive	63.0

Therefore, our models may have difficultly fitting this data. Next, we looked at the label confidence and found that the confidence was lower for our minority classes than it was for our majority class. I did attempt to use this for our "class_weight" when fitting our data, but it didn't seem to have a benefit when fitting our models on our training split and testing on the validation set.

Next, I attempted to tune the hyperparameters for each of the models. The tuning process found the following parameters to be ideal for the random forest model:

	Criterion	BootStrap	NTree	Max_Depth	Min_Samp_Split	Min_Samp_Leaf	Accuracy
4	entropy	True	50	10	5	2	62.711864

Figure 10: Result of fine-tuning the Random Forest model.

I next compared this to a model fitted with the default parameters, computed the confusion matrixes for both and compared. Comparison is below:

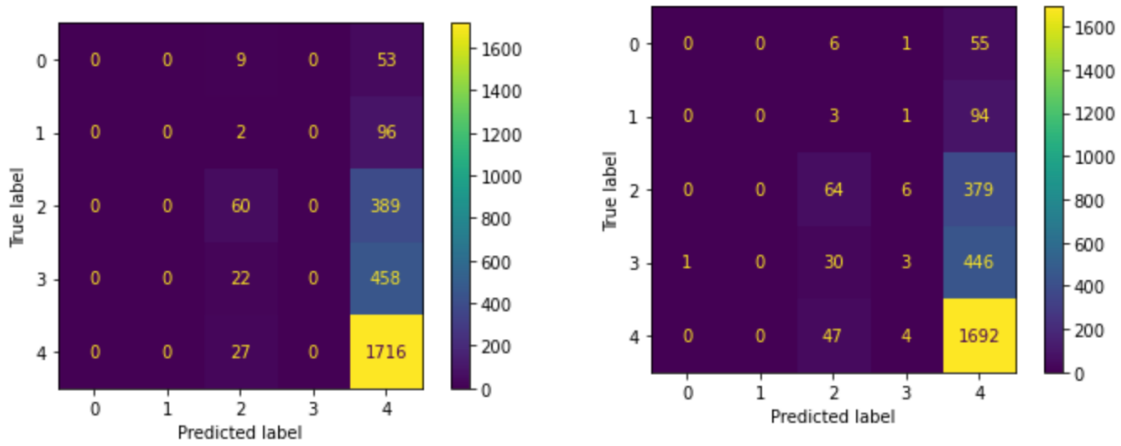


Figure 9: Confusion matrix for fitted model (left) and default parameter model (right) for Random Forest model.

The fitting has made a slight improvement to the accuracy of our model. Our fitted model has an accuracy of 62.7% and the model with default parameters has an accuracy of 62.1%. A null model on our split data would have an accuracy of 61.5%. It appears both slightly outperform that.

The next model I tried to fit was the Gradient Boost model. I only attempted to optimize the number of estimators for this model, as the literature suggested this one is the

most effective in preventing overfitting. The ideal number of estimators was determined as below:

	N_Estimator	Accuracy
3	31	62.076271

Figure 10: Results of fine-tuning the Gradient Boost model.

Next, I tried comparing this model to one with default parameters, calculating the confusion matrix and comparing. Results are below:

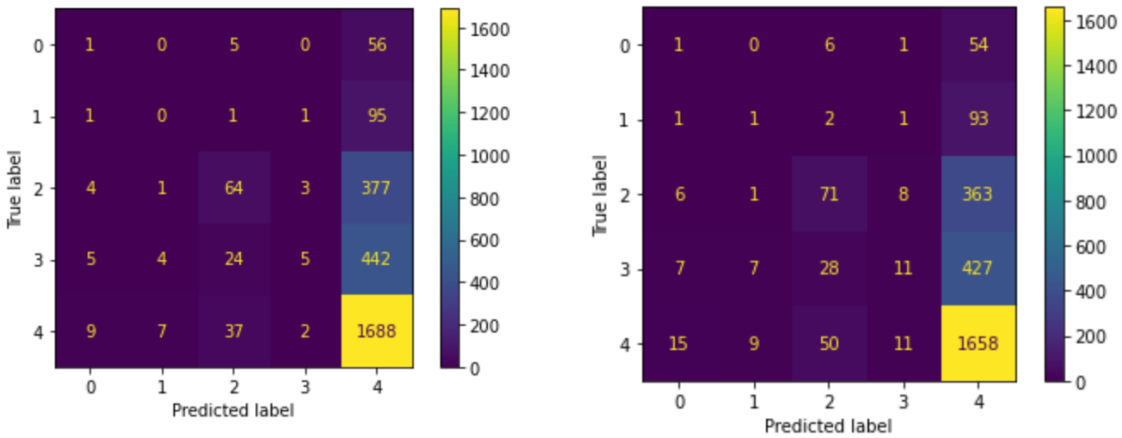


Figure 11: Confusion matrix for fitted model (left) and default parameter model (right) for Gradient Boost model.

The fitting has made a slight improvement to our model. Fitted the accuracy has an accuracy of 62.1% and with default parameters the accuracy is around 61.5%. It appears our default parameter mode has a lot more misclassified data. It looks like our tuned model for boosting outperforms the null model, which has an accuracy of 61.5%.

Lastly, I tried to tune the Support Vector Machines model. The ideal conditions were found to be as below according to my basic grid-search:

	Kernel	Gamma	C	Accuracy
32	sigmoid	scale	0.5	61.617232

Figure 12: Results of finetuning the SVM model.

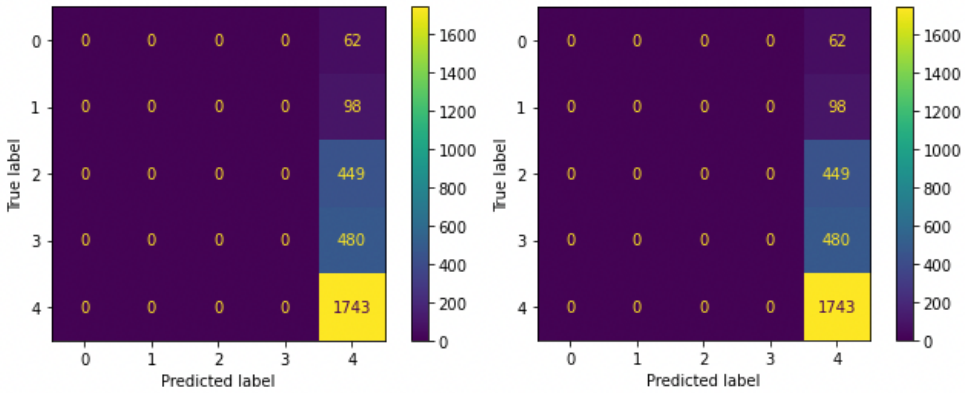


Figure 13: Confusion matrix for fitted model (left) and default parameter model (right) for the SVM model.

It doesn't look like either of our models perform any better than a null model. It appears that all models are having a tough time fitting against this data due to how unbalanced the data is or that there aren't sufficient strong features to pull predictions away from the majority class. This could be since there is more confidence in the positive class data than there is for the other classes. We could try taking the confidence of each class into account when fitting to see if this makes any difference. I'll try fitting to our default hyperparameter Random Forest model to see if introducing this can help us train our model.

Class weights were calculated as the mean of the confidence interval for each class divided by the largest of the four (which was positive). Results are below:

	4	3	2	1	0
0	1.0	0.849013	0.847266	0.736167	0.710174

Figure 14: Class weights calculated to give more weight to the more confident classes.

I then tried fitting the data with these class weights. It seems the results without class weights was slightly more accurate than without. Accuracy for data with class_weights was 62.5% and 62.7% for the model without:

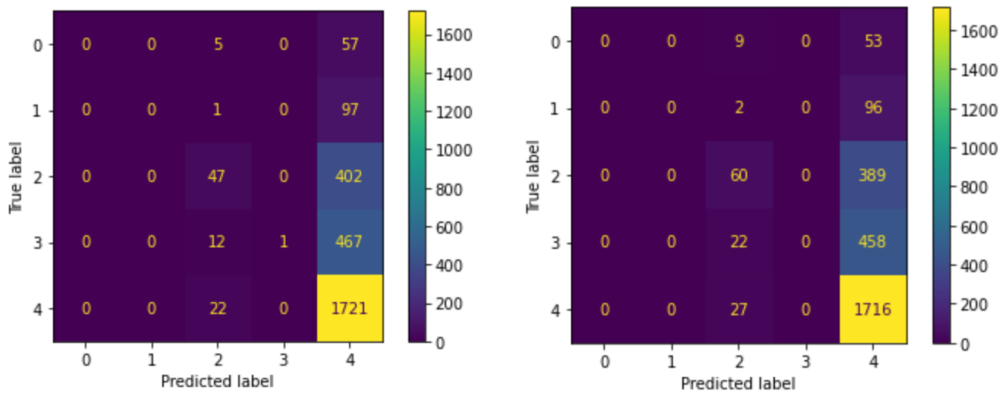


Figure 15: Confusion matrices for class weighted fit (left) and non-class weighted fit(right) for the random forest.

Lastly, I decided to use the fitted settings for the Random Forest model and Gradient Boost, fit them using our entire training set and make predictions for the test set. I decided to use the fitted settings as the accuracy of our models was slightly improved by the fitting. The

accuracy of the random forest model was slightly higher overall, so I am more likely to submit the data from this model as my final submission. SVC model didn't seem to perform any better than a null model, so I will not proceed further with that model.

The results of the Kaggle submission are tabulated below:

Model	Accuracy
Random Forest	0.71102
Gradient Boost	0.70912

Figure 16: Accuracy results from Kaggle.

I decided to submit the data for the Random Forest model for the final submission. This was because the test/training split indicated that the random forest model operated slightly better, which was backed up by the accuracy from the Kaggle submissions. Still, the accuracy of the fitted model wasn't great. It only slightly outperforms the null model. If more time was available, there are several more things I'd like to attempt to do. These include:

- Attempt a dimension reduction tool such as PCA, which could simplify the data we're trying to fit.
- Perform more EDA. It's clear from the data above that tuning hyperparameters won't take a poorly fitted model and make it work perfectly. There may be more optimisation that could be done during the EDA process to make it easier to classify our minority classes.
- More models. Maybe a decision tree or SVM wasn't the best choice for this data as the data is so imbalanced. There may be other models better suited for this. Maybe try a Neural Network instead.
- More features. Try to extract more information from the images, which could be used to make it easier to classify our test data.

6 Conclusion

In conclusion, it appears the optimal classifier from our test work above is the Random Forest classifier, as it achieved the highest accuracy for our test/training split sets. This operated best when all features were introduced as Random Forest has its own built-in feature selection tools. There were many features in the data set which contained no information, but there would have been no benefit in stripping them out.

The lesson I learned was training a classification model on very unbalanced data was very difficult, and that fine-tuning hyperparameters wouldn't be able to turn a poorly fitted/overfitted model into one that works well. The tuning of the hyperparameters only made a very slight improvement to the model. In this case, the best option may have been to continue EDA and find or introduce a feature that would provide a lot of information to our model.

There are several things I would have liked to try to get a better fit for the model. These would include other modelling techniques. For example, a decision tree of SVC may

not have been the best way to handle a dataset like this. I would have also liked to try dimension tools such as PCA to reduce the number of unrequired features. Though I believe the benefit may not have been significant due to the models we were using.

Lastly, it may be worth attempting to oversample and under-sample. However, I feel these may not have helped us a tremendous amount either. Under-sampling would remove information from our training set, and oversampling could increase the amount of noise.

The above experiment shows how random forest would be a good starting point for classification issues but is not guaranteed to be the best solution. The above demonstrates that tuning hyperparameters is not guaranteed to fix the problems with a bad fit. Perhaps more time spent on EDA and introducing features may have allowed for a better fitting model.

7 References

1. Shape Matching using Hu Moments (C++ / Python). (2021, November 30th) available from https://learnopencv.com/shape-matching-using-hu-moments-c-python/
2. Image moment. (2021, November 30th) available from https://en.wikipedia.org/wiki/Image_moment#Rotation_invariant_moments
3. Huang, Z., & Leng, J. (2010) AnaHu'slysis of Hu's moment invariants on image scaling and rotation, available from https://www.researchgate.net/publication/224146066_Analysis_of_Hu's_moment_invariants_on_image_scaling224146066_Analysis_of_Hu's_moment_invariants_on_image_scaling_and_rotation_and_rotation
4. Bynolfsson. P., et al. (2017) Haralick texture features from apparent diffusion coefficient (ADC) MRI images depend on imaging and pre-processing parameters.
5. Colour Histogram. (2021, November 30th) available from https://en.wikipedia.org/wiki/Color_histogram
6. Computing image colourfulness with OpenCV and Python. (2021, November 30th) Available from https://www.pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/
7. Deng, H., (2018) An Introduction to Random Forest, available from https://towardsdatascience.com/random-forest-3a55c3aca46d
8. Sklearn.ensemble.RandomForestClassifier., (2021, November 30th) available from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
9. Singh, H., 2021, Understanding Gradient Boosting Machines Available from https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab
10. Chelliah, I., 2020, An Introduction to Support Vector Machines Available from https://towardsdatascience.com/an-introduction-to-support-vector-machine-3f353241303b
11. Sklearn.SVM.SVC., available from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
12. Glen, S. 2019, Decision Tree vs Random Forest vs Gradient Boosting Machines Simply explained available from https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained

13. Simple Tutorial on SVM and Parameter Tuning in Python and R. Available from https://www.hackerearth.com/blog/developers/simple-tutorial-svm-parameter-tuning-python-r/

8 Appendix

Appendix 1: Summary of training dataset.

Column	Mean	Std_Dev	RSD	Median	Max	Min
533	0.6097	0.2694	44.18	0.6885	1	0
0	0.1453	0.2616	179.99	0.0127	1	0
532	0.3892	0.2452	63.01	0.3694	1	0
7	0.1202	0.2112	175.67	0.0234	1	0
537	0.4232	0.2017	47.67	0.3974	1	0
516	0.3399	0.1713	50.41	0.3214	1	0
523	0.57	0.158	27.72	0.573	1	0
517	0.4207	0.1501	35.68	0.4188	1	0
535	0.4361	0.1395	32	0.4388	1	0
522	0.5247	0.1389	26.47	0.5304	1	0
515	0.255	0.1351	52.99	0.2327	1	0
518	0.25	0.1342	53.67	0.2276	1	0
199	0.0535	0.1334	249.48	0.0034	1	0
538	0.2242	0.1251	55.82	0.2077	1	0
536	0.2213	0.125	56.47	0.2049	1	0
6	0.0617	0.1233	199.74	0.0106	1	0
534	0.2651	0.1224	46.14	0.2505	1	0
5	0.06	0.1182	196.97	0.0099	1	0
520	0.7408	0.1175	15.86	0.7596	1	0
4	0.0573	0.1145	199.67	0.0093	1	0
198	0.043	0.1112	258.84	0.002	1	0
56	0.0488	0.1104	226.48	0.0064	1	0
519	0.8912	0.1036	11.63	0.9219	1	0
20	0.0577	0.0999	173.2	0.0156	1	0
11	0.0536	0.0987	184.08	0.0123	1	0
14	0.0456	0.0975	213.56	0.0078	1	0
58	0.0379	0.0965	254.82	0.0016	1	0
3	0.0458	0.0965	210.45	0.0065	1	0
63	0.0215	0.0962	447.42	0	1	0
15	0.0403	0.0961	238.69	0.0049	1	0
57	0.0382	0.0954	250.15	0.0027	1	0
13	0.0492	0.095	192.95	0.0107	1	0
197	0.0372	0.094	252.9	0.0024	1	0
1	0.0357	0.0938	262.87	0.0034	1	0

19	0.0541	0.0926	171.29	0.0156	1	0
12	0.05	0.0916	183.18	0.0118	1	0
2	0.0393	0.0908	230.89	0.0049	1	0
21	0.0469	0.0901	192.07	0.0104	1	0
207	0.026	0.0894	343.44	0.0001	1	0
36	0.0472	0.0893	189.12	0.0091	1	0
28	0.0489	0.0879	179.65	0.0122	1	0
37	0.04	0.0878	219.78	0.005	1	0
23	0.0333	0.086	258.54	0.0026	1	0
29	0.0432	0.0855	198.01	0.0081	1	0
22	0.0398	0.0854	214.54	0.0062	1	0
42	0.0439	0.0852	194.21	0.0081	1	0
514	0.912	0.0852	9.34	0.9379	1	0
27	0.0476	0.0843	177.13	0.0132	1	0
34	0.0472	0.084	177.75	0.0122	1	0
35	0.0458	0.0836	182.48	0.0111	1	0
18	0.0463	0.0834	180.06	0.0127	1	0
205	0.0288	0.0829	287.86	0.0011	1	0
206	0.0273	0.0825	302.05	0.0004	1	0
60	0.029	0.0824	284.1	0.0002	1	0
26	0.047	0.0811	172.61	0.0135	1	0
30	0.0346	0.08	231.58	0.0039	1	0
51	0.0337	0.0799	236.68	0.0028	1	0
521	0.1022	0.0795	77.75	0.0818	1	0
59	0.0292	0.079	269.94	0.0006	1	0
9	0.0364	0.0773	212.25	0.0068	1	0
135	0.0262	0.0766	291.85	0.0021	1	0
62	0.0205	0.0765	373.43	0	1	0
61	0.024	0.0762	317.71	0	1	0
10	0.0379	0.0758	199.97	0.0078	1	0
215	0.0182	0.0752	413.12	0	1	0
196	0.0286	0.075	262.61	0.002	1	0
204	0.0283	0.0749	265.06	0.0016	1	0
33	0.0387	0.0748	193.42	0.0094	1	0
120	0.0246	0.0741	301.16	0.0019	1	0
17	0.0386	0.0737	190.78	0.009	1	0
43	0.0362	0.0732	202.26	0.0058	1	0
25	0.0385	0.073	189.41	0.0099	1	0
223	0.0148	0.0728	492.43	0	1	0
73	0.0258	0.072	278.4	0.0024	1	0
214	0.0198	0.0713	360.02	0	1	0
44	0.0333	0.0704	211.59	0.004	1	0

41	0.0341	0.0702	205.5	0.0067	1	0
31	0.0253	0.0698	276.37	0.001	1	0
45	0.0282	0.0692	245.22	0.0019	1	0
39	0.0219	0.069	315.23	0.0002	1	0
211	0.0235	0.0689	293.07	0.0009	1	0
524	0.9672	0.0687	7.1	0.9903	1	0
71	0.0222	0.0684	308.56	0.0027	1	0
47	0.0193	0.0677	349.77	0	1	0
49	0.0302	0.0673	223.28	0.0041	1	0
40	0.0285	0.0669	234.72	0.0052	1	0
219	0.0199	0.0665	334.31	0.0003	1	0
213	0.0204	0.0662	323.99	0.0002	1	0
230	0.0136	0.0661	485.11	0	1	0
203	0.0252	0.0659	261.92	0.0016	1	0
195	0.0233	0.0658	282.15	0.0016	1	0
70	0.0202	0.0655	324.55	0.0013	1	0
46	0.0231	0.0655	283.6	0.0005	1	0
209	0.0231	0.0654	282.92	0.0013	1	0
52	0.025	0.0651	260.77	0.0013	1	0
222	0.0157	0.0645	411.42	0	1	0
212	0.0206	0.0642	311.07	0.0005	1	0
54	0.0187	0.0641	343.6	0.0001	1	0
220	0.0183	0.0637	347.35	0.0001	1	0
201	0.0227	0.0636	280.81	0.0016	1	0
231	0.0113	0.0635	561.15	0	1	0
248	0.0186	0.0632	338.78	0.001	1	0
210	0.0217	0.0626	288.04	0.0011	1	0
38	0.0251	0.062	247.36	0.0016	1	0
229	0.0142	0.0617	434.33	0	1	0
512	0.0155	0.0617	397.1	0.0013	1	0
32	0.0259	0.061	235.49	0.0051	1	0
221	0.0165	0.0605	365.84	0	1	0
218	0.0188	0.0605	321.09	0.0005	1	0
104	0.0184	0.0604	327.65	0.0015	1	0
55	0.0152	0.0603	397.88	0	1	0
134	0.0173	0.0603	347.92	0.0008	1	0
98	0.0178	0.0603	339.54	0.0001	1	0
50	0.0273	0.0595	217.92	0.0033	1	0
68	0.0215	0.0588	273.61	0.0021	1	0
225	0.0157	0.0585	372.3	0.0003	1	0
260	0.019	0.0585	307.25	0.0014	1	0
67	0.0207	0.0581	280.36	0.0019	1	0

69	0.0195	0.0579	297.23	0.0016	1	0
99	0.0147	0.0567	384.47	0	1	0
53	0.0197	0.0567	287.8	0.0004	1	0
202	0.0224	0.0567	253.22	0.0017	1	0
513	0.0561	0.0564	100.48	0.0397	1	0
133	0.0174	0.0556	318.99	0.001	1	0
121	0.0128	0.0555	434.52	0	1	0
250	0.0092	0.0552	599.49	0	1	0
90	0.0163	0.0552	339.7	0.0002	1	0
228	0.0134	0.0542	404.93	0	1	0
75	0.0177	0.0542	306.14	0.001	1	0
74	0.0202	0.0542	267.81	0.0016	1	0
122	0.011	0.054	491.85	0	1	0
251	0.0088	0.0539	610.96	0	1	0
81	0.0187	0.0539	288.91	0.0012	1	0
82	0.0171	0.0538	313.84	0.0006	1	0
139	0.0145	0.0536	370.59	0.0004	1	0
89	0.0174	0.0536	307.35	0.0007	1	0
194	0.0184	0.0536	290.82	0.0013	1	0
171	0.0074	0.0529	713.9	0	1	0
227	0.0131	0.0528	403.45	0	1	0
252	0.0081	0.0525	648.3	0	1	0
131	0.0172	0.0525	304.74	0.0012	1	0
226	0.0137	0.0523	382.28	0.0001	1	0
237	0.0102	0.052	509.65	0	1	0
208	0.0154	0.0515	334.6	0.0016	1	0
83	0.0151	0.0513	341.05	0.0002	1	0
238	0.0086	0.0511	592.22	0	1	0
249	0.0091	0.0509	558.22	0	1	0
224	0.0147	0.0509	345.69	0.0017	1	0
16	0.0212	0.0509	239.81	0.0038	1	0
236	0.0112	0.0506	450.36	0	1	0
325	0.0152	0.0506	333.08	0.0011	1	0
253	0.007	0.0505	719.39	0	1	0
234	0.0111	0.0501	453.34	0	1	0
263	0.0149	0.05	336.51	0.0013	1	0
66	0.0163	0.05	306.23	0.0015	1	0
161	0.0101	0.0498	493.75	0	1	0
235	0.0107	0.0498	466.55	0	1	0
132	0.0168	0.0498	295.88	0.0013	1	0
123	0.0085	0.0497	584.99	0	1	0
115	0.0107	0.0497	463.76	0	1	0

255	0.0049	0.0496	1014.58	0	1	0
142	0.0088	0.0493	559.66	0	1	0
100	0.0106	0.0493	464.86	0	1	0
92	0.0114	0.0491	429.28	0	1	0
91	0.0132	0.0491	372.06	0	1	0
239	0.007	0.049	704.82	0	1	0
78	0.0094	0.0486	514.1	0.0001	1	0
107	0.0115	0.0485	420.96	0	1	0
93	0.0089	0.0484	541.58	0	1	0
217	0.0148	0.0484	327.17	0.0006	1	0
193	0.0158	0.0482	305.53	0.0012	1	0
245	0.0075	0.0481	643.29	0	1	0
338	0.0118	0.0481	408.47	0.0004	1	0
97	0.0143	0.0478	335.15	0.0002	1	0
88	0.0154	0.0478	309.91	0.0014	1	0
114	0.0114	0.0477	418.27	0	1	0
246	0.0065	0.0475	726.64	0	1	0
8	0.0172	0.0472	275.08	0.0023	1	0
24	0.0192	0.0468	243.16	0.0034	1	0
138	0.0143	0.0466	326.55	0.0006	1	0
254	0.0053	0.0465	880.17	0	1	0
106	0.0116	0.0459	394.6	0	1	0
48	0.0187	0.0459	244.91	0.0026	1	0
124	0.0067	0.0458	686.25	0	1	0
525	0.0555	0.0457	82.3	0.0456	1	0
113	0.0115	0.0456	396.99	0	1	0
330	0.0126	0.0454	360.63	0.0008	1	0
76	0.0132	0.0454	344.32	0.0005	1	0
244	0.0075	0.0453	607.18	0	1	0
140	0.011	0.0453	411.31	0.0002	1	0
85	0.0092	0.0452	492.32	0	1	0
80	0.0135	0.0452	335.34	0.0015	1	0
327	0.0119	0.045	377.27	0.0009	1	0
145	0.0124	0.0436	353.01	0.0004	1	0
86	0.0075	0.0435	582.72	0	1	0
243	0.0077	0.0435	563.77	0	1	0
84	0.0111	0.0435	392.96	0.0001	1	0
200	0.013	0.0435	333.76	0.0011	1	0
276	0.0057	0.043	752.68	0	1	0
96	0.0134	0.043	322.01	0.0014	1	0
129	0.0126	0.0429	340.25	0.001	1	0
240	0.0109	0.0428	391.82	0.0007	1	0

247	0.0046	0.0427	929	0	1	0
143	0.0064	0.0426	662.17	0	1	0
116	0.0077	0.0426	553.63	0	1	0
108	0.0088	0.0425	485.96	0	1	0
323	0.0124	0.0423	342.09	0.001	1	0
137	0.0132	0.0423	319.46	0.0008	1	0
141	0.0086	0.0422	488.87	0.0001	1	0
262	0.0107	0.042	392.12	0.0005	1	0
152	0.0109	0.042	384.86	0.0009	1	0
72	0.0129	0.042	325.61	0.0014	1	0
241	0.0081	0.0418	513.23	0	1	0
332	0.0091	0.0418	460.85	0.0003	1	0
216	0.0113	0.0407	360.3	0.001	1	0
105	0.0112	0.0404	360.38	0.0001	1	0
273	0.0077	0.0402	522.29	0.0002	1	0
346	0.0082	0.0402	489.93	0.0002	1	0
266	0.0093	0.0401	428.96	0.0003	1	0
242	0.0073	0.04	550.47	0	1	0
109	0.0057	0.0396	693.13	0	1	0
377	0.0067	0.0395	590.43	0	1	0
268	0.0069	0.0395	569.01	0.0001	1	0
130	0.0118	0.0395	334.05	0.0009	1	0
233	0.0087	0.0393	451.04	0.0001	1	0
65	0.0125	0.0393	314.42	0.0012	1	0
326	0.0098	0.0389	394.6	0.0005	1	0
77	0.0094	0.0385	410.54	0.0002	1	0
376	0.0098	0.0385	393.46	0.0005	1	0
169	0.0064	0.0382	598.69	0	1	0
136	0.0103	0.0381	370.95	0.0008	1	0
160	0.0106	0.038	358.63	0.001	1	0
64	0.0127	0.038	300.39	0.0014	1	0
374	0.0053	0.0378	710.34	0	1	0
177	0.0058	0.0375	643.2	0	1	0
162	0.0062	0.0375	602.76	0	1	0
155	0.0063	0.0374	590.62	0	1	0
147	0.0081	0.0374	460.47	0	1	0
272	0.0091	0.0372	407.96	0.0011	1	0
144	0.0099	0.0372	376.96	0.001	1	0
232	0.0107	0.0372	347.75	0.0011	1	0
188	0.0034	0.037	1092.62	0	1	0
333	0.0074	0.037	498.01	0.0001	1	0
172	0.0041	0.0369	906.43	0	1	0

170	0.0055	0.0369	666.55	0	1	0
368	0.0084	0.0369	436.8	0.0006	1	0
146	0.0087	0.0367	423.98	0.0001	1	0
342	0.0051	0.0363	715.4	0	1	0
269	0.0051	0.0362	709.55	0	1	0
101	0.0063	0.0362	578.72	0	1	0
178	0.0046	0.036	789.45	0	1	0
371	0.0061	0.0357	582.17	0	1	0
355	0.0069	0.0357	519.04	0	1	0
125	0.0039	0.0356	912.28	0	1	0
334	0.0067	0.0356	534.64	0	1	0
322	0.0104	0.0353	340.54	0.0009	1	0
331	0.0085	0.0352	412.38	0.0004	1	0
261	0.0103	0.0352	343.15	0.0006	1	0
87	0.0041	0.0349	857.27	0	1	0
154	0.0069	0.0348	505.48	0	1	0
157	0.0047	0.0347	732.4	0	1	0
329	0.01	0.0344	344.72	0.0009	1	0
340	0.0064	0.0343	538.4	0.0001	1	0
378	0.0058	0.0342	588.44	0	1	0
164	0.0045	0.034	764.39	0	1	0
258	0.01	0.0338	339.37	0.0008	1	0
354	0.0062	0.0335	536.66	0.0001	1	0
153	0.0071	0.0335	469.43	0.0001	1	0
259	0.01	0.0335	334.8	0.0008	1	0
163	0.0049	0.0334	681.13	0	1	0
379	0.0055	0.0333	603.35	0	1	0
345	0.007	0.0331	476.94	0.0003	1	0
270	0.0041	0.0329	804.81	0	1	0
168	0.0079	0.0329	417.46	0.0006	1	0
347	0.0061	0.0327	540.19	0	1	0
156	0.0053	0.0326	619.77	0	1	0
257	0.0095	0.0323	339.15	0.0009	1	0
187	0.0033	0.0321	978.03	0	1	0
359	0.0037	0.0321	856.86	0	1	0
349	0.005	0.032	644.84	0	1	0
151	0.0035	0.0319	909.68	0	1	0
117	0.0041	0.0319	782.52	0	1	0
353	0.0062	0.0318	511.71	0.0002	1	0
375	0.0033	0.0316	965.15	0	1	0
275	0.0045	0.0316	699.37	0	1	0
184	0.0071	0.0313	442.48	0.0004	1	0

267	0.0067	0.0312	467.27	0.0001	1	0
94	0.0044	0.0311	713.46	0	1	0
336	0.0067	0.0311	464.09	0.0007	1	0
186	0.0031	0.031	1002.7	0	1	0
380	0.0042	0.0309	736.95	0	1	0
265	0.008	0.0308	385.93	0.0005	1	0
112	0.008	0.0308	384.85	0.0004	1	0
150	0.0035	0.0304	862.15	0	1	0
360	0.0072	0.0303	422.53	0.0007	1	0
280	0.0066	0.0302	459.18	0.0006	1	0
321	0.0078	0.0302	386.38	0.0007	1	0
189	0.0022	0.0301	1354.83	0	1	0
335	0.0037	0.0301	815.26	0	1	0
148	0.0059	0.0301	514.07	0	1	0
126	0.0022	0.03	1390.36	0	1	0
79	0.0046	0.03	652.08	0	1	0
179	0.0035	0.0299	846.45	0	1	0
324	0.0084	0.0299	354.38	0.0007	1	0
312	0.0084	0.0296	350.68	0.0005	1	0
149	0.0047	0.0294	624.5	0	1	0
363	0.0047	0.0293	624.15	0	1	0
382	0.0031	0.0292	927.93	0	1	0
373	0.0042	0.0292	701.83	0	1	0
176	0.0053	0.0292	546.29	0.0002	1	0
277	0.0031	0.0291	948.01	0	1	0
95	0.0035	0.029	838.16	0	1	0
118	0.0027	0.0287	1058.32	0	1	0
357	0.0037	0.0287	780.04	0	1	0
274	0.0046	0.0284	619.43	0	1	0
344	0.006	0.0284	471.4	0.0006	1	0
185	0.0035	0.0283	815.33	0	1	0
361	0.0054	0.0283	520.22	0.0001	1	0
287	0.0019	0.0282	1464.75	0	1	0
381	0.0033	0.0274	823.29	0	1	0
383	0.0022	0.0273	1217.27	0	1	0
110	0.0028	0.0273	973.68	0	1	0
281	0.004	0.0273	682.94	0	1	0
288	0.0053	0.0272	508.24	0.0006	1	0
328	0.0061	0.0272	443.2	0.0006	1	0
298	0.0026	0.0271	1033.32	0	1	0
296	0.005	0.0271	540.28	0.0004	1	0
337	0.0067	0.0271	401.97	0.0004	1	0

351	0.0026	0.0265	998.12	0	1	0
350	0.0033	0.0264	797.11	0	1	0
128	0.0061	0.0262	431.77	0.0004	1	0
364	0.0036	0.0261	731.02	0	1	0
278	0.0023	0.026	1142.9	0	1	0
367	0.0027	0.026	966.75	0	1	0
358	0.0036	0.0259	714.09	0	1	0
343	0.0029	0.0255	870.54	0	1	0
339	0.0054	0.0255	471.77	0.0001	1	0
356	0.0038	0.0254	671.13	0	1	0
372	0.0033	0.0249	743.8	0	1	0
264	0.0066	0.0249	379.36	0.0007	1	0
180	0.0023	0.0248	1080.51	0	1	0
127	0.0014	0.0247	1772.72	0	1	0
341	0.0038	0.0244	648.31	0	1	0
301	0.0015	0.0242	1614.51	0	1	0
271	0.0024	0.024	990.85	0	1	0
352	0.0052	0.0236	456.96	0.0006	1	0
181	0.0017	0.0235	1405.1	0	1	0
175	0.0019	0.0234	1243.11	0	1	0
320	0.0053	0.0234	441.1	0.0003	1	0
158	0.0023	0.0233	994.03	0	1	0
289	0.0028	0.0233	822.6	0	1	0
165	0.0025	0.0232	941.79	0	1	0
174	0.0018	0.023	1311.1	0	1	0
102	0.0029	0.0229	792.29	0	1	0
365	0.0031	0.0229	734.88	0	1	0
307	0.0014	0.0228	1685.81	0	1	0
290	0.0024	0.0225	929.47	0	1	0
166	0.0018	0.0224	1209.39	0	1	0
306	0.0014	0.0222	1541.92	0	1	0
283	0.0025	0.0222	900.81	0	1	0
348	0.0033	0.0222	675.26	0	1	0
370	0.0032	0.0219	679.01	0	1	0
256	0.0061	0.0219	356.91	0.0007	1	0
302	0.0012	0.0217	1790.52	0	1	0
366	0.0025	0.0217	863.58	0	1	0
304	0.0037	0.0216	590.67	0.0002	1	0
291	0.0019	0.0215	1136.81	0	1	0
305	0.0021	0.0215	1034.08	0	1	0
191	0.0011	0.0213	1968.74	0	1	0
279	0.0015	0.021	1426.23	0	1	0

297	0.0022	0.0208	939.88	0	1	0
362	0.0031	0.0208	664.12	0	1	0
192	0.0052	0.0208	397.58	0.0004	1	0
103	0.0021	0.0204	976.03	0	1	0
310	0.001	0.0203	1956.62	0	1	0
284	0.0018	0.0203	1122.56	0	1	0
286	0.0015	0.02	1327.38	0	1	0
111	0.0015	0.0199	1315.65	0	1	0
159	0.0016	0.0199	1228.85	0	1	0
190	0.0012	0.0198	1718.01	0	1	0
293	0.0013	0.0193	1465.44	0	1	0
285	0.0015	0.0191	1254.84	0	1	0
282	0.0023	0.019	828.85	0	1	0
173	0.0015	0.0189	1224.08	0	1	0
369	0.0026	0.0189	729.18	0	1	0
299	0.0012	0.0187	1561.34	0	1	0
167	0.0013	0.0184	1394.14	0	1	0
308	0.001	0.0182	1873.53	0	1	0
309	0.0008	0.0179	2352.84	0	1	0
313	0.001	0.0179	1771.43	0	1	0
294	0.001	0.0174	1746.56	0	1	0
182	0.0008	0.0173	2169.32	0	1	0
292	0.0013	0.0173	1300.84	0	1	0
314	0.0008	0.0166	2013.05	0	1	0
183	0.0008	0.016	1952.94	0	1	0
303	0.0007	0.0158	2339.29	0	1	0
300	0.0009	0.0156	1743.28	0	1	0
119	0.001	0.0154	1539.58	0	1	0
316	0.0005	0.0149	2796.61	0	1	0
295	0.0008	0.0148	1946.72	0	1	0
527	0.0003	0.0121	3749.13	0	1	0
526	0.0009	0.0121	1373.54	0.0001	1	0
311	0.0003	0.0119	4148.87	0	1	0
319	0.0002	0.0117	5063.42	0	1	0
528	0.0003	0.0113	4344.92	0	1	0
317	0.0003	0.0112	3949.45	0	1	0
315	0.0004	0.011	3053.03	0	1	0
318	0.0002	0.0109	4688.06	0	1	0
530	0.0148	0.0107	72.49	0.0146	1	0
529	0.0036	0.0103	283.79	0.0035	1	0
531	0.3654	0.0077	2.1	0.3654	1	0
497	0	0	0	0	0	0

496	0	0	0	0	0	0
495	0	0	0	0	0	0
490	0	0	0	0	0	0
501	0	0	0	0	0	0
494	0	0	0	0	0	0
493	0	0	0	0	0	0
492	0	0	0	0	0	0
491	0	0	0	0	0	0
498	0	0	0	0	0	0
499	0	0	0	0	0	0
500	0	0	0	0	0	0
502	0	0	0	0	0	0
406	0	0	0	0	0	0
504	0	0	0	0	0	0
489	0	0	0	0	0	0
488	0	0	0	0	0	0
487	0	0	0	0	0	0
486	0	0	0	0	0	0
485	0	0	0	0	0	0
484	0	0	0	0	0	0
483	0	0	0	0	0	0
482	0	0	0	0	0	0
481	0	0	0	0	0	0
480	0	0	0	0	0	0
479	0	0	0	0	0	0
478	0	0	0	0	0	0
477	0	0	0	0	0	0
476	0	0	0	0	0	0
503	0	0	0	0	0	0
508	0	0	0	0	0	0
505	0	0	0	0	0	0
396	0	0	0	0	0	0
384	0	0	0	0	0	0
385	0	0	0	0	0	0
386	0	0	0	0	0	0
387	0	0	0	0	0	0
388	0	0	0	0	0	0
389	0	0	0	0	0	0
390	0	0	0	0	0	0
391	0	0	0	0	0	0
392	0	0	0	0	0	0
393	0	0	0	0	0	0

394	0	0	0	0	0	0
395	0	0	0	0	0	0
397	0	0	0	0	0	0
506	0	0	0	0	0	0
398	0	0	0	0	0	0
399	0	0	0	0	0	0
400	0	0	0	0	0	0
401	0	0	0	0	0	0
402	0	0	0	0	0	0
403	0	0	0	0	0	0
404	0	0	0	0	0	0
511	0	0	0	0	0	0
510	0	0	0	0	0	0
509	0	0	0	0	0	0
474	0	0	0	0	0	0
507	0	0	0	0	0	0
475	0	0	0	0	0	0
473	0	0	0	0	0	0
407	0	0	0	0	0	0
423	0	0	0	0	0	0
437	0	0	0	0	0	0
436	0	0	0	0	0	0
435	0	0	0	0	0	0
434	0	0	0	0	0	0
433	0	0	0	0	0	0
432	0	0	0	0	0	0
431	0	0	0	0	0	0
430	0	0	0	0	0	0
429	0	0	0	0	0	0
428	0	0	0	0	0	0
427	0	0	0	0	0	0
426	0	0	0	0	0	0
425	0	0	0	0	0	0
424	0	0	0	0	0	0
422	0	0	0	0	0	0
439	0	0	0	0	0	0
421	0	0	0	0	0	0
420	0	0	0	0	0	0
419	0	0	0	0	0	0
418	0	0	0	0	0	0
417	0	0	0	0	0	0
416	0	0	0	0	0	0

415	0	0	0	0	0	0
414	0	0	0	0	0	0
413	0	0	0	0	0	0
412	0	0	0	0	0	0
411	0	0	0	0	0	0
410	0	0	0	0	0	0
409	0	0	0	0	0	0
408	0	0	0	0	0	0
438	0	0	0	0	0	0
440	0	0	0	0	0	0
472	0	0	0	0	0	0
457	0	0	0	0	0	0
471	0	0	0	0	0	0
470	0	0	0	0	0	0
469	0	0	0	0	0	0
468	0	0	0	0	0	0
467	0	0	0	0	0	0
466	0	0	0	0	0	0
465	0	0	0	0	0	0
464	0	0	0	0	0	0
463	0	0	0	0	0	0
462	0	0	0	0	0	0
405	0	0	0	0	0	0
460	0	0	0	0	0	0
459	0	0	0	0	0	0
458	0	0	0	0	0	0
456	0	0	0	0	0	0
441	0	0	0	0	0	0
455	0	0	0	0	0	0
454	0	0	0	0	0	0
453	0	0	0	0	0	0
452	0	0	0	0	0	0
451	0	0	0	0	0	0
450	0	0	0	0	0	0
449	0	0	0	0	0	0
448	0	0	0	0	0	0
447	0	0	0	0	0	0
446	0	0	0	0	0	0
445	0	0	0	0	0	0
444	0	0	0	0	0	0
443	0	0	0	0	0	0
442	0	0	0	0	0	0

461	0	0	0	0	0	0