Using an ensemble of Neural Networks and Bayes Classifiers to predict post popularity of Instagram.

Candidate ID: 20948

May 8, 2021

1 Abstract

Given that Instagram is marketed as an image centric social network we would expect that a lot of the decision as to whether to engage with a Post is image dependent. Our analysis shows that this is not the case and that an ensemble method based mainly on the text data that accompanies an image (Captions, Mentions, and Hashtags) can provide a better predictive model than that of a Convolutional neural network trained on the image itself.

2 Introduction

Instagram is a social network that centers around sharing an image with an optional caption and tags. The specific intentions of why people use Instagram are more of a Psychological nature but the overarching element is that, in general, if given the option of uploading a photo that will receive a lot of attention, or little attention, most people would choose the former. This makes intuitive sense as if you had aspirations of receiving little attention for your posts then you wouldn't upload them in the first place. To this end, I aim to use an ensemble of a Convolutional Neural Network, an Artificial Neural Network, multiple Multinomial Bayes Classifiers D. J. Hand **and** Yu (2001), as well as multiple decision tree based model, to see if I can predict which posts will be popular, and hopefully why they are popular. I will refer to *Post* to mean the image, caption, and meta-data as a package, and not solely the image.

3 Paper Structure

The structure of this paper is as follows. Section 4 first describes the data collection, pre-processing, and separation. Section 7.1 introduces a number of adjustments to the Naive Bayes classifier individually to the Captions, Mentions, and Hashtags, and then as an ensemble where the Vectorisers are applied to the data first, combined, and then analysed as one input per image. Section 7.3 Introduces the use of 4 classifiers to analyse the meta-data that each image includes; such as Date, Time, Number of hashtags, etc. We then move on to using an Artificial Neural Network in Section 7.3.5 on the same image meta-data to compare the results with Section 7.3. Up to this point we have only looked at the meta-data and text that accompanies the image. Section 7.5 takes the analysis one step further and introduces a Convolutional Neural Network to analyse the image itself. Lastly we look at the limitations of my approaches and extensions I would make to the project in Section 8.

4 Data

4.1 Collection

Data was collected pseudo-randomly[1] by navigating to the aggregate pages for the hashtags #Selfie and #London and then selecting every account from the first column. I only selected accounts that wrote captions in English[2].

[1]Pseudo-random as I downloaded posts when I was free and not at a specified random time of the day. It could be argued that a bias was given to the accounts that post during my least unencumbered time of the day.

[2]This was not exhaustive as I did a cursory check on their most recent 5 posts to check the caption language. I did check the language of the caption during preprocessing using the *langdetect* package but this often gives an incorrect output as it struggles to classify slang and Unicode characters so I removed it and continued with the assumption that a person would be consistent with the language used in all of their captions.

4.2 Class Definition

Instagram uses the terminology *Likes* and *Followers* to mean the number of interactions others had with your post, and the number of people who have subscribed to your feed specifically, respectively. The models I will be using require labels for each of our Posts to allow it to learn which were successful and which were unsuccessful, I will refer to these as *Class(es).* The Class is the ratio of Likes to Followers. I have taken this step to normalise the Like count as we can not compare in absolute terms as a celebrity is more likely to receive 1,000 Likes for a photo than a smaller account. These 1,000 Likes are more significant for the smaller account.

Data availability is limited to only the current number of Likes for each post, and the current number of Followers the account has. This means that using the current number of Followers to calculate the ratio for older posts becomes increasingly inaccurate the older the post. For example, if an account now has lots of Followers and we are looking at a post made near the start of the account when the Follower count was much lower then it would suggest that the post is unpopular. This may not have been the case though, it was just a matter of the fact that at the time of posting, the account had few Followers and has since become more popular. Therefore, I had to estimate the number of Followers at any given time point during the account's lifespan, indeed on the date that each post was made.

After considering how I expect Follower growth to occur, and asking 20 other social network users, I believe the best approximation is to use a Logarithmic function with the limits [0, Current number of Followers][3]. I chose this function as it is probable that most accounts on Instagram were not the person's first social network account. It is likely that the person already had other social network accounts to which they already had a following so could post on those networks that they had created an Instagram account and request that those same people follow them on Instagram too. This would lead to an influx of Followers at the initial stages of the account's life which is not organic. This large spike would then flatten off as all of their current friends, family, colleagues, et al had agreed to Follow them on Instagram.

Therefore, the estimation of Class is; Class $= \frac{\text{Current number of Post Likes}}{\text{Estimated Followers on date of posting}}$

[3]As log(0) is undefined I had to set the number of Followers for day 0 as exactly 0, which is slightly inaccurate but negligible, and necessary

There are two further approximations made to this estimate:

1. The estimated Followers count is for the date of posting. This ignores the fact that some people may Like the post but not be a Follower of the account.

2. The number of Likes is the current number of Likes for that post and not those received in a short period following the posting. Therefore if someone were to Like older photos of an account that were posted before they started Following said account then the number would be inflated slightly. However, I believe this is a minority case and given a large enough dataset these occurrences should become insignificant.

It is possible for a person to Like a post for an account they do not Follow, which would cause our Class ratio to exceed 1. Although I believe this is more likely for celebrity accounts or a result of our estimation efforts for the estimated Posting Date Followers. I will only analyse Posts whose estimated Class ratio is ≤ 0.35. I will use Class values of $0.0, 0.025, 0.05, ..., 0.35$ and round all estimated Class ratios to their nearest respective interval. Ideally the more Classes we have the less rounding occurs but as I increase the number of intervals the number of images for each Class decreases requiring an even bigger dataset to train and validate our models on. If we did not increase the dataset size, each class would have less training images associated with it and hence we would expect training and validation loss to increase.

4.3 Separation

The data was first split into a 90% Training set and 10% Testing set. The Training set was then split into a 75% Training set and 25% Validation set. This led to a 67.5% Training set, 22.5% Validation set and a 10% Testing set. The end result is: 34,263 training images, 11,421 validation images, and 5,706 testing images.

These sets were then saved to separate .csv files to avoid data leakage when conducting further analysis.

Data splitting was done using the stratified method. If our dataset is not evenly distributed for each Class then we may introduce bias during the training if our dataset split is chosen randomly. Random sampling may result in over representation of some Classes and when it comes time to train, the model will have insufficient training data for some Classes and hence will do poorly identifying those Classes. In the extreme case the model may determine that to minimise it's error it is better off just predicting the same Class for every input.

Stratified sampling tries to retain this balance by taking proportional random samples from each Class in the data so that the ”training”, ”testing” and ”validation” sets have the same distribution of Classes as the original dataset did.

In future I would like to trial Synthetic Minority Oversampling Technique (SMOTE) which is a type of data augmentation procedure that creates synthetic replicas of the under represented Classes to attempt to reduce the Class imbalance (Chawla 2002).

I chose to avoid using any data augmentation as I wanted to keep this as vanilla as I could to see if real-world data was prevalent enough to sufficiently train a model, instead of injecting augmented data - which would have likely improved accuracy, but would have had a different data balance than the real-world.

4.4 Cross Validation

Cross-validation splits the dataset into a training set and a validation set and applies a model. Then a new split is taken and the same procedure is applied whereby the first validation set is now consumed into the training set and a different part of the training set becomes the new validation set (Refaeilzadeh, Tang **and** Liu 2009). This is repeated k times and the results are averaged. It is this k that is referred to in k-fold cross validation. Cross-validation reduces variance in the predictions as we are training on more data than if we were to only split the data into one training and one validation set. However, this does introduce a slight bias if the data is not completely independent, which ours is not.

I tested cross-validation but I decided against k-fold cross validation in favour of leave-one-out cross validation as it offered a practical compromise between data acquisition speed and accuracy. In an ideal world we would curate as much data as possible . However, under the physical limits of systems I have access to and the time constraints of the project this was not feasible to compete against the 14 million images that ImageNet contains, for example

(Deng **andothers** 2009). Given this inability to collect millions of images it would be convention to use k-fold cross validation to allow us to both train and validate on all of our data which would be the most efficient as no data is wasted. The main problem with this approach is that whilst more accurate on a canonical image classification task it makes no considerations for the person behind the Post, and any characteristics that accompany that profile's Posts. Often a profile on Instagram has a theme, beyond the actual content. Some profiles are dominated by close up selfies, some by far away, group selfies, and some are centered around highly saturated, vibrant photos. Therefore, when given the choice of using fewer profiles but all of the content (under k-fold cross validation) versus using more profiles but only a portion of the content (under leave-one-out cross validation) I opted for leave-one-out as it meant that I did not create a model that was training on only a couple of profile themes. My model, using more profiles should generalise better than one narrowly trained on fewer profiles.

I think this raises an important point about Machine Learning in general that is worth emphasising. We should not blindly follow tutorials (most of which enthuse that k-fold cross validation is best) without giving consideration for the source of our data and any characteristics it may have. The characteristics in our case come from the fact that our input images come as sets/collections. Machine Learning is as much an art form that requires human experience and knowledge as it is a science that can be used in a cookie-cutter style for every problem without regard for underlying detail.

5 Exploratory Data Analysis

Before conducting any thorough analysis it is good practice to get an overview of the data using visualisations. This is often an easy way to spot if a large cluster of data points are missing a value for any parameters, and to see if our Classes are relatively well balanced. Our model will stand a much better chance of predicting that a Post belongs to Class X if we show it sufficient examples of what a Post that is in Class X looks like.

Initially I disregarded any Posts with a Ratio above 1, but the Empirical Cumulative Distribution Function in Figure 2 shows that 95% of the data belonged to Classes of 0.3 or below. Therefore, I decided to disregard any Posts with a Class above 0.35. Having Classes that are infrequent, causes

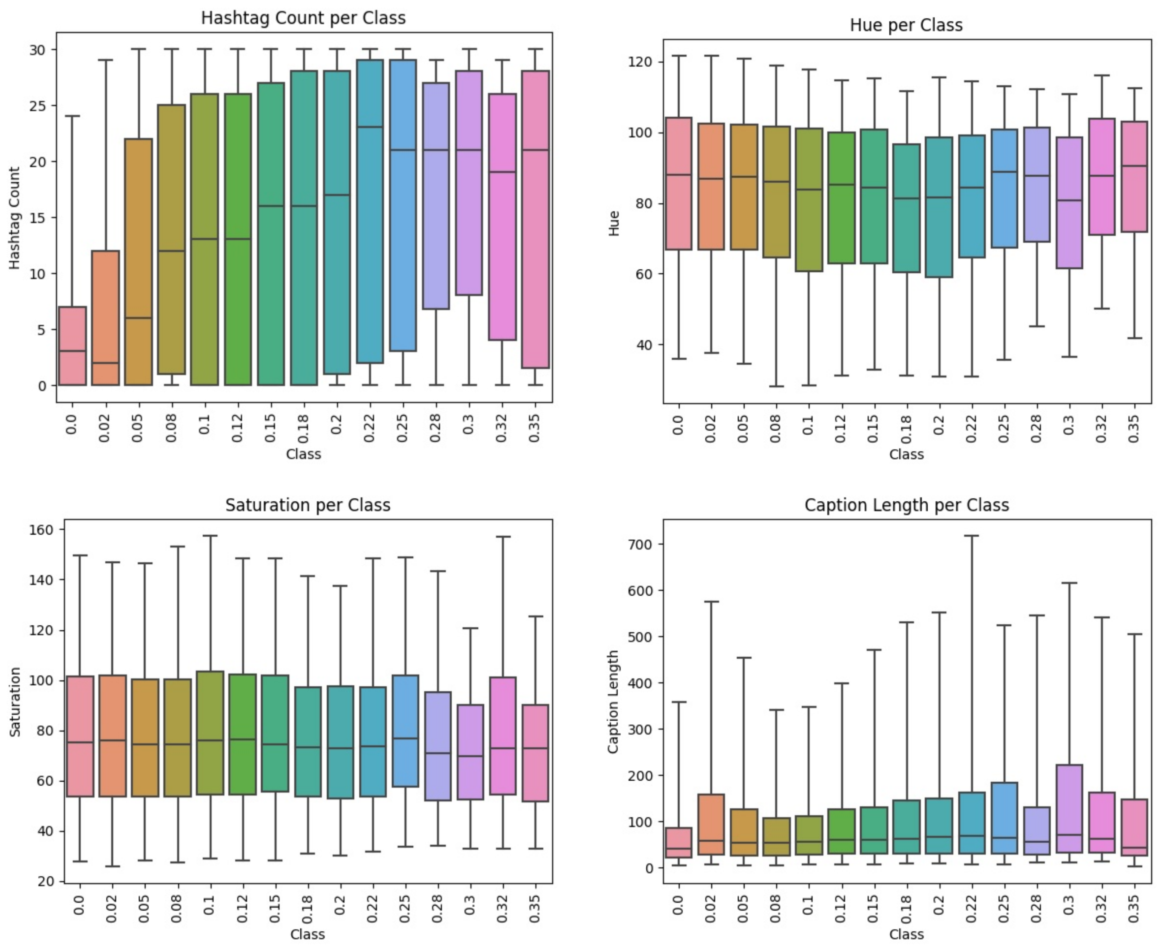
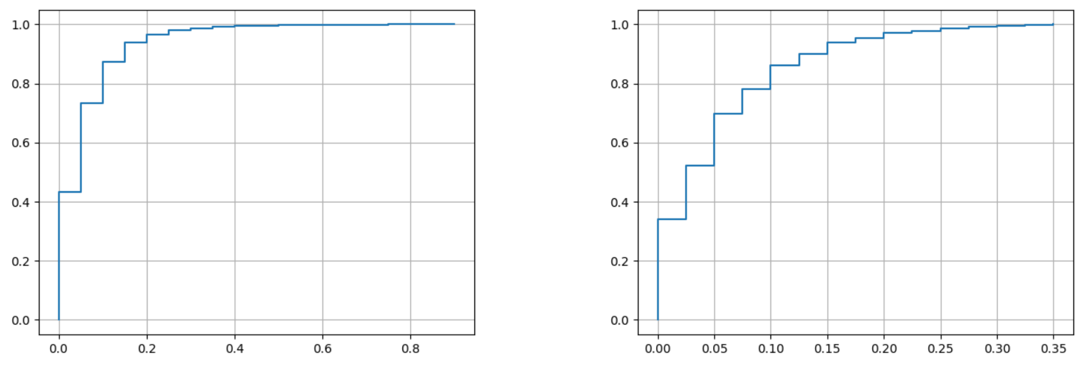


Figure 1: Box-plots showing the distribution of features for each Class in the whole dataset. Because I have used a stratified sampling method for the train-validation-test splitting it suffices to show only the above plots as the distribution should be almost identical in the splits, as it is in the full data set. I removed outliers from the plots to make interpretation clearer. Quantiles were set at 5% and 95%.



(a) Interval: 0.05 (b) Interval: 0.025

Figure 2: Empirical Distribution showing the cumulative proportion of each class. We can see the Class imbalance by the height of each vertical jump. Balanced Classes would have similar vertical steps. This is often a problem but given that so much of our data is in the intervals below 0.15 I am more concerned that it can classify those popular Classes correctly than the infrequent higher Classes. This is another reason I did not include SMOTE as a priority.

most of our data to be in the first two intervals and hence, we do not get enough granularity regarding which specific smaller intervals the model struggled to predict. As well as only analysing Posts with Classes below 0.35 I also reduced the Class width to 0.025 rather than the original 0.05. This should spread out the predictions over any confusion matrices.

Figure 2 also emphasises the benefit of reducing the interval width. In plot 2(b) we can see that the distribution of the data is more evenly distributed over all of the Classes than in plot 2(a).

6 Preprocessing

6.1 Captions

Each Post has the option to attach to it a caption describing the image. These can vary from 0 to 2,200 characters and are truncated after 125 characters. I analysed if the caption content, and length contributed to a change in the Class. Comments may compel an audience to engage with a post and so not only the length of the comment is important but the tone and language used too. I used Natural Language Processing to analyse the semantic nature of the caption content in Section 7.1

I ignored[4] any Posts that contained Non-English captions as the model would have required more data to retain an unbiased distribution of English to Non-English captions. This would have become a bigger problem as we consider all the languages than Non-English includes so given that English is a majority language and data was easy to find there was little reason to incur this vastly increased complexity.

6.2 Caption Cleaning

1. Split Hashtags from caption, split Mentions from caption, and remove newline characters.
2. Remove stop words: Stop words are common words that are not particularly valuable when trying to classify the tone of a sentence. Examples of these are; "be", "at", "the", "us".
3. Remove Unicode characters. I decided to remove most Unicode characters as it added complexity that I believe the classifiers would have struggled with. These characters can be punctuation marks but are often Emojis, such as faces, clouds, love hearts etc. This was a difficult decision as there is some emotion involved in the choice of Unicode character to use. However, through Exploratory Data Analysis it became apparent that there was an overuse of these characters and they were often repeated on many statuses by the same person so contributed no new information to any particular post. We will see later when discussing Term Frequency in the Naive Bayes Classifier why these repeated Unicode characters would have been ignored even if I had included them because they were so prevalent.

6.3 Caption Preprocessing

1. Remove Punctuation: The CountVectoriser does have inbuilt punctuation removal but I chose to remove it myself first before sending the captions to the model so I could be sure that each model was getting

[4]I chose profiles whose captions were in English, but I did not check every Post by that profile so a few Non-English captions did make it through.

the exact same input. This is in contrast to allowing each model to remove the punctuation itself which may vary in quality depending on the classifier used.

2. Lemmatize: Lemmatisation is the process of reducing all words down to their base (known as the lemma). An example is reducing "jumping", "jumped", and "jumper" all to "jump". This is in contrast to the other popular method of Stemming where the inflection of the word is ignored and is just reduced down to its "stem", such problems that occur with stemming is the change from "saw" to "s", whereas Lemmatisation would reduce this down to "see" or "saw" depending on the context. Lemmatising allows us to gain the majority of the information from the word which then gives us more occurrences of the base word to analyse, hopefully increasing the accuracy of our model more than the reduced accuracy caused by removing the inflection.
3. Tokenise: Our classifiers can not process a sentence (string) as an input. They require the sentence to be broken into it's individual words - this is what Tokenising does.

6.4 Feature Engineering

The Post also contains meta-data about the image which includes: *Date, Time, Hashtags (An identifying topic keyword), and Mentions (A direct link to another user's profile).* In addition to these I derived some more information using Feature Engineering. I created the features *Hue, Saturation, Days Since Last Posting, Hashtag Count, Mentions Count, Caption Length.* Below I will explain the reason I feel these might be important.

- Hue & Saturation: Hue refers to the actual colour that we seen, red, green, blue, and everything in between. Saturation is the measure of the amount of grey that each colour is made up of. Low saturation includes more grey and hence the image becomes darker. Higher saturation aligns more with the Hue we have chosen for that pixel. The effect of changing Saturation is that High saturated images appear more vibrant and "pop" more than lower saturated images. I wanted to test if these more vibrant images captured the readers attention more and increased engagement.
- Days Since Last Posting: The incentive to Like a Post may be changed

by the amount that a person posts. We are not shown everything by everyone we follow. The algorithms of the Social Networks often show us more content from people that we engage with, or people who routinely post. Services such as YouTube favour creators who post on a routine schedule and I wondered if Instagram was similar. Therefore, my incentive to Like a post may be changed by the number of times I have seen a person's posts, and indeed my ability to be able to choose to engage with the Post is somewhat decided by Instagram in whether they decide to show me the Post in the first place. If I follower a person who rarely posts then I might Like their newest Post in the hope to encourage them to become more regular. Days Since Last Posting gives an integer number of days between the then current Post and the date of the previous Post.

- Hashtag Count & Mention Count: Hashtags and Mentions will create the illusion of community in your posts and may encourage people to engage, further it is exactly how we've found the data in the first place - by using the Hashtag aggregate pages, which gives you access to a greater audience who then may engage with your Posts. However, on a personal level, if I am mentioned in a Post alongside 30 other people, it would not make me feel very special or unique, more like one of a crowd. This would reduce my incentive to Like the post. Therefore, I included these features to see if there is a correlation between Class and Hashtag/Mention Count. I believe there will be a saturation point.

- Caption Length. Whilst Instagram gives a person the option of including a caption up to 2,200 characters, it is unlikely that anyone will read a caption this long, and if they did I believe that increase in probability of a person Liking the Post diminishes with caption length. In our fast paced world I wanted to test if the more popular Posts were combined with short, easier to read captions, or did people require more detail about the image in the form of a longer caption before making the decision of whether to Like a Post.

7 Models

7.1 Text Classifiers

I chose classifier hyper parameters by using Random Search with candidate settings chosen from common choices in the literature (Bergstra **and** Bengio 2012). These can be seen in Table 15 in the Appendix. The tuner then tests n[5] random combinations of the candidate settings and retains the combination that gave the highest training accuracy.

7.1.1 Naive Bayes

Naive Bayes classifier is an application of Bayes theorem which uses probability to calculate the Class labels for each of our inputs - either Captions, Mentions, or Hashtags - in our case.

The classifier is known as *Naive Bayes* as we are making the strong assumption that each word is independent of each other. This is a strong claim to make as the English language follows a set of rules - often which would break this independence rule. Naive Bayes is surprisingly effective considering the small number of parameters it requires. However, it is often beaten by other methods such as Random Forests that we will discuss later (Caruana **and** Niculescu-Mizil 2006).

I used 4 approaches to applying the Multinomial Naive Bayes; Count Vectoriser, Word Level TF-IDF, Ngram TF-IDF, and Character Level TF-IDF. I applied each of these methods to the Captions, Hashtags, and Mentions separately at first and then using an ensemble stacked method afterwards (Section 7.2).

Count Vectoriser: Takes a simple count of every word in the input and records this in a sparse matrix. It is a sparse matrix because as we increase the number of sentences the number of unique words will increase, but no sentence will include all of those unique words and hence each row contains many empty elements.

Count Vectoriser is simplistic, but fast, and gives equal weight to all words. It tries to classify any test inputs by doing the same count of tokens in the test input and cross referencing that with the training data sparse matrix.

[5] Where n is defined by the user under the method setting *n_iter* in the sklearn package

Whatever it feels is the closest match is then taken as likely to be the same Class as the training input and the test input is assigned with the same Class.

Word Level TF-IDF: Term Frequency-Inverse Document Frequency (TF-IDF) is a measure of how important each word is to a particular document given how frequent it occurs in the document compared to how frequently the word occurs in the corpus of words from the whole collection of inputs.

If a word is very common, similar to stop-words, then it doesn't tell us much new information about the string. Inverse document frequency tells us how common a word is in the whole of the corpus. As a word becomes rarer it's IDF tends to 0 and the TF-IDF then becomes larger for words that are rare, but appear in a particular document. This gives us a good indication that whatever that word may be, it should hold a lot of information as to why that document received the Class label it did. Similar can be said for Character Level TF-IDF also.

N-gram TF-IDF Some words work in synergy and their meaning/intent changes when paired together. Therefore, using n-grams we can concatenate n words together before applying TF-IDF. I used 2 and 3 length n-grams in my analysis. The same rules apply as did with Word Level TF-IDF but the sparse matrix now contains combinations of 2 and 3 consecutive words.

7.1.2 Text classifier results

I used the F1 score as a measure of accuracy rather than the Receive Operating Characteristic (ROC) as our problem is a multi-class problem and the ROC was developed for binary classification problems. Landgrebe **and** Duin (2007) conducted research into using pairwise calculations to implement ROC in multi-class problems but found that whilst it did perform well the computational expense increased to the power of the number of classes so is infeasible for problems with more than a few classes.

The F1 score does have its drawbacks though. It does not consider the relative cost of precision versus recall. D. Hand **and** Christen (2018) talks about this problem in the case of predicting if a patient is well versus ill. In that we'd much rather predict a well person as ill than to predict an ill person as well. However, this is less of a problem in our case as incorrect predictions are not fatal, but it still poses a problem if our algorithm is predicting a Post will be popular - which might encourage a large sponsorship deal - for it to then be vastly less popular, and this is wasted advertising money. Whilst less damaging, it still has consequences though so we should be reserved in relying solo on F1 score alone.

Results can be seen in Subsection 11.1. The Count Vectoriser performed marginally better than the others for Caption analysis with a weighted average F1 score of 0.292 versus the Character Level which came second with 0.289. N-gram and Word Level performed a few percentage points worse. From this I infer that repetition of words - which the Word Level classifier would interpret as a progressively less useful word - is not a major factor in deterring a person to Like a Post. This is further corroborated by the higher Count Vector accuracy in that those Posts with Captions who repeat words get a higher ratio than those that follow more traditional English language rules. I conjectured this might be the case as our colloquial speech is moving in the direction of repetition for emphasis.

When it came to analysing Mentions all of the classifiers performed worse than on the Captions. This is likely due to the nature of there being less information in the Mention compared to captions, where most were a few sentences long. Most notable is the poor performance of the N-gram method as shown in Table 9. It makes sense that N-gram did so poorly on Mentions as the order the Mentions are written in is arbitrary and so there are no bi-gram or tri-gram like structures as there would be in Captions.

Interestingly the Hashtag analysis was much more accurate than the Mentions analysis[6]. Even though both methods contained only single words - that can be input in any arbitrary order (and hence bi-gram and tri-grams should not have been useful) the model drew a lot of useful information from the tags. Count Vector and Word Level performed the best with F1 scores of 0.400, and 0.371 respectively, and did outperform Character Level and N-gram as expected, those still had scores around 0.320, which is much better than the Mentions analysis.

We can see the poor performance of Mentions in the Confusion Matrices in Figure 8. The model could not gain any useful information from the Mentions and it decided it's best option to maximise accuracy was to predict the same Class for every image, where the Class it chose to predict was the one that

[6]Whilst any single tag can contain multiple concatenated words, I did not separate these out so the classifiers treated those as one word. Splitting these first may have increased accuracy slightly

was most common in the training dataset.

The confusion matrices for Captions (Figure 6), especially Character Level and Count Vectors do show the ability of the model to predict different Classes. Ideally we would prefer a the diagonal from top left to bottom right to contain the most predictions but the ability to predict different Classes does give us hope. These models we created from the analysis of 5,076 testing images. I'd previously done them for small testing sets and as the testing set size increased the model moved away from predicting the same Class for every image approach to a more diffused appearance. I believe adding more data would continue this trend and at some point it would predict very well. The limitations of this project though - as discussed in Section 9 talk more about why this was not done.

Overall, and to me at least, unexpectedly the Mentions analysis performed the worst, beaten by Captions and then Hashtags. Initially I expected Captions to contain the most useful information but it seems that Hashtags do. On reflection, because of the architecture of Instagram this is not implausible. Lots of users will search for a Hashtag and browse Posts related only to that Hashtag. Therefore, if you include more Hashtags, you have a better chance of reaching more people, and getting more Likes. This can be further seen in Figure 1(a), as Hashtag count increases, the Class ratio increases. There is a plateau around 20 Hashtags though which suggests a saturation point.

7.2 Ensemble Text Classifiers

I created an ensemble model for each approach where I applied the Vectoriser to the Captions, Hashtags, and Mentions separately, then combined them using *hstack* before fitting the Multinomial Naive Bayes model. The hope was that this would prove more accurate as it had access to more data about the input before making a prediction. The single prediction made by the ensemble model was the average of the predictions by the individual models, rounded to the nearest interval boundary/Class.

Type	**Accuracy**
Count	46%
Word Level	45%
N-gram	42%
Character Level	42%
Text stacked: Average	44%
Text stacked & ANN: Average	39%

Table 1: Mentions, Captions, and Tags were pre-processed, combined, and then analysed using the usual classifiers. I then ensembled this with the predictions of the media meta-data Artificial Neural Network, and took the average prediction to give the accuracy in the final row.

The accuracy scores for the ensemble methods all performed better than the same methods when applied individually to the Captions, Mentions, and Tags. We are starting to see the the ensemble method does work, and performs better than any solo classifier. It does perform slightly worse when we include the predictions from the Artificial Neural Network in Subsection 7.3.5, but when we perform different weightings later in Section 7.6 we will see that we can choose to ignore the ANN altogether if it decreases overall performance.

7.3 Meta-data Classifiers

7.3.1 XGBoost

XGBoost was originally designed by Tianqi Chen as a solution for a Machine Learning competition and gained popularity for its unique features (Chen **and** Guestrin 2016).

1. Penalisation of trees
2. Proportional shrinking of tree nodes
3. Newton Boosting
4. Extra randomisation parameter
5. ability to be distributed over multiple computation devices

XGBoost works by allowing multiple, simpler trees to make predictions about the input, penalising the worst trees (using an L1 or L2 regularisation term) and then combining the simpler trees and their predictions into a new leaf of the overall tree. This continues iteratively until convergence at which point we should have one tree where all of the components are the more successful of the previous iterations.

7.3.2 Decision Tree

Initially a decision tree splits the input data into n categories on some feature of the data. Then each of those categories is split further until we get to the end of tree where no more splits are done. We now have a "blueprint" to read off what the expected Class is for a new input depending on it's trajectory down the tree.

We can improve the decision tree by setting the maximum and minimum number of splits that should be made, the minimum samples per leaf, the splitter type, etc.

7.3.3 Random Forest

An extension to decision trees is the random forest model. A random forest is an ensemble of decision trees where we train multiple decision trees on the same inputs, where each decision tree is slightly different, and take the mode prediction for the respective input and assign that as the Class prediction for that input.

It is hoped this mode vote will reduce variance in the predictions and the result will not be swayed drastically by any single decision tree. This is in comparison to training a single decision tree which may - for some unknown reason - predict a wildly inaccurate prediction, but as we have no alternatives we just take that prediction as is and assign it to the image, further we expect a random forest to over fit less than a single decision tree.

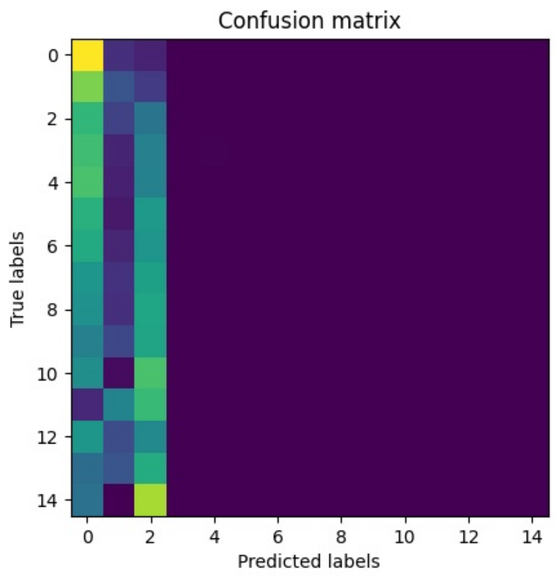
7.3.4 Ada Boost

AdaBoost is a model that tries to learn from its mistakes. It is an ensemble method in itself in that it contains many weak learners, that have weights associated to them. We take a proportional average of each of these weak learners to make the final prediction. As long as each weak learner is slightly better than random guessing then eventually the model will iteratively learn, and converge to a robust and strong model.

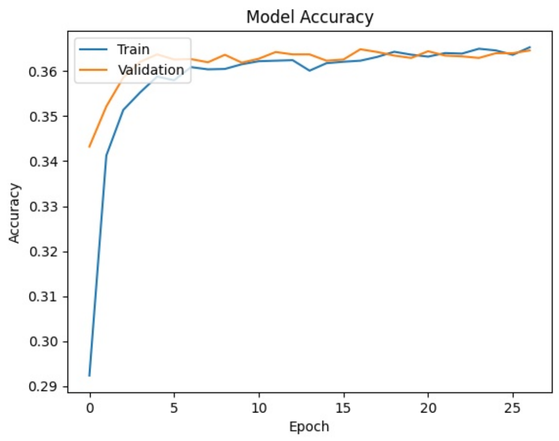
The results for these models can be found in Table 2.

7.3.5 Artificial Neural Network

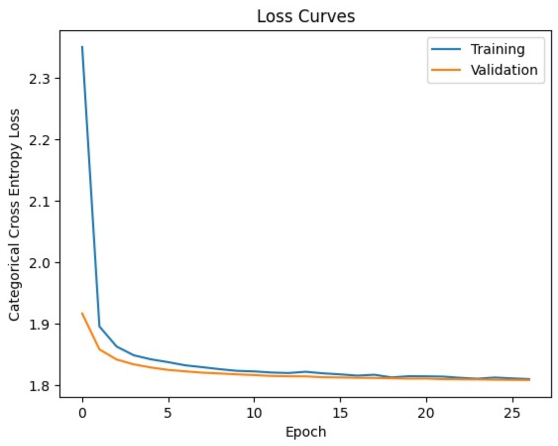
I trained an Artificial Neural Network on the meta-data features to predict the probability of each image belonging to each Class, solely on just the meta-data. The design of this network can be seen in Table 16



(a) Confusion Matrix: 0.025 interval



(b) ANN Accuracy



(c) ANN Loss

Figure 3: Confusion matrix for Artificial Neural Network trained on Post meta-data, Accuracy, and Loss plots.

The ANN trained on the media meta-data had an accuracy 35%. We can see in the confusion matrix in Figure 3(a) that, similar to the text classifiers, the model is able to draw out enough information from the meta-data to make distinct decisions about which Class an image belongs to, and does not just predict the same Class for all images. I believe with more data this trend would increase and the Classes that are greater would have predictions attributed to them. I am not overly concerned that the higher Classes did not have predictions made to them. This is because they make up such a small part of the dataset that, whilst ideally we want to correctly classify them, priority should be given to more accurately predicting the lower Classes.

The model was trained with a patience of 10 which meant that the model would only stop training early (before the 100 epoch limit) if the validation accuracy did not improve for 10 consecutive epochs. As we can see in Figure 3(b) the majority of the accuracy had occurred by epoch 20 and the remaining epochs had considerable diminishing returns. Therefore, we could in future train the model for fewer epochs and gain almost identical accuracy, but more importantly, save time, and avoid over fitting.

7.4 Post Meta-data Ensemble Method

I created a voting classifier that took the predictions of the XGBoost classifier, Random Forest Model, Decision Tree model, and Ada-boost model. A voting classifier takes the predictions of the component classifiers and outputs a single Class prediction for each input.

7.4.1 Soft and Hard Voting

Hard voting takes the majority rule vote of the component classifiers and outputs that as the single prediction.

Soft voting takes the *argmax* of the sums of the probabilities. Each component classifier predicts the probability that the input belongs to each of the Classes, attributing higher weight to the Classes it thinks are most likely correct. We take a weighted average of these for each Class and each classifier and sum them element wise. We then pick whichever Class gives us the maximum weighted sum total for that input.

The main difference between the two voting types is that Hard voting does not distinguish between a classifier giving a very confident prediction and a slightly confident prediction. Consider the case of binary classes with three component classifiers. The three classifiers might predict (0.1, 0.9), (0.2, 0.8), (0.25, 0.75) where it is apparent that all classifiers were very confident that the input belonged to class 2. Under Hard voting we would indeed pick Class 2 as it was the majority prediction. Consider now the three classifiers predicted (0.49, 0.51), (0.45, 0.55), (0.48, 0.52). Again we would still choose class 2 under Hard voting, but it should be obvious that the classifiers were sitting more on the fence with their predictions than in case 1. Soft voting on the other hand would take this into consideration.

The voting classifier can weight the importance of the component classifiers differently. For example, if we believe that classifier A is more reliable than classifier B then we might like to give more importance to the prediction that classifier A makes in deciding the overall single Class prediction for the voting classifier. I have included 4 different ways I calculated these weights below. For each of these I applied *soft* and *hard* voting.

Equal weights: Equal weights gives every classifier's prediction the same importance in the decision process for making the final single prediction.

Accuracy weights: Accuracy weights are simply the accuracy of the individual classifier when fitted on their own without any normalisation.

Advantage weights: Advantage weights gives weight only to those classifiers that did better than the worst classifier. My thinking behind this solution is that the top $c - 1$ classifiers already did a better job than the worst at making predictions so by measuring their *extra* performance boost over the minimum we can take advantage of their prediction accuracy without diluting it with a poor classifier. The weight of each classifier was calculated by subtracting the accuracy of the worst classifier from the accuracy of each classifier. This results in the worst performing classifier receiving a weight of zero, and being ignored. This reduces variance, but does increase bias, however, so I re-ran the voting classifier using normalised weights.

Normalised weights: Normalised weights forced the weights to sum to 1. In addition it still retained the ability to down-weight the poor performing classifiers by only considering their contribution to total accuracy - which is naturally low.

Classifier	Type	Accuracy
XGBoost	Tuned	30.30%
	Untuned	36.25%
Random Forest	Tuned	36.78%
	Untuned	34.10%
	CV	28.64%
Decision Tree	Tuned	36.09%
	Untuned	25.63%
	CV	21.92%
AdaBoost	Tuned	34.36%
	Untuned	34.36%
	CV	33.46%
Ensemble	Soft Voting	37.27%
	Hard Voting	37.04%
Ensemble Advantage Weights	Soft Voting	36.66%
	Hard Voting	36.64%
Ensemble Estimated Weights	Soft Voting	37.21%
	Hard Voting	36.86%
Ensemble Normalised Weights	Soft Voting	37.29%
	Hard Voting	36.72%

Table 2: Tree based models' accuracy scores. The tuned models had their parameters decided by Random Search for their respective hyper parameters.

For all methods, the results with the interval size 0.05 had a higher accuracy than the 0.025 intervals, which makes sense. However, with wider intervals comes lower precision. The intervals of 0.025 are arguably more useful as users can be more informed about how many Likes they expect a Post to get when we use smaller intervals.

We can see that sometimes the untuned performed better than the tuned model. Therefore, for the ensemble method I used the model (either tuned or untuned) that performed best for each specific tree based model and then used those for the ensemble. This lead to the ensemble being comprised of an untuned XGboost classifier, a tuned random forest, a tuned decision tree, and a tuned/untuned AdaBoost model.

We can see that the results are all about the same, but if I were to choose one, it would be the unweighted ensemble method as it performed consistently well under both Hard and Soft Voting.

I did test to see whether any features were consistently more useful than others, (Tables 9, and 10), but the results were erratic and I did not include them on the who, but included the plots in the appendix for your interest.

7.5 Convolutional Neural Network

I used a pre-trained ResNet 50 model from the Keras library to which I added a *Flatten()* layer and a final *Dense()* layer to reduce the number of Classes down to our intended 15 and then used a final Softmax activation function. The architecture of this network can be found in He (2016) and does not warrant repeating here in its entirety.

The model was trained with a learning rate scheduler that would reduce the learning rate when it started to plateau. I used a batch size of 1024, where each image was resized to 250 pixels by 250 pixels.

The model stopped training after 6 epochs due to the early stopping call back which had a patience hyper parameter value of 5. The final learning rate was 0.1^6. With an accuracy of 31% the CNN did comparatively well to the single text classifiers but performed worse than every other method. This suggests that the actual image holds little information as to why a Post receives a like or not. This is interesting given that Instagram is marketed as an image focused social network. Whereas, according to our results the text data that accompanies the image is more valuable from a prediction viewpoint.

7.5.1 Data Augmentations

I applied a standardising augmentation to the image data so as to reduce all pixel values to the interval $[0, 1]$, but I refrained from committing too much time to further augmentations such as rotations, translations, flips, crops, etc. I did this because the number of combinations for these augmentations scales very quickly and finding the optimal would not have been feasible.

Shorten **and** Khoshgoftaar (2019) eloquently makes this point and it is worth quoting directly:

> With a large list of potential augmentations and a mostly continuous space of magnitudes, it is easy to conceptualize the enormous size of the augmentation search space. Combining augmentations

such as cropping, flipping, color shifts, and random erasing can result in massively inflated dataset sizes. However, this is not guaranteed to be advantageous. In domains with very limited data, this could result in further over fitting.

Solutions to this, and an extension I would make to this paper is to use Adversarial Training. This would use two competing models that test different combinations of potential augmentations where they try to learn from each other as to which augmentations prove the most successful at correctly classifying the input images, and then change their own behaviour.

7.6 Final ensemble model

I iterated over 200 different combinations or normalised weights for the Naive Bayes ensemble method, the CNN, and the meta-data ANN. The prediction for each image was the sum of the proportional weights that each individual component had predicted for each Post.

I then selected the weights that produced the highest accuracy and plotted the confusion matrix as can be seen in Figure 5. The model has a nice distribution over the lower classes and did a very good job of predicting Class 0 (0.025 when we undo the label encoding).

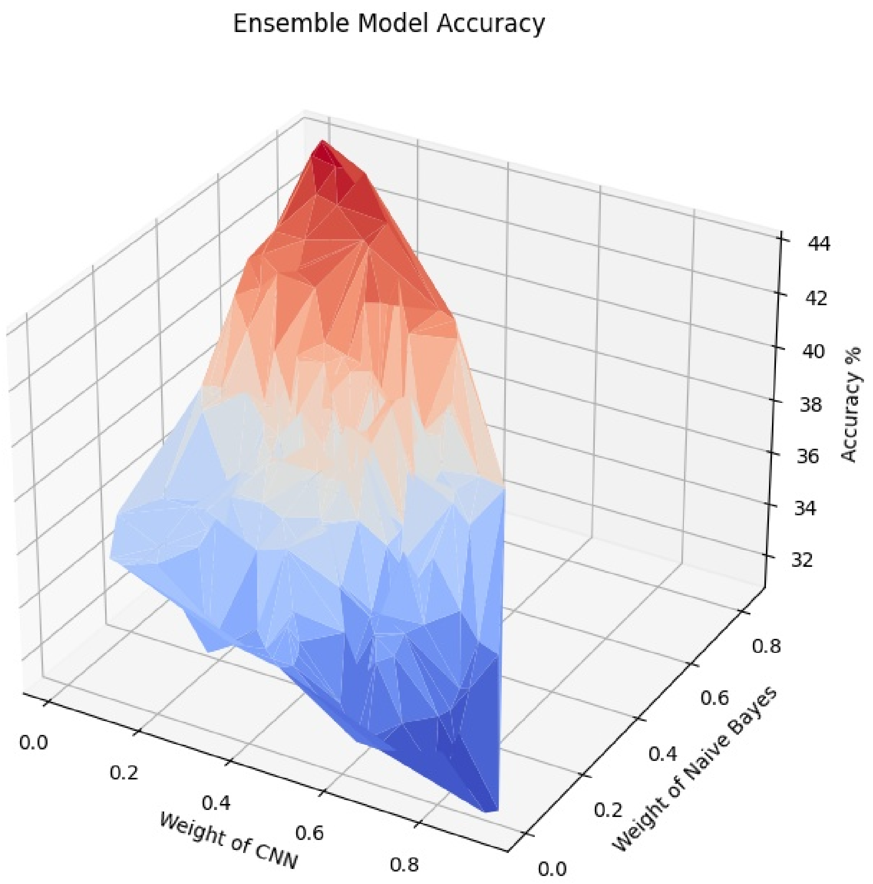


Figure 4: Ensemble model accuracy for varying weights of the CNN, ANN, and text classifiers. The weight given to the ANN is (1-weight given to CNN - weight given to Naive Bayes classifiers).

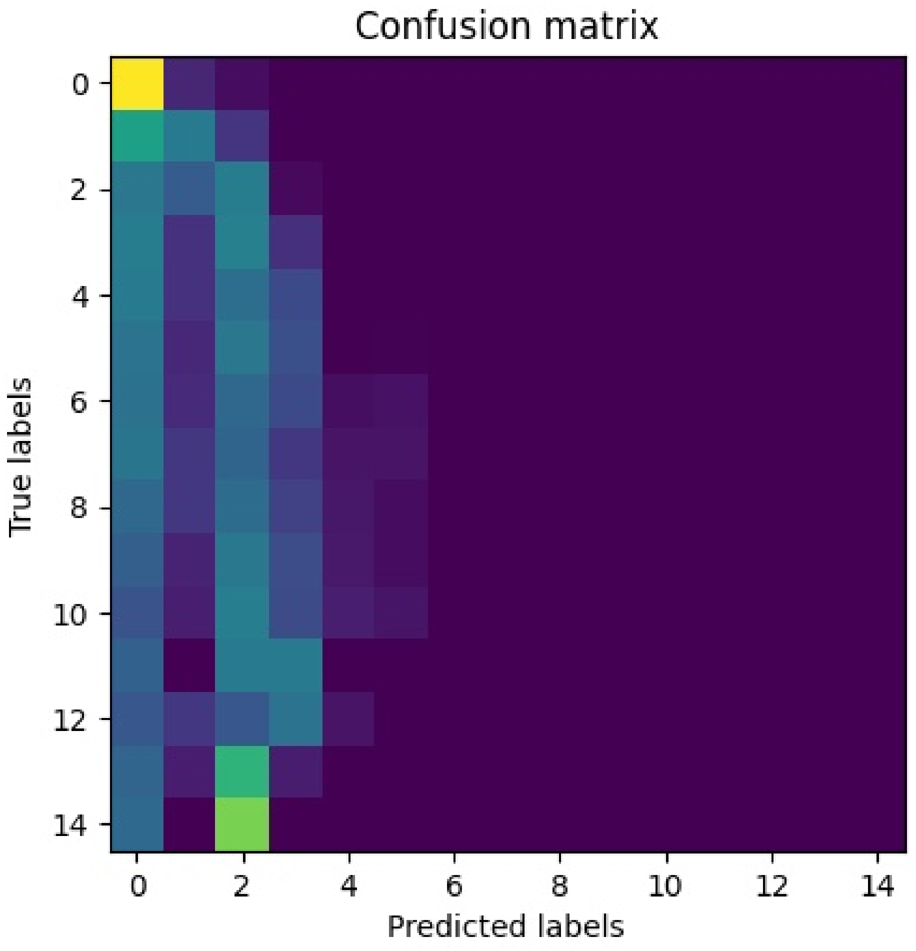


Figure 5: Confusion matrix for the most accurate ensemble model.

The ensemble model worked best when the majority of the weight was given to the sub-ensemble model that contained just the stacked Naive Bayes classifiers, and no weight given to the Convolutional Neural Network.

8 Extensions & Corrections

8.1 Retain punctuation

I would use classifiers that were able to understand the punctuation used in captions in the hope of deriving more information and thus value from the captions. However, I also feel that general colloquial English speak is migrating to a point of excess punctuation - more so on the side of exclamation marks and question marks. We seem to be moving to a style of writing where multiple are used at the end of sentences. I feel there is diminishing returns in terms of emphasis on using more "!" or "?". We seem to also be moving to a situation of overuse, which I think will cause considerably more sentences to end with an "!" that are not technically necessary under conventional literary rules which would then cause the frequency to rise and the marginal information we can gain from analysing "!" falls. I would argue then that analysing punctuation is not a priority given other extensions that are available.

8.2 Data augmentation

As alluded to in Subsection 7.5.1, Adversarial Training looks to be a potential improvement to this paper through the use of multiple models who have different loss functions. More specifically; Adversarial Attacking, as described by Zajac (2019) who showed that the success of adversarial attacks increases as image resolution increases. Given that our images are often high resolution I predict adversarial attacking to be very relevant to the ideas in this paper.

8.3 Improve Class imbalance

In Section 4.3 I touched on SMOTE and using different augmentation/sampling techniques to reduce the Class imbalances. I avoided it for this paper but I would like to address the Class imbalance in future as "The combination of SMOTE and under-sampling performs better than plain under-sampling" (Chawla 2002).

8.4 Cost complexity pruning for decision tree

I improved my decision tree classifier using random search of a candidate hyper parameter space but did not get time to include pruning. Cost complexity pruning aims to reduce the size of our tree and retain only those leaves and branches that substantially improve our accuracy, over and above the complexity that including said branch or leaf incurs (Bradford 1998). The result is a well pruned tree that does not over fit as much to the training data and generalises well to the test data.

8.5 Learning rate scheduler tuning

I did not tune the learning rate scheduler, but would like to in future. Similar to how I tuned the decision tree based classifiers, I would use Random Search over many values of *minimum learning rate*, *patience*, *cooldown*, and *factor*, the definitions of which can be found in the sci-kit learn documentation.

8.6 Improve caption preprocessing

A few Non-English captions fell through my filtering system and given the infrequent nature it is likely that those were wrongly classified, reducing the accuracy of the model. I would like to improve the filtering system, and find a more robust way to deal with Unicode characters.

9 Limitations

Rounding class estimates to 0.05, more granular classes would be better. I rounded the Class ratio for each post to the nearest 0.025. This was gave us a manageable amount of Classes to train on, and then analyse. However, this rounding does reduce accuracy in our predictions. For any Class value, c, all the values in the interval $c - 0.125 \geq c < c + 0.125$ would be rounded to c which gives us a rounding error of 0.125. For a personal profile this may be satisfactory, but for a promotional page 0.125 may be too high. This 0.125 will represent a smaller proportional of the total Class value as the Class value increases but at low levels it could become a problem. If a promotional profile has 100,000 Followers and our model predicts a particular Post would receive a Class Ratio of 0.1 (10,000 Likes) the true value could be as low as 8,750 Likes which is a considerable difference in engagement for a post if the person is making a claim to a company that they are a good profile to advertise their product on. This limits our models to giving a "ball-park" figure for predicted engagement for new Posts.

The data collection was done using a capped, and slow, network. Whilst the model would have been more accurate with a few hundred thousand images, I could only achieve 34,263 training images. I believe as a proof of concept though this data set size suffices to draw the conclusions we have made.

10 Conclusion

Whilst our model did not on the surface perform exceptionally well when compared to other image based analyses done on databases such as CIFAR and ImageNet that often achieve results of Foret (2020) and Pham (2020) which got 99.70% and 90.2% respectively, but when considering that these data sets have been around for 10 years and are canonical it should not be a huge surprise. Further, when we consider that we have 20 Classes for our dataset, which would make the accuracy of random guessing 5%, our model does not seem so unimpressive.

The ensemble model we have seen here is not perfect, and can undoubtedly be improved upon, and I hope it will be, but I think this should be taken as not solely proof of concept, but also a reason to be hopeful. The *now* canonical data sets have only come close to being ”solved” because of the incremental work of many thousands of people. If my model for this Instagram data set can be the foundations of future research into how human curated data sets - in the form of Profiles - can be beneficial to predicting human behaviour, then I think our model, and this project, have been very successful.

I expected the actual image to be more useful for the prediction task than it turned out to be. Indeed it was the text that went alongside the image that was most valuable. Even in this digital age with high definition images, it seems to be that the connections we make, and the way we convey stories, is still done through text. Sharing an image is nice, but the text seemed to be the key way the initial meaning of the Post was conveyed - which was then interpreted by the reader, and it’s this they used to make the decision as to whether to Like the post or not.

11 Appendix

11.1 Text classifier classification reports

Whilst we do have 15 classes, the classifiers never predicted any of the images belong to any Class greater than 6, and hence the rows were all zeros, so have been omitted below for succinctness.

11.1.1 Captions

	Precision	Recall	F1	Support
0	0.357	0.861	0.504	1623
1	0.337	0.217	0.264	1146
2	0.359	0.143	0.205	879
3	0.211	0.010	0.019	404
4	0.372	0.042	0.075	381
5	0.000	0.000	0.000	164
6	0.400	0.010	0.020	193
Accuracy			0.353	5076
Weighted Average			0.264	5076

Table 3: Classification Report for Naive Bayes Word-Level TF-IDF on Captions

	Precision	Recall	F1	Support
0	0.386	0.813	0.523	1623
1	0.330	0.328	0.329	1146
2	0.363	0.163	0.225	879
3	0.195	0.020	0.036	404
4	0.333	0.052	0.091	381
5	0.167	0.006	0.012	164
6	0.364	0.021	0.039	193
Accuracy			0.369	5076
Weighted Average			0.292	5076

Table 4: Classification Report for Naive Bayes Count Vectors on Captions

	Precision	Recall	F1	Support
0	0.344	0.925	0.502	1623
1	0.400	0.152	0.220	1146
2	0.395	0.109	0.171	879
3	0.118	0.005	0.010	404
4	0.158	0.008	0.015	381
5	0.000	0.000	0.000	164
6	0.000	0.000	0.000	193
Accuracy			0.350	5076
Weighted Average			0.242	5076

Table 5: Classification Report for Naive Bayes Ngram on Captions

	Precision	Recall	F1	Support
0	0.378	0.834	0.520	1623
1	0.381	0.221	0.280	1146
2	0.315	0.209	0.251	879
3	0.226	0.017	0.032	404
4	0.226	0.063	0.099	381
5	0.000	0.000	0.000	164
6	0.312	0.104	0.156	193
Accuracy			0.363	5076
Weighted Average			0.289	5076

Table 6: Classification Report for Naive Bayes Character Level on Captions

11.1.2 Mentions

	Precision	Recall	F1	Support
0	0.336	0.980	0.501	1623
1	0.536	0.079	0.137	1146
2	0.415	0.061	0.107	879
3	0.250	0.010	0.019	404
4	0.312	0.026	0.048	381
5	0.000	0.000	0.000	164
6	0.000	0.000	0.000	193
Accuracy			0.344	5076
Weighted Average			0.215	5076

Table 7: Classification Report for Naive Bayes Count Vectors on Mentions

	Precision	Recall	F1	Support
0	0.335	0.983	0.500	1623
1	0.526	0.079	0.138	1146
2	0.397	0.055	0.096	879
3	0.600	0.007	0.015	404
4	0.438	0.018	0.035	381
5	0.000	0.000	0.000	164
6	0.000	0.000	0.000	193
Accuracy			0.344	5076
Weighted Average			0.211	5076

Table 8: Classification Report for Naive Bayes Word Level TF-IDF on Mentions

	Precision	Recall	F1	Support
0	0.323	0.998	0.488	1623
1	0.472	0.015	0.029	1146
2	0.529	0.010	0.020	879
3	0.000	0.000	0.000	404
4	0.333	0.003	0.005	381
5	0.000	0.000	0.000	164
6	0.000	0.000	0.000	193
Accuracy			0.324	5076
Weighted Average			0.166	5076

Table 9: Classification Report for Naive Bayes Ngram on Mentions.

	Precision	Recall	F1	Support
0	0.330	0.937	0.488	1623
1	0.366	0.114	0.174	1146
2	0.400	0.034	0.063	879
3	0.100	0.002	0.005	404
4	0.348	0.021	0.040	381
5	0.000	0.000	0.000	164
6	0.500	0.005	0.010	193
Accuracy			0.333	5076
Weighted Average			0.210	5076

Table 10: Classification Report for Naive Bayes Character Level on Mentions

11.1.3 Hashtags

	Precision	Recall	F1	Support
0	0.488	0.864	0.624	1623
1	0.488	0.320	0.387	1146
2	0.388	0.341	0.363	879
3	0.305	0.131	0.183	404
4	0.325	0.286	0.304	381
5	0.263	0.030	0.055	164
6	0.313	0.161	0.212	193
Accuracy			0.448	5076
Weighted Average			0.400	5076

Table 11: Classification Report for Naive Bayes Count Vectors on Hashtags

	Precision	Recall	F1	Support
0	0.488	0.864	0.624	1623
1	0.488	0.320	0.387	1146
2	0.388	0.341	0.363	879
3	0.305	0.131	0.183	404
4	0.325	0.286	0.304	381
5	0.263	0.030	0.055	164
6	0.313	0.161	0.212	193
Accuracy			0.448	5076
Weighted Average			0.400	5076

Table 12: Classification Report for Naive Bayes Count Vectors on Hashtags

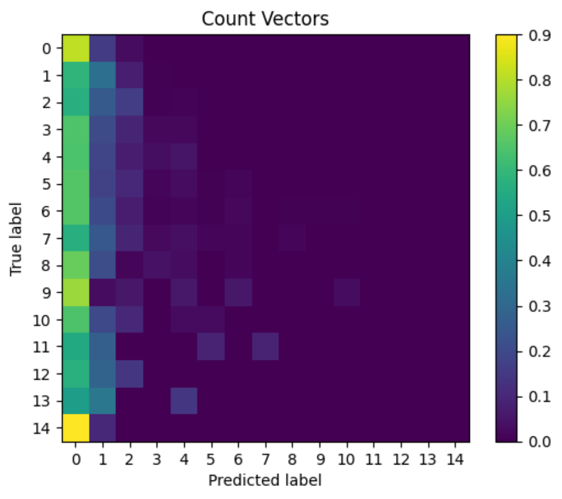
	Precision	Recall	F1	Support
0	0.403	0.965	0.569	1623
1	0.416	0.147	0.218	1146
2	0.510	0.234	0.321	879
3	0.300	0.067	0.109	404
4	0.315	0.168	0.219	381
5	0.333	0.037	0.066	164
6	0.318	0.073	0.118	193
7	0.200	0.024	0.042	85
Accuracy			0.405	5076
Weighted Average			0.319	5076

Table 13: Classification Report for Naive Bayes N-gram on Hashtags

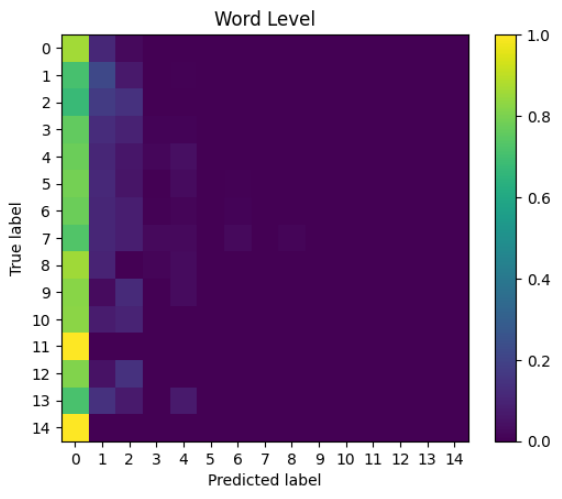
	Precision	Recall	F1	Support
0	0.424	0.869	0.570	1623
1	0.442	0.251	0.321	1146
2	0.318	0.274	0.295	879
3	0.271	0.064	0.104	404
4	0.299	0.152	0.202	381
5	0.000	0.000	0.000	164
6	0.341	0.078	0.127	193
7	0.000	0.000	0.000	85
Accuracy			0.402	5076
Weighted Average			0.334	5076

Table 14: Classification Report for Naive Bayes Character Level on Hashtags

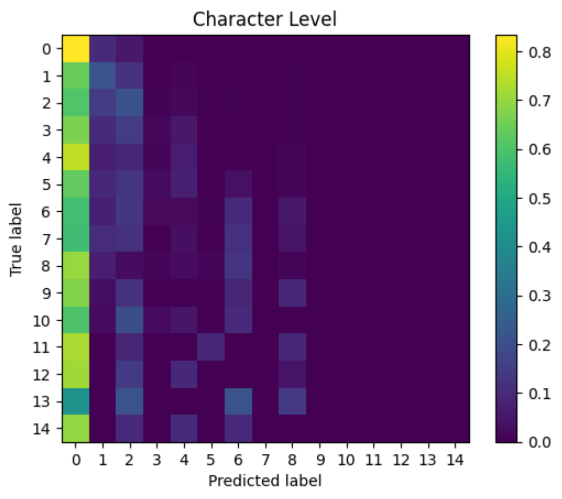
11.2 Confusion Matrices



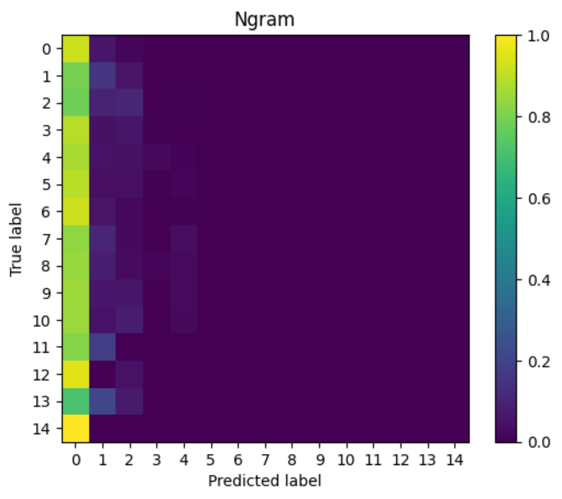
(a) Count Vectors



(b) Word Level

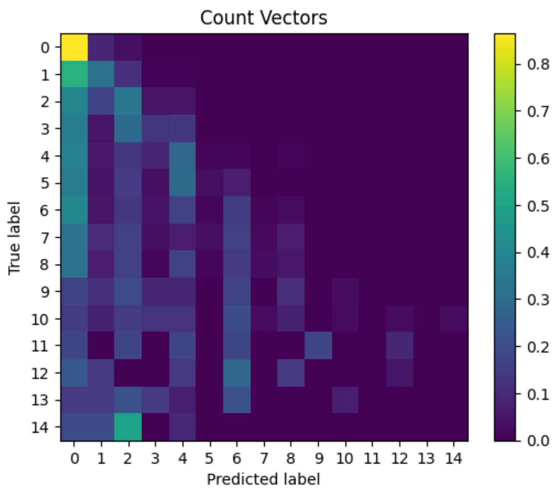


(c) Character Level

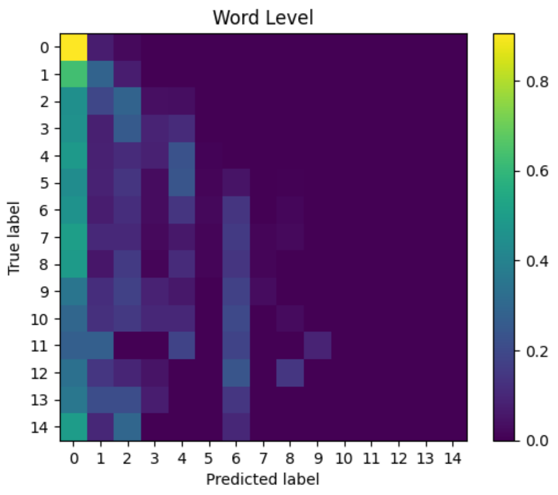


(d) Ngram

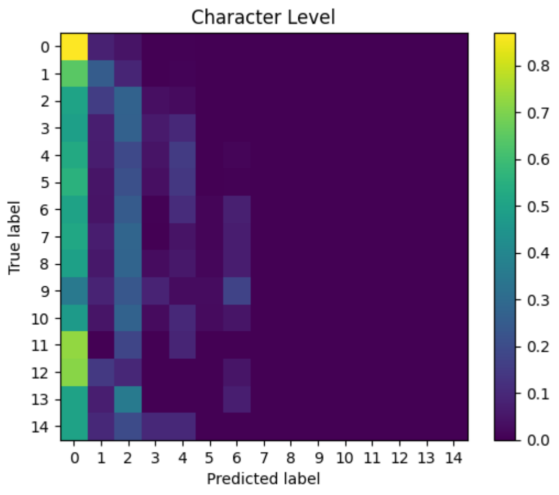
Figure 6: Confusion Matrices for single text classifiers applied to captions



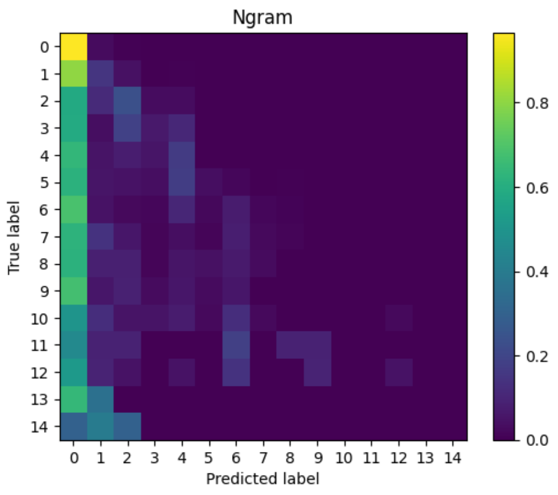
(a) Count Vectors



(b) Word Level

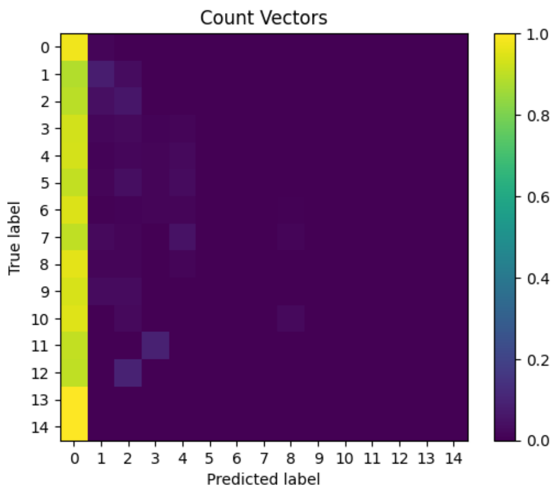


(c) Character Level

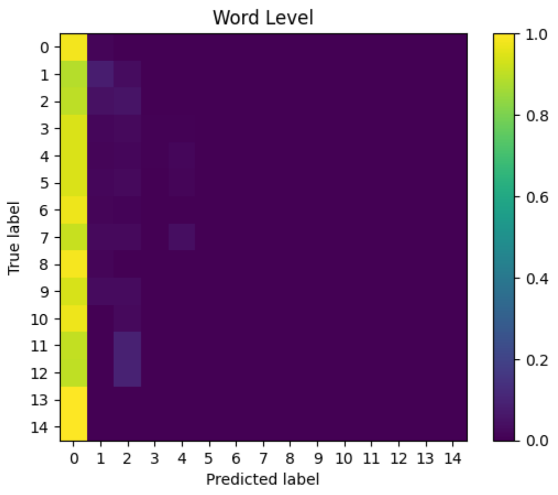


(d) Ngram

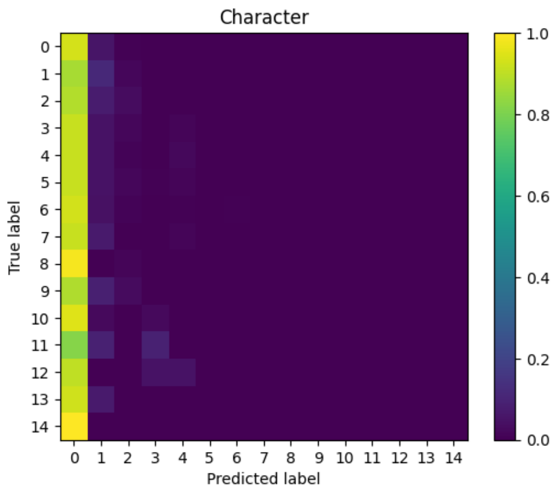
Figure 7: Confusion Matrices for single text classifiers applied to tags



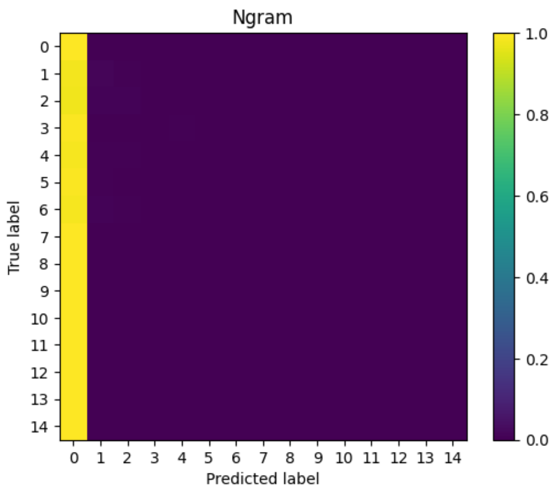
(a) Count Vectors



(b) Word Level



(c) Character Level



(d) Ngram

Figure 8: Confusion Matrices for single text classifiers applied to mentions

11.3 Text classifier hyper parameters

	XGboost	**Decision Tree**	**Random Forest**	**AdaBoost**
Number of estimators	linspace(200, 2000, 10)	linspace(200, 2000, 100)	linspace(200, 2000, 100)	linspace(200, 2000, 100)
Max Depth	linspace(10, 110, 11)	linspace(10, 110, 11)	linspace(10, 110, 11)	
Gamma	linspace(0, 1, 20)			
Learning Rate	linspace(0, 1, 20)			linspace(0, 1, 10)
Regularisation Alpha	linspace(0, 1, 20)			
Regularisation Lambda	linspace(0, 1, 20)			
Min Samples Split		[2, 5, 10]	[2, 5, 10]	
Max Features		[”auto”, ”sqrt”, ”log2”]	[”auto”, ”sqrt”]	
Max Leaf Nodes		[2, 5, 10]		
Min Samples Leaf		[2, 5, 10]		
Splitter		[”best”, ”random”]		
Boostrap			[”True”, ”False”]	
Algorithm				[”SAMME”, ”SAMME.R”]

Table 15

11.4 Artificial Neural Network

Layer	**Output Shape**	**Parameters**
Dense	(None, 32)	224
Dropout	(None, 32)	0
Dense	(None, 32)	1056
Dropout	(None, 32)	0
Dense	(None, 64)	2112
Dense	(None, 20)	1300

Table 16

11.5 Feature Importances

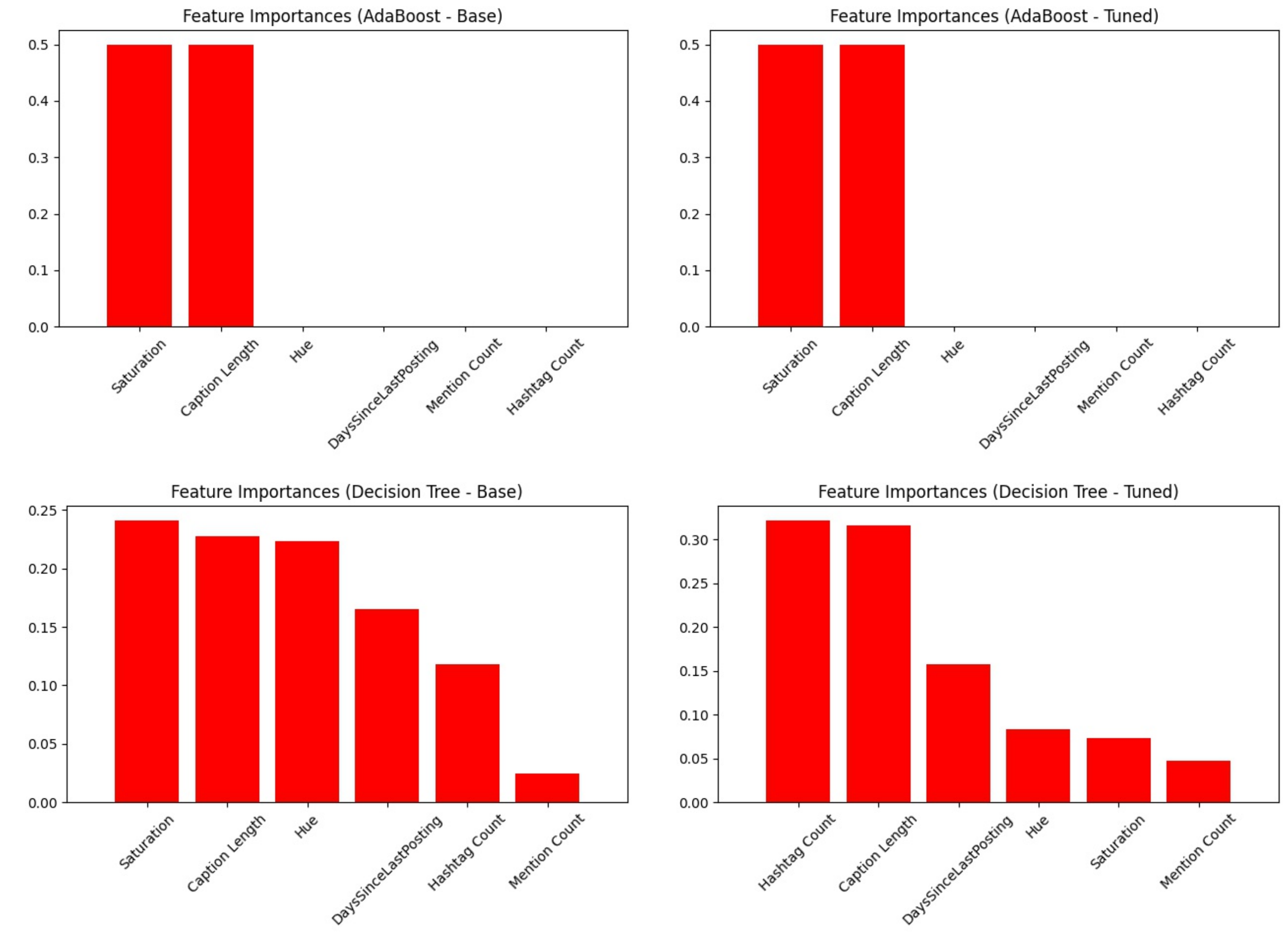


Figure 9: Bar plots showing the comparison of feature importance for 4 AdaBoost and Decision Trees between the base - un-tuned classifier and the tuned classifier after Random Search.

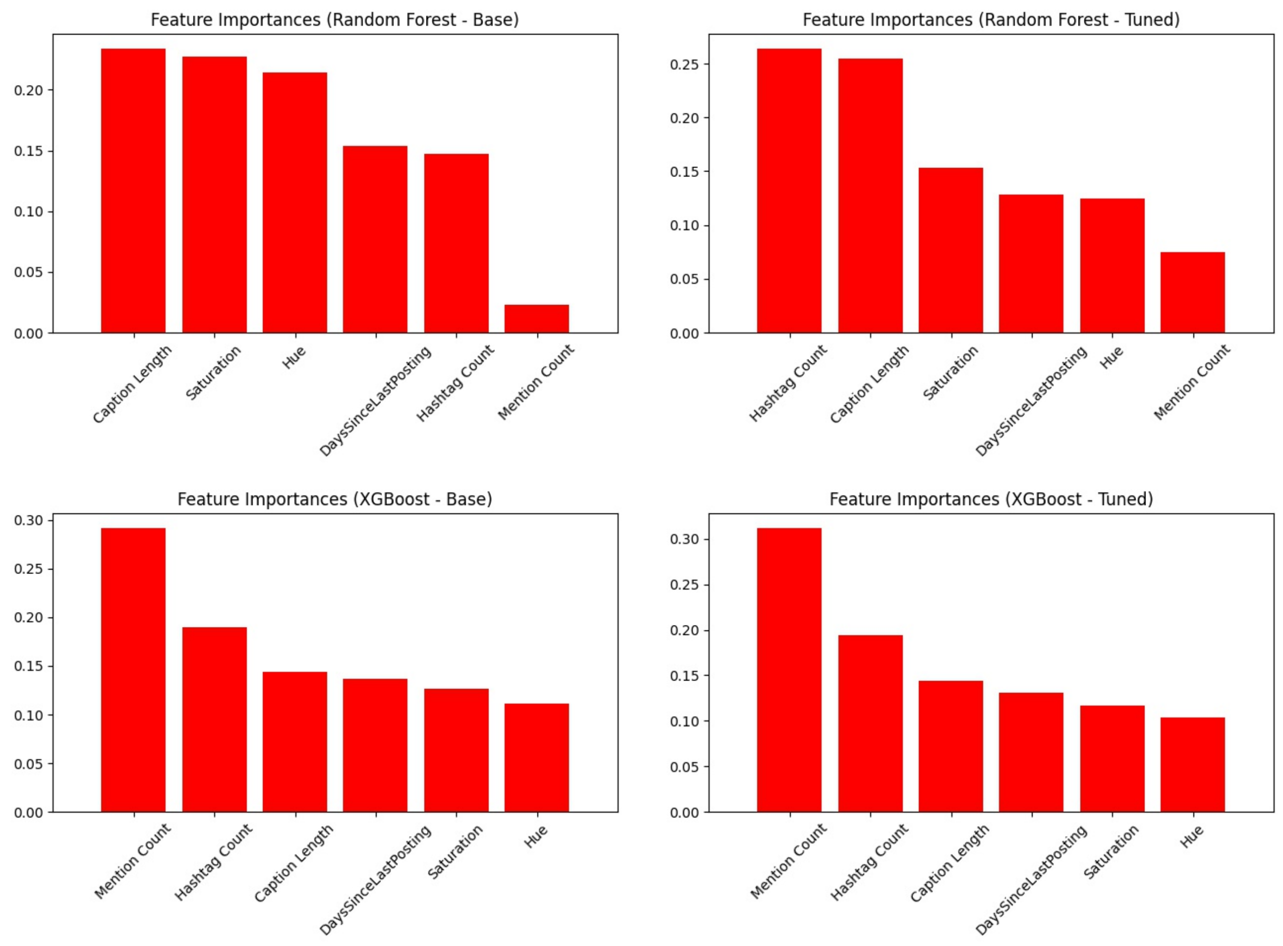


Figure 10: Bar plots showing the comparison of feature importance for Random Forest and XGBoost between the base - un-tuned classifier and the tuned classifier after Random Search.

12 Future Technical Changes

Parsing the data from the individual JSON files to the csv took around 1,100 seconds for 25,000 images which was very slow. In future I would do this using Pyspark and Spark dataframes, which could then be converted CSVs or Pandas dataframes for further processing in the rest of the project.

References

Bradford, Jeffrey P (1998). “Pruning decision trees with misclassification costs”. **in**: *European Conference on Machine Learning*. Springer, **pages** 131–136.

Hand, David J **and** Keming Yu (2001). “Idiot’s Bayes—not so stupid after all?” **in**: *International statistical review* 69.3, **pages** 385–398.

Chawla, Nitesh V (2002). “SMOTE: synthetic minority over-sampling technique”. **in**: *Journal of artificial intelligence research* 16, **pages** 321–357.

Caruana, Rich **and** Alexandru Niculescu-Mizil (2006). “An empirical comparison of supervised learning algorithms”. **in**: *Proceedings of the 23rd international conference on Machine learning*, **pages** 161–168.

Landgrebe, Thomas CW **and** Robert PW Duin (2007). “Approximating the multiclass ROC by pairwise analysis”. **in**: *Pattern recognition letters* 28.13, **pages** 1747–1758.

Deng, Jia **andothers** (2009). “Imagenet: A large-scale hierarchical image database”. **in**: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, **pages** 248–255.

Refaeilzadeh, Payam, Lei Tang **and** Huan Liu (2009). “Cross-validation.” **in**: *Encyclopedia of database systems* 5, **pages** 532–538.

Bergstra, James **and** Yoshua Bengio (2012). “Random search for hyper-parameter optimization.” **in**: *Journal of machine learning research* 13.2.

Chen, Tianqi **and** Carlos Guestrin (2016). “Xgboost: A scalable tree boosting system”. **in**: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, **pages** 785–794.

He, Kaiming (2016). “Deep residual learning for image recognition”. **in**: *Proceedings of the IEEE conference on computer vision and pattern recognition*, **pages** 770–778.

Hand, David **and** Peter Christen (2018). “A note on using the F-measure for evaluating record linkage algorithms”. **in**: *Statistics and Computing* 28.3, **pages** 539–547.

Shorten, Connor **and** Taghi M Khoshgoftaar (2019). “A survey on image data augmentation for deep learning”. **in**: *Journal of Big Data* 6.1, **pages** 1–48.

Zajac, Micha (2019). “Adversarial framing for image and video classification”. **in**: *Proceedings of the AAAI Conference on Artificial Intelligence*. **volume** 33. 01, **pages** 10077–10078.

Foret, Pierre (2020). “Sharpness-Aware Minimization for Efficiently Improving Generalization”. **in**: *arXiv preprint arXiv:2010.01412*.

Pham, Hieu (2020). “Meta pseudo labels”. **in**: *arXiv preprint arXiv:2003.10580*.