Supervised Learning Report

Skyler Ebelt
Georgia Institute of Technology
GT ID: 904077010

I. AI Use Statement

In order to complete this project I allowed AI to modify several components of this project, I marked each usage in the code. I used AI generated code via Microsoft Copilot which was featured on a search via Microsoft Edge. This code simply helped me convert Float64 to Float32. Next I used Claude Sonnet 4 to create an easily reusable method to track ram usage, each instance of this code is marked as being AI generated. I worked to understand every atrribute I implemented with the help of AI.

II. Introduction

A. Datasets Utilized

This analysis was completed using two datasets. One dataset focuses on hotel booking demand and contains relevant features for analysis and prediction [1]. The second dataset focuses on car accident data, this data garners much more observations than that of the hotel data [2]. Both datasets provide the opportunity to derive models and subsequently insights into a wide variety of trends. For me personally, I much prefer classification over regression, I used both datasets to make predictions of classifications. With the hotel data, I sought to classify whether or not a booking would be canceled. In the case of the accident data, I worked on classifying severity of car accidents. Insights into both of these datasets could be useful to business practitioners in these industries. For an engineer working in the humanities space, predicting which hotel bookings may be canceled could be useful in launching and maintaining campaigns to combat this trend. Using these accident models to model for severity could be useful for informing decisions around insurance rates in a car insurance setting. It may even be useful in informing vehicle users by being able to track which conditions produce the highest severity accidents.

B. Objectives

The objectives of this project where to establish four classification models utilizing the two aforementioned data sets (hotel data [1], accidents data [2]). The purpose of this was to compare model performance between every instance. This can range from, but is not limited to, measuring how the models performance between datasets vary. How the models using the same data can vary in performance. Additionally how each model displays bias/variance trade offs. Metrics were calculated to answer and inform on these queries. This exercise informs the efficacy of each model, their respective use case, their strengths and weaknesses. This comparative analysis gives a direct view into each of these topics which is the main objective for this paper.

III. Data & Methodology

A. Hotel Data

The hotel dataset[1] features approx. 119,000 observations with 32 features. Columns which may incur leakage were removed. These were comprised of the columns "reservation_status" as well as "reservation_status_date". The target selected for this dataset was the feature "is_canceled". This is a classification problem, as a result all models built pertaining to this dataset were stratified on this target, and were designed to make predictions to classify observations which the target variable were unknown to the model. The very next step after dropping columns which present leakage was properly splitting the data into processed training and test splits to ensure no leakage could occur in modelling. Further preprocessning involved converting Float64 variables to Float32, as well as removing high cardinality object variables. Due to runtime constraints, I opted to go for a very tight handling of cardinality, designating that 20 or greater unique values in column would be considered high cardinality, and subsequently would be removed from one hot encoding. High cardinality object variables were then encoded using target encoding.

B. Accidents Data

The accients dataset[2] features approx 7,000,000 observations with 46 features. This dataset possesses a far greater amount of information and as a result has required a lot more special attention in ensuring these models function as intended. Columns which presented leakage issues were dropped, these were comprised of variables which included post-outcome data such as "end_time" and "weather_timestamp". The same process which occured during preprocessing the hotel data, was followed for the accidents data. High cardinality was defined in the same way, opting for target encoding with object columns which featured 20 or more unique values. Once again Float64 variables were converted to Float32.

C. Evaluation

Accuracy, PR-AUC (for imbalance), and ROC-AUC were selected to evaluate and compare model performance. Cross model evaluation occured after each model was established and verified to compare and contrast how models results differed between eachother. Additionally, this provides a

view into how the data being used in modeling can affect the output of each model. This code establishes seeds and random states which I associated with my GTID (4077010), this was the strategy used in another class I took, I liked this and decided to implement it here. Finally, I really struggled with runtime during this project, frequently I would notice using 100% of my CPU during operations (especially in kernel svm). I took the liberty of heavily sampling some cases while (hopefully) following the guidelines. The most prominent case of this can be found in the case of both kernel svm models. While time train seems to remain low, performing grid searches and calculating learning curves proved to best my machine, leading to many headaches. I am currently running Windows 10 Pro, with 32 GB of ram, and an Intel Core i7 9700 @ 3.00 GHZ. Additionally I am working out of Jupyter notebooks, with my conda environment being initialized on n.2 NVME drive. I would like to add a personal note, I was very confused about how to go about ensuring these ran in accordance with the guidelines and decided I must have to sample further. After tightening parameter grids with little improvement, I followed the instructions to halve the sample until finding a suitable runtime, this resulted in very small samples. In summation, after this machine serving me well for 5 years, it may be time to look into an upgrade, please excuse my small samples.

IV. Decision Trees

A. Implementation

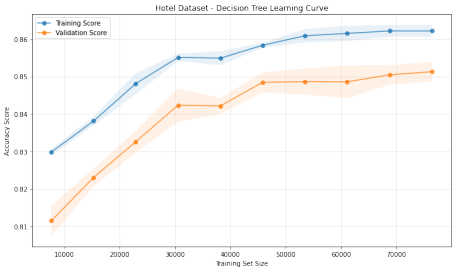

Fig. 1. Hotel DT Model Learning Curve

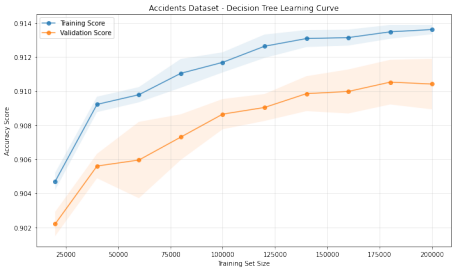

Fig. 2. Accident DT Model Learning Curve

For this implementation of decision trees, Gini based impurity criterion was selected. Gini provides comparable results within the predictive capabilities of decision tree algorithms. Unlike similar impurity criterion however, Gini requires fewer logarithmic calculations during tree construction [3]. Because of this, I opted to select this methodology due to computational constraints, with the goal of sampling less and hopefully keep parameter grids wider for a more thorough analysis. I opted for post-pruning via ccp_alpha in order to keep cost-complexity minimal.

The parameter grid for this model was kept wide, ensuring a well suited model could be derived from the established hyper parameters. max_depth was given values of [6,10,14,18], min_samples_leaf was given the values of [100,200], min_samples_split was given the values of [100,200,400], finally ccp_alpha was given the values of [0.0, 1e-4, 5e-4, 1e-3]. The only caveat here is that the hotel model included max_features values of ['sqrt', 'log2', 0.5], this was removed from the accidents model to ensure runtime was not excessive. These values were selected to establish a well generalized model, while keeping runtime within the provided constraints, the hotel model used a cv value of 5 while the accidents model utilized a cv value of 3. The nature of the accident data set required cutting back of cv folds and certain hyper parameters to conform to runtime constraints. Neither model in this instance required sampling.

B. Results & Analysis

TABLE I
Decision Tree Confusion Matrix: Hotel Dataset

	Predicted Not Canceled	Predicted Canceled
Actual Not Canceled	13489	1544
Actual Canceled	1893	6952

TABLE II
Decision Tree Confusion Matrix: Accidents Dataset

	Severity 1	Severity 2	Severity 3	Severity 4
Severity 1	4014	8521	906	32
Severity 2	1079	1182848	41559	5910
Severity 3	170	50299	209090	309
Severity 4	33	22734	2874	15301

The optimal hyperparameters identified through grid search were max_depth=18, min_samples_leaf=50, min_samples_split=100, max_features=0.5, and ccp_alpha=0.0 for the hotel dataset, while the accidents dataset achieved optimal performance with max_depth=10, min_samples_leaf=100, min_samples_split=100, and ccp_alpha=0. Looking into the complexity curves suggests that both models benefited from a ccp_alpha value of 0. This suggests overfitting had already been adequately addressed via other parameters with both models requiring no pruning in their final state. This shows evidence of a well generalized, and computationally affordable model.

Analyzing the learning curve output, it can be observed that the hotel dataset showed signs of mild overfitting with training accuracy above validation accuracy, with this trend slowly improving as training sizes increases. Validation accuracy topped out at approx. 85% while training accuracy reached approx 87%. The accident model showed similar

trends but with a smaller bias-variance gap suggesting that the larger access to data allowed for the model to become slightly more generalized.

Features importance for both the hotel and accident data revealed that encoded variables had the most impact on predictive capability. This would suggest the method of approaching encoding worked well in this instance as the model was able to generalize trends in these variables even after processing. Run time was sparse in both cases with hotel model requiring approx. 198 seconds to train and 0.008 seconds to predict, whereas the accident model required approx 352 seconds to train and 0.612 seconds to predict. Neither model demanded very much ram with each operation requiring less than 10mb of RAM. Hotel performance achieved 85.6% accuracy with ROC-AUC of 0.933 and PR-AUC of 0.904, with a prevalence of 37%. Accidents performance reached 91.3% accuracy with ROC-AUC of 0.942. These experiments outline the strength of decision trees, being able to be well generalized on a multitude of data, fast to implement, nearly instantaneous predict time, and very computationally inexpensive. A large drawback which can be directly observed in this experiment however, is the necessity for more observations. These results would suggest the hotel model would have benefited from access to more data, whereas the accident model featured high predictability likely due to the larger amount of information.

V. KNN

A. Implementation

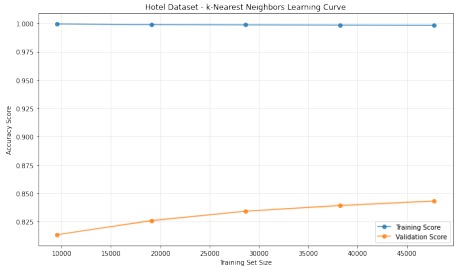

Fig. 3. Hotel KNN Model Learning Curve

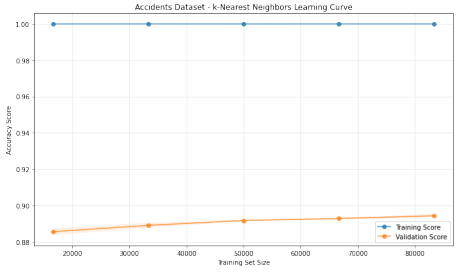

Fig. 4. Accident KNN Model Learning Curve

This implementation of KNN, as directed in the guidelines, utilized the brute algorithm with the Manhattan distance metric. Brute does well with high dimensionality and mixed feature types which is likely why we were instructed to use this method [4]. However, as I found when performing this analysis (and with some reading of the previously cited work) this method is highly exhaustive and incredibly computationally expensive. In an instance such as mine, where CPU performance is lacking, this KNN method seems to be bottle necked, resulting in slow train times, and computation times.

Hyperparameter tuning allowed the model to explore values of k from 3-101 for the hotel model and 5-21 for the accidents data set. This was done to tighten the model in light of the computational complexity, ensuring suitable runtime.

B. Results & Analysis

TABLE III
KNN CONFUSION MATRIX: HOTEL DATASET

	Predicted Not Canceled	Predicted Canceled
Actual Not Canceled	13721	1312
Actual Canceled	2045	6800

TABLE IV
KNN CONFUSION MATRIX: ACCIDENTS DATASET

	Severity 2	Severity 3	Severity 4
Severity 1	91	4	0
Severity 2	9663	289	6
Severity 3	628	1472	2
Severity 4	260	24	47

Complexity analysis showed that the optimal values of k for the hotel model were 21, and 11 in the case of the accident model. Analysis of the learning curve outlines how smaller values of k leads to a biased model, while larger values lead to more generalization. At k = 1 we can observe the model memorizing the data in the case of both hotel and accident models. Both models here seemed to work well with the amount of data provided, unlike the example of the DT hotel model. Runtime proved to be heftier in both models here, with the hotel model requiring 458 seconds to train and 35.2 seconds to predict. The accidents model showed a similar pattern of increased run time, requiring 580 seconds to train however only 11.4 seconds to predict. Modeling the hotel data showed an 85.9% accuracy with a ROC-AUC of 0.928 and PR-AUC of 0.909. Modeling accidents showed an accuracy of 89.6% on a reduced sample.

Due to the aforementioned limitation imposed by my machines CPU, I required sampling the accidents data further (used half of the 250,000 sample required in the guidelines as instructed). This enabled this model to go from timing out (I suspect kernels were dying due to CPU overload) to running in 10 minutes which is within the runtime constraint. Fortunately, and unsurprisingly, the hotel model did not require any further sampling. Both models used just shy of 5 GB of ram, again supporting my hypothesis of a CPU bottleneck. While running this code I kept my eye on my resource manager, KNN and SVM frequently maxed out my cpu usage, especially when calculating learning and validation curves.

Bias-variance tradeoffs can be easily seen, with lower values of k showing a tendency to fit too strongly to the data. Not generalizing trends but rather overfitting to the training data. Larger values of k show the model begin to reverse this creating more generalized models. Dimensionality proves to be difficult here as well. Due to the large amount of features in the accidents dataset , this can create too much noise for the model, reducing the meaning of each neighbor. Because of this distance weighting is essential for these models, as without this too many neighbors would contribute too much to each prediction. Essentially reducing predictive power. Due to an explosion within runtime, learning curves proved to be impossible for me to calculate, despite the heavy sampling.

VI. Linear & Kernel SVM

A. Implementation

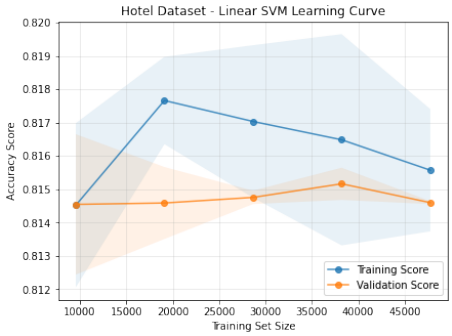

Fig. 5. Hotel Linear SVM model Learning Curve

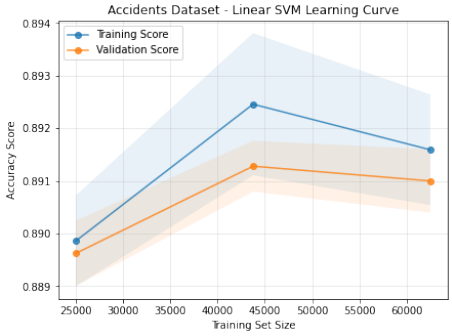

Fig. 6. Accident Linear SVM Model Learning Curve

For these models LinearSVC and RBF SVM models were established. Data was heavily sampled and learning curves were not able to be calculated, seeking to reduce computational load, ensuring the model would run within runtime constraints. Hyperparameter tuning focused on values of C to maximize margin and minimize classification errors. Additionally class_weight = 'balanced' was selected to handle imbalanced classes. Class weight functionality automatically penalizes any class which is inversely proportional to frequency, this allows the model to address and adapt to any imbalance.

Both SVM types saw heavy constraints in parameter grids, similarly the hotel model had a tight grid to try to adhere to runtime limits. Heavy sampling was required to ensure this code would even run (feedback here would be great) I sampled as instructed in the guidelines until finally settling on 12,500 observations for both RBF models. 25,000 samples for both models resulted in dead kernels after CPU overload, 12,500 records allowed the models to run very quickly. Describing this rapid change in performance between two similarly small sample sizes is frankly a little beyond me. Advice or feedback on how I could have better handled this would be greatly appreciated. I was able to use the whole hotel dataset but required halving the sample size required for accidents three times down to 125,000 records. For the linear hotel model, C values of [0.1, 1, 10] were used and [1, 10, 100] for the accidents model. Maximum iteration for both linear models were 10,000 and 5,000 for the linear hotel and accidents model respectively. This ensure convergence while keeping runtime down. Both RBF models were constrained to a C value of 2, this was done out of necessity due to the tendency of the poly models to overload my CPU. Runtime for linear models were 38 seconds and 120 seconds for hotels and accidents respectively, both used about 7.7 and 13 GB of ram respectively. RBF SVM saw a huge dip (very small sample) to 48 and 85 seconds on hotels and accidents respectively with both using only about 383 MB and 7 GB of RAM respectively.

B. Results & Analysis

TABLE V
Linear SVM Confusion Matrix: Hotel Dataset

	Predicted Not Canceled	Predicted Canceled
Actual Not Canceled	13719	1314
Actual Canceled	3116	5729

TABLE VI
RBF SVM Confusion Matrix: Hotel Dataset

	Predicted Not Canceled	Predicted Canceled
Actual Not Canceled	12836	2197
Actual Canceled	1683	7162

TABLE VII
Linear SVM Confusion Matrix: Accidents Dataset

	Severity 1	Severity 2	Severity 3	Severity 4
Severity 1	101	107	8	2
Severity 2	97	18444	1018	358
Severity 3	28	695	3447	33
Severity 4	4	275	53	330

Reviewing the model complexity curves outlines how regularization varies between datasets. It was found for the hotel data that the optimal c value was c = 1, whereas c = 0.1 showed underfitting and c = 10 showing overfitting. On the accidents data however, c = 10 proved to be the optimal selection, showing that a larger more complex data

TABLE VIII
RBF SVM CONFUSION MATRIX: ACCIDENTS DATASET

	Severity 1	Severity 2	Severity 3	Severity 4
Severity 1	129	58	20	11
Severity 2	1354	15215	1492	1856
Severity 3	74	375	3606	148
Severity 4	15	176	48	423

set benefited from less regularization. Comparing performance of linear models shows solid performance with 81.4% accuracy on hotel data, and 89.3% on accident data. Kernel SVM proved to be beneficial to hotel data, showing a slight improvement to 83.7%, but degraded the performance of the accident model reducing it to 77.5% accuracy. This may be attributed to the small sample failing to capture enough trends to properly generalize. It is also possible that the model simply did not/would not do a better job of generalizing this data.

Regularization effects can be observed within tuning values of c. Lower values of c created larger margins allowing the hotel model to properly generalize, whereas the accidents model required a larger c value. Likely due to more complex data. Feature scaling has been demonstrated in preprocessing and performance sensitivity. Significant performance differences showcases SVMs sensitivity in variations between the magnitude of features. The efficacy of the linear kernel might suggest that proper encoding and scaling allowed the model to identify and linearly separate relationships, rendering the use of the RBF kernel unnecessary. Using a class weighting of balanced seems to be effective based on models performance. These models however seem to struggle under the heavy load of the data provided, at least on my machine. Training times fast by comparison, requiring only 38 and 48 seconds for linear and RBF hotel models respectively, and 120 and 80 seconds for accidents linear and RBF models respectively. Both had near instant predict times. Where these models struggled (and eventually timed out killing the kernel) was in grid searches, it seemed this is where the bulk of the runtime was occurring. This leads me to believe, with the difficulty in adhering to runtime constraints, these models were not effective in capturing the trends of this data especially when compared to DT or NN.

VII. ARTIFICIAL NEURAL NETWORKS

A. Implementation

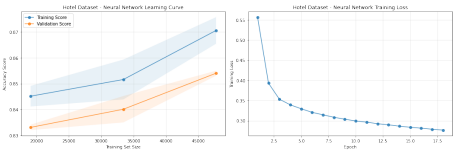

Fig. 7. Hotel NN Model Learning Curve & Training Loss

For this implementation stochastic gradient descent with early stopping was instructed to be used. L2 regularization was utilized in accordance with the guidelines. Learning rate was initialized as 0.1 and a batch size of 512 to try to allow for convergence while reducing noise. A patience of 3 epochs as instructed was implemented, which allowed for these models to adequately explore parameter space. Validation fraction was set to 10% in order to properly assess convergence of the models. Shallow models used hidden layer sizes of (200,200) for hotel data and (512,512) for accidents data whereas deep models used the values (150,100,75,50). While modeling accidents, the model was afforded 50 epochs to balance the time to train while enabling convergence to still occur.

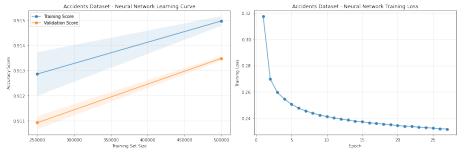

Fig. 8. Accident NN Model Learning Curve & Training Loss

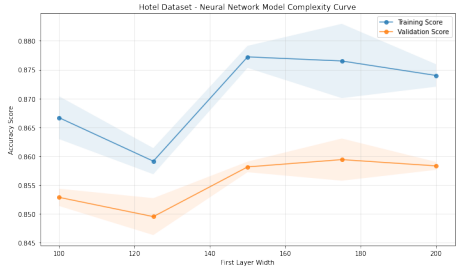

Fig. 9. Neural Network Hotel Model Complexity Curve Over First Layer Depth

B. Results & Analysis

TABLE IX
NEURAL NETWORK CONFUSION MATRIX: HOTEL DATASET

	Predicted Not Canceled	Predicted Canceled
Actual Not Canceled	13493	1540
Actual Canceled	1761	7084

TABLE X
NEURAL NETWORK CONFUSION MATRIX: ACCIDENTS DATASET

	Severity 1	Severity 2	Severity 3	Severity 4
Severity 1	96	109	13	0
Severity 2	13	19127	706	71
Severity 3	3	746	3436	18
Severity 4	1	346	53	262

Analyzing model complexity shows that dataset specific optimal architectures occurs, although do note that some hyper parameters were modified in the case of the accidents model to ensure runtime conformed to expectations. The hotel data seemed to prefer the deeper configuration, reaching 86.2% accuracy when doing so. The accidents model seemed to prefer the shallow model however, achieving a whopping 91.7% accuracy as a result. Because of this, it would seem the broader nature of the accident data benefits

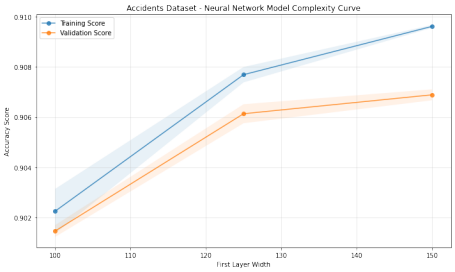

Fig. 10. Neural Network Accidents Model Complexity Curve Over First Layer Depth

TABLE XI
ALGORITHM PERFORMANCE SUMMARY

Algorithm	Hotel Dataset		Accidents Dataset	
	Acc.	ROC-AUC	Acc.	ROC-AUC
Decision Tree	85.6%	0.933	91.3%	0.942
k-NN	85.9%	0.928	89.6%	—
Linear SVM	81.4%	—	89.3%	—
RBF SVM	83.7%	0.912	88.9%	0.929
Neural Network	**86.2%**	**0.937**	**91.7%**	**0.958**

from a larger representational capacity. Time to train the hotel model required only 13 seconds, while the accident model required 210 seconds both models used only a little more than 5 GB of RAM. The hotel model required 18 epochs to converge whilst the accidents model required 27 epochs to converge. This provides further evidence of the complexity within the accidents data, likely stemming from multiclass data. Calibrating probability shows that neural networks have produced the most reliable estimations of confidence with ROC-AUC values of 0.937 for hotel data and 0.958 for accidents data. This describes strong discriminative capability. Early stopping proved to be highly beneficial in these models, stopping after having identified optimal points at which to stop. These results have produced smooth convergence curves without a loss of validation further showing the strength of these models.

VIII. CROSS ALGORITHM ANALYSIS

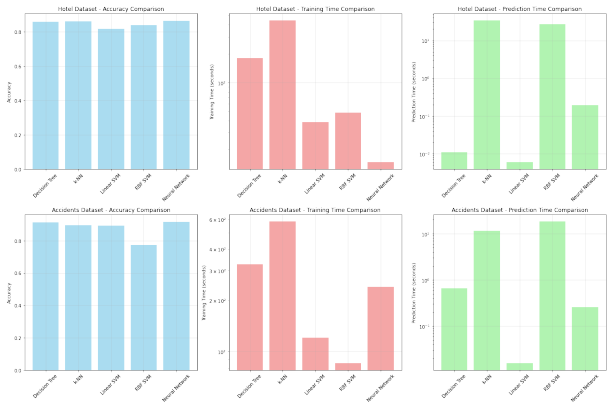

Fig. 11. Comparison of All Models

Upon considering Fig. 9, which depicts all models performance side by side, we can clearly see significant increases in performance across all models trained on accidents data. Another notable attribute of this table is that it can clearly be seen that neural networks outperformed all other models, but were followed close behind by decision trees. These models appear to be relatively quick to train and are capable of handling a larger amount of data as no additional sampling was required. KNN and SVM both were severely limited in performance, almost entirely from cutting sample sizes and parameter grids due to runtime constraints. DT and NN did really well to handle the large amount of data without chewing through too many resources, not only is the accuracy good but the performance in all regards really makes these models stand out here. Figures 10 & 11 outline how model complexity varies between datasets and hyperparameters.

IX. CONCLUSION

A. Type I & Type II Error

In the case of the hotel dataset, and its practical business use cases, false positives and false negatives can present difficulty in practice. Failing to identify a customer who will cancel their booking can result in the loss of revenue. Simultaneously, identifying an individual as potentially being at risk of a cancellation, and subsequently being exposed to campaigns, or outreach unexpected and unwanted by the individual can lead to a negative experience with the hotel. The current 0.5 probability threshold may not fully optimize business outcomes, and is intended to be as balanced as possible.

Accident severity miss-classification can create public safety implications. Currently, severity is often classified as being lower than the actual value. This means in real world applications proper resources may not be allocated to those in need of them as a result of this miss-classification. Understanding these drawbacks, further developing these models, and directly addressing these issues would be required for these models if they were to see any practical application in the real world.

B. Key Findings

Neural networks was the best performer across these analyses, enabling high predictability without being computationally over bearing. Of the four algorithms explored in this report, decision trees and neural networks offer similar results but with a few trade offs in the case of both. Decision trees did not have as high performance as NN, however it is much easier to interpret and tune. NN, again, has proven itself to be a higher scorer, however it is less interpretable and requiring much more fine tuning. SVM and KNN did not provide results worth the compute cost, trends in the data were not generalized by these models as well as they were by DT and NN

REFERENCES

[1] Hotel Booking Demand: Data: Hotel booking demand (H1/H2), Ant´onio, Almeida & Nunes (2019), Data in Brief 22:41–49, doi:10.1016/j.dib.2018.11.126.

[2] US Accidents: Data: US Accidents (since 2016), Sobhan Moosavi et al., CC BY-NC-SA 4.0, retrieved from Kaggle US Accidents.
[3] S. Tangirala, "International Journal of Advanced Computer Science and Applications," Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm, vol. 11, pp. 612–619, Nov. 2020. doi:10.14569/issn.2156-5570
[4] Halder, R.K., Uddin, M.N., Uddin, M.A. et al. Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. J Big Data 11, 113 (2024). https://doi.org/10.1186/s40537-024-00973-y