Final Project Report: Machine Learning Model Development & Evaluation

Project Group: Arthur Castillo, Cristian Ceja Bolanos
Date: August 25th, 2025
Course: CSCI 164
Instructor: David Ruby

Dataset Exploration

Selected Datasets

Dataset 1: Gender by Name

Overview

The Gender by Name database includes 4 different features: Name, Gender, Count, and Probability. All the data within the database is from government sources from the following 4 countries: UK, US, Canada, and Australia. Since our model will be used to predict if a person is Male or Female based on their name, the target variable will be "MajorGender" and we will be using classification models. For our data, we decide to exclude the 4th feature "Probability" as it wasn't very clear what it represents and didn't seem to effect our results. The reason we chose this dataset was because it had a straightforward goal that could be used for understating more about how different models work with data that has a small set of features, but many instances.

Comparison and benchmark of name-to-gender inference services

This study underscores the challenges of cultural diversity in name-based gender classification, aligning with our findings. However, our use of Logistic Regression with optimized n-gram ranges improved generalization compared to their SVM approach.

The features that are used are Name, Gender, and Count. They are represented in a structure that has rows representing the name, gender and count of occurrences.

1. Categorical data (Encoding necessary)
    - Gender(M/F): Gender of the individual (Male/Female)
    - Name: The name of an individual

2. Integer data (Encoding necessary)
    - Count: Number of occurrences of a name with the specified gender
3. Target Variable
    - MajorGender(binary): Binary number determined by the gender with highest count for each name(0=M, 1=F)

Processing Steps

The data has no missing values, but we still used "dropna()" as a precaution. The data has to be carefully processed as it requires a particular way to transform it. One of the main features from the preprocessing steps is to use grouping to organize the data so it can be used with vectorization. To do this, we had to group "Name" and "Gender" by summing the counts for each name occurrence. Then we find the gender(M/F) with the highest count for each name. Once we have the greater occurrence, we know that is the most likely gender for that name and store it (MajorGender").

Next, we must use vectorization to use the TfidfVectorizer model with character n-grams(1-3). The reason we use this is to break down the string "Name" into 1-3 n-grams so that they can be converted into numerical features for easier machine learning model implementations. The ultimate goal is to convert the string into n-grams with numerical vectors that models can use to learn patterns and make predictions.

Model Development with Scikit-Learn

Models

Model 1: Logistic Regression

The Logistic Regression Model was our first pick for this database because we knew logistic regression is good for regularization. Since we had a binary value for Male and Female, we knew it would be a probably perform very well with what we were trying to achieve. Not to mention, logistic regression tends to be pretty efficient. This is crucial for our models, as we found out later, because it makes working on a large set of data a lot less time-consuming.

Model 2: Multinomial Naive Bayes

The Multinomial Naive Bayes models were our second pick because they work well with text classification. This is ideal for this dataset as one of its main features is the "Name"

string text. It particularly worked well with the TfidfVectorization after some minor tweaking on the vectorization data.

Model 3: K Nearest Neighbor

The K Nearest Neighbor model was our last choice and this was mainly because we wanted to see how it compared against the other 3 models. Objectively, it didn't quite seem like a good model for our database as it works pretty slow with big datasets and is impacted by dimensionality.

Validation Techniques

Originally, we had our testing/training splits at default 80 training and 20 testing. This had to be reduced as we adjusted the code for the hyperparameter tunning and model comparisons. The reason for this was because we kept running into issues with the K-NN model tunning. Ultimately, we had to reduce the training to 50 so that the K-NN model could actually finish running since it was taking HOURS to finish.

For the tuning, we used GridSearchCV with a 5-fold for cross validation. This helped to optimize the model parameters and get a more consistent outcome. Initially, we tested the parameters separately to make sure they each performed as intended. This proved to be very wise as we were running into a lot of errors, especially with the K-NN model. We had to keep adjusting other parts of our code and add/delete certain parts in order for the K-NN model to run decently. The logistic regression model took about 1 minute to complete, while the bayes took a bit less. The K-NN model was still running for over an hour and even crashed a couple times before we eventually tuned it to about 10 minutes.

Hyperparameter Tuning & Model Comparison

During our investigation, we discovered the model with the highest accuracy was the Logistic Regression model with an accuracy of ~86%. This was almost 10% better than the second highest model and a little over 10% better than the lowest model. It swept scores across the board by almost 10% in all other scoring metrics with its highest being the ROC-AUC score of 0.93.

	Accuracy	F1-Score	Precision	ROC-AUC
Logistic Reg.	0.8601	0.8883	0.8890	0.9280
K-NN	0.7703	0.8228	0.7965	0.8251
Naive Bayes	0.7528	0.8108	0.7792	0.8241

Logistic Regression

For the logistic regression, the default parameters were already pretty decent. But once we applied the grid search, we were able to pinpoint the exact best tuning for it. It turned out the best 'C' value was 10, and 'penalty' was 'l2'. This was likely because it was more improved generalization of the data. Since the score was for accuracy and ROC-AUC were pretty high, it meant that the data was strongly separated into their respective class. The main aid in its success was likely due to the optimization of dimensionality and how well it managed the sparse matrix.

K-NN

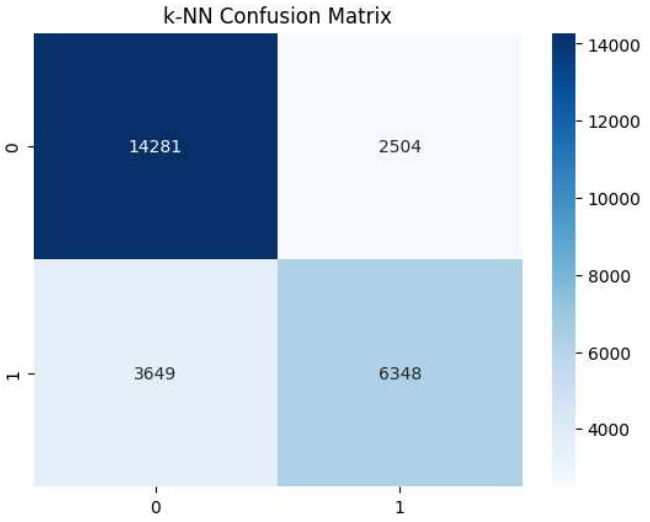
For K-nearest neighbor, the initial performance was pretty bad because it didn't even finish running. The amount of RAM it was consuming was a lot and sometimes it would even cause the session to crash. There was obviously a lot of tunning that had to be done to this model. Ultimately, we had to reduce the training size to 50% and reduce the number of k we wanted to test in the grid search. This ended up being a big help as it cut the run time massively and gave us actual results. The best parameters were k = 5 and 'uniform' for weights.

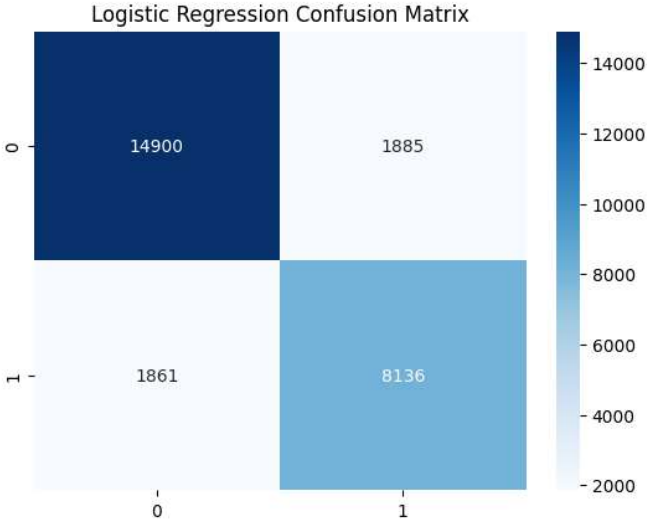
Multinomial Naive Bayes

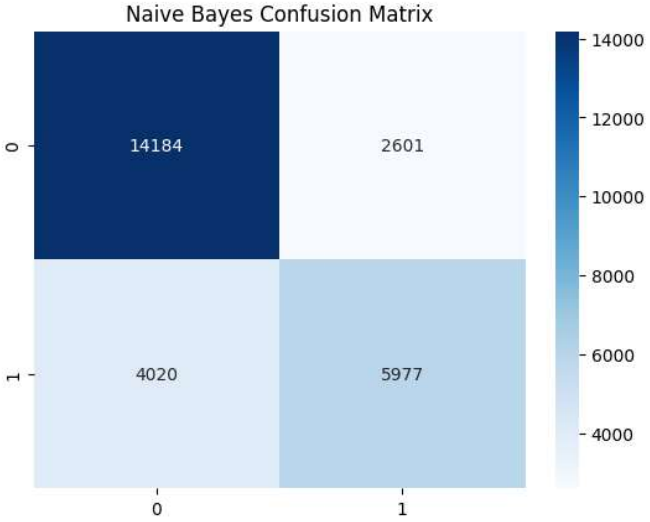
For multinomial naive bayes, this model crashed on us at the beginning because we forgot to use the multinomial model for the vectorization. After fixing this minor issue, the naive bayes model performed the fastest. It took the least time to run even though it didn't have the best scores. Its best parameter was an alpha value of '0.1'. Its low scores were likely due to its n-gram assumptions for names and their independence.

Confusion Matrix

The confusion matrices for each of the models are displayed below. The results are in line with the scores we got along the 4 different scoring metrics. Logistic regression leads the way with the best scores across the matrix. Having the highest true positive and true negative scores as well as the lowest scores for the false negative and false positives. The K-NN and naive bayes models both performed similar in some instances. Overall, it seems like the K-NN models performed better than the naive bayes in most sections.


k-NN Confusion Matrix
14281
2504
3649
6348
0
1
0
1
14000
12000
10000
8000
6000
4000


Logistic Regression Confusion Matrix
14900
1885
1861
8136
0
1
0
1
14000
12000
10000
8000
6000
4000
2000


Naive Bayes Confusion Matrix
14184
2601
4020
5977
0
1
0
1
14000
12000
10000
8000
6000
4000

Dataset 2: Estimation of Obesity Levels Based On Eating Habits and Physical Condition

Overview

This dataset contains 16 features of data about the eating habits and physical condition of individuals from Mexico, Peru, and Colombia. The target variable for this dataset is categorical, and measures obesity level in 7 distinct categories: Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II, and Obesity Type III.

The features of this dataset are comprised of 4 Categorical, 6 Continuous, 2 integer, and 4 Binary data types.

1. Categorical data (Encoding necessary)
    - Gender: Gender of the individual (Male or Female)
    - CAEC: Describes how often an individual eats between meals (no, sometimes, frequently, and always)
    - CALC: Describes how often an individual drinks alcohol (no, sometimes, frequently, and always)
    - MTRANS: Describes the type of transportation the individual uses and is measures (Public_Transporation, Walking, automobile, motorbike, and bike)
2. Continuous data (no encoding necessary)
    - Age: Age in years
    - Height: Height in meters
    - Weight: Weight measures in kilograms
    - NCP: Number of main meals consumed daily
    - CH20: Amount of water consumed daily
    - FAF: Frequency of physical activity
3. Integer data (no encoding necessary)
    - FCVC: Frequency of vegetables consumed in meals
    - TUE: How much time a person spends on technological devices and is measured in hours.
4. Binary data (encoding necessary)
    - family_history_with_overweight: Whether the individual has a family history with obesity.
    - FAVC: whether an individual regularly eats high calorie food
    - SMOKE: Whether an individual smokes
    - SCC: Whether an individual monitors their calorie consumption
    - All of the binary data types are measured with ‘yes’ or ‘no’

Source: [UCI Repository, https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition]

Preprocessing Steps

This dataset contains no missing values but as a precaution *.dropna()* was used on the dataset to drop any data with missing values. I chose to drop any data that was missing since there was no way to estimate what values could be since every individual is different

The Categorical data types and Binary data both require encoding. I used label encoding for CAEC, CALC, and NObeyesdad since these data types follow normal ordering meaning that each category follows a progressive ranking. For all the Binary Values and Gender, I used label encoding as well since all these values needed to be changed into zeros and ones. I used One-Shot encoding for MTRANS since MTRANS has no natural ordering.

For normalization I used Min-Max scaling since the ranges of each data types of values were different (e.g. Height, Weight, and Age).

Model Development with Scikit-Learn

Estimation of Obesity: Machine Learning

Model 1: Logistic Regression

Logistic regression was chosen as the first machine learning algorithm for this dataset due to logistic regression being well-suited for categorical target variables. In addition, logistic regression works well with mixed data types which this dataset contains. Initially, I ran logistic regression with all default parameters since I wanted a baseline for judging logistic regression performance. Then, after collecting performance data about the logistic regression performance on default parameters I used grid search to find the best hyperparameters for Logistic regression. For the hyperparameters, I tested the regularization strength, the type of regularization, and the optimization algorithm. Grid search found that a regularization strength of 10, lasso regularization, and saga for the optimization algorithm to produce the best accuracy.

Model 2: KNN-neighbors

KNN was the second machine learning algorithm used for the predicting obesity dataset due to KNN performing very well with classification and KNN allowing for much flexibility with datasets. The only parameter I set for the default KNN algorithm was n_neighbors which I gave a value of 7. I chose 7 since the dataset was quite large and many data types were involved, I didn't want any outliers to affect the performance of my algorithm. After collecting all the performance data from KNN I then ran a grid search to find the best hyperparameters for the algorithm. I tested different values for n_neighbors, weights, algorithm, and p (power parameter). The best hyperparameters for KNN was for the algorithm to be set to auto, n_neighbors to 3, p to 1, and the weights to distance.

Model 3: Multilayer Perceptron

For the third machine learning algorithm, I chose MLP since MLP performs better on larger sets of data. I had 1800 instances input into the training set which was a good size for the MLP algorithm to analyze. After running MLP with default settings and collecting the data, I ran a grid search finding the best hyperparameters for hidden_layer_sizes and activation. Originally, I tried more parameters with more values, but MLP is very computationally expensive and took too long to load on my computer. Nevertheless, MLP produced the best results out of any of the machine learning algorithms.

Validation Techniques

I used a train-test-split for dividing up my dataset into 3 groups. The training set was set at 80% and the test and validation set were set at 10%. Since my dataset had over 2000 instances of data, I thought this split would be enough to give my training algorithms enough instances to be well-trained and for there to be enough data points to properly test my algorithm. I followed a general pattern for my three algorithms by running a default case and collecting the accuracy of default case on the training, testing, and validation split. Then, I tuned the algorithms by selecting the best parameters using the grid search algorithm. Afterwards, I did the same test I did to the default algorithm on the tuned algorithm to see if there were any differences. In my code, I used cross-validation to make sure that I was getting an accurate representation of the performance of my dataset by setting cv equal to 5 in the grid search algorithm.

Hyperparameter Tuning & Model Comparison

Hyperparameter Optimization

For hyperparameter optimization, I used the grid search algorithm since I had a good understanding of how it worked and how to use it for each of the machine learning algorithms. For each of the machine learning algorithms, I selected as many parameters as I could, each containing at least 2 or more possible options. However, due to MLP being very computationally expensive, I had to lower the number of parameters I had originally intended to use. For the scoring method for grid search, I chose accuracy since the goal of my project was to use machine learning algorithms to correctly classify the level of obesity of the individual based off the data provided in the dataset.

Comparison of Models

Untuned Performance: Logistic regression -> KNN -> MLP

Training set: 74.51, 84, 99.94

Testing set: 72.17, 79.25, 93.37

Validation set: 72.64, 79.25, 94.34

For the untuned versions of each model MLP had the highest accuracy of 99%, followed by KNN with a score of 83% accuracy, and lastly logistic regression with a score of 74%. MLP scored significantly higher than KNN and logistic regression, this might be due to the fact that my dataset features a more complicated relationship between each feature that MLP is suited to recognize. Logistic regression works by creating a linear relationship between the features and target variables which in the case of my dataset does not work well. KNN doesn't analyze patterns in data and with so many feature relationships KNN will miss since it only looks at the distance in between points.

Tuned Performance: Logistic regression -> KNN -> MLP

Training set: 96.5, 100, 99.82

Testing set: 92.45, 86.32, 97.17

Validation set: 95.75, 85.38, 94.33

For the tuned version of the algorithms, Logistic Regression and KNN saw massive improvements in accuracy while MLP stayed mostly the same. Logistic Regression saw the biggest improvement in accuracy, this is due to the regularization strength being set to a bigger value causing features to have more weight to them. Lasso regularization and saga optimization algorithms definitely had a hand in improving the accuracy of the dataset as

well by zeroing out irrelevant feature coefficients. Overall, MLP still performed the best but it did have the longest run time out of all the machine learning algorithms,

Precision

Models -> Obesity Levels	LR (Untuned)	LR (Tuned)	KNN (Untuned)	KNN (Tuned)	MLP (Untuned)	MLP (Tuned)
0	0.76	0.94	0.76	0.94	0.97	0.97
1	0.65	0.96	0.86	0.85	0.89	0.97
2	0.76	0.97	0.73	0.86	0.93	0.97
3	0.83	1	0.92	1.00	1.00	1.00
4	0.91	1	0.97	0.97	1.00	1.00
5	0.45	0.67	0.61	0.68	0.79	0.88
6	0.43	0.85	0.61	0.62	0.94	1.00

All of the machine learning algorithms were much more precise near the center and beginning of the obesity classification range yet at the of the obesity classification range, with MLP being the most precise. After tuning the models, their precision numbers increase meaning that the algorithm were getting much better at predicting positive values without overshooting the target variable.

Prior Work Review

SOURCE: https://drpress.org/ojs/index.php/HSET/article/view/28659/28140

This study analyzed the main factors influencing obesity using the estimation of obesity dataset. They analyzed each feature of the dataset independently so that they could see which factor had the greatest impact on obesity levels. Logistic regression was used for the machine learning algorithm due to its interpretability and it's usefulness if categorizing both continuous and categorical variables. In their study, they used a confusion matrix and a ROC curve to judge the efficacy of the model in differentiating between disparate obesity

categories. The conclusion of their study found that physical activity, vegetable intake, and family obesity levels have a great impact on the obesity level. They noted that the model struggled with differentiating between closely related obesity levels.

In my project, Logistic Regression struggled with predicting values that were on the extreme ends of the obesity level scale. In the center of the obesity scale logistic regression seemed to perform the best. The study's author did use ROC curves which I have not used which may explain the difference in our answers. Overall, I found Logistic Regression to be very accurate only after its been tuned with Grid Search. The accuracy of my logistic regression model increased from 72% to 92%.