Statistical Learning - Final Project

Jannik Wirtheim

Introduction

The favorite fruit of Germans is the apple (Janson, 2017). A representative survey found that 79% of Germans eat apples occasionally, followed by bananas (78%) and strawberries (77%). Globally, apples rank fourth in production, with approximately 96 million tons in 2022 (The Science Agriculture, 2024). When buying apples, three-quarters of Germans prioritize the country of origin, and about half consider organic quality. Musacchi (2018) found that fruit quality is a dynamic concept shaped by customer needs. Nearly all apple quality characteristics can be measured or classified. Beyond vision-based factors like size and color, eating quality includes sweetness, acidity, and ripeness is important. Understanding these factors not only strengthens consumer trust and their willingness to rebuy, also companies benefits by reducing food waste through better classification of profitable apples.

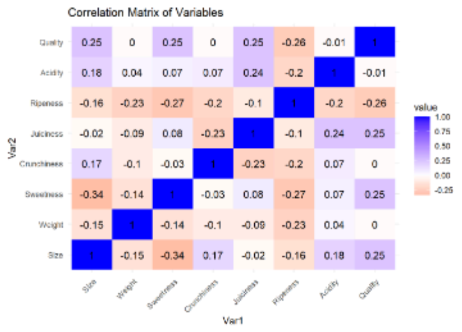
Over time, both customer perceptions and the methods for investigating problems have evolved. Traditional statistical approaches like Regression and Decision Trees can strongly identify patterns in data. But in comparison to Deep Learning (DL) this performance might suffer. However, this heavy approaches aren't often necessary for simpler tasks like apple quality classification. Due to time complexity, computational inefficiency, and lack of interpretability, traditional methods may be more suitable for easy tasks. To compare these models, the open-source Apple Quality dataset by Nidula Elgiriyewithana on Kaggle is perfect (Elgiriyewithana, 2024).

Analysis

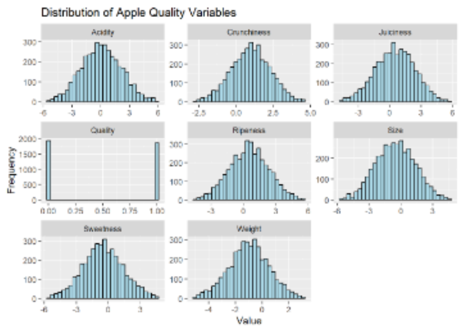
As Musacchi (2018) described, most metrics for classifying apple quality can be quantified. This dataset contains various apple characteristics, including optical features like size and weight, but primarily focuses on perception-based metrics such as sweetness, crunchiness, juiciness, ripeness and acidity. All are encoded as numeric data, while 'Quality' is a categorical variable. In this class 'good' indicates that the apple meets consumer expectations, while 'bad' means it does not. The data is generously provided by an American agriculture company.

According to the Kaggle Data Card, the data has been cleaned and scaled, but further preprocessing is needed. As the saying goes, “Garbage in, garbage out”, some preprocessing steps were applied. First, missing values were checked. Only one row contained NA values and was removed. The dataset was also examined for duplicates, but all rows were unique. Next, the target variable “Quality” was encoded as binary for easier model handling, and “Quality” and “Acidity” were converted to numeric values, ensuring the dataset contained only numerical data. To enhance robustness and generalizability, continuous variables were analyzed for outliers. These often do not represent the broader population but can impact model performance. To visually detect their influence, Boxplots were used. The interquartile range (IQR) marks potential outliers as dots beyond 1.5 times it’s size (see Code “Exploratory Analysis”). A total of 231 outliers were detected. Given the dataset size (n = 4.000), dropping them could improve model performance while the loss of information is negligible. After dropping the A_id column feature selection was done. The final dataset included seven numeric predictors and one binary target variable. It is nearly balanced (“bad” = 1.928; “good” = 1.862).

Interpreting the descriptive statistics of individual values is difficult due to their standardization. The mean of the target variable (μ = 0.49) reflects the balanced distribution. Plotting the dataset shows that all independent variables are approximately normally distributed. The correlation matrix shows only weak positive and negative correlations. “Quality” correlated weakly with Size (r = .25), Sweetness (r = .25), and Juiciness (r = .25) but negatively with Ripeness (r = -.26). Size and Sweetness had a stronger negative correlation (r = -.34), as did Sweetness and Ripeness (r = -.27).



(a) **Figure 1:** Correlation Matrix of Variables



(a) **Figure 2:** Histogram of Variable Distribution

Methods

To “describe the models architecture and training approach” I decided to not tune hyperparameters automatically. Since the plain model already performs well, only concepts covered in the lecture were used to keep the Deep Neural Network (DNN) within scope.

To address the binary classification problem, a fully connected feedforward DNN was built using seven numerical predictors as input features to classify the two possible outcomes. The network consists of multiple hidden layers to make it "deep", sufficiently transform the input data and learn meaningful patterns. The input layer takes the seven continuous predictors. This transitions into the first hidden layer (HL1) with 128 neurons, followed by a ReLU activation function, which introduces non-linearity to improve learning. Each subsequent layer applies ReLU after reducing the number of neurons by half, compressing the data progressively until the final output layer, which consists of a single neuron. For binary classification, a sigmoid activation converts outputs into probabilities. Predictions above 0.5 are 'good,' otherwise 'bad'. The model was trained using the Adam optimizer, which is a well-known optimized gradient descent algorithm dynamically adjusting learning rates and utilizes momentum for faster convergence. Since this is a binary classification task, the Binary Cross-Entropy (BCE) loss function was used to measure how well the model differentiates between the two classes. To ensure robust validation during training, 10-fold cross-validation (CV) on the training set was applied. In this method, the model was trained on ten different subsets of the initial dataset. At each of the ten iterations, nine folds were used for training, while one fold was used for validation. Regularization techniques were implemented to prevent the model from overfitting or underfitting. Overfitting occurs when the model performs well on training data but poorly on unseen data. In contrast, underfitting means the model fails to capture the complexity and patterns in the data, leading to poor predictive performance. Dropout prevents the model from relying too heavily on specific neurons, improving generalization. Initially at 0.4, with a decrease by 0.1 per layer. L2 Regularization penalizes large weights, reducing the influence of less relevant input features (see Table 1). Early Stopping halts training if no improvement occurs for a set number of epochs. To evaluate the performance of the models, two metrics were used. On the one hand, accuracy, which measures the proportion of correct predictions, and on the other hand, AUC (Area Under the ROC Curve), which assesses the model's ability to distinguish between the two classes. Additionally, Loss was monitored to measure the model's overall error during training and validation.

Changes were made through trial and error by evaluating each model's training performance to adjust parameters. Model 1 showed stable performance but underfitted due to short training. Extending training to 100 epochs (Model 2) improved predictions on training data but worsened validation performance, indicating overfitting. To counter this, Model 3 introduced Early Stopping and L2 Regularization. Initially, performance dropped, but tuning these parameters (Model 4) improved results. Though Early Stopping never triggered due to a low epoch limit. Increasing epochs in Model 5 finally yielded the best results on both training and validation sets. The table below summarizes the tuning configurations:

Table 1: Manual Model Tuning [1]

Model	HL	Epo	BS	LR	L2	ES	P	MinD	Train Acc (%)	Train AUC (%)	Vali Acc (%)	Vali AUC (%)
1	3	50	32	0.0001	-	-	-	-	91.6	95.2	87.7	91.6
2	3	100	32	0.0001	-	-	-	-	98.7	99.8	93.2	95.2
3	4	100	32	0.001	0.0001	15	5	0.01	88.1	95.0	87.2	94.3
4	4	50	32	0.001	0.001	100	10	0.001	92.4	97.5	91.8	97.0
5	4	100	32	0.001	0.001	92	10	0.001	93.9	98.2	94.5	98.3

To assess whether a DNN was necessary, a simpler K-Nearest-Neighbor (KNN) model was built. This model assigns a data point to a class based on local probability. The key hyperparameter k determines the number of closest neighbors considered for classification. To find the optimal k, Leave-One-Out-CV on the training set was applied. Similar to the DNN's CV, the model was trained on multiple folds, but here, each iteration used a single observation as the validation set while the rest served as training data. This approach identified k = 10 as optimal (see Code "Task 2 - K-nearest-neighbor").

Results

The best DNN (Model 5) was selected based on highest AUC due to its relevance in cluster separation. The resulting model achieves an optimal balance between generalization and performance, as seen in the loss curves (see Code 'Task 1 - Deep Feedforward Neural Network'). Early stopping (92 epochs) and regularization prevented overfitting. Applying the model to the test set (n = 790) led to comparable results. To evaluate its performance, several metrics were considered alongside Accuracy and AUC. Recall measures how many actual positive cases were correctly identified, while Precision checks how many predicted positive cases are actually correct. The F1-score represents the harmonic mean of both metrics. When classification aims to reduce food waste, Recall should be weighted more heavily to ensure that fewer good apples are wrongly classified as bad. Looking at the results, the DNN consistently outperforms the KNN across all metrics (see Table 2). The largest absolute percentage difference is observed in AUC (+11.77%), indicating that the DNN is significantly better at distinguishing between classes. Additionally, notable improvements are seen in Accuracy (+8.74%), Precision (+7.77%), F1-score (+8.87%) and Recall (+10.0%). This high absolute difference in Recall highlights the importance of the DNN in reducing food waste. The ROC curve for the DNN is tightly clustered near the top-left corner, while KNN's curve appears more linear, suggesting that its classification certainty is lower (see Code "Evaluation - Deep NN").

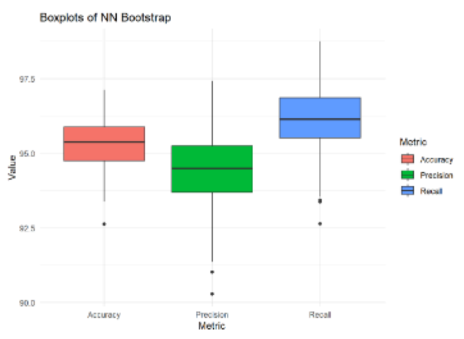
[1] HL: Hidden Layers, Epo: Epochs, BS: Batch Size, LR: Learning Rate, L2: L2 Regularization, ES: Early Stopping, P: Dropout Rate, MinD: Minimum Delta for Early Stopping, Train Acc: Training Accuracy, Train AUC: Training AUC, Vali Acc: Validation Accuracy, Vali AUC: Validation AUC

Table 2: Result Comparison

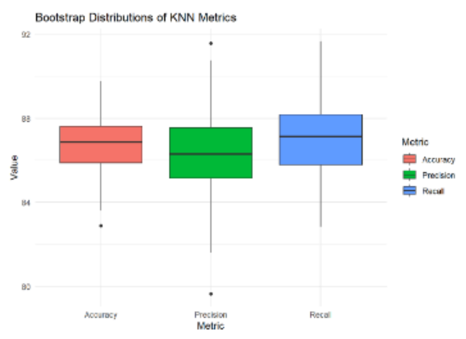
Model	Accuracy	Precision	Recall	F1-Score	AUC
Neural Network	95.70%	94.28%	97.18%	95.71%	98.73%
K-Nearest Neighbor	86.96%	86.51%	87.18%	86.84%	86.96%

Uncertainty measurements provide insights into the variability of results. Comparing the 95% confidence intervals (CI), we see that none of them overlap when comparing both models, reinforcing the statistical significance of the DNN (see Code "Evaluation"). Additionally, the 95% CI for the DNN is narrower, indicating more stability, whereas KNN shows greater variability across metrics. This is also reflected in the boxplots, where the spread of DNN metrics is significantly narrower than that of KNN (see Figure 3 & 4). The density plots further confirm this. The DNN's peak density is sharper, while KNN's distributions are more spread out, indicating that its predictions fluctuate more depending on the training sample.

From a practical perspective, whether DL is necessary depends on the trade-off between performance and computational efficiency. The DNN demonstrates significantly better performance but requires more computational resources for training and deployment, while also suffering from lower interpretability. In contrast, KNN is computationally simpler and easier to interpret. But its lower performance might be unacceptable in scenarios requiring high classification accuracy. Given the substantial performance improvements across all metrics and the significantly lower uncertainty, DL is a justified choice. However, if interpretability is key, simpler statistical learning models should be preferred.



(a) **Figure 3:** Uncertainty Boxplot of DNN



(a) **Figure 4:** Uncertainty Boxplot of KNN

Reflection

Building my second DNN with Torch (and first in R) felt smoother, with faster progress. It further deepened my understanding of DL structures, with several "aha" moments, especially around Dropout and Adam. When manually tuning the model, I realized the importance of regularization to avoid overfitting. Another learning was, that performance isn't always the key. Often it's a trade-off between time, cost, study complexity and accuracy. Hardware limitations were a challenge. My old laptop struggled with Torch and frequently crashed, so I used distributed computing to run tasks remotely on my high-performance computer at home. Even then, training and tuning took time, requiring me to complete other tasks on the side (or take a coffee break). Writing a non-technical report was another challenge, particularly when explaining DNN architecture and its specific terminology to a non-technical audience. For future work, I would automate hyperparameter tuning with Grid or Random Search and explore concepts beyond the lecture, such as Learning Rate Schedulers. Experimenting with different architectures would also be a priority, especially if the plain DNN doesn't immediately achieve ~90% accuracy. Additionally, I would compare the DNN's performance against more simpler classification models, like Decision Trees or Logistic Regression.

References

Elgiriyewithana, N. (2024). *Apple quality.* Kaggle. https://www.kaggle.com/dsv/7384155

Janson, M. (2017). *Lieblingsobst der Deutschen im Jahr 2017.* Retrieved February 8, 2025, from https://de.statista.com/infografik/12081/lieblingsobst/

Musacchi, S., & Serra, S. (2018). Apple fruit quality: Overview on pre-harvest factors. Scientia Horticulturae, 234, 409–430. https://doi.org/10.1016/j.scienta.2017.12.057