The COVID-19 Face Mask Image Classification

Christine Hou
chou37@wisc.edu

Yue Yin
yin58@wisc.edu

Seanna Zhang
szhang577@wisc.edu

Abstract

The project aims to explore the applicable machine learning model for classifying whether individuals wear the mask appropriately, incorrectly, or did not wear it at all. To process, we download 4559 raw images from Kaggle. We processed the image data set into two representation ways, the raw pixel representation and color representation. 70% of each data set is used for training, while the remaining 30% is used for testing. 20% of the training set in each data set is accounted for validation. We selected K-Nearest Neighbors, Decision Tree, Bagging (with Decision Tree), Random Forest, XGBoost, and Logistic Regression algorithms to train and test. The tuning strategy is used in each algorithm to improve the test accuracy. In addition, We evaluate the model prediction outcome by the F1 Score and McNemar's Test. Consequently, we find that the XGBoost, with a test accuracy of 94.88%, is the best machine learning model for this project.

1. Introduction

In December 2019, a new infectious disease, caused by the virus named SARS-CoV-2, was reported in Wuhan, Hubei, China. When the number of cases continued to rise around the world, the Centers for Disease Control and Prevention (CDC) announced the regulation requiring people to wear the mask in crowded outdoor settings and inside the buildings for adequate protection (https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/about-face-coverings.html) [8]. Currently, everyone is required to wear the mask while entering the buildings, and there is security personnel ensure that the mask is worn correctly. However, we are concerned such manual inspection is inefficient and wasteful of human effort. With the help of machine learning today, we believe that face mask recognition could achieve the goal while saving human resources.

Wearing masks has become the primary public health measure to prevent the spread of the current COVID-19 pandemic. Indoor spaces, such as malls, schools, and hospitals, require people to wear masks. Such requirement enforcement is often supervised by personnel. As we claimed early, such inspection is inefficient since it requires inspectors to keep their attention on everyone at all times. From our user experience, we found out many cell phones can recognize whether the owner wears a mask and change the unlock option. Inspiring by such observation, we came up with the idea of applying this face mask detection to the public space. By developing such a tool, we aim to reduce the work of personnel and increase the efficiency of scrutinizing mask-wearing, especially correct mask-wearing behavior. It is critical because we may not eradicate the COVID-19 in years, which means people need to wear masks to protect themselves for a long time. In addition, we believe that by ensuring correct mask-wearing behavior with other factors, such as vaccination, we will significantly reduce the spread of COVID-19.

To be precise with our project goal, we intend to discover the most suitable machine learning algorithms to classify whether people are wearing the mask correctly, wearing it incorrectly, or not wearing it at all. Other people with the same idea or interest can build the real-time face mask recognition from our project result. We download 4559 images with three class labels for our project to start. The class labels are balanced distributed in the data set. We then encode each mask-wearing behavior with numbers to represent the classification results: 0 means correct, 1 means incorrect, and 2 means no mask.

We trained K-Nearest Neighbors, Decision Tree, Bagging (with Decision Tree), Random Forest, XGBoost, and Logistic Regression algorithm to discover the best machine learning classification model for our project. We select two different ways to represent the raw image data: pixel and color features. The pixel feature is stored into raw_images.csv, and the color feature is saved in features.csv. Therefore, we have two training sections in the project, each accountable for one data set. Following that, we employ 70% of each data set for training and the remainder for testing. In addition, we employ 20% of the training set in each data set for validation. The previously described machine learning algorithms are trained using the two features input gathered in the first phase. We test six models on two data sets and use a tuning strategy

`RandomizedSearchCV` to determine the ideal parameters for each model. Then, we compare the test accuracy from the two training sections to find which input feature gives a better classification result. Finally, we use the F1 score and McNemar's Test to pick the most suitable model.

2. Related Work

Indeed, face recognition is a hot topic for research, and many researchers did the studies using different models. In 2003, M.J.Aitkenheada and A.J.S.McDonald developed and tested the neural-network based facial recognition program (FADER—FAce DEtection and Recognition)[1]. They used 1000 images of the face and 1000 images without the face as input, and the detection rate is more than 94%. Three different neural network models were used to train, and the Hebbian (a neuroscientific theory) connection strength adjustment model gave the highest accuracy of 74.1%.

Also, many intriguing projects use deep learning and machine learning algorithms to conduct face mask recognition. One study [10] used Convolutional Neural Networks (CNN), Support Vector Machine (SVM), and K-Nearest Neighbor (K-NN) models to extract deep features. The authors compared accuracy and precision to obtain the most satisfactory model, and the highest classification rate reached 97.1%.

Another effort in 2019[7] is also using machine learning approaches to identify the face mask. The authors used the Viola-Jones algorithm to detect faces from an image. Furthermore, they addressed and used Principal Component Analysis (PCA) algorithm on masked and non-masked face recognition.

3. Proposed Method

3.1. K-Nearest Neighbors

The first model we decide to use is K-Nearest Neighbors (K-NN). It is one of the supervised machine learning algorithms that are easy to implement while achieving high quality classification results. Given the training features $x^{[i]}$ with its corresponding labels $y^{[i]}$, for $i = 1, ..., n$, the K-NN algorithm simply remembers all of the training data and labels. During the fitting step, as we give K-NN algorithm a query point $x^{[q]}$, the algorithm would loop through all training samples and calculate the distance between current data samples and the query point. Finally, it would pick k smallest distance and its corresponding k training samples and decide query point $x^{[q]}$'s label as the most common labels from k training samples.

For our particular problem, we have five hyperparameters for tuning. k is the number of neighbors the algorithm will pick. We also search for optimal distance metrics with the leaf size and power parameter (p). The power parameter is particularly used for the Minkowski distance metric. However, when $p = 1$, it is same as using Manhattan distance metric, and when $p = 2$, it is equivalent to Euclidean distance [4]. Finally, we also consider the weight function in the K-NN algorithm, which is either to value all points in the neighborhood equally or to weigh the points by inverse proportional to its distance.

3.2. Decision Tree

The second model we use for our data set is the Decision Tree. Unlike the K-NN algorithm, it is an eager supervised learning algorithm, which means it tries to learn from training data and construct a generalized target function to predict newly added data. The decision tree utilizes the Divide-and-Conquer algorithm. Starting from the root node, the algorithm splits training samples into different branches based on features that lead to the most significant information gain. It repeats this splitting recursively until the data at each leaf node belongs to the same label, or there is no improvement.

In our case, we have three possible parameters for tuning. We use the Gini information criterion for splitting, but the number of min samples to split and the maximum depth of the tree is unknown for optimizing. Due to our large-scale data set size, our maximum depth searching range is limited up to 20. We also look for optimized value for the minimum impurity decrease. It means the algorithm would split the node if such split causes the impurity to decrease more than the given value [5].

3.3. Ensemble Methods

As we utilize the Decision Tree classifier on image features, we may wonder which elements lead to more accurate results for splitting. This is a natural idea, as we know that image features on the lips will have different changes due to the presence and absence of the mask. We hope that our model can consider these changes. The ensemble methods would actually allow us to take a set of the decision trees into account and pick the features that are important for prediction. Thus, we decide to use three ensemble methods, Bagging, Random Forest, and XGBoost, to improve the Decision Tree model.

3.3.1 Bagging

Bagging, also called Bootstrap Aggregating, is an ensemble method to fit the base model (in our case, Decision Tree) into random bootstrapped samples from the original data set. Then, we aggregated the predictions from each individual model to build up the final predictor. Due to the size of our image data set, we only consider 50 base estimators in this ensemble method. We also decided to use out-of-bag samples to validate its prediction scores and estimate the generalized error.

3.3.2 Random Forest

Random Forest is another ensemble method we apply to the Decision Tree model. It extends the bagging method that not randomly selects subsets of samples but also the subsets of features for each bootstrapped instance. This randomness helps to make each decision tree more independent and improves the ensemble prediction. For our particular case, we select 100 trees to train and choose three parameters for optimizing: the minimum number of samples to split, the minimum impurity decrease value, and the maximum tree depth.

3.3.3 XGBoost

The final ensemble method we use is XGBoost, which stands for Extreme Gradient Boosting. Precisely, XGBoost is a scalable gradient boosted decision tree machine learning library [3]. Similar to the random forest, gradient boosting is built from multiple decision trees. The difference is that gradient boosting creates the final predictor from many weak learners through several boosting rounds. The gradient decision tree iteratively trained the new model based on the error residuals from the previous model. It weighted the sum of all the tree predictions until there was no improvement. There are various booster parameters and learning task parameters; the former helps the booster while the latter one decides the learning scenario [6]. We let the random search tuning strategy help us find the best combination of these parameters.

3.4. Logistic Regression

Finally, we have our last model, Logistic Regression. It is a classical and widely used algorithm for classification. Specifically, we use multinomial logistic regression to generalize the logistic regression with three classes. The difference between logistic regression and other classification algorithms is that logistic regression calculates the probability of categorical membership by estimating the maximum likelihood.

The multinomial logistic regression is generalized from binary logistic regression. Explicitly, the model formula we would use is

$$P(Y_i = k) = \frac{\exp\left(\beta_{0k} + x_i \beta_k^{'}\right)}{\sum_{j=1}^{m} \exp\left(\beta_{0j} + x_i \beta_j^{'}\right)}$$

where x is the set of features and Y is the label. β_k is the coefficients vector of X for the kth category of Y. In our case, we are estimating the probabilities for $m = 3$ categories [2]. These unknown coefficients vector is estimated by minimizing the negative log likelihood

$$\sum_{i=1}^{N} \sum_{k=1}^{M} -y_k^{[i]} \log\left(a_k^{[i]}\right)$$

where N is the training sample size, M denotes the number of groups (labels), $x_k^{[i]}$ represents the features of the ith sample with kth label (group k) in the training set, $y_k^{[i]}$ denotes the corresponding label. a is the activation function output

$$h(x) = \sigma(\beta^T x + \beta_0) = a$$

[11]

4. Experiments

This section presents the information about our data set, software, and hardware. Since this project is based on the analysis of images, we also demonstrate our step to process the raw image data to make it practical for our model training.

4.1. Dataset

The dataset we choose is from Kaggle named "Face-Mask Recognition Dataset (FMRD)" (link: https://www.kaggle.com/hadjadjib/facemask-recognition-dataset) [9]. It consisted of 4559 images with three class labels, "Correct," "Incorrect," and "NoMask." The Table1 shows the raw data structure of FMRD.

Label	Count
Correct	1559
Incorrect	1500
NoMask	1500
Total	4559

Table 1: This is the raw data structure of image data set

According to the table, the data from FMRD are generally balanced, so we simply do a stratification split after the data processing to make sure each training, validation, and test set would have balanced class labels. The data processing steps will be explained in the following subsection. The ratio of training, validation, and test dataset are 0.56:0.14:0.30.

4.2. Data Processing

Before we are able to train any practical machine learning algorithm, we need to transform the input image into a vector of some numbers that represent the contents of the picture. We first simply take the input image, resize it to a fixed size, and then flatten three color channel components into one single array input. In other words, this is simply a raw pixel representation. The data was processed and stored into file "raw_image.csv" for later training. The overall size after transformation is 3.42 GB. Since this representation leads to a large data size, we also try another representative method, hoping to get a good result while saving running

time. The other representation is the color histogram, coming from the idea that the color of the mask and human face would be different, so the image's color distribution may be an appropriate feature for training. This transformation has a much better storing size, 58.4 MB. The data is stored in file "features.csv." At the same time, we also save labels information into the file "labels.csv." We then read all data information from these three csv files and train the data.

4.3. Software

For the data processing and model training, we simply use Jupyter Notebook with Python 3.8 or 3.9, depending on each of our systems. We have used NumPy and Pandas for data processing, reading, and splitting, matplotlib and mlxtend for plotting the graph we need, scipy for general computing, and scikit-learn and xgboost for various model training.

4.4. Hardware

Each group member uses their own laptop, all for Mac with different chips.

5. Results and Discussion

After processing the images we access online, we obtain the images into a 4559×30000 matrix for pixel representation and a 4559×512 matrix for color representation. We use `train_test_split` to divide each of two data into 70% of training and 30% of testing, and use `train_test_split` again to obtain 20% of training set as validation set. The split is stratified to make sure that training, testing, and validation sets can have the same proportions of 0 (correct), 1 (incorrect), and 2 (no mask) as observed in the original data. For randomness, we use the random seed of 123. To conduct the hyperparameter tuning, we choose `RandomizedSearchCV`.

We trained and tuned models on both input features and compared their accuracy results. Table 2 shows the comparison. According to the table, we find out that the relatively large dataset raw_images.csv has higher accuracy in general, so for convenience and length of report, we will only present our training and tuning process on pixel representation in the following sections.

Models	features.csv	raw_images.csv
K-NN	83.26%	92.62%
Decision Tree	87.21%	90.13%
Bagging	89.18%	91.23%
Random Forest	90.72%	94.44%
XGBoost	91.74%	94.88%
Logistic Regression	78.22%	93.20%

Table 2: Accuracy Comparisons on Two Input Features

5.1. Training and Hyperparameter Tuning

5.1.1 K-NN

We use `KNeighborsClassifier` to build the K-NN model. Two parts are involved in finding the best parameter k and the other parameters for the model. Firstly, we compare each error rate corresponding to k and choose the parameter k with the lowest error rate. We examine k from 1 to 29 and compute the corresponding error rates. From Figure 1, we can see that the lowest error rate occurs when k is 4, and the error rate increases when k is larger than 4. Therefore, using the error rate, we train the K-NN model with $k = 4$ with $p = 2$ (euclidean_distance) and uniform weight function. This model is the base K-NN model.

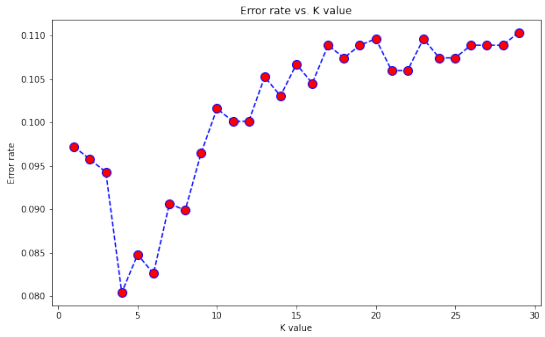

Figure 1: KNN - Error Rate vs. Neighbours

Then, we use the tuning strategy to explore parameters that produce a better performance of the K-NN model. Five parameters are involved in the tuning process: the number of neighbors, leaf size, power parameter (p), weight function (weights), and distance metric to use for the model (metric). Based on the error rate we checked before, we only search for the optimal value of k from 3 to 7 because larger k has a larger error rate, which may not work well for the model. In addition, we set leaf size from 20 to 31 coupling with p = [1,2], weights = ['uniform','distance'] and metric = ['minkowski','chebyshev']. After tuning, we get the best parameter for the K-NN model shown in the Table 3.

n_neighbors	leaf_size	p	weights	metric
6	20	1	'distance'	'minkowski'

Table 3: Best Parameters in K-NN Model

Using the K-NN model from the error rate method, we obtain its test accuracy of 91.96%. When we further apply the tuning strategy, we get the test accuracy of 92.62%. We see an increase in test accuracy, so we use the model with 92.62% for future accuracy comparison. Its F1 score is equal to 0.9263.

5.1.2 Decision Tree

We use `DecisionTreeClassifier` for the Decision Tree Model. Without tuning strategy, the parameters are default with the random seed of 123, and we get the test accuracy of 87.87%. Then, we apply the tuning strategy. There are three hyperparameters involved: max_depth, min_sample_split, and min_impurity_decrease. We search for the optimal max_depth from 1 to 20 coupling with min_sample_split from 2 to 11 and min_impurity_decrease from 0 to 0.5. Finally, we find the best parameters shown in the Table 4.

min_samples_split	min_impurity_decrease	max_depth
2	0.011493972900871952	4

Table 4: Best Parameters in Decision Tree

After tuning, the Decision Tree model produces the test accuracy of 90.13%, and we also compute its F1 score equal to 0.9018.

5.1.3 Bagging on Decision Tree

By using `BaggingClassifier` from scikit-learn, we want to apply bagging to the previous best decision tree to improve its performance. Considering the size of our dataset, we only use 50 base estimators, and the other parameters are as shown in Table 5:

oob_score	bootstrap	bootstrap_features	n_jobs
True	True	False	1

Table 5: Tuning Parameters in Bagging

The test accuracy is slightly improved to reach 91.23%, as well as the 0.9121 F1 score, so the ensemble method has some positive effect on the performance of the decision tree.

5.1.4 Random Forest

For Random Forest, we use `RandomForestClassifier` with 100 estimators and a random seed of 123 to build the model without tuning strategy, and it achieves the test accuracy of 94.44%. To check whether the model after tuning can have higher test accuracy, we do similar works to Decision Tree that there are three hyperparameters involved: max_depth, min_sample_split, and min_impurity_decrease. We search for the optimal max_depth from 1 to 20 coupling with min_sample_split from 2 to 11 and min_impurity_decrease from 0 to 0.5. Finally, we find the best parameters shown in the Table 6.

After tuning, the Random Forest model produces the test accuracy of 90.42%. The model with tuning has lower test accuracy than the model without adjusting, so we decide to use the model with the test accuracy of 94.44% as our final Random Forest model for future algorithm comparison. Additionally, we compute its corresponding F1 score, equal to 0.94438.

min_samples_split	min_impurity_decrease	max_depth
15	0.013666043770271419	2

Table 6: Best Parameters in Random Forest

It's normal to see that the Random Forest model has higher test accuracy than the Decision Tree model. The random forest relies on assembling many decision trees to arrive at its final predictor. This aggregation usually improves the weak prediction happening in a single decision tree. Even though the Decision Tree is easier to understand, Random Forest works better on the large dataset such as our input. The ensemble algorithms, including the other classifiers (bagging, xgboost), are more robust than a single decision tree.

5.1.5 XGBoost

For XGBoost, we use the `XGBClassifier` from xgboost package. We tune 12 hyperparameters such as n_estimators and alpha. The exact values for each parameter are shown in the Table 7.

Hyperparameter	Value Range
n_estimators	30, 50, 100, 300, 500
min_child_weight	4,5
reg_lambda	10^{-8} - 1
alpha	10^{-8} - 1
gamma	0.3 - 0.6
subsample	0.6 - 1.1
colsample_bytree	0.6 - 1.1
max_depth	2,3,4,6,7
objective	'reg:squarederror', 'reg:tweedie'
booster	'gbtree', 'gblinear'
eval_metric	'rmse'
eta	0.3 - 0.6

Table 7: Tuning Parameters in XGBoost

Due to the large size of both dataset and hyperparameters, we use random search `RandomizedSearchCV` to find the best hyperparameters out of 15 iterations of 5-fold cross validation to make it less expensive. After tuning, we found the best hyperparameters, and they are shown in Table 8.

The after-tuning model has a test accuracy of 94.88% and an F1 score of 0.9487.

Hyperparameter	Value
n_estimators	500
min_child_weight	4
reg_lambda	0.0576480249714419
alpha	0.3929444207680876
gamma	0.5
subsample	1.0
colsample_bytree	1.0
max_depth	2
objective	'reg:squarederror'
booster	'gbtree'
eval_metric	'rmse'
eta	0.3

Table 8: Best parameters in XGBoost

5.1.6 Logistic Regression

We import `LogisticRegression` from scikit-learn library. The tuning strategy changes the L_1 regularization parameter C, the boolean value to decide if added a constant term to the decision function (fit_intercept), and a boolean value to decide whether to use the previous solution (warm_start) (Table 9).

Hyperparameter	Value Range
C	scipy.stats.expon(scale=.01)
fit_intercept	True, False
warm_start	True, False

Table 9: Tuning Parameters in Logistic Regression

After doing a random search of 15 iterations with 5-fold cross-validation, we find the best hyperparameters shown in Table 10.

C	fit_intercept	warm_start
0.002573	True	False

Table 10: Best Parameters in Logistic Regression

The model using the best hyperparameters achieves a test accuracy of 93.20% and an F1 score of 0.9320.

5.2. Model Evaluation and Selection

5.2.1 McNemar's Test on Random Forest and XGBoost

Since the performances of the random forest and XGBoost models are best according to the F1 score, we use McNemar's Test to check whether these two models have the same results for our particular problem. Our null hypothesis is that the performances of the two models are equal. Figure 2, Figure 3, and Table 11 show some details.

		Random Forest	
		correct	wrong
XGBoost	correct	1277	15
	wrong	21	55

Table 11: Confusion Matrix - XGBoost & Random Forest

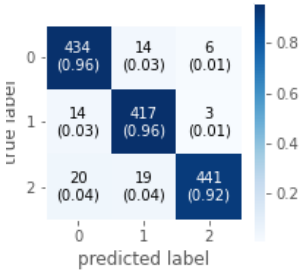

Figure 2: McNemar's Test - Random Forest

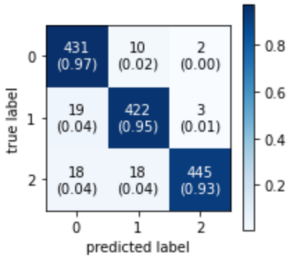

Figure 3: McNemar's Test - XGBoost

The p-value we get from the test is 0.4047, which is greater than $\alpha = 0.05$, so we fail to reject the null hypothesis and conclude that the performances of the two models are the same. Thus, based on Figure 12, we will choose the model with the slightly higher F1 score as our final model– XGBoost.

Random Forest	XGBoost
0.94438	0.9487

Table 12: F1 Score Comparison

5.3. Discussion

Although we achieve a high accuracy in two models, it is extremely computationally expensive to fit the models and tune the hyperparameters due to the large size of our dataset, which will be one of the weaknesses of our project. In addition, we separate our dataset into training, test, and validation datasets, which may lead to a pessimistic bias in our result. Future improvement can include other different feature representations, such as features of shape or texture, which may reduce the number of input variables while achieving

high prediction results and saving the cost.

At the same time, we acknowledge that Convolutional Neural Networks (CNN) is a robust model for image classification. Due to the time limit, we haven't had a chance to try CNN in our data set. If any other who have a similar interest in face mask recognition, we believe CNN would be a worthy option for a try.

6. Conclusions

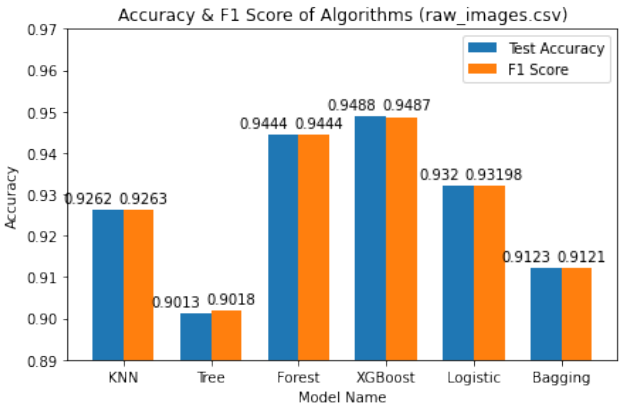

Figure 4: Test Accuracy & F1 Score

We have accomplished our initial motivation in the above analysis by finding the optimal way to classify images of people wearing masks, not wearing masks, and wearing masks incorrectly. We investigate six potential algorithms, including K-NN, Decision Tree, Bagging, Random Forest, XGBoost, and Logistic Regression. We train them on the training dataset and tune hyperparameters using random search. Finally, we evaluate different models using both test accuracy and F1 score, and compare the best two models Random Forest and XGBoost by McNemar's Test. We conclude that XGBoost with an F1 score of 0.9487 will be our final model, although Random Forest has the same good performance. All the accuracy and F1 scores are shown in the Figure 4 above.

We aim to efficiently scrutinize mask-wearing behavior during the pandemic to protect public health. We hope to improve our classification algorithm to make it less computationally expensive in the future. It would be appreciated and excited if other people engaging in real-time face mask recognition could utilize our classification model and apply it to public places such as malls and airports in the future.

7. Acknowledgements

We give many thanks to user *hadjadji b* on Kaggle for publishing the face mask dataset and allowing unrestricted use of them.

8. Contributions

Seanna Zhang found the original image dataset online, extracted the feature datasets, and cleaned up the data for further analysis. Christine Hou trained and evaluated the K-NN, Decision Tree, and Random Forest models with hyperparameter tuning. Yue Yin trained and assessed the rest models– XGBoost, Logistic Regression, and Bagging on Decision Tree, and did the McNemar's Test.

For the report, Seanna wrote the Proposed Method and Experiments section. Christine finished the Abstract, Introduction, Related Work, and Results and Discussion part. Yue Yin completed the rest of the Results and Discussion, Conclusions, Acknowledgements, and Contributions.

References

[1] M. J. Aitkenhead and A. J. S. McDonald. A neural network face recognition system. *Engineering Applications of Artificial Intelligence*, 16:167–76, 2003.

[2] M. Carpita, M. Sandri, A. Simonetto, and P. Zuccolotto. Cahpter 14-football mining with r. *Data Mining Applications with R*, pages 397 – 433, 2014.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[4] S.-L. Developers. sklearn.neighbors.kneighborsclassifier.

[5] S.-L. Developers. sklearn.tree.decisiontreeclassifier.

[6] X. Developers. Xgboost parameters.

[7] S. Ejaz, M. Rabiul Islam, M. Sifatullah, and A. Sarker. Implementation of principal component analysis on masked and non-masked face recognition. *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, pages 1–5, 2019.

[8] C. for Disease Control and P. (CDC). Your guide to masks, 2021.

[9] hadjadji b. Face-mask recognition dataset, 2021.

[10] A. Oumina, N. E. Makhfi, and M. Hamdi. Control the covid-19 pandemic: Face mask detection using transfer learning. *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, pages 1–5, 2020.

[11] S. Raschka. Lecture slides 08 logistics regression and multiclass classification, 2021.