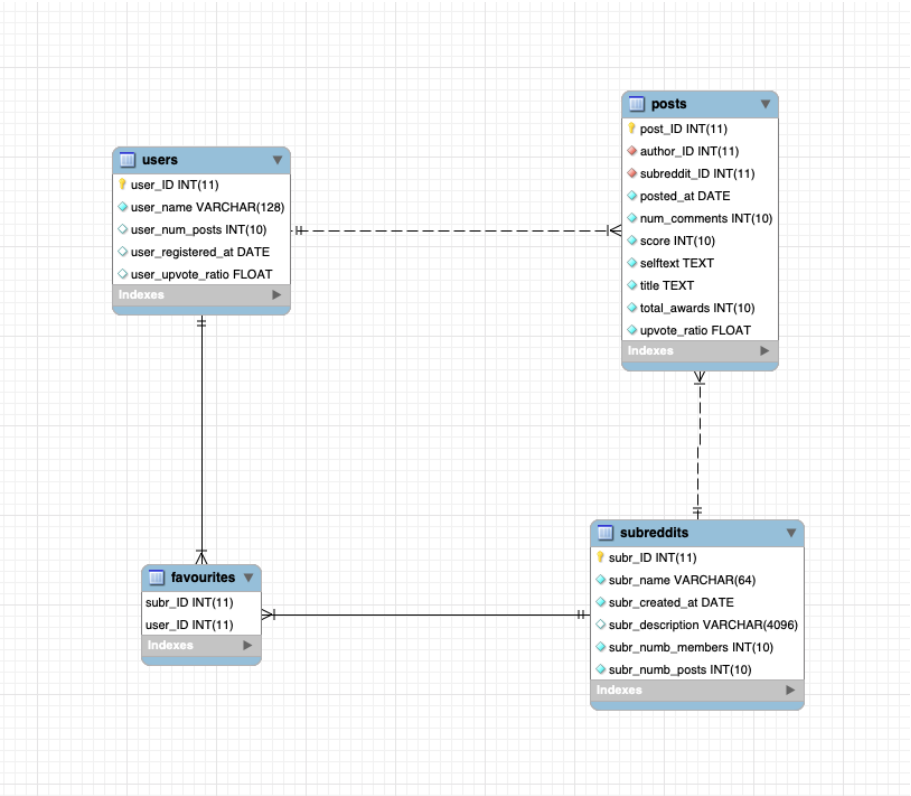
CMT655 Assessment 3

Business rules



Entity Relationship diagram obtained by reverse engineering the c2075016_covid_reddit database

From the dataset it was inferred that each post is authored by exactly one user and is posted in exactly one subreddit. Each user in the dataset has authored one or more posts, and each subreddit in the dataset contains one or more posts. There is a many-to-many relationship between users and subreddits in terms of favourites.

The database was transformed into NF1 by creating a separate “favourites” linking table to capture the many-to-many relationship between users and favourited subreddits, removing the array of users that had favourited each subreddit. Each row was assigned a unique primary key “post_ID” to distinguish between rows, ensuring that there are no repeated rows in the dataset. There were no repeated columns by default. NF2 was automatically achieved because there were no composite primary keys being used at this stage. Finally, NF3 was achieved by resolving the functional dependencies “author” -> “user_num_posts”, “user_registered_at”, “user_upvote_ratio” and “subreddit” -> “subr_created_at”, “subr_description”, “subr_faved_by”, “subr_numb_members”, “subr_numb_posts”, by separating these into “user” and “subreddits” tables, with primary keys “user_ID” and “subreddit_ID” respectively. The subreddit and author names in the “favourites” table were replaced with a composite primary key of foreign key constraints referencing “user.user_ID” and “subreddits.subreddit_ID” and IDs were used in the “posts” table to refer to the subreddit and author of each post, reflecting the one-to-many relationships between users and posts and subreddits and posts. It was initially thought that the post score, upvote_ratio and total_awards_received could be functionally dependent, but evidently score=upvotes-

downvotes, while upvote_ratio=upvotes/(upvotes+downvotes), while total_awards_received is unrelated to upvotes and downvotes (Reddit [no date]). Simple counterexamples can disprove both “upvote_ratio”->”score” and the converse (Table 1).

Upvotes	Downvotes	Score	Upvote Ratio
1	0	1	1
2	1	1	0.6667
2	0	2	1

Table 1: Counterexamples showing that upvote_ratio and score are not functionally dependent, since row 1 and row 2 have the same score but different upvote ratios, and rows 1 and 3 have the same upvote ratio but different scores.

As a result, there were no unnormalized relations in the database.

I used the domain constraint of storing dates in the DATE format rather than as a string to prevent the insertion of invalid dates. Another domain constraint was using UNSIGNED INTs for discrete values such as a user’s number of posts, preventing the insertion of meaningless negative values. All primary keys were given the property AUTO_INCREMENT and entity integrity constraint NOT NULL, to prevent NULL values from being inserted into these columns and automatically generate a primary key if a row without a primary key value is inserted. This improves the usability of the database, since the arbitrary surrogate keys may not be known by the user. The use of surrogate primary keys (instead of usernames and subreddit names) improves performance and allow usernames and subreddit names to be updated without violating referential integrity constraints (Dragos-Paul 2011). Since surrogate keys are arbitrary, there is no need to update them, so the MySQL default of ON UPDATE RESTRICT was maintained to prevent changes to primary keys referenced by foreign key constraints.

I used foreign key constraints in the posts and favourites tables to ensure referential integrity, by both preventing the insertion of records referencing non-existent primary key values and controlling the deletion of records from parent tables that are referenced in a child table. I chose to use ON DELETE CASCADE for all foreign key constraints because this models the real-life possibility that posts, accounts and subreddits may be deleted from Reddit and adheres best to GDPR. However, the deletion of such records would lose valuable data, and invalidate the analysis of trends in the data over time, and therefore the database design could be improved by adding a “date_deleted” column for accounts and posts and “date_favourited” and “date_unfavourited” columns in the favourites table to capture historic data instead. However, this would introduce the need for another primary key in the favourites table since a user may favourite and unfavourite the same subreddit multiple times.

I dealt with the missing data (NULLs) in the post selftext using data imputation (Gonulal 2019), inserting empty strings for the selftext of posts where this was NULL in the original dataset. This allowed the rows with empty selftext to be treated the same as other rows, without the need to deal with NULLs in the database and avoided the loss of data associated with the common missing data practices of listwise or pairwise deletion (Gonulal 2019). I set all fields to NOT NULL to prevent the insertion of incomplete data. I suspected that some

of the users that favourite subreddits may not have an associated post, resulting in missing data, but this transpired not to be the case.

I used the TEXT data type for post title and selftext because row size limits of 65535B were exceeded by some posts and rendering it impossible to balance varchar field sizes to accommodate both title and selftext without data loss (MySQL [no date]a). However, each TEXT field contributes only 9-12B towards row length, since data is stored on disk, but this negatively impacts performance (MySQL [no date]b). The use of TEXT rather than MEDIUMTEXT or LONGTEXT saved memory (MySQL [no date]c).

The presence of non-ASCII characters in the posts was dealt with by setting DEFAULT CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci on every table, which allowed the insertion of emojis and other symbols rather than losing them through cleaning. The presence of HTML entities such as “&” (Web Design Group [no date]) indicated that the data had been corrupted to some extent, but I did not replace these due to time constraints.

The use of transactions also prevented bad data from being inserted into the database, because if an error was thrown during table population, the whole set of queries could be rolled back and reapplied after correction (Factor 2012).

Finally, the Python-MySQL interaction would have been more maintainable if the table creation code was saved in a stored procedure and called by PyMySQL in the Python script, thereby increasing cohesion.

Training and Test VIEWS

The training and test VIEWS were designed to contain the same proportion of COVID:non-COVID posts as the overall dataset, because the dataset was relatively balanced (5824 COVID-related posts and 14116 non-COVID-related posts) (Sun et al. 2009). The training and testing data was chosen by ordering the COVID and non-COVID entries by date and assigning the top proportion for training. This ensured the model was trained on the widest possible variety of both COVID and non-COVID subreddits, assuming the distribution of posts in each subreddit over time is approximately the same. This also ensures the reproducibility of the split, as would be achieved by using a random seed (Bansal 2020). I also created a separate SQL function assign_labels (Figure 1), which determined from post_IDs whether posts were from COVID or non-COVID subreddits based on the whether the subreddit title or description contained “corona”, “covid” or “lockdown”. This function was created before the VIEWS at route ‘/’ (Figure 2). This function’s values were conditioned upon during VIEW creation to ensure each VIEW contained the desired proportion of COVID and non-COVID posts (Figure 3) and to populate the label column. This introduced modularity into the SQL, allowing labelling code reuse and improving maintainability by increasing cohesion.

```
def create_labelling_function():

    print("Creating labelling function...")

    with connection.cursor() as cur:
        q="""DROP FUNCTION IF EXISTS assign_label;"""
        cur.execute(q)

        q= """CREATE FUNCTION assign_label(ID_of_post INT)
            RETURNS INT
            DETERMINISTIC
            BEGIN
                DECLARE label INT;
                SET @subreddit_name = (SELECT subr_name FROM posts INNER JOIN subreddits ON
                subreddits.subr_ID=posts.subreddit_ID WHERE post_ID=ID_of_post);
                SET @subreddit_description = (SELECT subr_description FROM posts INNER JOIN subreddits ON
                subreddits.subr_ID=posts.subreddit_ID WHERE post_ID=ID_of_post);

                IF @subreddit_name LIKE '%covid%' OR @subreddit_name LIKE '%corona%' OR @subreddit_name LIKE '%lockdown%' OR
                @subreddit_description LIKE '%covid%' OR @subreddit_description LIKE '%corona%' OR @subreddit_description LIKE
                '%lockdown%'
                THEN SET label=1;
                ELSE
                    SET label=0;
                END IF;
                RETURN label;
            END;"""
        cur.execute(q)

    print("Successfully created labelling function")
```

Figure 1: Python function create_labelling_function which created the SQL function assign_labels, which determined whether a post was COVID-related given its post_ID.

```
@app.route('/')
def form():
    reset_views()
    # Added this in to create the labelling function.
    create_labelling_function()

    create_training_view()
    create_test_view()
    check_views()
    return render_template('index.html')
```

Figure 2: Route '/' from main.py showing the calling of create_labelling_function before the views were created in create_training_view and create_test_view.

```
    #
    q="""CREATE VIEW training_set AS
            (SELECT CONCAT_WS('\n', title, selftext) AS full_text, assign_label(post_ID) AS label FROM posts
                WHERE assign_label(post_ID)=1
                ORDER BY posted_at ASC, full_text ASC
                LIMIT {0})
            UNION ALL
            (SELECT CONCAT_WS('\n', title, selftext) AS full_text, assign_label(post_ID) AS label FROM posts
                WHERE assign_label(post_ID)=0
                ORDER BY posted_at ASC, full_text ASC
                LIMIT {1});
    """
```

Figure 3: Query implemented by the Python function create_training_view that shows an example of the use of assign_label in the creation of the training and test VIEWs.

One problem was ensuring that rows in the training and test views were disjoint, which was achieved by ordering posts by posted_at followed by full_text (since some posts shared dates) and selecting the top proportion of posts (Figure 3), with the orders reversed between training and test VIEWs. I also chose to use placeholders for the limits, so the

train/test proportion could be set in the Python code and the desired number of posts calculated, facilitating train/test proportion variation during experimentation.

Machine Learning Algorithm

I considered using both CountVectorizer and TfidfVectorizer in various configurations for data vectorization, as shown below (Table 2).

Vectorizer	Time to vectorize data (seconds)
CountVectorizer with 100 words	0.8054
CountVectorizer with 1000 words	0.7507
CountVectorizer with 100 strings of length up to 5 words	10.1075
CountVectorizer with 1000 strings of length up to 5 words	10.0490
CountVectorizer with 100 strings of length up to 5 characters	23.5719
TfidfVectorizer 100 words	0.7272
TfidfVectorizer 1000 words	0.7867
TfidfVectorizer with 100 strings of length up to 5 words	10.2764
TfidfVectorizer with 100 strings of length up to 5 characters	16.2255

Table 2: Time taken by CountVectorizer and TfidfVectorizer to vectorize the dataset in various configurations, with the grey-filled rows deemed too slow.

To keep the time to perform the classification experiment and load the page reasonable, vectorizer configurations taking over 5 seconds to vectorize the dataset were eliminated. After experimenting with various classifiers using CountVectorizer, with 100 and 1000 words as features (Table 3), I chose the four most promising for further investigation. I started with a 2:1 train-test split because evidence suggests that this is optimal for high-dimensional classifiers with an accuracy of over 85% (Dobbin and Simon 2011).

Vectorizer	CountVectorizer (100 words)		CountVectorizer (1000 words)	
Classifier	Time	F1 Score	Time	F1 Score
DecisionTreeClassifier(max_depth=20)	0.059829712	0.787900026	0.206284046	0.815655368
KNeighborsClassifier(10)	2.311132193	0.791228146	3.27589798	0.761590019
MLPClassifier(alpha=1, max_iter=1000)	5.041053057	0.809041204	27.12834883	0.848049853
GaussianNB()	0.073451996	0.574722135	0.523475885	0.564742647
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	1.112083912	0.817053092	2.52767396	0.826446996
AdaBoostClassifier()	1.398600101	0.804731412	15.9410212	0.820474528
SVC(kernel="linear", C=0.025)	9.114061117	0.763398852	83.11252785	0.792030147
SVC(gamma=2, C=1)	19.3241961	0.716327331	(took too long)	(took too long)

QuadraticDiscriminantAnalysis()	0.297460079	0.509887875	5.260378122	0.52677497

Table 3: Training time and F1 score for various classifiers using data vectorized by CountVectorizer.

Classifiers taking significantly longer than 5 seconds to train and those with F1-scores below 0.8 for both configurations were eliminated. Further investigations were performed, varying vectorizer, number of features and test-train proportion (Tables 4-7). Experiments using DecisionTreeClassifier and RandomForestClassifier were run 10 times and an average F1 score taken, since they are non-deterministic.

Vectorizer	CountVectorizer(100 words)		CountVectorizer (1000 words)	
Classifier	Time (seconds)	F1 Score	Time (seconds)	F1 Score
DecisionTreeClassifier(max_depth=20)	0.0598	0.7888*	0.2063	0.8270*
MLPClassifier(alpha=1, max_iter=1000)	5.0411	0.8090	27.1283	0.8480
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	1.1121	0.7969*	2.5277	0.8298*
AdaBoostClassifier()	1.3986	0.8047	15.9410	0.8205

Table 4: Training time and F1-score for the most promising classifiers using data vectorized by CountVectorizer and a train/test split of 2:1 (=average of 10 runs).*

Vectorizer	TfidfVectorizer(100 words)		TfidfVectorizer (1000 words)	
Classifier	Time (seconds)	F1 Score	Time (seconds)	F1 Score
DecisionTreeClassifier(max_depth=20)	0.0869	0.8045*	0.2197	0.8262*
MLPClassifier(alpha=1, max_iter=1000)	2.9763	0.8094	16.2185	0.8345
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	1.7765	0.8031*	2.6430	0.8316*
AdaBoostClassifier()	1.5324	0.8243	15.7904	0.8366

Table 5: Training time and F1-score for the most promising classifiers using data vectorized by TfidfVectorizer and a train/test split of 2:1 (=average of 10 runs).*

Vectorizer	CountVectorizer(100 words)		CountVectorizer (1000 words)	
Classifier	Time (seconds)	F1 Score	Time (seconds)	F1 Score
DecisionTreeClassifier(max_depth=20)	0.0655	0.8232*	0.1365	0.8371*
MLPClassifier(alpha=1, max_iter=1000)	6.9094	0.8262	20.3743	0.8507
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	1.3075	0.8232*	2.9631	0.8374*
AdaBoostClassifier()	2.3507	0.8032	16.2164	0.8011

Table 6: Training time and F1-score for the most promising classifiers using data vectorized by CountVectorizer and a train/test split of 4:1 (=average of 10 runs).*

Vectorizer	TfidfVectorizer(100 words)		TfidfVectorizer (1000 words)	
Classifier	Time (seconds)	F1 Score	Time (seconds)	F1 Score
DecisionTreeClassifier(max_depth=20)	0.0863	0.8274*	0.2486	0.8398*
MLPClassifier(alpha=1, max_iter=1000)	3.3779	0.8114	14.5359	0.8289
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	2.5455	0.8225*	2.8811	0.8428*
AdaBoostClassifier()	2.3904	0.8213	19.0238	0.8264

Table 7: Training time and F1-score for the most promising classifiers using data vectorized by TfidfVectorizer and a train/test split of 4:1 (=average of 10 runs).*

MLPClassifier and AdaBoostClassifier were too slow with max_features=1000 and their F1-scores with max_features=100 were outperformed by DecisionTree and RandomForest with max_features=1000, so MLP and AdaBoost were eliminated. Furthermore, TfidfClassifier appears to be better than CountVectorizer for the same number of features, classifier and train/test proportion (which follows since TF-IDF accounts for overall word frequency in the dataset relative to the post (Scikit learn [no date]) so CountVectorizer was eliminated. The 4:1 test-train proportion also outperformed 2:1. On average, RandomForestClassifier outperformed DecisionTreeClassifier, with training time increasing by approximately a factor of n_estimators (which follows since RandomForestClassifier chooses the average classification from n_estimators random decision trees to limit overfitting (Deng 2018)). However, the training time using n_estimators=10 was deemed acceptable so RandomForestClassifier with n_estimators=10 was chosen.

Max_depth	10		20		30		50	
Max_features	t	f1	t	f1				
10	0.3219	0.5510	0.4774	0.6704	0.7021	0.7318	0.8863	0.7992
50	0.8830	0.7572	1.5528	0.8283	2.1956	0.8494	3.3082	0.8611
100	1.6791	0.7924	3.0261	0.8440	4.0253	0.8567	6.1172	0.8618
200	2.8683	0.8096	5.5142	0.8509				

Table 8: Average training time and F1-score over 10 runs using RandomForestClassifier with various max_depth and max_features values and n_estimators=10.

Finally, parameters max_depth and max_features were varied in order to find optimal values (Table 8), with max_depth=50, max_features=50 being selected. These were set as the defaults in the form allowing users to choose the parameters.

Evaluation Metrics

I used vectorization and training time to eliminate vectorizers and classifiers taking over 5 seconds to ensure the application routes return pages in a reasonable time. I evaluated the models using F1-score since the application's purpose is unknown and the dataset is slightly imbalanced (Huilgol 2019). However, were the application aiming to all block posts about COVID, for example, then the model should be optimised on recall instead. Whilst using Matthews Correlation Coefficient (MCC) may have been a better approach (Chicco and

Jurman 2020), preliminary investigations showed that in this use case where specificity is high (around 97% for RandomForestClassifier), using MCC would have negligibly impacted the outcome (Table 9).

Vectorizer	CountVectorizer (100 words)	
Classifier	Time	MCC
DecisionTreeClassifier(max_depth=20)	0.0695	0.6604
KNeighborsClassifier(10)	3.9060	0.6193
MLPClassifier(alpha=1, max_iter=1000)	10.1951	0.6730
GaussianNB()	0.0710	0.3306
RandomForestClassifier(max_depth=20, n_estimators=10, max_features=100)	1.3031	0.6661
AdaBoostClassifier()	1.6399	0.6431
SVC(kernel="linear", C=0.025)	12.8781	0.5892
SVC(gamma=2, C=1)	32.3955	0.5631
QuadraticDiscriminantAnalysis()	0.2850	0.2414

Table 9: Preliminary results comparing various classifiers using the Matthews Correlation Coefficient

Interaction between Python, MongoDB and the browser

The model and vectorizer binaries and associated data were stored in a MongoDB database, allowing these binaries to be retrieved and reused at the "/submitted" route using pickle.loads. I reused the model and vectorizer binaries generated previously to ensure reproducibility (since RandomForestClassifier is not deterministic), and save time on data vectorization and classifier retraining. I stored the results in MongoDB in the following form:

```
{
    "model": {
        "binary":< model_binary>,
        "classifier": <classifier_name>,
        "parameters": {
            <parameter_key_value_pairs>
        }
    },
    "vectorizer": {
        "binary": vectorizer_binary,
        "type": <vectorizer_name>,
        "parameters": {
            <parameter_key_value_pairs>
        }
    },
    "train_test_proportion":<proportion>,
    "evaluation_metrics": {
        "time":<time>,
        "accuracy": <accuracy>,
        "precision": <precision>,
        "recall": <recall>,
```

```
                "f1score": <f1score>
                }
}
```

I included implementation details for each model such as the vectorizer and classifier information with input parameters and the train-test proportion to indicate how the model and vectorized data was generated, allowing a similar models to be generated in the future and informed comparisons of models with modified parameters. This is particularly relevant given the feature allowing users to choose their own parameters for RandomForestClassifier, as discussed below. I included multiple evaluation metrics, not just F1-score to improve the application's flexibility, so if its purpose changes, requiring models to be optimised on a different metric, data from previous experiments can be used.

On the "Retrieve results" and "Experiment done" pages, I displayed all of the above information for each model except the unreadable model and vectorizer binaries, in order to make the data shown to the user as informative as possible. However, the readability of the data on the "Retrieve results" page would be enhanced if inserted into a table using a template. Therefore, at "/report" (Figure 4), I only queried for each model's metadata, thereby avoiding using unnecessary bandwidth during transmission (MongoDB documentation [no date]). Similarly, at "/submitted" (Figure 5), I only retrieved the model and vectorizer binaries to be compiled using pickle.loads because none of the metadata was used.

```
projection={"_id":0, "model.classifier":1, "model.parameters":1, "vectorizer.type":1, "vectorizer.parameters":1,
"train_test_proportion":1, "evaluation_metrics":1}
res=list(mdb.results.find({}, projection).sort("evaluation_metrics.f1score", -1).limit(3))
```

Figure 4: MongoDB query at the "/report" route retrieving the metadata only for models with the best three F1 scores.

```
projection={"_id":0, "model.binary":1, "vectorizer.binary":1}
result=list(mdb.results.find({}, projection).sort("evaluation_metrics.f1score", -1).limit(1))[0]
```

Figure 5: MongoDB query at the "/submitted" route retrieving the model binaries only for the model with the highest F1 score.

One challenge in the MongoDB interaction was the size limit of 16MB for MongoDB documents (MongoDB Documentation [no date]), which was overcome by limiting the vectorizer to 1000 features.

<u>Covid-or-not Functionality</u>

I initially implemented the covid-or-not functionality using AdaBoostClassifier, CountVectorizer(max_features=1000, ngram_range=[1,5], analyzer="char") and a 2:1 train/test proportion. However, this took 13 seconds to load "/experiment_done", increasing to 25 seconds with 1000 features (Figure 6). Following this, given the large number of variables involved in the optimisation of the classifier, I obtained the results

Tables 4-7 from experimentation in Jupyter Notebook.

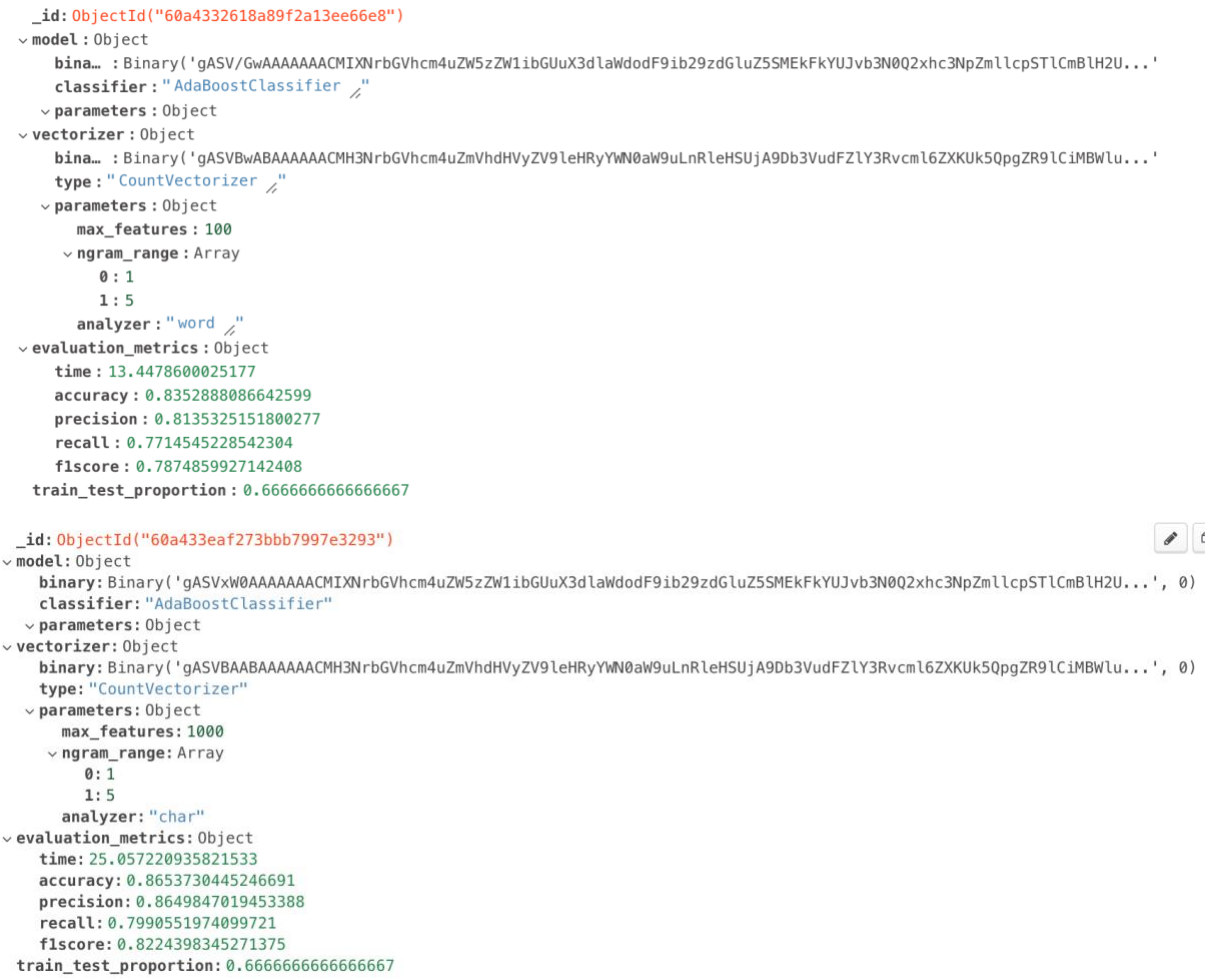


Figure 6: Screenshots from MongoDB Compass showing the results of the initial investigations using AdaBoostClassifier and CountVectorizer.

The additional feature allowing users to choose parameters for RandomForestClassifier facilitates the creation of better models for the covid-or-not functionality. Selecting the model with the highest F1-score for the covid-or-not prediction ensures that the quality of this feature (in terms of F1-score) can only increase as further classification experiments are performed. However, using words rather than character or word sequences as features means the current classifier cannot recognise the more colloquial “corona” as COVID-related and has inconsistencies, like labelling “infection” and “social distancing” COVID, but “infectious” and “socially distanced” non-COVID. Additionally, this single-word approach makes the classifier less able to recognise COVID-related phrases, such as “contact tracing”, but using of TfidfVectorizer means the classifier is not using a bag of words model, so is able, for example, to label “stay at home” COVID, whilst labelling each constituent word non-COVID.

Furthermore, the assumption of equivalence between COVID-related posts and COVID-related subreddits resulted in 538 posts containing the strings “covid”, “corona” or “lockdown” being labelled non-COVID, 376 of which were in the conspiracy subreddit. This could bias the classifier against recognising less scientific COVID-related posts and classifiers

labelling such posts COVID-related would be ranked less highly. Additionally, since a large proportion of Reddit users are from the USA (Tanovska 2021), the classifier is likely to be biased towards recognising American COVID phraseology rather than British.

Data Ethics

Whilst the model in this project was trained to recognise whether a post is about COVID-19 or not, this topic was arbitrary, and the same algorithms could be used to classify text corresponding to any subject or sentiment, albeit using different datasets or labels. This section focusses on the first action given in the UK Data Ethics Framework, to "*define and understand public benefit and user need"* (Central Digital and Data Office 2020). I believe this to be the weakest area of this project because there is wide-ranging discussion in the literature about unintended consequences of using text classification algorithms (Sap et al. 2019) and human rights considerations, such as limiting freedom of speech (Heldt 2019) which I have not addressed. This project is not fulfilling any specific user need, and the public benefit of a covid-or-not predictor has not been clearly defined, giving the project a score of 0 against this action.

Fairness

This project could be used for censorship by a state controlling social media who do not wish to admit to the prevalence of COVID-19 in their country. Wider applications include Facebook's removal of hate speech (Forbes Technology Council 2017) and censorship of undesirable opinions from social media (Heldt 2019). The specific groups that benefit from the removal of hate speech are those who are targeted by online abuse, such as ethnic or religious minorities, both because they are not subjected to the abuse removed, but also because seeing abusive content online might encourage others to participate in abuse themselves. The benefit to the wider public would be ensuring people feel safe from abuse when using social media. However, the automated removal of hate speech can have unintended consequences, which can prove discriminatory against the groups such algorithms were trying to protect. For example, racial bias can occur in hate speech detection due to insensitivity to dialect and the reclamation of certain terms that would otherwise be considered racist (Sap et al. 2019). This could therefore reinforce social and ethical problems and inequalities rather than solving them. Furthermore, the misuse of the algorithm for censorship could lead to discrimination through the "*suppression of diverse or even unpopular speech"* (Forbes Technology Council 2017), which violates human rights and democratic values. This would suppress these views in real life, since opinions are informed by the social media ones consumes (Gündüç 2020). Furthermore, censorship may result in biasing the datasets which are used for downstream AI projects (Yang and Roberts 2021), further consolidating (intentional or unintentional) bias or discrimination, or the deplatforming of certain views.

Accountability

This project had no formal accountability structures, which would have been improved had users of the application been consulted to determine user need (Central Digital and Data Office 2020). Following the Stop Hate For Profit boycott, Facebook pledged to improve their accountability by submitting to a third party audit of their content moderation systems (Facebook Business 2020), demonstrating the extent to which such accountability matters

to many users. Furthermore, Hutchinson et al. (2021) suggest that datasets used to train classifiers should undergo the same development process as software, including requirements analysis involving stakeholders and therefore users, in order to provide accountability, since datasets may exhibit trends that reflect historic human biases, with the models based on them therefore reinforcing rather than combat these trends. For example, without accountability based on user need, biases in the labelling of the Reddit dataset would be reflected in the resulting models.

Transparency

The UK Data Ethics Framework recommends making "*user need and public benefit of the project transparent*". For this project, user need must be identified before it can be communicated to the public. As for Facebook, their Community Standards (Facebook, [no date]) explain how their content moderation delivers positive social outcomes for the public and communicates their understanding of the user need (Central Digital and Data Office 2020), and they publish reports showing the extent to which their moderation software is reflecting their values (Facebook 2021). This transparency is partial however (Turilli and Floridi 2009), as Facebook do not disclose exactly how posts are screened, but given the inherent lack of interpretability of many machine learning models, it is difficult to explain to the average user how a model works, which can impact trust. Although Random Forests are more difficult to visually interpret than Decision Trees (Kowsari et al. 2019), meaningful methods have been devised for their interpretation (Robinson et al. 2017).

Solution (Accountability):

Consult with users to determine user need before the project starts (Central Digital and Data Office 2020) including involving users in the design of the dataset to ensure that it reflects their needs and does not contain biases that compromise this (Hutchinson et al. 2021). The elimination of biases, for example, better recognising American phraseology and the marking of conspiracy posts as non-COVID, would broaden the ability of the classifier to recognise COVID posts from different segments of social media users, thereby reducing any resulting discrimination.

References

Bansal, J. 2020. *How to Use Random Seeds Effectively*. Available at: https://towardsdatascience.com/how-to-use-random-seeds-effectively-54a4cd855a79 [Accessed on: 31 May 2021].

Central Digital and Data Office. 2020. *Data Ethics Framework*. Government Digital Service.

Chicco, D. and Jurman, G. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 6.

Deng, H. 2018. *Why random forests outperform decision trees.* Available at: https://towardsdatascience.com/why-random-forests-outperform-decision-trees-1b0f175a0b5 [Accessed on: 31 May 2021].

Dobbin, K. and Simon, R. 2011. Optimally splitting cases for training and testing high dimensional classifiers. *BMC Medical Genomics* 4(31).

Dragos-Paul, P. 2011. Natural versus Surrogate Keys. Performance and Usability. *Database Systems Journal* 2(2) pp. 55-63.

Facebook Business. 2020, *Sharing Our Actions on Stopping Hate.* Available at: https://www.facebook.com/business/news/sharing-actions-on-stopping-hate [Accessed on: 31 May 2021].

Facebook. [No date]. *Community Standards.* Available at: https://www.facebook.com/communitystandards/introduction [Accessed on: 29 May 2021]

Facebook. 2021. *Community Standards Enforcement Report* Available at: https://transparency.fb.com/data/community-standards-enforcement/?from=https%3A%2F%2Ftransparency.facebook.com%2Fcommunity-standards-enforcement [Accessed on: 31 May 2021].

Factor, P. 2012. *Handling Constraint Violations and Errors in SQL Server.* Available at: https://www.red-gate.com/simple-talk/sql/t-sql-programming/handling-constraint-violations-and-errors-in-sql-server/ [Accessed on: 1 June 2021].

Forbes Technology Council. 2017. *13 Pros And Cons Of Integrating AI For Social Media Scrubbing.* Available at: https://www.forbes.com/sites/forbestechcouncil/2017/08/17/13-pros-and-cons-of-integrating-ai-for-social-media-scrubbing/?sh=40ce3d9d19ce [Accessed on: 29 May 2021]

Ghoneim, S. 2019*. Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?*. Available at: https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124 [Accessed: 28 May 2021].

Gonulal, T. 2019. Missing Data Management Practices in L2 Research: The Good, The Bad and The Ugly. *Erzincan University Education Faculty Journal* 21(1) pp. 56-73.

Gündüç, S. 2020. The Effect of Social Media on Shaping Individuals Opinion Formation. *Complex Networks and Their Applications VIII* pp. 367-386.

Heldt, A.P. 2019. Upload-Filters – Bypassing Classical Concepts of Censorship?. *Journal of Intellectual Property, Information Technology and Electronic Commerce Law* 10(1) pp. 57-65.

Huilgol, P. 2019. *Accuracy vs. F1-Score.* Available at: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2#:~:text=Accuracy%20is%20used%20when%20the,as%20in%20the%20above%20case [Accessed on: 31 May 2021].

Hutchinson, B. et al. 2021. Towards Accountability for Machine Learning Datasets: Practices from Software Engineering and Infrastructure. *Conference on Fairness, Accountability, and Transparency (FAccT '21).* Canada, 3-10 March, 2021. New York: ACM, pp. 560-575.

Kowsari, K. et al. 2019. Text Classification Algorithms: A Survey. *Information (Switzerland)* 10(4), 150.

MongoDB Documentation. [No date]. *Documents*. Available at: https://docs.mongodb.com/manual/core/document/#:~:text=Document%20Size%20Limit,MongoDB%20provides%20the%20GridFS%20API [Accessed on: 31 May 2021].

MySQL. [No date]a. *12.5 Limits on Table Column Count and Row Size.* Available at: https://dev.mysql.com/doc/mysql-reslimits-excerpt/5.7/en/column-count-limit.html [Accessed on: 27 April 2021].

MySQL. [No date]b. *11.3.4 The BLOB and TEXT Types.* Available at: https://dev.mysql.com/doc/refman/8.0/en/blob.html [Accessed on: 31 May 2021].

MySQL. [No date]c. 11.7 *Data Type Storage Requirements.* Available at: https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html [Accessed on: 31 May 2021].

Reddit. [No date]. *Frequently Asked Questions.* Available at: https://www.reddit.com/wiki/faq [Accessed on: 28 May 2021].

Robinson, R. et al. 2017. Comparison of the Predictive Performance and Interpretability of Random Forest and Linear Models on Benchmark Data Sets. *Journal of Chemical Information and Modeling* 57 pp. 1773-1792.

Sap, M. et al. 2019. The Risk of Racial Bias in Hate Speech Detection. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, 28 July – 2 August, 2019 pp. 1668-1678.

Scikit learn. [No date]. *sklearn.feature_extraction.text.TfidfTransformer* Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html [Accessed on: 31 May 2021].

Sun, Y. et al. 2009. Classification of Imbalanced Data: A Review. *International Journal of Pattern Recognition* 23(4) pp.687-719.

Tanovska, H. 2021. *Regional distribution of desktop traffic to Reddit.com as of December 2020, by country.* Available at: https://www.statista.com/statistics/325144/reddit-global-active-user-distribution/ [Accessed on: 31 May 2021].

Turilli, M. and Floridi, L. 2009. The ethics of information transparency. *Ethics and Information Technology* 11 pp. 105-112.

Web Design Group. [No date]. *Special Entities.* Available at: https://www.htmlhelp.com/reference/html40/entities/special.html [Accessed on: 31 May 2021].

Yang, E. and Roberts, M. 2021. Censorship of Online Encyclopedias: Implications for NLP Models. *Conference on Fairness, Accountability, and Transparency (FAccT '21).* Canada, 3-10 March, 2021. New York: ACM, pp. 537-548.