CS7641 Assignment 1 – Supervised Learning

Nian Liu
Georgia Institute of Technology
nliu319@gatech.edu

***Abstract*—This project implements 3 commonly used supervised learning models to perform classification tasks on 2 distinct tabular datasets, with the goal of elucidating each model's characteristics. We found that for a dataset that has distinct feature values depending on the class, all 3 models performed very well, but the training time and amount of effort needed for tuning differs significantly. On the other hand, the other dataset which has a complex decision boundary reveals performance differences among the models while also causing models to be more prone to overfitting. Overall, we have found that it is not always advantageous to apply models with more expressiveness. Rather, finding the right model tailored to the specific dataset can result in optimal performance while minimizing any costs associated with training and tuning.**

I. Introduction

Supervised learning is a class of machine learning techniques where the algorithm can learn from paired input – output data, generalize to predict unseen output data from input alone, and ultimately carry out tasks without explicitly being programmed to do so. They have become ubiquitous in our everyday life ranging from machine translation to object detection. This project explores several commonly used supervised learning models to perform classification tasks. Specifically, we implemented neural networks (NN), support vector machines (SVM), and k-nearest neighbors (k-NN) on 2 multi-class classification datasets. Each of these algorithms have their own advantages and disadvantages in terms of train/test performance, training and testing runtime, ease of implementation, and difficulty to tune. For instance, NNs are generally thought to be the most expressive in that it can model any input–output relation given the correct architecture. However, this typically comes at a cost of increased model complexity, longer training runtime, and more tedious hyperparameter tuning. SVM, on the other hand, is a simpler model but can also be ubiquitous given the proper non-linear kernel transformations tailored to the dataset. It typically runs faster with fewer hyperparameters to tune, and in most tabular data classification tasks can achieve similar performance compared to NNs. However, its training time scales with input data on the order of O(n^3), which makes it challenging to train on large datasets. k-NN differs from the other 2 models in that it relies on the data itself to form the decision boundary, rather than modeling it as a mathematical expression. It is potentially the easiest to implement among the 3 while retaining good performance. However, k-NNs are typically much more memory intensive and have longer inference times. The goal of this project is to compare and contrast these algorithms on the 2 chosen datasets, each of which having distinct characteristics. Analysis will be provided based on trends observed in each model's learning curves and validation curves, as well as any other findings that illustrate the models' unique attributes. All data processing and analyses were performed using standard packages from scikit-learn [1]. Our key learning is that it is important to analyze each model's pros and cons in the context of the specific dataset. In the following sections, we will first introduce the 2 datasets, mention any preprocessing that was carried out, and form rudimentary hypotheses on how each model will perform on them. Then we will describe the general workflow applied to all analyses, which includes hyperparameter tuning as well as model training and testing. Finally, we will present the results from each model on each dataset and compare the results.

II. Description of datasets

A. Dry bean

The dry bean dataset was originally published in [2], where 16 distinct geometric features were automatically extracted from dry bean images and used to classify 7 different types of beans. The dataset contains 13,611 entries and was split into 80% training data and 20% test data using a stratified method to ensure the train-test data had similar class distributions. The test set was only used during final testing and was not involved in any model tuning or training. For the training data, features were standardized by subtracting the mean and scaling to unit variance. The same mean and scaling factor were then applied to the test data, such that we rely only on the training data for preprocessing. The dry bean target classes were encoded into integer labels 0-6 in order to feed into each model. Fig. 1 (left) and Fig. 2 summarizes the distribution of the classes and normalized features, respectively. As shown in Fig. 2, there is good separation in nearly all feature values among the different classes, and thus we expect the 3 models tested in this report to have good predictability on this dataset with minimal difference. As a result, we hypothesize that SVM and k-NN are better suited for this dataset when factoring in other considerations such as training runtime.

B. Wine quality

The wine quality dataset was originally published in [3], where 11 distinct features derived from physicochemical measurements of wine were used to classify wine quality. An integer score between 3 and 8 for quality was assigned to each wine, and was determined by sensory assessors (i.e. through wine tasting). The original dataset contained both red and white wine, but were analyzed separately due to their

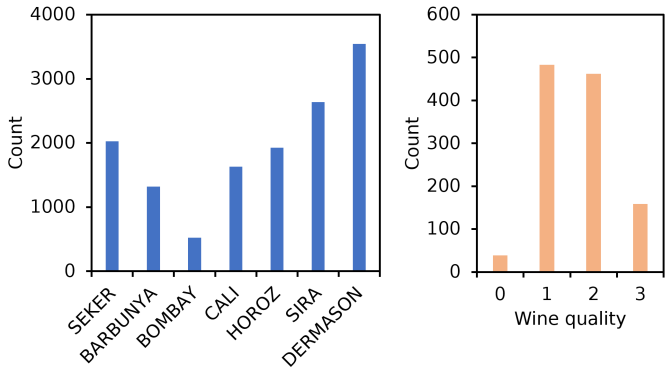

Fig. 1. Class distribution of the dry bean dataset (left) and the wine quality dataset (right).

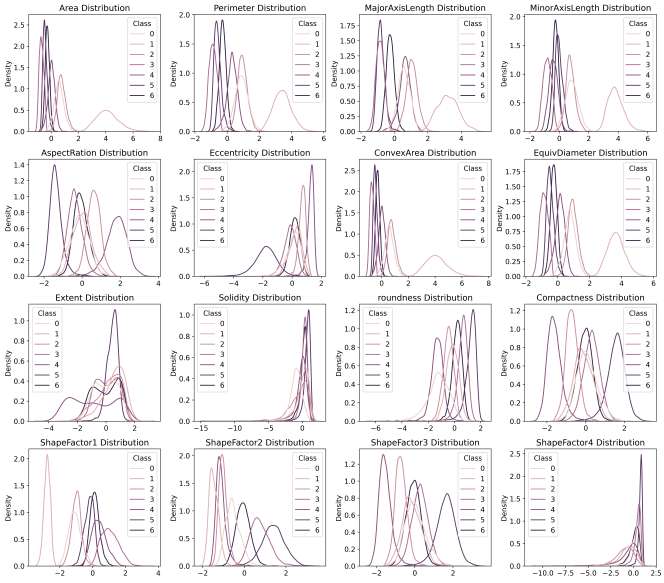

Fig. 2. Distribution of normalized feature values for the dry bean dataset.

differences. Here we chose to analyze only the red wine dataset for simplicity, resulting in 1,599 total entries. Since the original dataset was imbalanced with only a few quality 3 or 8 datapoints (3 and 16, respectively), those with quality labels 3 and 4 were consolidated and those with quality labels 7 and 8 were also consolidated, resulting in 4 classes ($\leq$4, 5, 6, $\geq$7). The class labels were then shifted for 0-indexing ($\leq$4: 0, 5: 1, 6: 2, $\geq$7: 3) in order to train models. All other preprocessing, including the train-test split, was performed similar to that of the dry bean dataset. Summaries for this dataset are shown in Fig. 1 (right) for class distribution and Fig. 3 for normalized feature value distribution. The various classes have a large degree of overlap across many features except volatile acidity, citric acid, and alcohol. This observation combined with the subjectivity in the quality rating makes the wine quality dataset very challenging for proper classification. Hence, we expect all models to perform much worse. In this situation, we hypothesize that NNs, with their increased model expressiveness, may be able to better model the complex relation between input features and output labels, resulting in better performance.

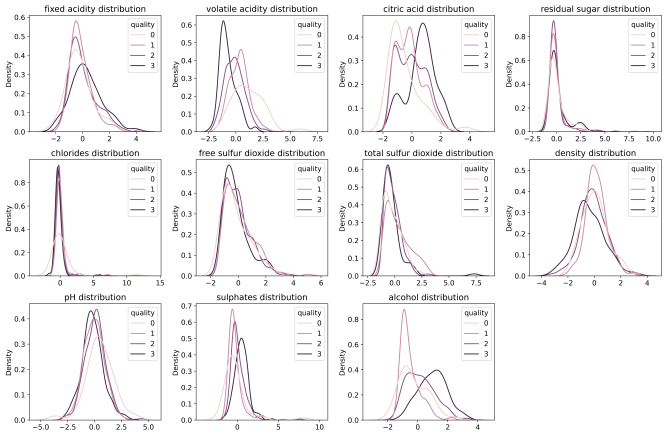

Fig. 3. Distribution of normalized feature values for the wine quality dataset.

III. General Workflow

For each of the 3 algorithms, we selected several hyperparameters that affect model performance and listed them in Table I. The ranges tested for each hyperparameter differs between datasets. For NN specifically, we fixed several hyperparameters to reduce the amount of tuning needed. ReLU was used as the activation function for all units as it is currently the most popular and have shown good performance across a wide range of applications. SGD was used as the optimizer with momentum fixed at 0.9. All other hyperparameters across the 3 algorithms not lised in Table I were left at their default values. Models were initially tuned simultaneously using a crude grid search approach, where only 3-4 hyperparameter values spanning multiple orders of magnitude were tested. 5-fold cross validation was used to assess model performance during tuning. The set of hyperparameter values that resulted in the highest validation accuracy was selected as the starting point for further manual tuning. During manual tuning, each hyperparameter was varied independently, keeping all others constant. Values were tested at more refined intervals in order to properly define the validation curve and illustrate the bias-variance trade-off. Once again, the hyperparameter value resulting in the best 5-fold cross validation accuracy was selected. In cases where the validation accuracy was similar, other factors such as how the hyperparameter impacts runtime were considered. Note that due to the minor imbalance in classes seen in both datasets, using f1 score as the cross validation metric would be a better reflection of the models' performance, as it captures false positives and false negatives. However during experimentation, we found that while the score value differs depending on which metric was used, the optimal hyperparameter value did not vary significantly. Therefore, for the purposes of tuning, all results are reported with validation accuracy. During testing, however, other metrics

were reported to better characterize the performance of these models (see below).

After manual tuning, each model was trained on the entire training set using the set of optimal hyperparameter values. Early stopping was applied to NN training depending on when the validation loss reached its lowest point. Learning curves were generated using optimal hyperparameter values to illustrate additional characteristics of the models, such as whether overfitting was an issue. Finally, the performance of the trained models was tested with the test set, and we report the test accuracy, test recall, and test f1 score, along with the confusion matrix for evaluation and comparison.

TABLE I
HYPERPARAMETERS CONSIDERED FOR EACH MODEL

Model	Hyperparameters considered
NN	No. hidden layers, no. units per layer, dropout percent, learning rate, weight decay
SVM	Regularization constant, kernel type, degree (polynomial kernel only)
k-NN	No. neighbors, distance function, neighbor weighting

IV. DRY BEAN DATASET MODEL RESULTS

A. Neural networks

For the NN model, initial grid search results indicated that having 2 hidden layers with 256 hidden units each along with a 0.1 dropout and no weight decay produced the best validation accuracy. This set of values served as the baseline for further manual tuning and the results are shown in Fig. 4. Validation curves exploring architecture complexity (Fig. 4, top left) indicate that the architecture can be significantly simplified without sacrificing accuracy. Specifically, a single hidden layer with 16 units has nearly identical performance compared to the baseline case. Doing so greatly reduces the number of model parameters, which benefits training time and prevents overfitting. The results also illustrate the bias-variance trade-off, with increasing model complexity from adding hidden layers leading to worse validation accuracy (high variance regime). On the other hand, a drop in accuracy occurs when the model is oversimplified with only 8 hidden units per layer (high bias regime). Due to the significant improvements to training time by updating the model to only include 1 hidden layer with 16 hidden units, all subsequent manual tuning used the simplified architecture. We also explored whether adding regularization either through dropout or weight decay can benefit model performance. As shown in the top right panel of Fig. 4, regularization did not appreciably affect validation accuracy with the simplified model architecture. In this case, overfitting was not a significant issue, as indicated by the loss curves in Fig. 4 where the training and validation accuracy are highly similar. This is likely due to the model being simple enough, having relatively few parameters compared to the overall training set size. Because of these observations, we kept the dropout percent and weight decay at the baseline values (0.1 and 0, respectively). Finally, learning rate was also explored since it affects the convergence rate and training stability. As shown in Fig. 4, the model can accommodate learning rates up to 0.1, which leads to the optimal convergence rate. Any further increases in learning rate results in instability, as indicated by the erratic behavior of the loss curves. The final set of hyperparameter values chosen for the NN model on the dry bean dataset is shown in Table II. An NN model using these values converges after around 80 epochs of training and has a smooth loss curve (Fig. 5 left). The near perfect overlap between the training and validation curves suggest that there is essentially no overfitting, and that the model can generalize very well on this dataset. Both the training and validation accuracies are also high, suggesting the model is optimal. We also plotted the learning curve as well (Fig. 5 right). Both the training and validation accuracy plateaus as training size increases, suggesting that using the full training set to train the final NN model is suitable.

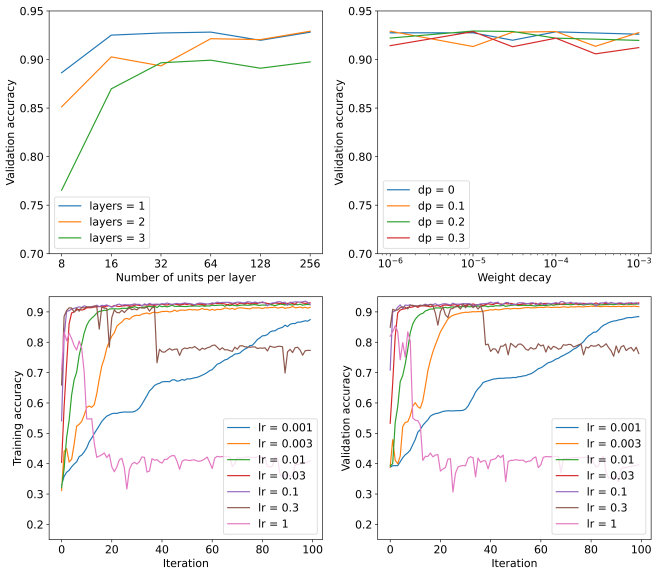

Fig. 4. Manual hyperparameter tuning results for NN on the dry bean dataset. Top left: validation accuracy versus number of units per layer at different numbers of hidden layers; top right: validation accuracy versus weight decay at different dropout percentages; bottom left: training accuracy as a function of epoch with varying learning rates; bottom right: validation accuracy as a function of epoch with varying learning rates. Validation accuracy was obtained using 5-fold cross validation.

B. Support vector machines

Initial hyperparameter value grid search for the SVM model suggested that a radial basis function (rbf) kernel (with $\gamma = 0.1$) and a regularization constant of $C = 10$ gave the best results in terms of validation accuracy. However, during manual tuning, it was found that all of the kernels tested, which included linear, polynomial of degrees 2, 3, and 4, and rbf, had very similar validation accuracy (Fig. 6, top left). To determine the best kernel function to use, we also examined the runtime during training and cross validation. The SVM implementation used in this project employs the sequential minimal optimization algorithm [4], which is iterative. Therefore, runtime is a

TABLE II
OPTIMIZED HYPERPARAMETER VALUES ON THE DRY BEAN DATASET

Model	Hyperparameter	Value
NN	no. hidden layers	1
	no. units per layer	16
	dropout percent	0.1
	learning rate	0.1
	weight decay	0
	momentum	0.9
SVM	kernel	polynomial, degree 3
	regularization constant	10
k-NN	no. neighbors	22
	distance function	euclidean
	neighbor weighting	inverse distance

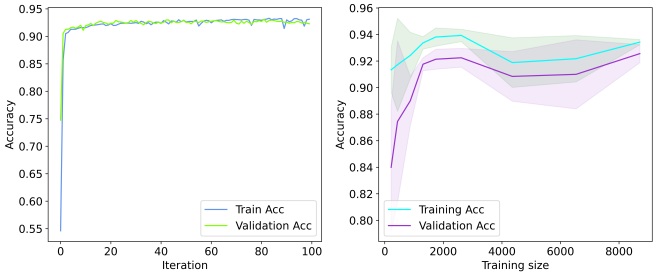

Fig. 5. NN training results using optimized hyperparameter values on the dry bean dataset. Left: training and validation accuracy as functions of epoch; right: training and validation accuracy as functions of training set size. Validation accuracy was obtained using 5-fold cross validation.

function of the number of iterations needed to find the global optimum during training and the complexity of the kernel function calculations (sample size and number of features also affect runtime but are both fixed here). In this case, a degree 3 polynomial kernel function outperformed all others with the lowest runtime, and hence was chosen as the optimal kernel type (Fig. 6, top right). Regularization strength, tuned through the hyperparameter C was also an important factor to consider, where a lower C value indicates stronger regularization. A high regularization strength can lead to model underfitting, as indicated by the drop in accuracy for low C values in Fig. 6. Conversely, at high C values, the validation curve plateus, suggesting that the SVM model does not overfit the data even with minimal regularization, which is consistent with the NN results. However, there is an optimal C value in terms of runtime, namely $C = 10$. Generally, increasing C lowers the tolerance for misclassification and hence more iterations are needed to find the proper hyperplane leading to slower convergence. It was surprising to see that lowering C also increases runtime. According to the scikit-learn documentation, lowering C results in more support vectors and hence will increase prediction time during validation, which is also undesirable. Given these considerations, we chose to keep the value of C at 10 for the optimal runtime and high validation accuracy. Table II summarizes the optimal hyperparameter values for SVM after manual tuning. Using these hyperparameters, a learning curve was built and shown in Fig. 6. Similar to the NN case, SVM does not show signs of overfitting. However, unlike NN, SVM does benefit from increasing training set sizes, likely due to its increased sensitivity to training data. We believe this sensitivity comes from the discrete nature of classification using hyperplanes, where each datapoint is distinctly classified into a specific class based on its position relative to the hyperplane. By contrast, NN outputs probability distributions over the classes for each datapoint. Classification is performed by assigning the max probability class to the datapoint and the model will predict the same class so long as the output probability of that class is $> \frac{1}{7}$ (for the dry bean dataset specifically). Hence, there is a certain amount of tolerance built into NN predictions that is not present in SVM.

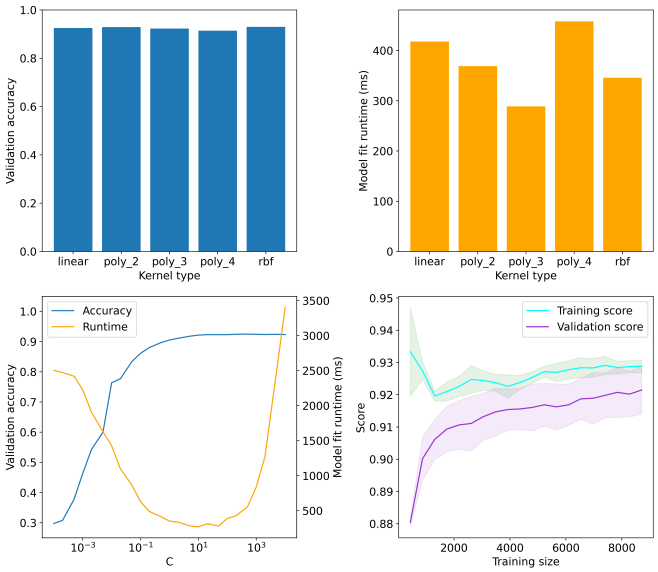

Fig. 6. SVM hyperparameter tuning and training results on the dry bean dataset. Top left: validation accuracy with various kernel functions; top right: model runtime with various kernel functions; bottom left: validation accuracy and model runtime as functions of the regularization constant C; bottom right: training and validation accuracy as functions of training set size using optimal hyperparameter values. Validation accuracy was obtained using 5-fold cross validation.

C. K-nearest neighbors

Grid search for k-NN indicated that using 22 neighbors (i.e. $k = 22$), an euclidean distance function, and weighting neighbors based on their inverse distance to the query point resulted in the best model performance. In this case, these values perfectly aligned with manual tuning (Fig. 7). The euclidean and manhattan distance functions clearly outperformed cosine similarity. This is likely due to the cosine similarity metric being more sensitive to noise and cannot effectively separate datapoints that are close to each other [5]. It is also interesting to note that the validation curves are smoother in the top right panel in Fig. 7 compared to the top left panel. In other words, by weighting neighbors based on inverse distance, the model becomes less sensitive to small changes in the k value. This is because the classification decision continues to be dominated by the same several datapoints that are closest

in the weighted case. Finally, the shapes of the validation curves with respect to k clearly indicate a bias-variance trade-off. At low values of k, the model overfits the data by only looking at a very close vicinity around the query point, and hence the model becomes sensitive to minor variations within the training data (high variance). On the other hand at high values of k, the model underfits by effectively averaging a large portion of the training data, losing any intricacies along the decision boundary (high bias). Intermediate values of k balance this trade-off, leading to optimal validation accuracy. The final set of optimized hyperparameter values is listed in Table II. Learning curves generated using these values for k-NN are shown in Fig. 7, bottom left. Interestingly, the learning curve's training accuracy does not seem to follow the expected trends where it should monotonically decrease as the training set size increases. This is likely due to the large k value chosen for the model. At small training data sizes with a high k, the model averages a large portion of the training data for each query point, as opposed to only the closest neighbors, introducing a regularization effect and pushing the model to underfit. As the training size increases, k remains constant and hence the regularization effect diminishes, leading to better training accuracy.

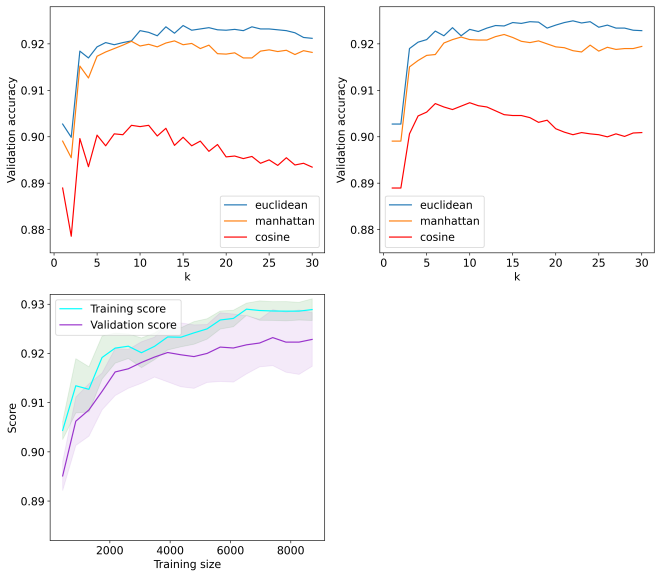

Fig. 7. k-NN hyperparameter tuning and training results on the dry bean dataset. Top left: validation accuracy as a function of number of nearest neighbors for various distance functions. All neighbors were weighed equally; top right: validation accuracy as a function of number of nearest neighbors for various distance functions. All neighbors were weighed by their inverse distance to the query point; bottom left: training and validation accuracy as functions of training set size using optimal hyperparameter values. Validation accuracy was obtained using 5-fold cross validation.

D. Comparison of test results across models

Table III summarizes the results obtained by applying each trained model (with optimized hyperparameter values) on the test set. Consistent with the original hypothesis on this dataset, all models performed very well, reaching >90% scores on prediction accuracy, recall, and f1 score. The confusion matrices shown in Fig. 8 also show very good performance across all 3 models, with minimal misclassified data across the entire test set. All models were able to classify the minority class well (class 1, with the lowest training and testing samples), indicating that they were able to overcome the slight data imbalance issue. However, despite the similarities in performance, model runtimes during training, as well as the amount of effort needed to tune each model varies drastically. Training time for NN is significantly longer than the other 2 models (>50-fold) due to its iterative nature and a much larger number of trainable weights that needs to be updated through back-propagation. NN training time could be reduced by lowering the number of epochs at the slight expense of performance, but it is still difficult to reach the same level of training speed as SVM and k-NN since training time only scales linearly with epochs. Additionally, the number of hyperparameters that affect model performance, which includes those that are part of the model itself as well as those that are related to the optimizer, is also greater for NN than the others, making model tuning more tedious. Overall, for this simple dry bean dataset, the increased expressiveness that NNs offer do not outweigh the increase in training and tuning cost. Simpler models such as SVM or k-NN are much better suited in these situations where the classification task is straightforward.

TABLE III
TEST RESULTS ON THE DRY BEAN DATASET

Model	Accuracy	Recall	F1 score	Runtime (ms)[a]
NN	0.94	0.93	0.93	16300[b] + 35
SVM	0.92	0.94	0.95	265 + 75
k-NN	0.92	0.95	0.95	0.7 + 33

[a]Runtime is expressed in training time + test time.
[b]NN was trained for 80 epochs.

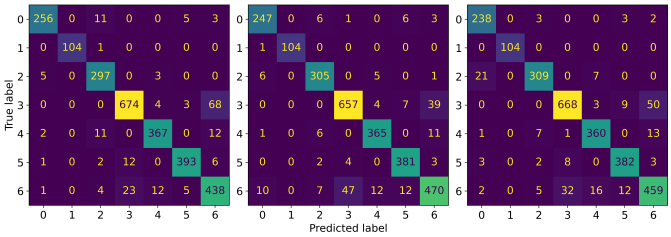

Fig. 8. Confusion matrices on the dry bean test data. Left: NN; middle: SVM; right: k-NN.

V. WINE QUALITY DATASET MODEL RESULTS

A. Neural networks

Following the general workflow, we performed an initial grid search for crude hyperparameter tuning. An NN model having 3 hidden layers with 256 hidden units each along with no dropout and 10^{-4} weight decay was found to be the best. Results from further manual tuning are shown in

Fig. 9. Here, we also saw an opportunity to reduce model complexity by using only 1 hidden layer with 128 hidden units, which is a large reduction compared to the grid search baseline without sacrificing validation accuracy (Fig. 9, top left). The general trends on how model complexity affects validation performance is similar to what was observed in the dry bean dataset, where it is favorable to implement fewer hidden layers but sufficiently high hidden units is needed for the best outcome. It should be noted that with the wine quality dataset, the minimum number of hidden units required is around 64 – 128, which is larger compared to 16 for the dry bean dataset. This is expected since the current dataset has a more complex input–output relationship, requiring more trainable weights. Interestingly, it was found that the common regularization techniques do not seem to benefit validation performance (Fig. 9, top right), despite the model's validation accuracy being worse than the training accuracy (Fig. 9, bottom panels). Additionally, unlike that of training performance, increases in validation performance slows substantially after the first few training epochs. These characteristics point to model overfitting on this dataset. Combined with the ineffectiveness of model regularization, we believe that data augmentation, for instance synthetic minority oversampling (SMOTE, [6]), could potentially improve generalization and close the gap between training and validation performance. Finally, the optimal learning rate was set as 0.1 based on both the training and validation loss curves (Fig. 9, bottom panels). A similar reasoning of balancing convergence rate and training stability applies here. The optimal set of hyperparameter values are shown in Table IV. Fig. 10 shows the loss curves (left) and learning curves (right) based on a model with optimized hyperparameters. Once again, the discrepancy between training and validation performance is evident. The training curves corroborate all discussions so far in that the model is still in the high variance regime even when using all of the training data. This explains why the wine quality dataset seems to be much more prone to overfitting compared to the dry bean dataset.

TABLE IV
OPTIMIZED HYPERPARAMETER VALUES ON THE WINE QUALITY DATASET

Model	Hyperparameter	Value
NN	no. hidden layers	1
	no. units per layer	128
	dropout percent	0
	learning rate	0.1
	weight decay	10^{-4}
	momentum	0.9
SVM	kernel	rbf, $\gamma = 0.01$
	regularization constant	200
k-NN	no. neighbors	48
	distance function	manhattan
	neighbor weighting	inverse distance

B. Support vector machines

On the wine quality dataset, initial SVM hyperparameter tuning with a grid search approach suggested the rbf kernel (with $\gamma = 0.01$) and $C = 100$. Once again, during manual

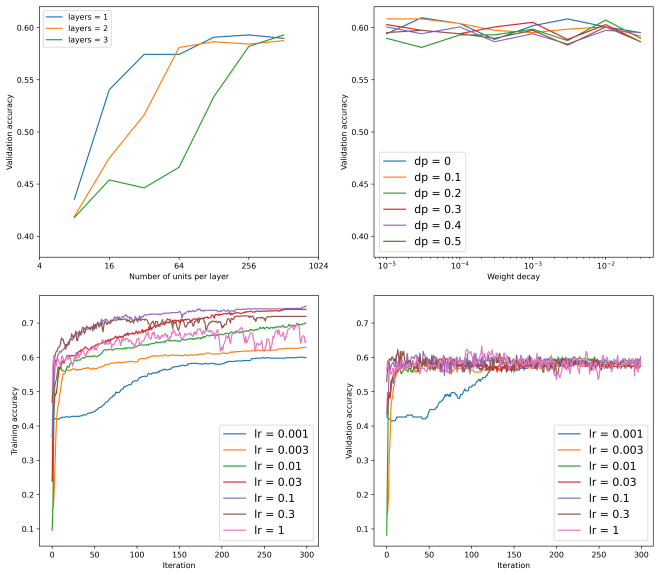

Fig. 9. Manual hyperparameter tuning results for NN on the wine quality dataset. Top left: validation accuracy versus number of units per layer at different numbers of hidden layers; top right: validation accuracy versus weight decay at different dropout percentages; bottom left: training accuracy as a function of epoch with varying learning rates; bottom right: validation accuracy as a function of epoch with varying learning rates. Validation accuracy was obtained using 5-fold cross validation.

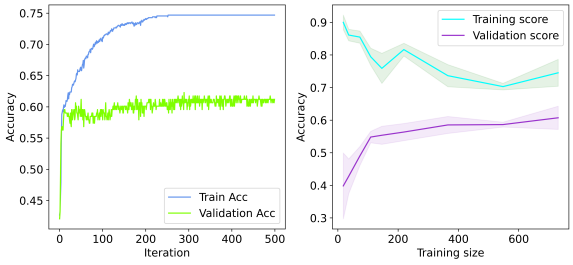

Fig. 10. NN training results using optimized hyperparameter values on the wine quality dataset. Left: training and validation accuracy as functions of epoch; right: training and validation accuracy as functions of training set size. Validation accuracy was obtained using 5-fold cross validation.

tuning, it was found that all kernel types tested resulted in similar validation accuracy with the rbf kernel having the lowest runtime (Fig. 11, top panels). Therefore, there is no need to modify the kernel function from the baseline value. As for the C value, Fig. 11 shows a clear trade-off in terms of validation accuracy, where extreme low and high values of C results in underfitting and overfitting, respectively. The appearance of the high variance regime at high C values differs from the trends seen in the dry bean dataset, where the validation curve plateaus. As explained before, the wine quality dataset is more susceptible to overfitting, which places a lower bound on the regularization strength needed (upper bound on C). The runtime curve as a function of C aligns with expectations and was previously explained in section IV-B. Fortunately, the optimal C value in terms of validation

accuracy lies before the point at which the runtime increases substantially. We chose $C = 200$ for this model based on the tuning results (Table IV). The learning curve produced by SVM utilizing optimized hyperparameters is shown in Fig. 11 in the bottom right panel. The trend is large similar to the NN case and a similar analysis can be applied here.

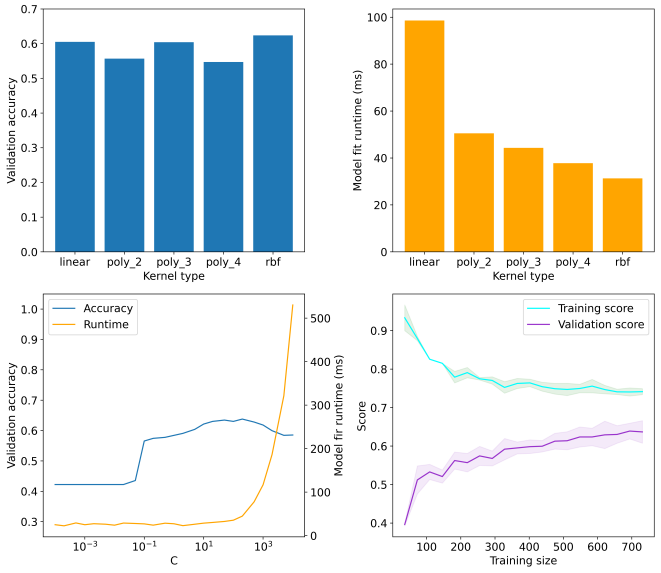

Fig. 11. SVM hyperparameter tuning and training results on the wine quality dataset. Top left: validation accuracy with various kernel functions; top right: model runtime with various kernel functions; bottom left: validation accuracy and model runtime as functions of the regularization constant C; bottom right: training and validation accuracy as functions of training set size using optimal hyperparameter values. Validation accuracy was obtained using 5-fold cross validation.

C. K-nearest neighbors

Starting with the grid search, we obtained $k = 48$, manhattan distance function, and inverse distance weighting as our baseline hyperparameter values. Here, manual tuning reveals several interesting results (Fig. 12). While the 3 distance functions are equivalent in terms of validation performance, whether or not to weigh neighbors by inverse distance plays a much bigger role. Since inverse weighting significantly increased validation accuracy, this suggests that the decision boundary is driven more by the local structure of nearby datapoints, leading to complex and irregular boundary shapes. Given that the feature values are similar across the various classes, this result is not surprising, and the boundary has to be sensitive to small changes within the feature values. The complexity of the decision boundary shape explains why the wine quality dataset is prone to overfitting. In general, the more the boundary conforms to the local irregularities of the data, the more difficult it is for the model to generalize. On the other hand, the optimal k value for this dataset is a lot higher than that of dry bean, providing the additional regularization effect needed to alleviate overfitting. Additionally, another advantage of k-NN here is that the inverse distance weighting also makes the model less sensitive to class imbalance observed in this dataset. This is because doing so can provide more weight to the minority class datapoints for correct classification if the query point is nearby. Overall, manual tuning resulted in the same optimal hyperparameter values as the baseline, and hence no values were changed (Table IV). The k-NN learning curves here also display the same upward trend for the training accuracy. This is an effect of choosing a large k value and was previously explained in section IV-C.

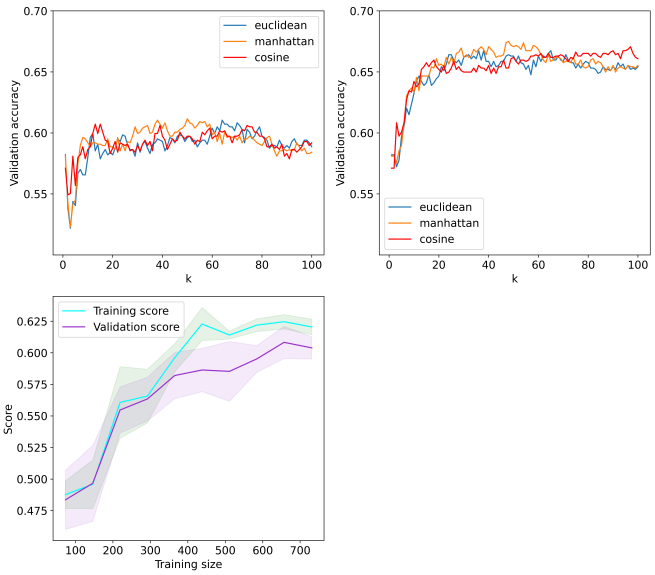

Fig. 12. k-NN hyperparameter tuning and training results on the wine quality dataset. Top left: validation accuracy as a function of number of nearest neighbors for various distance functions. All neighbors were weighed equally; top right: validation accuracy as a function of number of nearest neighbors for various distance functions. All neighbors were weighed by their inverse distance to the query point; bottom left: training and validation accuracy as functions of training set size using optimal hyperparameter values. Validation accuracy was obtained using 5-fold cross validation.

D. Comparison of test results across models

Table V summarizes the test results after training each model with optimized hyperparameters. All performance metrics were much worse on this dataset compared to the dry bean dataset, which is expected. However, contrary to the original hypothesis, NN with its increased expressiveness did not perform the best, and instead k-NN demonstrated a notable increase in test results for both correctly classifying each class (accuracy) and minimizing false positives and false negatives (recall and f1 score). As mentioned previously, the decision boundary for the wine quality dataset is complex, and likely difficult to be modeled by a parameterized mathematical expression, which is how SVM carries out its task. k-NN, on the other hand, can better model the local intricacies and irregularities of the boundary, especially when inverse distance weighting is enabled. As for NN, while it is expressive enough to be able to model such complexities (since NNs with 1 hidden layer can model any continuous function), it may require a lot more hidden units than what was tested here, which in turn makes converging to the appropriate weight

values even more challenging. Practically speaking, given the already long computation times for NN, it would not be suitable for this particular dataset. Test confusion matrices for all 3 models are shown in Fig. 13. None of the models were able to classify class 0 at all as it accounted for only $<4\%$ of the entire dataset. However, k-NN has improved classification results on not only the majority classes but also on class 3, which is the second least abundant class. This further illustrates k-NN's ability to classify based on rare instances in the data as the algorithm gives more weights to these datapoints when needed.

TABLE V
TEST RESULTS ON THE WINE QUALITY DATASET

Model	Accuracy	Recall	F1 score	Runtime (ms)[a]
NN	0.62	0.42	0.43	6710[b] + 3
SVM	0.63	0.47	0.47	52 + 10
k-NN	0.68	0.49	0.50	1 + 4

[a]Runtime is expressed in training time + test time.
[b]NN was trained for 300 epochs.

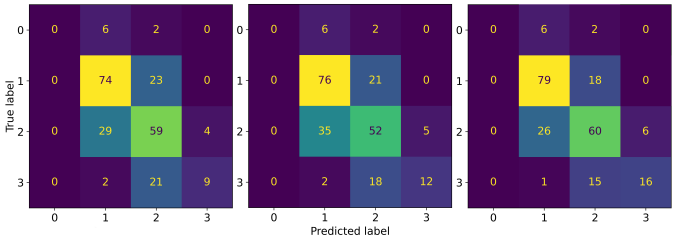

Fig. 13. Confusion matrices on the wine quality test data. Left: NN; middle: SVM; right: k-NN.

VI. KEY LEARNINGS AND FUTURE WORK

In conclusion, this project explores 3 machine learning models, NN, SVM, and k-NN, on 2 separate classification datasets, dry bean and wine quality. In most cases, the results conform with our expectation: (1) all models performed very well on the dry bean dataset where there are clear separation of feature values depending on the class, although tuning was still required for each to attain the optimal performance; (2) validation and test performance was generally much poorer on the wine quality dataset, with results from all 3 modeling efforts suggesting the complex decision boundary shape as well as issues with overfitting being the primary causes of modeling difficulty. A major learning here is that while NN is the most expressive model and has gained significant popularity due to that reason, it is not suited for all applications. In instances where the dataset is easy to classify and does not require high expressiveness, implementing NNs will only lead to drastic increases in training time, without providing any performance benefits. In other instances where the decision boundary is difficult to be modeled by mathematical relations, NNs can also struggle and may need a high number of model weights. K-NN, on the other hand, was able to handle this situation surprisingly well as it solely relies on the data itself to draw the decision boundary.

For future work, as mentioned throughout the analysis for the wine quality dataset, we could explore data augmentation to understand whether that can help with generalization when the common model regularization methods fail. In cases where there is an imbalance in the classes, performing synthetic minority oversampling can be a good method to augment the data. Furthermore, as the authors in the original publication discussed, SMOTE can be used with random undersampling to simultaneously oversample the minority class and undersample the majority class, respectively [6]. This can be done to balance the classes while maintaining a constant training dataset size. Since the learning curves also suggest that all models benefit from additional data, we could perform SMOTE only to artificially increase the amount of training data. Comparing both sampling methods could provide a better understanding of why this dataset is challenging to work with and potentially lead to better validation and test results.

REFERENCES

[1] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011
[2] M. Koklu and I. Ali, "Multiclass classification of dry beans using computer vision and machine learning techniques," Computers and Electronics in Agriculture, vol. 174, 2020.
[3] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," Decision Support Systems, vol. 47, pp. 547–553, 2009.
[4] C. C. Chang, C. J. Lin, "LIBSVM: A library for support vector machines," ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 1-27, 2011
[5] V. B. Surya Prasatha et al., "Effects of Distance Measure Choice on KNN Classifier Performance - A Review," Big Data, vol. 7, 2019.
[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of artificial intelligence research, pp. 321-357, 2002.