A Web-Accessible Machine Learning Model for Assessing Lung Cancer Risk

Harry Albert hsa1@williams.edu
Computer Science

1. Introduction

In 2020 in the United States, 197,453 new cases of Lung and Bronchus cancer were reported, and 136,084 people died of this cancer (CDC, 2023). While Lung cancer can be cured if it is found in its early stages, median survival is less than two years if it is not found until its later stages (Dritsas and Trigka, 2022). Thus, early recognition of and treatment for lung cancer is vitally important. Although there have been papers written on the use of machine learning in conjunction with common (and easily diagnosable) symptoms in the prediction of lung cancer (Dritsas and Trigka, 2022), the results of said papers have not been published in an accessible and easily usable way.

In this paper, I aim to mend this gap in the research. I will create a neural network classification model which, using some easily self-diagnosable symptoms, can classify individuals as having or not having lung cancer with a high degree of accuracy. While this model will not be able to diagnose users with lung cancer, it can serve as a basis to prompt users to seek actual medical help (hopefully at a sufficiently early stage of lung cancer). My model will achieve a performance in line with the state of the art machine learning models presented in previous papers. I will create a website for this model using Vercel (Ver, 2023), Node.js (Tilkov and Vinoski, 2010), Material UI (mat, 2023), and DALL-E 2 (to create the logo for my website) (Ramesh et al., 2022). Users will be able to interact with the model using this site. I will also create a decision tree to classify users as having or not having lung cancer. My hypothesis is that this tree, when visualized, will contain smoking as one of the highest data splits, showing that smoking is a good predictor of lung cancer. I hope that this will serve to discourage some users from smoking, or at least encourage them to question their smoking habits.

2. Preliminaries

In analyzing the data and attempting to create an effective classifier, I used many different technologies and methods. Specifically, I employed standardization and SMOTE in my data preprocessing, used a neural network and a decision tree for categorization, and calculated precision, recall, f1, and accuracy to quantify the success of the categorization models. To read in and modify the data, I used pandas (McKinney et al., 2011) and numpy (Oliphant et al., 2006). I used imbalanced-learn (Lemaître et al., 2017) to perform SMOTE on the dataset. I used scikit-learn (Kramer and Kramer, 2016) to train my models. In this section,

I will describe each of the methods I used in order to provide a basis of understanding for the rest of the paper.

Standardization is a method employed to give each feature in the data a mean of 0 and a standard deviation of 1. The formula for standardization is

$$X' = \frac{X - \mu}{\sigma}$$

where μ represents the mean of X and σ represents the standard deviation.

The relu function is often used as the activation function for a neural network (terms which will be defined in the following paragraphs). The relu function is defined as

$$f(x) = \max(0, x)$$

In practice, this means that, when fed through the relu function, any negative value will become zero and any positive value will go unchanged. The derivative of the relu function when $x < 0$ is 0, and the derivative when $x \geq 0$ is 1.

A neural network is, at its core, a directed, acyclic graph (DAG) with some additional structure. A graph is a DAG if: it is simple (has a maximum of one edge between each vertex/node), it only contains directed edges, and there are no directed cycles (i.e. there is no path from any vertex back to itself). In addition to these requirements, a neural network can be partitioned into layers $L_1, L_2, \ldots, L_n$, such that any vertex in layer L_i can only have vertices in layer L_{i-1} as its parents. It is often the case that a vertex in layer L_i has all the vertices in layer L_{i-1} as its parents.

Each edge from any vertex in L_{i-1} to any vertex in L_i represents a learnable weight/parameter, forming the weight matrix $[\theta]^{L_{i-1} \to L_i}$. To compute any layer's value, you simply apply the activation function (in this case relu) to the layer's weight matrix multiplied by the previous layer's values:

$$X_{L_i} = \text{relu}(X_{L_{i-1}}[\theta]^{L_{i-1} \to L_i})$$

It is important to note that each layer will contain an additional node which simply has a value of 1. This node will serve as an intercept term which can be modified for the following layer's values. The value of the last (rightmost) node(s) in the network serve as the network's output.

To update the neural network, you simply perform gradient descent on each parameter in the neural network some number of times (epochs). The formula for gradient descent is

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla L(\theta^{(t)})$$

where α is the learning rate. To compute the gradient of your parameters, you use back-propagation (as computing the gradients by perturbing each parameter requires k forward passes through the network, where k is the number of parameters). Recall that a neural network can, at its core, be represented by a DAG. To perform backpropagation, you first

initialize $\frac{\partial L}{\partial L} = 1$, where L is the last layer in the network. You then move in reverse topological order via breadth first search, applying the chain rule to each parameter in order to ascertain said parameter's gradient. This gives us the gradient of each parameter in the network in only one passthrough of the network.

In stochastic gradient descent, you modify the update rule. Instead of updating θ based on the gradient of the entire dataset, you split the dataset into smaller batches and update the gradient based on the average gradient in each of these batches. So, the update rule now looks like

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla L_{batch_i}(\theta^{(t)})$$

You apply this rule for each batch in the dataset, and repeat this process for some specified number of epochs.

Lastly, Nesterov's Momentum modifies the update rule as follows:

$$m_{t+1} = \mu m_t + \alpha \nabla L_{batch_i}(\theta^t - \mu m_t) \quad (\dagger)$$
$$\theta^{(t+1)} = \theta^{(t)} - m_{t+1} \quad (\ddagger)$$

where μ represents the momentum constant, a parameter which can be tweaked by the user (Dozat, 2016). In (†), you update the momentum m by adding the previous momentum — scaled by the momentum constant μ — to the α scaled gradient of the loss. This loss is calculated based on an initial guess for $\theta^{(t+1)}$, which you approximate by subtracting the current scaled momentum μm_t from $\theta^{(t)}$. In ‡, you subtract the updated momentum from $\theta^{(t)}$ to obtain $\theta^{(t+1)}$.

Incorporating Nesterov's Momentum into the update rule allows the model to converge faster, as taking previous gradients into account allows the model to update θ by larger values in steeper regions of the loss function. It also reduces the speed at which θ is updated in shallower regions, reducing the risk of overshooting the target and providing stabler gradients on the whole. In addition, because Nesterov's Momentum effectively 'looks ahead' to the next position of θ when updating it, the positive effects of momentum (faster and more stable updates) are further solidified.

A decision tree is created through a process of splitting the data in an attempt to minimize a loss function. To create a decision tree, you begin with a node which contains all of your data, $[D] = ([X], Y)$. You then search through all possible ways of splitting the data on values in $[X]$. This best split is then used to create two children, $[D]^{s1}$ and $[D]^{s2}$. This process is continued recursively until you reach some maximum depth or until the loss function has been completely minimized. A decision tree with some maximum depth is called a limited depth decision tree, and a decision tree which can grow until the loss function has been minimized is an unlimited depth decision tree. Entropy is often used as the loss function to determine the best split. Entropy is calculated as follows:

$$H([D]) = -\sum_c p_c \log_2(p_c)$$

where p_c denotes the portion of data points in $[D]$ where Y has the class label c. The way to calculate the entropy of a data split is through the weighted entropy:

$$H([D]^{s1}, [D]^{s2}) = \frac{\text{len}([D]^{s1})H([D]^{s1}) + \text{len}([D]^{s2})H([D]^{s2})}{\text{len}([D]^{s1}) + \text{len}([D]^{s2})}$$

Each split, you aim to minimize this weighted entropy.

In measuring the performance of a machine learning model, the following metrics are often used: precision, recall, f1 score, and accuracy. Each of these scores takes into account either the number of true positives, true negatives, false positives, or false negatives. A true positive (TP) or a true negative (TN) represent a correct positive or negative prediction (i.e. correctly predicting that a patient does or does not have cancer). A false positive (FP) is when the model predicts a positive outcome but the outcome is actually negative, and a false negative (FN) is when the model predicts a negative outcume but the outcome is actually positive.

The formula for precision is

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision quantifies how accurately the model predicts positive outcomes, with a low precision score indicating that the model has predicted many false positives. However, this metric does not capture the amount of positive outcomes which the model misses (i.e. false negatives).

The formula for recall is

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

Recall quantifies how accurately the model can identify all positive outcomes, with a low recall score indicating that the model has incorrectly identified many positive outcomes as negative. This metric does not capture the amount of negative outcomes which the model incorrectly identifies as positive, meaning that a model which drastically overestimates positive outcomes would still score well on recall.

The formula for the f1 score is

$$2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

The f1 score is simply a weighted average of precision and recall, providing one metric which describes how the model is doing in both of these metrics. This means that, even if the model is performing very well in one metric to the detriment of the other (i.e. overestimating positives to perform better in recall), the f1 score will still suffer as a result.

The formula for accuracy is

$$\frac{\text{TP} + \text{TN}}{\text{Total Observations}}$$

Accuracy simply measures the proportion of correct predictions to total predictions. While accuracy is very intuitive, it can be misleading for imbalanced datasets. If a dataset has a large proportion of one outcome over another, a model which simply predicts that outcome everytime would still achieve a high accuracy. This is where precision, recall, and f1 scores provide a more nuanced picture of the model.

Synthetic Minority Over-sampling Technique (SMOTE) is an algorithm which can be used to address imbalances in outcomes in a dataset (Chawla et al., 2002). In performing SMOTE, you first randomly pick a point from the underrepresented class. You then find the k-nearest neighbors of that point (where k is a parameter set by the user). In the imbalanced learn implementation of SMOTE which I used in this paper (Lemaître et al., 2017), the k-nearest neighbors are found using the value difference metric (Stanfill and Waltz, 1986). The value difference metric computes the distance between two feature vectors, X and Y, using the following formula:

$$\delta(x, y) = \sum_{c=1}^{C} |p(c|x_f) - p(c|y_f)|^k$$

$$\Delta(X, Y) = \sum_{f=1}^{F} \delta(X_f, Y_f)^r$$

where x and y are two samples, f a given feature, C is the number of classes, F is the number of features, and r and k are exponents usually set to either 1 or 2. After you find the k-nearest neighbors to the point using this formula, you select one neighbor at random. You then draw a "line" between the original point and the selected point (or draw a line between the values of each feature in the two points), and select a random point on this line. This random point is then added to the dataset as a new value. This process of random selection, k-nearest neighbor searching, and point generation is repeated until the desired balance between classes has been achieved (which in many cases is simply a 50/50 split).

Finally, a causal DAG is a directed acyclic graph wherein each directed edge between A and B implies that A *might* be a cause of B. A lack of such an edge implies that B is independent of A given its direct causes. So, given any topological ordering of the causal Dag $V_1, \ldots, V_k$, the following property holds:

$$V_i \perp\!\!\!\perp \text{past}_g^*(V_i) | \text{past}_g(V_i)$$

where $\text{past}_g^*(V_i)$ denotes all variables appearing before V_i in the topological ordering, except for the parents of V_i. This implies that V_i is independent of all of its predecessors in the topological ordering given its parents, except for its parents.

A topological ordering is an ordering of a graph such that, for every direct edge from V_i to V_j, V_i appears before V_j. In other words, no node which has a direct edge to another node will appear after that node, and every node's parent(s) will appear before that node in a topological ordering.

3. Data

The data which I used to train my models was obtained from the Kaggle data set posted by Bhat (2021). This model contains 284 instances and 16 features (15 inputs for the model and one target). All of the features are categorical and binary except for age, which is numerical. The features are: Gender, Age, Smoking, Yellow Fingers, Anxiety, Peer Pressure, Chronic Disease, Fatigue, Allergy, Wheezing, Alcohol, Coughing, Shortness Of Breath, Swallowing Difficulty, Chest Pain, and Lung Cancer (the target). Unfortunately, the dataset does not include additional classifiers for other sexes or genders not captured by the male/female binary.

The dataset contains 39 people (12.6%) without lung cancer and 270 people (87.4%) with lung cancer. I attempted to reduce this skew in the outcomes by using SMOTE (which I will discuss in the next paragraph), but it should still be noted that there are very few unaltered data points for people without lung cancer. The following summary statistics are based on the dataset *before* SMOTE was performed. For the age feature, the mean is 62.7, the mode is 64, the minimum is 21, and the maximum is 87. The dataset is clearly skewed towards older people, an issue which would ideally be remedied in future research and/or new datasets. Finally, 56.3% of the people in the dataset are smokers, 50.4% have chronic disease, and 55.7% have allergies.

The data was read in and processed using pandas (McKinney et al., 2011). It should be noted that some of the feature names in the data had trailing spaces (specifically fatigue and allergy) which I manually removed. In processing the data, I converted each categorical (and binary) variable to map to either a 0 or a 1 (some of the variables previously mapped to either 1 or 2, or were associated with strings such as "MALE" instead of numbers). I then applied SMOTE — with the number of neighbors (k) set to 5 — to the dataset in order to equally balance the occurrences of lung cancer versus non lung cancer. The data was split into an X matrix, which contained every feature except lung cancer, and a Y matrix, which contained only lung cancer. The X and Y matrices were then split into training, validation, and testing sets (with a split of 56.25% for training, 18.75% for validation, and 25% for testing). These matrices were then standardized (using only the standard deviations and means from the testing set so as to avoid overfitting), and were finally converted into numpy matrices (Oliphant et al., 2006).

I also created a separate dataset which contained only the features Gender, Age, Smoking, and Lung Cancer. All of the processes described above were also applied to this reduced dataset.

4. Training And Validation Of Models

To create both the neural network and the decision trees, I used the python package scikit-learn (Kramer and Kramer, 2016). As my baseline for performance, I used the paper by Dritsas and Trigka (2022), as this paper analyzed the same data set that I analyzed (Bhat, 2021). The authors of this paper achieved an accuracy, precision, recall and F-Measure of 97.1%.

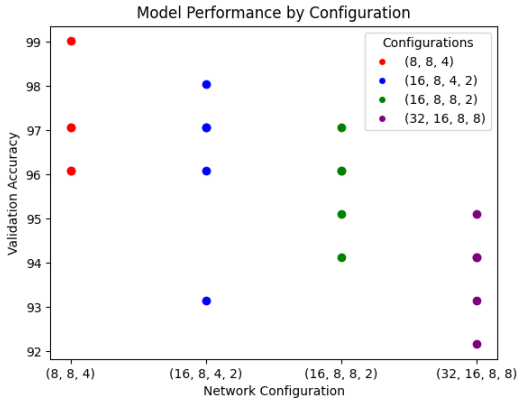

Figure 1: Accuracy of different model sizes when tested on the validation set. Created using Matplotlib (Hunter, 2007).

In order to train my neural network, I created a program to test out different configurations using a validation set. I created a program which trained each of the following neural network configurations 5 times, where each number represents one layer size within the network: (8, 8, 4), (16, 8, 4, 2), (16, 8, 8, 2), and (32, 16, 8, 8). I then saved the one model which performed the best on the validation set, which had the configuration (8, 8, 4). Each model was trained using 500 iterations (with data shuffling), the relu activation function, an alpha of 0.0001, the stochastic gradient descent solver, a batch size of 32, and nesterov's momentum with the momentum constant set to 0.9. I kept the alpha, batch size, and iterations constant as initial testing suggested that the modifying of these hyperperameters did not make a consistent or significant difference in the model's performance on the validation set. Thus, I selected seemingly reasonable values for these constants. The momentum constant was set to 0.9 because this is the constant suggested by scikit-learn. Figure 1 shows the accuracy of the various models which were trained when tested on the validation set.

I trained two separate decision trees. The trees made splits using entropy as the criterion. One tree had an infinite depth and one had a depth of 5. The tree with a depth of 5 was originally intended as an educational model which could be used to show that smoking is a good predictor of lung cancer. This was not successful, the possible reasons for which will be elaborated on in the Discussion and Conclusion Section.

5. Results

The best performing neural network had a configuration of (8, 8, 4). The network achieved a precision of 94.4%, a recall of 98.6%, an f1 of 96.5%, and an accuracy of 96.3%. This does not reach the baseline accuracy, precision, or F-Measure of 97.1%. However, my model does outperform the baseline model in terms of recall. This is an ambiguous result, as it is arguable that predicting the highest proportion of truly positive cases (i.e. recall)

Feature Perturbance Results										
Gender	Anxiety	Chronice Disease	Allergy	Age	Smoking	Peer Pressure	Fatigue	Wheezing	Coughing	Chest Pain
3	10	17	7	1	4	2	6	3	3	4

Figure 2: Table with how often a perturbation of any given feature changed the model's output.

is extremely important in lung cancer prediction. Failing to predict a positive case of lung cancer could be harmful or even deadly. On the other hand, the baseline is slightly better than my model in all other metrics. So, it is ambiguous whether my model performs strictly better or worse than the baseline. My model is available for public use at `https://lung-cancer-risk-assessment-tool.vercel.app/`.

To test which variables have the most impact on the model, I created a program which perturbs every possible combination of 3 features in the dataset, and outputs which perturbation changes the results. The initial sample which was perturbed had 0 for every non-continuous variable — "no" for all binary variables, and "Male" for gender. Age was set to the mean age of 62.7. A table which shows the number of times each feature appeared in a perturbation which changed the output of the model can be found in Figure 2. In short, it appears that chronic disease is, by a wide margin, the variable which is most likely to change the output of the model when perturbed, followed by anxiety, allergy and fatigue.

The unlimited depth decision tree achieved a precision of 93.0%, a recall of 95.7%, an f1 of 94.3%, and an accuracy of 94.1%. The decision tree with a depth of 5 achieved a precision of 95.5%, a recall of 92.8%, an f1 of 94.1%, and an accuracy of 94.1%. Both of these trees performed strictly worse than the baseline. However, the original intention with the decision trees was to provide an intuitive demonstration that smoking is a large risk factor for cancer. As such, I hypothesized that smoking would appear high up on these decision trees. However, smoking did not appear until very low in either tree. Instead, features such as age, gender, fatigue, and swallowing difficulty appeared higher up in the unlimited depth decision tree. Upon further thought, this makes intuitive sense. The causal relationship between the features in the dataset and lung cancer are far from simple, and smoking is not the only possible cause for lung cancer within the set of features in the data (Figure 3). While smoking might be a direct cause of cancer, it is also possible that its downstream effects, such as allergies, predict cancer (although the relationship between allergies and lung cancer is controversial (El-Zein et al., 2014)). In addition, high levels of smoking might cause some symptoms such as fatigue and difficulty swallowing (Corwin et al., 2002; ŞANLI et al., 2016), again making these potentially better predictors than smoking itself. So, it is not surprising that smoking does not appear as a high-level split in either decision trees.

6. Ablation Study

For my ablation study, I decided to remove every variable that could be caused by smoking (Figure 3). This left me with just Gender, Age, and Smoking. The reason I did this was that features such as fatigue and swallowing difficulty were appearing higher up on the

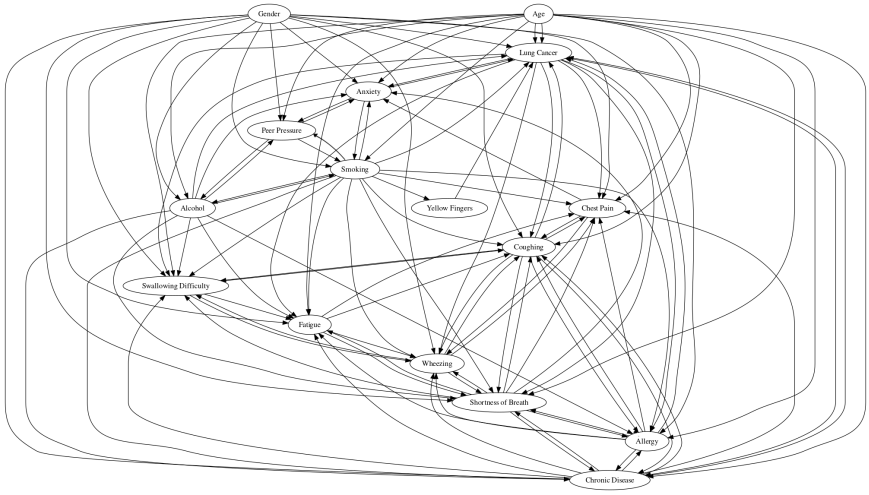

Figure 3: A possible causal graph for the variables in the dataset. This is not a causal DAG, as there are cycles. Created using Graphviz (Ellson et al., 2002).

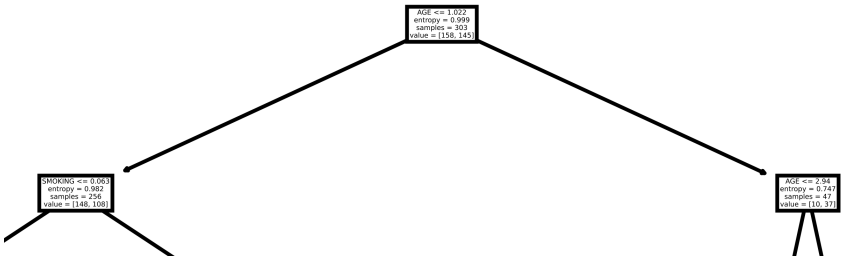

Figure 4: First two layers of limited depth decision tree trained on the reduced feature dataset. Created using Matplotlib (Hunter, 2007).

decision tree than smoking. Using just these features, I trained another neural network (using the same methods as described above), as well as another decision tree with a depth of 5. For the original dataset, the best neural network had a training accuracy of 97.7% , a validation accuracy of 99.0%, and a testing accuracy of 96.3%. With the new dataset, the best neural network had a training accuracy of 72.9%, a validation accuracy of 73.5%, and a testing accuracy of 71.9%. The best limited depth decision tree had a training accuracy of 86.1%, a validation accuracy of 79.4%, and a testing accuracy of 77.0%. As expected, smoking moved upwards in the decision tree to the second layer (Figure 4).

7. Discussion and Conclusion

Overall, I am happy with how the neural network has turned out. Although it is not a significant step above the baseline, it is also not clearly worse than the baseline. Fur-

ther, my initial goal of publicizing a state of the art model has been achieved (at `https://lung-cancer-risk-assessment-tool.vercel.app/`). However, the neural network is somewhat limited by both the size and the detail of the dataset which it was trained on. The dataset is small (with only 284 instances), and also does not include *how much* any given person smokes, but only whether that person smokes or not. The fact that so many of the features in the dataset are binary reduces the amount of personalization and specificity which can be incorporated into the model. In addition, the dataset is limited in age rage, as it has only 3 patients with ages less than 40. A future paper could address these issues by either creating or focusing on an existing dataset without some of these limitations. Further, I failed in my goal of creating a decision tree which could serve to show that smoking is a significant feature in assessing lung cancer. For the future, I realize that I should think about a possible causal graph of the features in a dataset before thinking about which feature might be significant in any model's predictions.

References

Cancer statistics at a glance. https://gis.cdc.gov/Cancer/USCS/#/AtAGlance/, 2023.

Vercel. https://vercel.com, 2023.

Material ui. https://mui.com/material-ui/, 2023.

Mysar Ahmad Bhat. Lung cancer. https://www.kaggle.com/datasets/mysarahmadbhat/lung-cancer, 2021.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357, 2002.

Elizabeth J Corwin, Laura Cousino Klein, and Kristin Rickelman. Predictors of fatigue in healthy young adults: moderating effects of cigarette smoking and gender. *Biological research for nursing*, 3(4):222–233, 2002.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Elias Dritsas and Maria Trigka. Lung cancer risk prediction with machine learning models. *Big Data and Cognitive Computing*, 6(4), 2022. ISSN 2504-2289. doi: 10.3390/bdcc6040139.

Mariam El-Zein, Marie-Elise Parent, Jack Siemiatycki, and Marie-Claude Rousseau. History of allergic diseases and lung cancer risk. *Annals of Allergy, Asthma & Immunology*, 112 (3):230–236, 2014.

John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, pages 483–484. Springer, 2002.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Oliver Kramer and Oliver Kramer. Scikit-learn. *Machine learning for evolution strategies*, pages 45–53, 2016.

Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1):559–563, 2017.

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

Travis E Oliphant et al. *Guide to numpy*, volume 1. Trelgol Publishing USA, 2006.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1 (2):3, 2022.

Arif ŞANLI, Eda Bekmez, Gazi Yildiz, Banu Atalay Erdoğan, Hüseyin Baki YILMAZ, and Gökhan ALTIN. Relationship between smoking and otorhinolaryngological symptoms. *The Turkish Journal of Ear Nose and Throat*, 26(1):28–33, 2016.

Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

Stefan Tilkov and Steve Vinoski. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.