CS7641 A1: Supervised Learning

Charles Leung
cleung75@gatech.edu

I. Introduction - Dataset Explanation

As a self proclaimed cooking savant, I have always been curious about how the intricacies of differing food attributes combine together and determine the outcome of a dish. Each trait of a particular food can greatly alter its outcome and whether or not it should/should not be used in everyday cooking. However, a single food item may house a variety of different species, and it may not be clear for the average human to differentiate one type from another. Utilizing the power of machine learning, one can harness predictive learning algorithms and data analysis tools to learn and categorize common food items into different species. In this paper, we will dive deeper into exploring the Mushroom Edibility dataset and the Rice Species dataset from UCI's Machine Learning Repository.

I have chosen these datasets to analyze as I believe they provide a deeper analysis to common food items that people tend to overlook. The Mushroom dataset contains qualities like smell, color, and size, that can help an average mushroom forager scour different edible mushroom species for their cooking endeavors in a safe manner. The rice dataset is interesting because it dives deep into metrics and attributes of a common household food staple. Analyzing its physical metrics on a miniscule grain of rice seems interesting, and determining whether or not the learner can perform when lengths/widths of every grain of rice can vary so greatly.

What is interesting to me, however, is understanding if quantitative features of a dataset will outperform and correlate to qualitative metrics. The rice dataset provides features that are almost purely physical. Features like area, perimeter, length, etc. are purely used to classify two different types of rice, Cammeo and Osmancik. In the mushroom dataset, however, there is a larger disbursement of qualitative data. The set here focuses on aspects of the mushroom itself, such as color, odor, bruises, and gill spacing. With the data being split up like this, it would be interesting to see how our learners perform with different qualities presented to each dataset.

With that being said, the goal of this paper will be to analyze the learner's performance with two different datasets, and determine how the complexity of different features play into the overall classification of different rice types and edibility of mushrooms. To do this, we will be utilizing three different learners: Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Neural Networks and compare the performance of each one per dataset. We will take advantage of Python library's sci-kit to create the learners. Then we will analyze the hyper-parameters associated with the learners and ultimately tune them to yield the highest accuracy via GridsearchCV.

k-NN is a simple non parametric algorithm that uses a distance based algorithm to fit a new data piece to the k nearest neighbors to it, classifying it based on the most common outcome based on the k nearest results in the training data.

SVM aims to find the best suited hyperplane to effectively divide the data in a space with high amounts of features (dimensions). To optimize this hyperplane, we optimize the kernel functions to determine the best linear/non-linear relationships between the data. While this is a robust learning mechanism, its complexity causes it to be less computationally efficient.

Neural Networks utilize pattern recognition techniques and activation functions thresholds to classify data into its respective categories. Its ability to model complex functions makes it a great learner to solve more complicated problems.

After consolidating the performance data, we will compare the overall performance between one dataset and another, and finally compare the algorithms performance against one another. Given the cleanliness of the data and its vast amount of features, we hypothesize that both datasets will yield good results in all of the learners. However, given the qualitative features present in the mushroom data, we believe that those features will provide less noise in the overall dataset and be stronger indicators to predict edibility in mushrooms, more than the numerical features present in the rice dataset.

II. Preprocessing Data

To give a better analysis of the dataset, we must first sift through our data to see if there are any missing values or an imbalance of data (meaning the data is heavily skewed on one result over another, causing the training to be biased).

A. Rice

The rice dataset contains 7 total features: area, perimeter, major/minor axis length, eccentricity, convex area, and extent. All of these are numerical ratings that describe the physical features of the grain of rice. The output of this dataset is either an Cammeo, or an Osmancik

grained rice. This means that we will be analyzing a binary classification problem, and our learner will only be predicting one of two outcomes. Of the 3810 instances, 42.7% of them are Cammeo outputs, and the rest are Osmancik. This means our dataset is balanced relatively well, which is important to note due to bias. In a biased dataset, where the majority of the testing data is skewed towards one result, the learner will naturally be geared to be trained to predict the majority class, causing poor performance in the minority class. In addition, there were no other missing values/entities in our dataset, which is an indication that this is good data. Because of this, little/no preprocessing is needed.

Figure 1 below describes the correlation between each feature. In the figure, a correlation of 1 indicates a direct relationship between the feature and the result being an Osmancik type rice. In this case, the features are pretty inversely related to the outcome (class), suggesting there is a strong relationship between the features and the outcome. It is also worth mentioning that there is some redundancy between different features, which may cause an unnecessarily complex model and cause overfitting. However, due to the limitations of the features we have, we will analyze first, and re-prune our data if our accuracy is suboptimal.

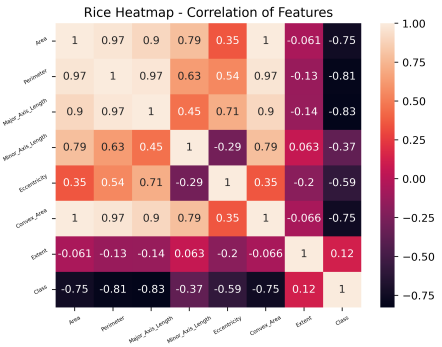

Fig. 1: Heat map of Rice dataset. Correlation of features compared with the outcome (Class) being an Osmancik rice type.

B. Mushrooms

The mushroom dataset contains 23 features and is more complex in nature. They are cap shape, cap surface, cap color, bruises, odor, gill attachments, gill spacing, gill sizes, gill color, stalk shape, stalk root, stalk surface above/below ring, veil color, ring number, ring type, spore print color, habitat, and population. The output of the dataset is either poisonous or edible, meaning this is another binary classification dataset. We will apply the same preprocessing techniques where applicable in the mushroom dataset by first analyzing the spread of the data. Of the 8124 instances, 48.2% of the data are poisonous, which is a near 50/50 split. This is a good indication that we do not need to resample data in efforts to reduce bias. In addition, there were also no missing data and entities in the dataset. One differing aspect between this data and the rice dataset, however, is that the mushroom data provides qualitative (string) features/outputs instead of quantitative. To allow for data digestion in sci-kit learners, and have unit variance in the qualitative data, we will discretize our features (using LabelEncoder in sklearn) into numerical representation of the data.

Figure 2 below describes the correlation between each feature. Similar to the rice dataset, a correlation of 1 indicates a direct relationship between the feature and the toxicity of the mushroom. From analyzing the heat map, there is no clear winner of one feature dominating the prediction over the others. This means that the features will need to work together to build an accurate prediction of each test case. In addition, when scanning the feature-feature correlation, there seems to be little redundancy that is of concern (mostly between -0.6 to 0.6, which is not too strong of a correlation). This means there is not much redundancy and we can keep the complexity of the data for now.

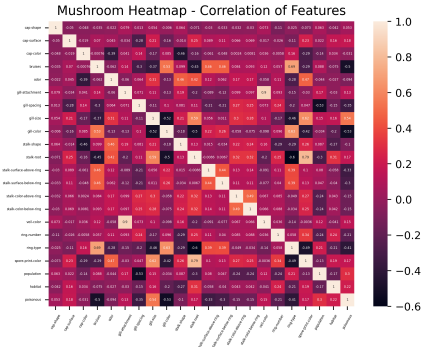

Fig. 2: Heat map of Mushroom dataset. Correlation of features compared with the outcome being poisonous.

III. K-Nearest Neighbors (KNN)

The following sections will analyze K-NN's performance with our dataset. The hyperparameters we will analyze and tune are the numer of neighbors (k), p value (distance calculator), and weights calculator.

A. Dataset 1 - Rice

The simplicity of the rice's dataset, combined with the binary classification goal makes KNN a decent tool to use when wanting to create the model quickly. As mentioned earlier, we will use Python's sklearn library, in particular, the default KNeighborsClassifier library. We will first create a KNN learner with default values (uniform weights and Minkowski distance calculator), and vary the hyperparameter k value and observe the results. We standardized the features and divided the data into 70% for training and 30% for testing, using a 5-fold cross-validation strategy to minimize bias. This k

value indicates the number of train data neighbors the test data will look at when predicting from the model. This validation curve is generated in figure 3 below.

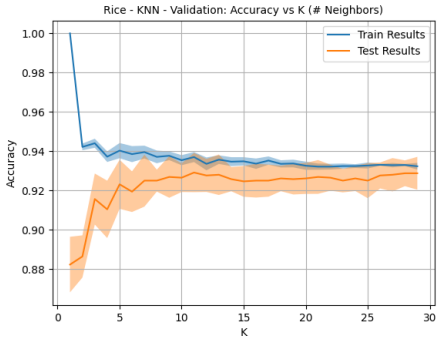

Fig. 3: KNN Validation Curve for rice dataset, varying K while keeping other parameters constant. Overfit happens at around $k < 5$.

As you can see from the graph, overfitting is prominent when k values decrease. These results make sense, as a k value decrease indicates it will generalize less and less when fitting test data, causing the model to be over reliant on the train data and ultimately decrease accuracy. From the figure, the point at which overfitting begins is when $k \leq 5$. When that happens, the accuracy dips to below 90%. Despite that, however, the outcome of the dataset looks very promising with a $> 90\%$ accuracy rate without any hyperperameter tuning.

Another hyper-parameter we tuned was the distance metric, which utilizes a different calculation when measuring distances between the testing data to the nearest neighbors. Tuning different calculation methods will allow us to explore and select the best one. In Scikit, there are three types of calculations, the Euclidean (which measures the shortest straight-line distance from one to another), the Manhattan (which measures distances by right angles), and the Minkowski (a combination of the first two). Here, we built three different learners (one for each metric), each with the same fixed neighbor parameter, and trained with a 5-fold cross validation strategy. The perceived accuracy is then calculated via the test dataset and graphed in the bar chart in figure 4 below.

From the graph it is a euclidean and Minkowski based KNN learner performs slightly better with a 0.001% margin between one another. We can finally take these results and use GridSearchCV to tune these parameters. For the grid, we will use neighbors = [1 to 100], p=[1 to 3], and weights = [uniform, weighted] (uniform gives equal weighting for k nodes regardless of how far they are from testing point, and weighted penalizes further distanced nodes). The optimized parameters ended up being n neighbors = 43, p = 3, and weights = uniform, with a final test accuracy of 92.3%, which is a decent score.

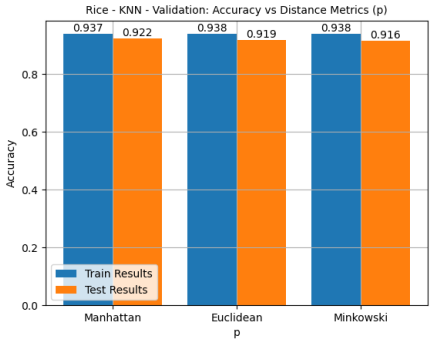

Fig. 4: KNN validation curve for rice dataset, varying p (distance metric calculation method) while keeping other parameters constant. Accuracy results did not change too much with the difference in calculation methods.

Finally, we rebuild a new instance of KNN using these optimal parameters and record the learning rate with respect to training examples to observe its performance over the number of samples, depicted in figure 5 below. From the graph, it shows the training and testing set starting at a lower accuracy, and overtime converging at around 0.935, indicating the model is optimized well and more generalized.

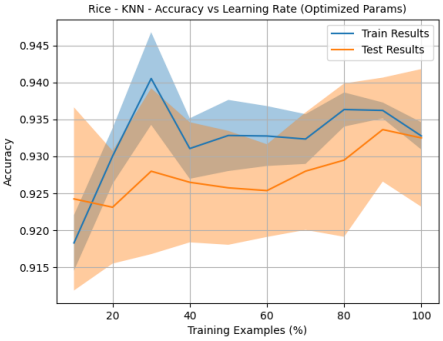

Fig. 5: KNN Learning curve for rice dataset with optimized parameters.

B. Dataset 2 - Mushroom

We take a similar approach with our mushroom dataset by first creating an arbitrary KNN model from sklearn, and then observing each hyperparameter graph, and finally optimizing the parameters using GridSearchCV. We again reuse a 70:30 split on training and testing data and enable 5-fold cross validation strategy, and then build out the validation curve, altering the number of neighbors (K), shown in figure 6 below.

Interestingly, the mushroom dataset appears to be extremely performant at low levels of k, with accuracy approaching 100% as k approaches 1. This means the model is fitting our mushroom data extremely well. As K increases, the accuracy begins to decrease, indicating the model is beginning to underfit as it observes more

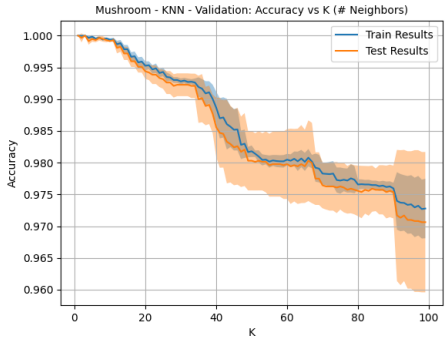

Fig. 6: KNN Validation Curve for mushroom dataset, varying K while keeping other parameters constant.

unseen data. Finding an optimal K value later on will be important to note, as low values of K tends to overfit due to the model being overtrained. However, a high value of K, as observed, compensates in accuracy as well. The key is to find something in the sweet spot (in this case, between 5 - 15), to optimize this parameter.

The second hyperparameter is the p value, and the approach taken was identical to the rice dataset's. The results in figure 7 below show that there isn't a significant depletion of performance when using a different distance calculator, meaning the hyperparameter here does not affect performance much.

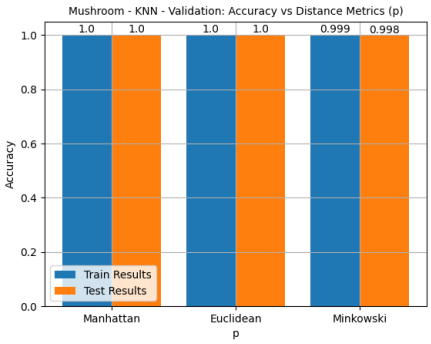

Fig. 7: KNN validation curve for mushroom dataset, varying p.

With the parameters analyzed, we use GridSearchCV to tune them. For the grid, we will use neighbors = [1 to 100], p=[1 to 3], and weights = [uniform, weighted] (uniform gives equal weighting for k nodes regardless of how far they are from testing point, and weighted penalizes further distanced nodes). The optimized parameters ended up being k = 1, p = 1, and uniform weights, with a final test accuracy of 100%, which is an excellent score. One thing to note is that the optimizer responded with using k = 1. While this is an acceptable parameter, having k = 1 tends to lead to overfitting, as the model might be too sensitive to noise and outliers in the data. However, since cross validation yielded 100% accuracy consistently, perhaps overfitting may not be an issue for this case. What I would do as a future enhancement, however, is to introduce some more data (and potentially more noise), and see how well it performs in that condition, and perhaps re-optimize if the dataset that was introduced to us was too perfect to begin with.

Finally, we can use the optimized parameters to construct our learner and see its learning rate. From the graph in figure 8 below, we can see that the training data is consistently at 100%, meaning that the model is fitting perfectly to the training data. The initial variance of the testing data converges quickly with the training set, at around 50%, then consistently remains at 100%, meaning that the model is performed optimally. Overall, kNN is a great choice for this dataset, suggesting that the classes were easily distinct and separable given the features we had in our dataset.

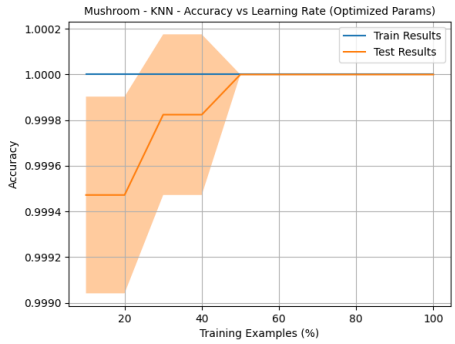

Fig. 8: KNN Learning curve for mushroom dataset with optimized parameters.

IV. Neural Networks

A. Dataset 1 - Rice

The approach we will take to explore Neural Networks will be similar to KNN. We can implement sklearn's MLPClassifier to build our learners, observe how its hyperparameters affect performance. Then make a prediction and use GridSearchCV to tune on hyperparameters. To save time on compute time and due to hardware limitations of my computer, we will stick to using a single layer neural network and optimize the number of neurons in a single layer, the alpha (regularization), activation function, and initial learning rate. Just like KNN, we split the data and use cross validation, as always. We begin by taking a look at the accuracy vs. alpha rate and accuracy vs. initial learning rate, shown in figure 9a and 9b below.

Alpha essentially promotes regularization in our neural network weights. Having large weights in or data increases the model complexity, which can cause overfitting easily. Promoting regularization enables our boundaries to be more smooth and helps promote generalization from noisy data. From figure 9a, we can see the rice data hitting an accuracy at around 93% before dropping as alpha continues to increase. When alpha increases, the

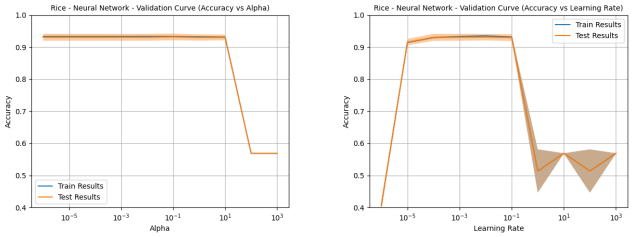

Fig. 9: Rice Dataset - Neural Network Validation Curves. From left to right: (a) Alpha (b) Learning Rate

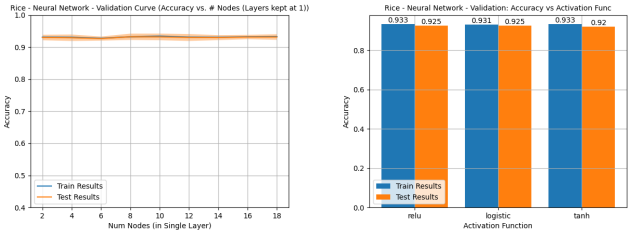

Fig. 10: Rice Dataset - Neural Network. From left to right: (a) Single layer NN validation curve for rice dataset alternating number of neurons (b) Accuracy vs Activation Function

penalties increase to the point where the model lacks in complexity, which is an indication of underfitting (located at alpha = 10^2).

Learning rate is the initial rate at which parameter weights are being updated. This is an interesting hyperparameter to control, as controlling the steps indicate how fast neural networks can find convergence in a solution and requires a fine balance. Having too slow of a learning rate will significantly increase runtime, and also run into the possibility of having the model converge at a local minima. Having too high of a learning rate will promote instability and oscillation and cause an unexpected divergence. Both alpha and learning rate go hand in hand - learning rate controls the rate at which weights are updated, and alpha regulate the values of the weights. In the graph, we can see the learning rate hitting a sweat spot at around 10^{-2}. With the learning rate increasing, we can see the accuracy plummet due to divergence, and similarly when learning rate decreases, it causes a potential incorrect convergence.

Another crucial element to observe is the number of neurons in the neural network, demonstrated by figure 10a, as well as the activation function selection in figure 10b. Interestingly, it seems like the number of neurons in a single layer neural network does not have too much of an effect on performance of the learner as it provides pretty consistent and stable results. This could suggest that the data provided is sufficient enough and of good quality that an increase of data may not be required, since the relationships are already mapped well. The activation function in 10b suggests that a simple threshold system of 1 or 0 as its activation (linear based) or a inverse tangent (non linear) activation function gives the most optimal result. Optimizing the parameters coincidentally will allow us to get a better analysis.

With that, we can begin to optimize the parameters alpha, learning rate, number of neurons, and activation (something we did not mention, but in short, is the determining function that controls the output for every input in a neuron). We then use GridsearchCV and throw these parameters into the solver to figure out the best parameters. The outcome in this case was specified with the following parameters: activation = tanh, alpha = 0.001, neurons = 16, initial learning rate = 0.01. The activation function being tanh suggests that features could also be non linear in nature, which is an interesting find. We will then take this optimized model and analyze its training journey illustrated in figure 11a below, along with the loss curve in figure 11b.

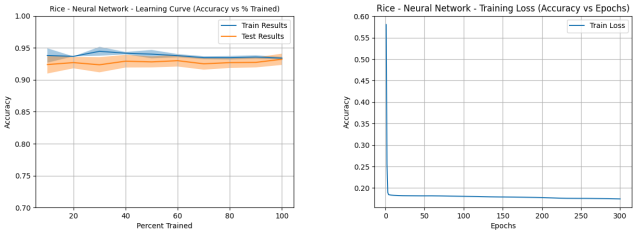

Fig. 11: Rice Dataset - Neural Network. From left to right: (a) Learning Curve (b) Loss Curve

The figure shows that results were relatively slow but stable in the increase of accuracy (in the testing set) throughout the course of the training period, ultimately converging near the tail end of the training period. This is a good indication that the learner effectively optimized, and successfully learned well. Another good indication is by observing the loss curve being stable and converging at a value (approx. 0.2) over a period of 300 epochs. This means that the network was able to adapt and learn the patterns of the dataset and minimize its discrepancy between predicted data and test data.

B. Dataset 2 - Mushrooms

In the mushroom dataset, we can repeat the same exercise as the rice dataset. We begin by taking a look at the accuracy vs. alpha rate and accuracy vs. initial learning rate, shown in figure 12a and 2b below.

Since the dataset did virtually perfect in our KNN learner, it was expected that it would also perform extraordinarily well in the neural network as well. This is because Neural Networks are more flexible and robust and can handle a more diverse amount of patterns (ie. linear vs. non linear), as demonstrated by the alpha and learning rate hyperparameter distribution. Similar to the rice dataset, signs of underfitting are observed due to the model being overly simplistic from the strict regulations, and ultimately missing the hidden intricacies of the data. Similarly in the learning rate, there also exists an optimal

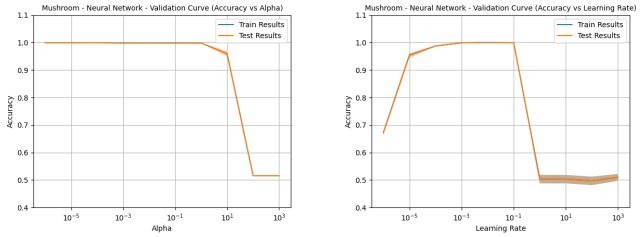

Fig. 12: Mushroom Dataset - Neural Network Validation Curves. From left to right: (a) Alpha (b) Learning Rate

learning rate that provides the maximum accuracy of 100%. At around $10^{-}2$.

Figure 13a shows the effect of alternating the number of neurons within our MLPClassifier. Since we know that the dataset is already clean and easily separable, as indicated from our KNN analysis, we expect that changing the intricacy of the neurons (nodes) will not make a significant difference, which is indicated by the results we got here. We can even see the slight increase in accuracy as we increase the neurons in our network. At around 6 neurons, the test results converge with our training results at an approx. 100% accuracy. Figure 13b showcases the different activation functions yielding relatively similar results in accuracy, suggesting that the data is of good quality and separable to the point where it is easy splitable with any activation function.

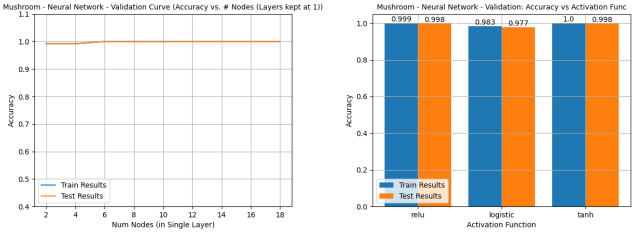

Fig. 13: Mushroom Dataset - Neural Network. From left to right: (a) Single layer NN validation curve for rice dataset alternating number of neurons (b) Accuracy vs Activation Function

We will then reoptimize using GridSearchCV to determine the most optimal alpha, learning rate, number of neurons, and activation parameters. By optimizing these values, we obtained the following values: activation = relu, alpha = 0.0001, neurons = 6, initial learning rate = 0.001. The low alpha and learning rate suggests that the tolerance for weights can be quite relaxed, indicating that the data is of good quality, and that the model performs well without any sign of overfitting. This optimization correlates well with our previous finding that the most optimal K value was 1. Meaning that the training and testing data were so consistent with one another, it was easy for the learners to pick up on the pattern to determine the outcome of the mushroom.

Finally, figure 14a and 14b depicts the final learning curve of the optimized neural network and its associated loss curve. As we can see from the learning curve, the testing data closely aligns with the training data throughout the entire training journey, and converges tightly at 100%, indicating a sign of a well trained and optimized model. This is also reflected in the training loss curve, where it steadily approaches and converges to 0, indicating that there was 0 training loss, which supports our evidence that the accuracy is at 100%. One thing that is worth mentioning is how quickly the learner converged to zero error - at the approx. 125 epoch it already reached an optimal solution. That speaks to how insanely clean and of high quality the data is, and how easy it was for the learners to establish a clear and effective boundary to perform the binary classification.

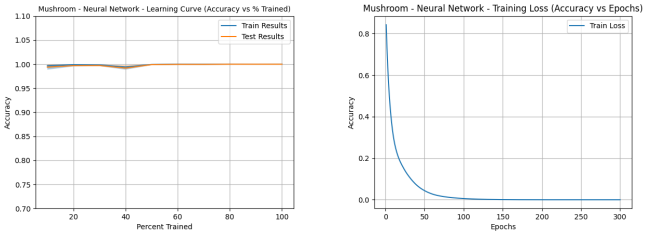

Fig. 14: Mushroom Dataset - Neural Network. From left to right: (a) Learning Curve (b) Loss Curve

V. Support Vector Machines (SVM)

The intent for a Support Vector Machine (SVM) provides a more robust learner, optimizing the hyperplane that maximizes the distance between different outputs within the features. Knowing that it is a more robust version than KNN, we predict that both the rice and the mushroom data will perform well here.

A. Dataset 1 - Rice

For our rice data, we will reuse our approaches in the previous analysis for consistency. This means we will first create an arbitrary SVM using sklearn's SVM library and vary a parameter while holding the other constant. The two hyperparameters I will analyze are the C value, and the kernel type (more explanations on what these mean below). Figure 15a depicts the results from holding each variable constant with a default SVM learner (linear kernel), while 15b builds/trains three learners, each with different kernels, containing default C values, and attempts to fit the test data and ultimately calculate its accuracy. As usual, we split the data using a 5 fold cross validation strategy in a 70-30 train to test split.

C values balances the margin in the hyperplane that divides the two classes of data, while also doing it in such a way for it to be general enough without overfitting. Similar to the alpha term in neural networks, the C value regularizes the margin - a large C value will tightly fit the training data more closely but runs the risk of overfitting, while a small C value will loosely fit

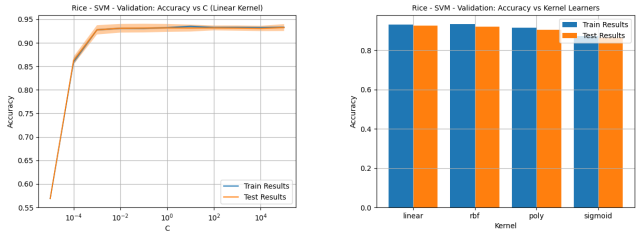

Fig. 15: Rice Dataset - SVM Validation Curves. From left to right: (a) Accuracy vs C (b) Accuracy vs Kernel

but risks misclassifications and errors. In our example, we can see that there is a nice convergence in test/train as C values increase, with high amounts of accuracy as well.

Kernels dictate the decision boundary and heavily influences the hyperplane shape that segregates classification data. Therefore, it is important we compare and contrast the differences obtained in the kernel methods used. According to the figure, it looks like all of them yielded pretty high results, with linear and rbf (gaussian) taking the lead in the highest accuracy. We think this makes sense and aligns with our other findings, as we know that the dataset seems to be simple enough for the learners to split the data, and therefore, it would not require a complicated kernel function like polynomial to segregate our data.

The next step is to use GridSearchCV to optimize these hyperparameters. Using a C grid range and the kernels that were discussed earlier, which combination will yield the highest accuracy according to our data? After passing through these functions, the results show that a C value of 1 and an rbf kernel generated the best results, yielding an accuracy of 92.5%, which is still a very good score. The learning curve with these optimized parameters are plotted below in figure 16. The figure shows a steady increase in the accuracy as the testing dataset is being trained and ultimately converges with the training dataset, which is a good indication that it is a good/optimized model with proper selected features.

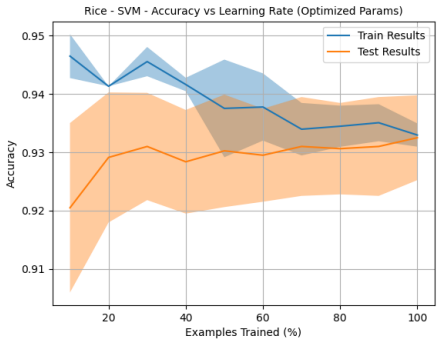

Fig. 16: SVM Learning curve for rice dataset with optimized parameters.

B. Dataset 2 - Mushroom

The same approach will also be used in analyzing and optimizing SVMs for our mushroom dataset. We employed a 5-fold cross-validation strategy, partitioning the data into a 70-30 train-test split. First, we analyze the C and the kernel parameters respectively, highlighted by figure 17a and b below.

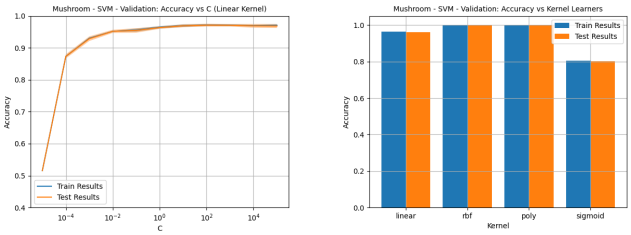

Fig. 17: Mushroom Dataset - SVM Validation Curves. From left to right: (a) Accuracy vs C (b) Accuracy vs Kernel

First starting with figure 17a, which compares the C values with accuracy. Since we played with the C values in a SVM learner that used a linear kernel, the performance slightly underperformed in comparison to the KNN and Neural Network learners we used in the past. Upon observing and comparing figure 17b, it looks like this is because there is a dip in performance using a linear kernel in comparison to a gaussian or polynomial kernel. I would suspect this may be because of the high dimensionality of features the mushroom contains. It is a rather complex dataset, but is so highly correlated and of such high/consistent quality that it can be trained easily and be super effective. That being said however, my theory would be because of its "complexity" in features, causing a non-linear separation that requires a higher order kernel.

We will now proceed to optimize the data just like what was done in the rice dataset using GridsearchCV. The same parameters were tuned just like in the rice dataset, and the algorithm output a C value of 1 and an rbf kernel type, yielding an increased accuracy to 100%. The mushroom dataset is once again displaying its high quality data and proving it through highly efficient learners across our analysis journey throughout all three systems. As expected, the dip in performance observed in figure 17a was rectified with the introduction of a rbf kernel, which showed 100% accuracy in figure 17b, which justifies the validity of the optimization. The learning curve is finally created by using a SVM learner with these optimized parameters plugged in. The results of the learning curve are depicted in figure 18.

Similar to the other learning curves in the mushroom dataset, the training dataset converged nicely with the training results at 100% accuracy, which proves it to be a well trained model. One noticeable difference is how much longer it took the testing data to converge. This leads us to the next section of comparing our algorithms performance in the next section.

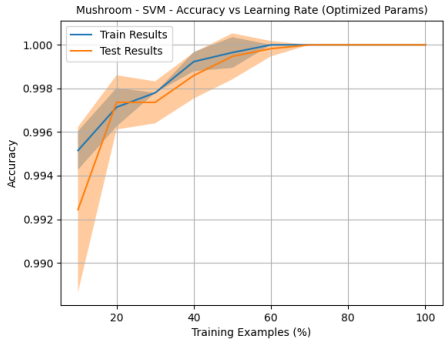

Fig. 18: SVM Learning curve for mushroom dataset with optimized parameters.

VI. Comparison of Algorithms

Table 1 below shows all of the accuracies and training time for a specific learner. What we will do here is compare the different learners and select the best one given its tradeoffs.

TABLE I: Learner performance comparison

Dataset	Learner	Accuracy (%)	Train Time (s)
Rice	Neural Net.	92.39%	2399
Rice	KNN	92.30%	34
Rice	SVC	92.48%	635
Mushroom	Neural Net	99.96%	5190
Mushroom	KNN	100%	131
Mushroom	SVC	100%	6989

Upon first glance at the model, neural networks and SVMs takes much longer time to train in comparison to KNN. As stated previously, KNN is the simplest and most intuitive model to train. In fact, there should not be too much training involved, as KNN only utilizes training data to store in a storage and memorizes its coordinates (traits). A neural network will require a lot of training to optimize for weights via gradient descent, and involve multiple calculations to account for loss. SVMs require optimizing the hyperplane to most efficiently separate the classes and be computationally intensive with large datasets.

Now comparing this with performance. With both datasets being very effective in building predictive models, they all yielded very similar accuracies - rice dataset yielded approx. 92.3% accuracy, whereas the Mushroom dataset yielded 100% accuracy. Given that the performance is quite similar, it would be most optimal to choose the KNN learner for both datasets given the significant reduction in runtime to build this model.

Going deeper another level, the cleanliness and consistency (little/no noise/outliers) in both datasets makes it a good candidate for a relatively simple learner like KNN. SVMs and Neural Network really plays more of a role in which data can be more complex, or when there is more noise in the data. Other supporting evidence can be found in the learning curves for the models. For example, when comparing figure 18 (Mushroom SVM learning curve) with figure 8 (Mushroom KNN learning curve), the convergence was much earlier compared to SVM, which maybe points out that this system may be overly robust for our dataset. Simplicity is the key here.

VII. Conclusion

The results of the experiment went quite well for both datasets. Due to the nature of a binary classification problem, combined with the cleanliness and quality of the data, we were able to utilize both dataset's great features and developed highly performant learners from it, validating our hypothesis that both datasets would yield good results in the learners. Time and time again, however, the mushroom dataset proves to be much more accurate and easy to train throughout all three systems and consistently yielding a higher accuracy. Because of this, our other hypothesis of it being the most performant out of the two is also verified. We believe this makes sense, because upon qualitatively analyzing the traits of mushrooms seems more diverse than just measuring the size of the rice. Each individual feature in the mushroom dataset provided specific traits that would naturally seem like an easier time to narrow down in terms of the species of the fungi, and thus its edibility. In the rice dataset, however, we are really only measuring physical qualities of the grains of rice, which can always present anomalies from it being more prone to outliers (genetic spread of rice shape). The mushroom dataset contains mini traits that are not strong indicators individually (as seen from figure 2), but perhaps an ensemble of them working together will yield extraordinary results like what was demonstrated here.

One way to verify this further and further enhance this experiment is to introduce external data that was completely unseen to the learner. By doing a 5 fold cross validation, we are still technically reusing a lot of samples that were previously trained before. If we inject new data as the training data, I would be curious to see how that would perform. Another thing to note was that we were not given unlimited compute power to fully optimize everything. For example, in neural networks, my computer was having a hard time trying to optimize the number of layers as well as neurons, running into memory and ultimately crashing the system when I attempted to solve it. Being able to optimize for everything can truly allow us to get the best model given our dataset.

VIII. Resources

[1] *API Reference.* Scikit-learn. https://scikit-learn.org.
[2] Nakamura, K. (2023). *ML LaTeX Template*.