Exploring Efficiency Amongst Supervised Models

[redacted], Jaxon Hornsey, [redacted]
Dept. of Computer of Science
University of Windsor, Windsor, Canada
{[redacted] hornseyj, [redacted]} @uwindsor.ca

***Abstract*— The need for efficient supervised models increases by the day as more complex problems are introduced and datasets continue to grow. With the vast amount of these models, it can become difficult to find the ideal algorithm for a chosen dataset let alone the most fitting hyperparameters to best suit your data. Experimentation using four popular supervised models was conducted to emphasize the advantages of using the cross-validation technique to determine the models best hyperparameters. To determine the best model for a general dataset all calculations were performed on one common dataset with the intention of analyzing and comparing their speeds and accuracies. This comparison was done between decision trees, k-nearest neighbors, support vector machines, and multi-layer perceptron using scikit-learn, a machine learning library. In essence, decision trees dominated both categories by a large margin and can be declared as the most efficient. Following this was the k-nearest neighbors coming at second, support vector machines in third place, and finally, multi-layer perceptron as the least efficient in both categories. In light of these results, the conclusion can be made that for a simple classification dataset, it is best to use a decision tree model over the other popular supervised models, however, it is crucial to modify hyper-parameters for maximum efficiency.**

I. INTRODUCTION

As more complex problems become easier to solve with the constant advancement of Artificial Intelligence (AI), the creation of algorithms that can utilize different types of data also grows. These numerous algorithms all have their own individual approach to a dataset, however, when they all perform on one dataset, the different accuracies and timings must be accounted for to see which one is the most efficient. Thus, identifying which algorithm should be used for a specific dataset that needs to be analyzed must be chosen attentively for maximum accuracy and speed.

The purpose of this report is to analyze and compare a variety of supervised models on a dataset that classifies if a mushroom is edible or poisonous. With the most efficient algorithm, we will be able to determine the best AI-supervised model to be used for different labelled datasets. To compare, we will look at the accuracy of each model using its best hyper-parameters and then compare them all with a cross-validation model. The variety of AI algorithms that will be used includes support vector machines, neural networks (perceptron), decision trees, and k-nearest neighbors. These algorithms will be tested in a simulated environment and modelled for proper analysis in terms of speed and accuracy.

II. RELEVANT LITERATURE REVIEW

With the various AI models that are used for modern day problems, it would come to no surprise how many experiments are done regarding efficiency comparisons between each algorithm. One book, Applied Predictive Modeling by Max Kuhn, and Kjell Johnson [1], expands on the practice of predictive modelling. From data-preprocessing to regression and classification models, the authors give a detailed outlook on study cases, measuring performances and even studying factors that may affect a model's performance. However, with the popularity of python in the span of a few years, the libraries used in the department of research have also changed. Libraries such as TensorFlow and scikit-learn have dominated in recent resources and tutorials found online. Companies such as IBM have even set up their own tutorials [2] so anyone can easily learn the implementations of AI models to solve machine learning problems.

III. EXPERIMENTAL SETUP & METHODOLOGY

For experimentation, we decided to use mushroom data that would classify whether a mushroom was edible or poisonous. The input dataset about mushroom classification was picked from the UCI Machine Learning repository. This dataset includes descriptions of hypothetical samples corresponding to 23 species of mushrooms [3].

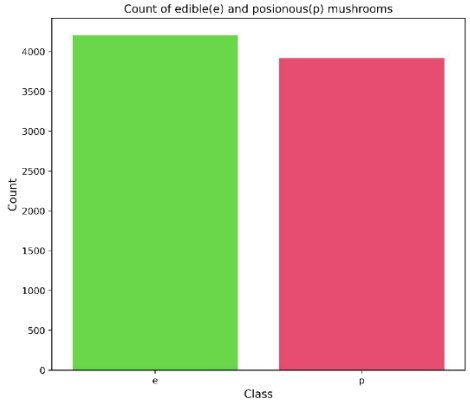

Fig. 1: Bar chart of mushroom class

From Fig.1 we can see that out of 8124 samples, 4208 are edible and the rest 3916 are poisonous.

The environment that was used for the experimentation of the different AI models used Python as its programming language. For all experiments, the Scikit-library library along with Pandas and NumPy libraries were used []. Each experiment of a different AI model uses the same dataset, the mushroom data [3]. The scikit-library library contains all the supervised models that will be implemented for the analyzation of the data. To hold the data and the data frame, we will be using the Pandas data frame. Finally, the NumPy library will be used to flatten the array of values when filtering the dataset. One computer will be used to run all the experiments, so the time is not altered based on attribute of different computers. As mentioned before, this paper is to find the best hyper-parameters for each supervised model and then analyze and compare them based on their speed and accuracy.

From the experimentation on the data, we will question which supervised model is the fastest and which is the most accurate. These two questions will then be expanded upon with the inquiry of whether our answers would change based on a smaller and larger dataset than our own. Furthermore, we will also speculate on why the chosen hyper-parameters were the best for each of the models.

A. *K Fold Cross Validation*

To test our machine learning models against each other, we made use of k-fold cross-validation model.

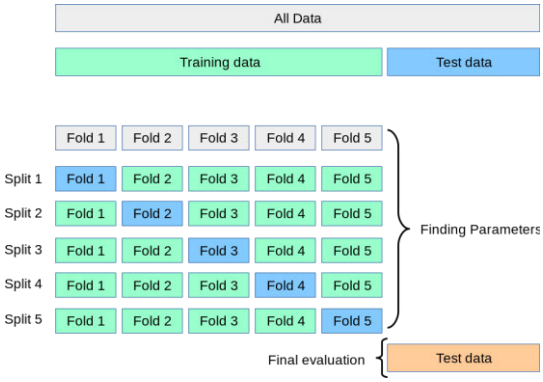

Fig. 2: displays 5 iterations done by K-fold cross-validation

This statistical method iterates over different training sets based on the iteration value given. Each iteration is a different training set that is part of the training data as seen in [7, Fig. 2]. The number of iterations can be changed as a parameter when calling the method and usually is the default value of 5. We changed this value from 5 to 10 to make the bias of the technique smaller without the variance being too high.

The Scikit-library provides a function called GridSearchCV. This function can have several parameters passed through it but in this paper, will be using 3. These parameters are model type, parameter grid, and the number of folds. For each model the Scikit-library provides, the parameter list for each hyper-parameter varies and we want to explore which values are the most optimized set.

B. *Models Setup*

1. *Decision Trees Setup*

a) *Decision Tree:*

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features [4]. Decision trees are a simple yet very effective method which frequently are used to solve classification and regression problems. It has a couple of models. The tree model that predicts a discrete variable is called as classification tree. The leaves of the tree represent class labels and the branches represent conjunctions of features that result into the class labels. The tree model that predicts a variable with continuous data is called as regression tree.

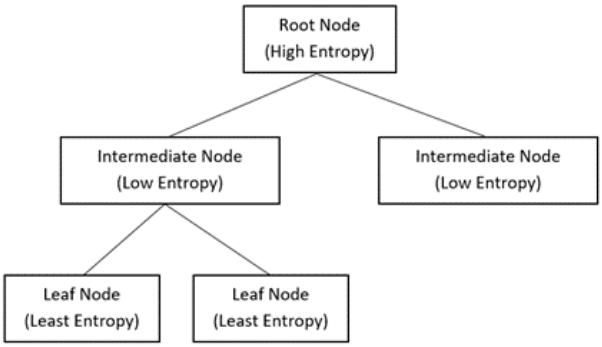

Fig. 3: A representation of a Decision Tree

The dataset is split based on classification features on each level of the tree. Each leaf of the tree is labelled with a class or a probability distribution over a class. Entropy is a measure of randomness in the dataset. Information gain is a measure of reduction in entropy after the dataset is split. Higher information gain is expected to get higher accuracy from the model.

b) *Decision Trees in Scikit-learn*:

The Decision Tree algorithm is implemented in the 'DecisionTreeClassifier' class of Scikit-library libraries. DecisionTreeClassifier can perform multiclass classification on a dataset. To utilize this class, a variable is assigned to call DecisionTreeClassifier class methods. This function has several hyperparameters, and we selected four parameters which are gini, splitter, max_depth and min_samples_split.

```
param_grid = {'criterion': ['gini','entropy'],
              'splitter': ['best','random'],
              'max_depth': [5,10,15],
              'min_samples_split': [4,8,16,32]}
```

Fig. 3.1: parameters to be exhaustively searched for the most efficient Decision Tree

Different hyperparameters and their combination of values alter the behavior of decision tree and its prediction

accuracy. The below-mentioned hyperparameters are tweaked and the accuracies are evaluated.

1. **criterion**: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
   Grid Search Options: ['gini','entropy']
2. **splitter:** The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.
   Grid Search Options: ['best','random']
3. **max_depth**: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
   Grid Search Options: [5,10,15]
4. **min_samples_split:** The minimum number of samples required to split an internal node.
   Grid Search Options: [4,8,16,32]

After finding the most efficient parameters, the fitting method will then be called taking the training variables as arguments. Finally, the prediction method takes a new set of data as its only parameter which is the testing data.

2. *K-Nearest Neighbors Setup*

a) *K-Nearest Neighbors:*

The K-Nearest Neighbors Algorithm can be used for various classification applications [14]. The important thing when applying this algorithm is to use supervised data as the algorithm is based off the supervised learning technique [12]. The supervised learning technique relies on labelled data; the algorithm will develop a function using the labelled data that will then predict new unlabeled data [13]. K-Nearest Neighbors (k-NN) is also a non-parametric algorithm [13].

The K-Nearest Neighbors Algorithm followed in our paper can generally be summarized by the following steps:

1. Pick a value k for the number of neighbors.
2. Find the Minkowski distance between k numbers of neighbors.
3. Use K nearest neighbors from part 2.
4. Count the classes from K neighbors.
5. Give the new unlabeled data the class that is most dominant with K neighbors.

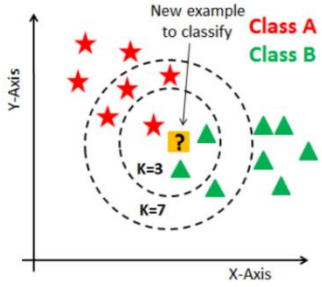

Fig. 4: Unlabeled data point being classified. [15]

Figure 4 is demonstrating how the unlabeled value will be given its class based on the dominant class within K-neighbors.

b) *K-Nearest Neighbors in Scikit-learn:*

To implement this algorithm the scikit-learn library was used. Similar steps were followed in the methodology for preparing our data for processing. Scikit-learn provides a function called KNeighborsClassifier this function has several parameters including:

1. **N_neighbors:** total number of neighbors to use | 1 to 25
2. **Weights:** a function used in prediction | Default=uniform
3. **Algorithm:** used to compute the nearest neighbors | Default = auto
4. **Leaf_size:** Can affect speed of construction and query | Default =30
5. **P:** power parameter for Minkowski metric | Default = euclidean_distance
6. **Metric:** The distant metric to use | Default = minkowski

Looking at some of the default parameters for this algorithm, we can see that the Minkowski will be using Euclidean distance which is appropriate for our task since we are using vector values. The parameter ‘algorithm’ changes the way the nearest neighbors are calculated, this library happily provides us with the best one automatically. For the sake of generalization, we will be working with the defaults the Scikit-learn has provided. The main parameter we will be focusing on for this algorithm is the number of neighbors used. N_neighbors is the only parameter because this factor has the largest effect over all other parameters. Since we are provided with default parameters, we will use them. Otherwise, there would be an unobservable difference in accuracy along with a drastic increase in run time.

When determining the best value for K there are some key things to keep in mind.

- If you are working with the same training set your results may vary dramatically, this is where k-folds cross-validation comes in
- The training size percentage will influence how well your algorithm is trained.

Due to these factors, the use of the GridSearchCV function provided by Scikit-learn, and its ability to provide the best hyperparameters is what makes it so useful in this situation.

3. *Support Vector Machines Setup*

a) *Supper Vector Machine:*

Support Vector Machines: Support Vector Machines (SVM) are supervised learning models that are used in this paper to analyze the data for classification. The goal of the algorithm is to find the optimal hyperplane that will be used to classify the new data points. There are two types of SVM models, one being linear and the other being non-linear.

The linear SVM model first selects two hyperplanes that separate the data with no points. After this, it defines the margin which is the width that the boundary could be

increased before hitting a data point. Finally, the decision boundary will be determined by taking the average lines between the two dashed lines. This can be seen in [29, Fig.5]

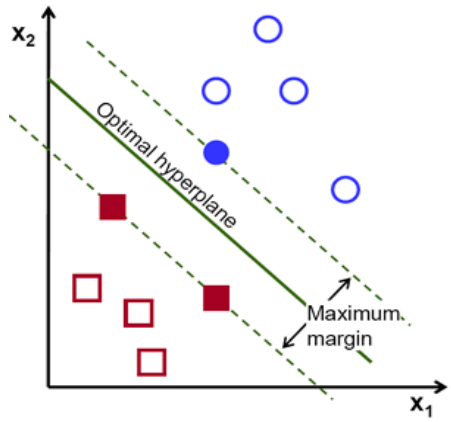

Fig. 5: A linear support vector machine

Non-linear SVM models are specified using kernel functions. When the data cannot be separated by a straight line, these models are used. By changing the data into a higher dimension, the non-linear spaces turn into linear spaces so that the data can be classified.

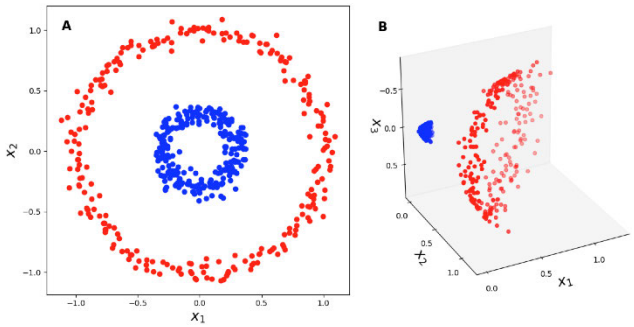

Fig. 6: A representation of a non-linear SVM model (radius basis function)

There are many kinds of kernel functions, ranging from radius basis function (RBF) [28, Fig.6] to polynomial. The difference between these kernel functions is their mathematical approaches. They make different decisions on the hyperplane decision boundaries between classes.

b) Support Vector Machines in Scikit-learn:

To use the support vector classifier class in the sklearn library, a variable must first be assigned to the calling of the built-in SVC function. This function has three parameters which are the kernel, the gamma, and the value of C. *After finding the most efficient parameters*, the fitting method will then be called taking the training variables as arguments. Finally, the prediction method takes a new set of data as its only parameter which is the testing data. A variable will then store the returned label for each object in the array its holding.

4. MultiLayer Perceptron Setup

a) Multilayer Perceptron:

Multilayer Perceptron [23] algorithm is an artificial neural network. Neural networks are a way of creating simple and effective models of biological brains for use in solving challenging calculations [27]. In this case, the difficult task is predicting whether a mushroom is edible or not from a supervised training set. The building blocks are neurons, weights and activation functions [23].

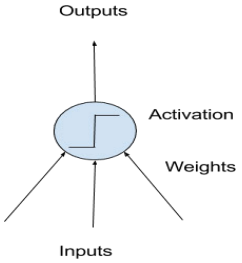

Fig. 7: Artificial nodes. [23]

The artificial nodes Fig.7 can be described as neurons, these are made up of weights and an activation function [23]. All nodes except input ones contain an activation function and a neuron weight [20]. Activation functions are used in determining the output value of a neural network [25]. The neuron weight is a determined value that multiplies the value of the input data within the hidden layers of the neural network [27]. The neurons will be arranged in a multi-layer format [23]. This neural network works by making connections between multiple layers of nodes within a directed graph [20]. What this means is that in between each layer the nodes are only able to go one way [20]. The way this algorithm predicts the class consists of 2 general phases. These phases are called feed-forward and backpropagation [20]. During feed-forward values are: 1. put into the input layer where their weights are altered 2. passed to the next layer until the outer layer is reached where the calculated value is the class output [20]. Back Propagation has 2 parts: 1. The error rate is calculated by computing the difference between the predicted and actual label, this attribute is called “loss function” 2. Minimize the error for the next output by updating neuron weights respectively [25]. When the neural network has completed these 2 phases cycle this process is called “epoch” [25]. Epoch will continue until a good accuracy percentage is achieved [25].

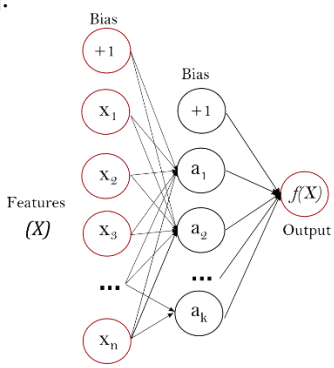

Fig. 8: Example of one hidden layer MLP [22]

Through Fig.8 We can see an example of a directed graph with one hidden layer. The leftmost layer is known as the input layer, this is made up of a set of neurons, and represents different input attributes [26]. Each node in the

hidden layer (outlined in black) will transform the values as mentioned.

b) Multilayer Perceptron in Scikit-learn:

MLP was also set up using the Scikit-learn library. The dataset we are using will be scaled due to MLP sensitivity to feature scaling [21]. The function StandardScaler will be used on both the test and training set for standardization [21]. Scikit-learn provides a function called MPLClassifier. This is a multi-layer perceptron algorithm that trains using backpropagation [26]. This function trains 2 arrays: array X contains the training sample and array Y holds the target class [26].

1. **Hidden_layer_sizes**: Number of neurons in the it hidden layer
2. **Grid Search Options**: [(150,100,50), (120,80,40), (100,50,30)]
3. **Max_iter:** Max number of iterations
   Grid Search Options: [50, 100, 150]
4. **Activation**: Activation function for hidden layer
   Grid Search Options: ['tanh', 'relu']
5. **Solver:** The way the weight is optimized
   Grid Search Options : ['sgd', 'adam']
6. **Alpha:** L2 Penalty Parameter
   Grid Search Options: [0.0001, 0.05]
7. **Learning_Rate:** Schedules for weight updates
   Grid Search Options: ['constant', 'adaptive'] [scikit]

IV. RESEARCH & DISCUSSION

A. *Impact of hyperparameters*

1. *Decision Trees:*

Variation in accuracy is observed with different hyperparameter setups of the decision tree. Provided is a pair of sample setups that illustrate the observation.

The *first setup has the following hyperparameters:*

1. **Criterion***: 'entropy'*
2. **max_depth:** 5
3. **min_samples_split:** 4
4. **Splitter:** 'best'

The second setup has the following hyperparameters:

1. **Criterion:** 'gini'
2. **max_depth:** 15
3. **min_samples_split:** 8
4. **Splitter:** 'random'

The "entropy" criterion, together with max_depth of just 5, min_samples_split of just 4 and 'best' splitter strategy, resulted in a lower accuracy of 97.35%. This combination falsely classified 34 poisonous mushrooms and 9 edible mushrooms, as shown in Fig. 9. The precision obtained is shown in Fig. 10, which is lower compared to the best hyperparameter combination of setup-2.

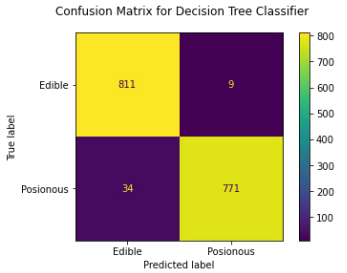

Fig. 9: Confusion Matrix (setup-1), decision trees

	precision	recall	f1-score	support
0.0	0.96	0.99	0.97	820
1.0	0.99	0.96	0.97	805
accuracy			0.97	1625
macro avg	0.97	0.97	0.97	1625
weighted avg	0.97	0.97	0.97	1625

Fig. 10: Classification Report (setup-1), decision trees

The "gini" criterion (signifying Gini impurity), together with max_depth of 15, min_samples_split of 8 and 'random' splitter strategy, resulted in the highest accuracy of 100%, with 0 incorrect classifications of both poisonous and edible mushrooms, as shown in Fig. 11. This hyperparameter combination also resulted in the highest precision of 100%, as shown in Fig. 12.

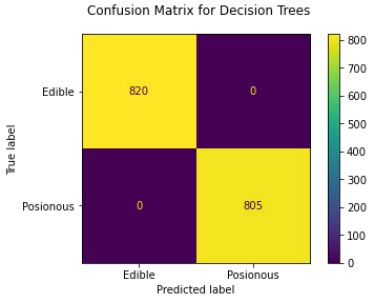

Fig. 11: Confusion Matrix (setup-2), decision trees

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	820
1.0	1.00	1.00	1.00	805
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Fig. 12: Classification Report (setup 2), decision trees

2. *K-Nearest Neighbor Model:*

The K nearest neighbor function tested on a set of 20% of our dataset. Both graphs are a result from training our function on the same data set, while altering the k value each version. Each graph uses a different training set. This is to emphasize the difference the k value can make.

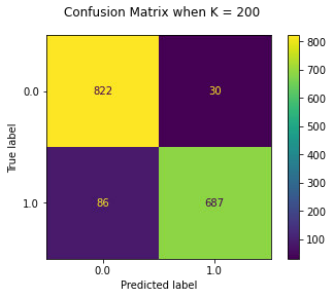

Fig. 13: Confusion Matrix (setup-1), k-nearest Neighbors

The k value within Fig 13 is set at 200 to show the effects of a higher k value on this dataset. We can see that the K nearest Neighbor algorithm incorrectly classified 86 poisonous mushrooms and 30 edible mushrooms. This test set also took 639ms.

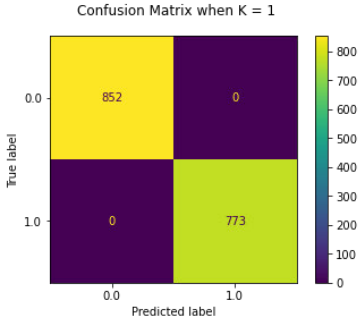

Fig. 14: Confusion Matrix (setup-2), k-nearest Neighbors

The k value within Fig.14 is set to 1. We can clearly see how meaningful the k value is to this method. It is also clear that a lower K value for this set of data will suit it better. When the k value parameter is set to 1 there are no wrong classifications. This is an indication there may be a fault in using solely one test set. This test set also took 443 Ms.

	precision	recall	f1-score	support
0.0	0.91	0.96	0.93	852
1.0	0.96	0.89	0.92	773
accuracy			0.93	1625
macro avg	0.93	0.93	0.93	1625
weighted avg	0.93	0.93	0.93	1625

Fig. 15: Classification Report (setup-1), k-nearest Neighbors

From the classification table presented in Fig. 15 we can see that we accurately predicted poisonous mushrooms 91% of the time vs 96% for edible mushrooms. This results in an average accuracy of 93%.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	852
1.0	1.00	1.00	1.00	773
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Fig. 16: Classification Report (setup-2), k-nearest Neighbors

In Fig.16 we can see that we have accurately predicted all the mushrooms correctly within our test set. Which is 7% better than when the k parameter is equal to 200. The random training set the algorithm selected could have been due to luck. We should also be observant of accuracy and keep in mind that these values may also be skewed by a lucky training set.

3. Support Vector Machine Model:

To compare how big of an impact hyperparameters can have on an SVM model, we will compare two sets of different hyperparameters. The variables for the first being:

1. kernel= 'linear'
2. C=1000
3. Gamma=0.01

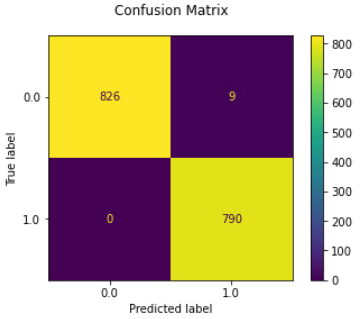

Fig. 17: Confusion Matrix (setup 1): SVM

From the Fig.17 we can see that changing the C and Gamma to 1000 and 0.01 respectively, will have accurate predictions. However, this test took 173.511 seconds. The kernel parameter does not make the SVM model any slower as a linear SVM model is known to do the opposite. Because of this, we can establish that the C and gamma parameters have a huge role in computing time.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	835
1.0	1.00	1.00	1.00	790
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Fig. 18: Classification Report (setup 2), SVM

Fig.18 displays the perfectly accurate results of the extremely slow test.

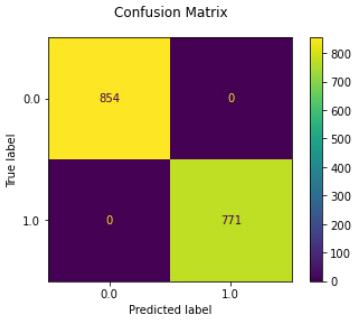

Fig. 19: Confusion Matrix (setup 2), SVM

For Fig. 19, changing the hyperparameter C to a much lower number and gamma to a higher, the accuracy remains the same. However, when this change occurs, the speed of the test changes to 1.7135 seconds.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	854
1.0	1.00	1.00	1.00	771
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Fig. 20: Classification Report (setup 2), SVM

Here in Fig. 20 we can see the accuracy being relatively the same when compared with the previous set of hyperparameters. The main factor to look out for in the change of hyperparameters in an SVM model will therefore be speed.

4. *Multi-layer Perceptron:*

The parameters for the first setup Fig 21 conducted were:

1. **hidden_layer_sizes:** (150, 100, 50),
2. **Solver:** 'adam',
3. **Alpha:** 0.0001,
4. **batch_size:** 'auto',
5. **learning_rate:** 'invscaling',
6. **max_iter:** 100,
7. **learning_rate_init:** .18

For the second setup Fig.23 the parameters were:

1. **Hidden_layer_sizes:** (150, 100, 50)
2. **Max_iter:** 100
3. **Activation:** tanh
4. **Solver:** adam
5. **Alpha:** 0001
6. **Learing_rate_init:** 0.001
7. Learning Rate: constant

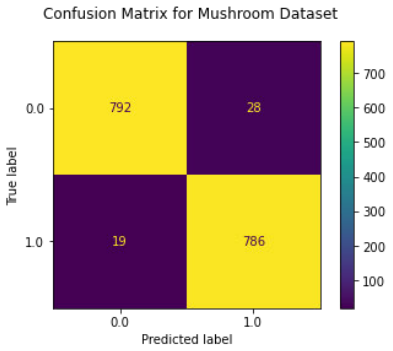

Fig. 21: Confusion Matrix (setup-1), multi-layer perceptron

As seen in Fig.21 when the learning_rate_init factor is changed, it starts to have a significant effect on the way the algorithm classifies mushrooms. For this algorithm to compute it took 2.17 Seconds.

	precision	recall	f1-score	support
0.0	0.98	0.97	0.97	820
1.0	0.97	0.98	0.97	805
accuracy			0.97	1625
macro avg	0.97	0.97	0.97	1625
weighted avg	0.97	0.97	0.97	1625

Fig. 22: Classification Report (setup 1), multi-layer perceptron

From the classification table presented in Fig.22 we can see that we correctly predicted the correct class of the mushroom with an average accuracy rate of 97%

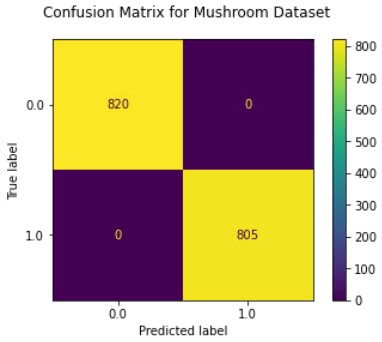

Fig. 23: Confusion Matrix (setup 2), multi-layer perceptron

From Fig.23 we can see that our second setup of parameters works extremely well. This setup was compiled in 1.692 seconds. So, there is no real difference in speed.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	820
1.0	1.00	1.00	1.00	805
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Fig. 24: Classification Report (setup 2), multi-layer perceptron

From the classification table presented in Fig.24 we can see second setup correctly identified mushrooms 100% of the time. Since this was performed on one test set results may be a lucky outcome. But we can see that when the learning rate is small the algorithm becomes more precise, which would make sense.

B. *GridSearchCV*

1. *Decision Trees Model*

The Grid search of the Decision Tree model with different hyperparameters discussed in Section 3 Part A was executed with cross-validation of 10. The exhaustive search yielded the following hyperparameter combination as the best.

1. **Criterion:** 'gini'
2. **max_depth:** 15
3. **min_samples_split:** 8
4. splitter: 'random'

These hyperparameters resulted in an average accuracy of 99.61%.

2. *K-Nearest Neighbors Model*

After running the simulation, the grid search returns a parameter list containing the best parameter values. For this experiment, it was determined that a **K value of 2** would yield the highest accuracy results. The results for classifying our mushroom data using the K-Nearest Neighbors algorithm are 95.792% accurate with an average time of 10.65 seconds

We can also view the difference between the spectrum of K values by cross comparing cross-validation results with different k values. Fig.25 can have a visualization of how K behaves.

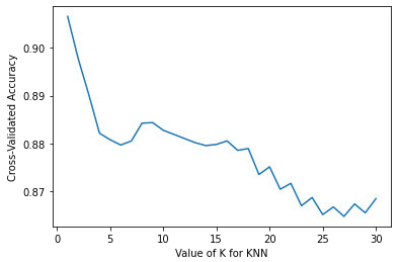

Fig. 25: Showing cross-validation Accuracies for different k values

3. *SVM model*

With the use of the GridSearchCV model, we can exhaustively search over specified parameters to find the most efficient ones. [Fig. 26]

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'linear', 'poly']}
```

Fig. 26: parameters to be exhaustively searched for the most efficient support vector machine

Using the GridSearchCV model, we determine the most efficient SVM would be:

1. **Kernel:** poly
2. **C:** 0.1
3. **Gamma:** 1

4. *Multi-layer perceptron*

After completing the Grid search cross-validation for the Multi-layer Perceptron algorithm with the parameter discussed in Section 3 Part D.

The best parameters are:
Grid Search Cross-Validation Results

1. **Hidden_layer_sizes:** (150, 100, 50)
2. **Max_iter:** 100
3. **Activation:** tanh
4. **Solver:** adam
5. **Alpha:** 0001
6. Learning Rate: constant

These parameters resulted in an average accuracy of

C. *Final Results*

Trials	DECISION TREES (SEC)
TRIAL 1	0.0576
TRIAL 2	0.0561
TRIAL 3	0.0571
TRIAL 4	0.0542
TRIAL 5	0.0554
AVERAGE	0.05608

Fig. 27.1: The best hyperparameter for the decision tree algorithm had an average time of 56 ms

Trials	K-NEAREST NEIGHBOUR (SEC)
TRIAL 1	0.9689
TRIAL 2	0.9949
TRIAL 3	0.9869
TRIAL 4	0.9558
TRIAL 5	0.9858
AVERAGE	0.97846

Fig. 27.2: The best hyperparameter for the K-Nearest Neighbor algorithm had an average time of 978 ms

Trials	SUPPORT VECTOR MACHINES SPEED (SEC)
TRIAL 1	1.9527
TRIAL 2	1.9766
TRIAL 3	1.9569
TRIAL 4	2.0011
TRIAL 5	1.9765
AVERAGE	1.97276

Fig. 27.3: The best hyperparameter for the Support Vector Machine algorithm had an average time of 1.972 seconds.

Trials	MULTI-LAYER PERCEPTRON SPEED (SEC)
TRIAL 1	18.4851
TRIAL 2	19.2989
TRIAL 3	18.3809
TRIAL 4	18.7939
TRIAL 5	18.729
AVERAGE	18.73756

Fig. 27.4: The best hyperparameter for the decision tree algorithm had an average time of 18.73756

MODELS	SCORES
DECISION TREES	0.9813
K-NEAREST NEIGHBOUR	0.9579
SUPPORT VECTOR MACHINES - POLY	0.9468
MULTI-LAYER PERCEPTRON	0.9449

Fig. 27.5: Model accuracy scores

[Fig. 27.5] shows the resulting accuracy score (out of 1) when each algorithm was running its best possible parameters.

Rankings Based On Score

1. Decision Trees
2. K-Nearest Neighbors
3. Support Vecotr Machine
4. Multi-Layer Perceptron

Ranking Based On Time

1. Decision Trees
2. K-Nearest Neighbors
3. Support Vector Machines
4. Multi-Layer Perceptron

From the rankings, based on score and time we can see that decision trees dominate in both categories. Because of this, if the dataset were larger, the decision trees would not only exceed in timing but also in accuracy. The difference in the decision trees and the next most efficient model is quite a large gap in terms of the factors we are looking for. The speed is nearly 1-second faster. The efficiency is also around 3 percent more accurate. Although these numbers may not seem too big, if these models were to ever be tested on datasets much larger than our mushroom dataset, these numbers would be extremely important as efficiency in research is key. Furthermore, we can declare that multi-layer perceptron would be the most inefficient on this type of data and would be very unreliable in larger datasets.

V. CONCLUSION

By experimenting with different supervised learning algorithms and tweaking each model's hyperparameter, highly efficient models were identified for the dataset under study. We can conclude that for datasets with related attributes and size, the Decision tree is the most effective algorithm to use resulting in an average time of 56ms and an average prediction accuracy of 98.13%. While the most inefficient algorithm to use would be the Multimulti-layer perceptron with an average time of 18.74sec and an average prediction accuracy of 94.49%. These differences result in a 1.038% and time 198.808% difference. We can only imagine what kind of effect this would have on a larger data set. Another conclusion that can be made is that when training data for classification it is always more effective to apply the grid search cross-validation technique for any algorithm since it returns the best hyperparameters for speed and accuracy.

VI. FUTURE WORK

The classification algorithms that we applied to the mushroom dataset could be utilized to build more useful applications. One such application is a mobile app that categorizes mushrooms as either edible or poisonous. A recent CBC article [5] and CDC report [6] pointed out the serious threat posed by mushroom poisoning, which is progressively getting worse. With the proposed mobile app, users would be given options to enter attributes of a mushroom to get predictions about its toxicity. The models discussed in this paper would provide an accurate classification. With the strides made in AI-based image recognition, the app can be tweaked to automatically pull as many attributes as possible. Such an app can help avoid or minimize accidental consumption of poisonous mushrooms. We can thus reduce the number of deaths, hospitalization, the strain on healthcare and sheer suffering caused by accidental ingestion of toxic mushrooms.

VII. REFERENCES

[1] Kuhn, Max, and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2018.

[2] Madhavan, Samaya, and Mark Sturdevant. "Learn Classification Algorithms Using Python and Scikit-Learn." *IBM Developer*, 4 July 2019, https://developer.ibm.com/tutorials/learn-classification-algorithms-using-python-and-scikit-learn/.

[3] UCI Machine Learning Repository: Mushroom Data Set. https://archive.ics.uci.edu/ml/datasets/mushroom. Accessed 25 Apr. 2022.

[4] "1.10. Decision Trees — Scikit-Learn 1.0.2 Documentation." Scikit-Learn, https://scikit-learn.org/stable/modules/tree.html#tree. Accessed 25 Apr. 2022.

[5] Pawson, Chad. "Calls to Health Officials in B.C. over Mushroom Poisonings Hit High in 2020 | CBC News." CBC, 8 Nov. 2021, https://www.cbc.ca/news/canada/british-columbia/mushroom-poisonings-2020-1.6240293.

[6] "Health Care Utilization and Outcomes Associated with Accidental Poisonous Mushroom Ingestions — United States, 2016–2018 | MMWR." Centers for Disease Control and Prevention, 11 Mar. 2021, https://www.cdc.gov/mmwr/volumes/70/wr/mm7010a1 htm .

[7] "3.1. Cross-Validation: Evaluating Estimator Performance — Scikit-Learn 1.0.2 Documentation." *Scikit-Learn*, https://scikit-learn.org/stable/modules/cross_validation html. Accessed 25 Apr. 2022.

[8] Arora, Surbhi. "SVM: Difference between Linear and Non-Linear Models - AITUDE." *Aitude*, 4 Feb. 2020, https://www.aitude.com/svm-difference-between-linear-and-non-linear-models/.

[9] Brownlee, Jason. "A Gentle Introduction to K-Fold Cross-Validation." *Machine Learning Mastery*, 23 May 2018, https://machinelearningmastery.com/k-fold-cross-validation/.

[10] "What Is the Difference Between Bias and Variance?" *Master's in Data Science*, https://www mastersindatascience.org/learning/difference-between-bias-and-variance/#:~:text=A%20model%20with%20high%20variance%20may%20represent%20the%20data%20set,overlooks%20regularities%20in%20the%20data. Accessed 25 Apr. 2022.

[11] Q. Sun, "How to deal with Cross-Validation based on KNN algorithm, Compute AUC based on Naive Bayes algorithm | by Qiping Sun | Medium," *Medium*, May 18, 2018. http://medium.com/@svanillasun/how-to-deal-with-cross-validation-based-on-knn-algorithm-compute-auc-based-on-naive-bayes-ff4b8284cff4 (accessed Apr. 25, 2022).

[12] E. Allibhai, "Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn," *towardsdatascience.com*. https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a (accessed Apr. 25, 2022).

[13] "K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint," *www.javatpoint.com*. http://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning (accessed Apr. 25, 2022).

[14] "sklearn neighbors.KNeighborsClassifier — scikit-learn 1.0.2 documentation," *scikit-learn*. http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier html (accessed Apr. 25, 2022).

[15] "Most Popular Distance Metrics Used in KNN and When to Use Them - KDnuggets," *KDnuggets*. http://www kdnuggets.com/2020/11/most-popular-distance-metrics-knn html (accessed Apr. 25, 2022).

[16] V. Beena, "Understanding Confusion matrix and applying it on KNN-Classifier on Iris Data set.," *ai.plainenglish.io*. https://ai.plainenglish.io/understanding-confusion-matrix-

and-applying-it-on-knn-classifier-on-iris-dataset-b57f85d05cd8 (accessed Apr. 25, 2022).

[17] A. Nair, “A Beginner’s Guide To Scikit-Learn’s MLPClassifier,” *Analytics India Magazine*, Jun. 20, 2019. http://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/ (accessed Apr. 25, 2022).

[18] K. Choudhury, “Deep Neural Multilayer Perceptron (MLP) with Scikit-learn,” *towardsdatascience.com*. https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-with-scikit-learn-2698e7 7155e (accessed Apr. 25, 2022).

[19] “NN - Multi-layer Perceptron Classifier (MLPClassifier) - Michael Fuchs Python,” *Michael Fuchs Python*. http://michael-fuchs-python netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/#loading-the-data (accessed Apr. 25, 2022).

[20] Techopedia, “What is a Multilayer Perceptron (MLP)? - Definition from Techopedia,” *Techopedia.com*, Mar. 30, 2017. http://www.techopedia.com/definition/20879/multilayer-perceptron-mlp (accessed Apr. 25, 2022).

[21] “1.17. Neural network models (supervised) — scikit-learn 1.0.2 documentation,” *scikit-learn*. http://scikit-learn.org/stable/modules/neural_networks_supervised html (accessed Apr. 25, 2022).

[22] “Scikit-Learn - Neural Network,” *CoderzColumn : Developed for Developers by Developers for the betterment of Development.* http://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-neural-network (accessed Apr. 25, 2022).

[23] “Crash Course On Multi-Layer Perceptron Neural Networks,” *Machine Learning Mastery*, May 16, 2016. https://machinelearningmastery.com/neural-networks-crash-course/ (accessed Apr. 25, 2022).

[24] S. Sharma, “Activation Functions in Neural Networks,” *towardsdatascience*, Aug. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accessed Apr. 25, 2022).

[25] S. Robinson, “Introduction to Neural Networks with Scikit-Learn,” *Stack Abuse*, Jan. 29, 2018. http://stackabuse.com/introduction-to-neural-networks-with-scikit-learn/ (accessed Apr. 25, 2022).

[26] “NN - Multi-layer Perceptron Classifier (MLPClassifier) - Michael Fuchs Python,” *Michael Fuchs Python*. http://michael-fuchs-python netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/ (accessed Apr. 25, 2022).

[27] DeepAI, “Weight (Artificial Neural Network) Definition | DeepAI,” *DeepAI*, 5AD. https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network (accessed Apr. 25, 2022).

[28] G. Gundersen, “Implicit Lifting and the Kernel Trick,” *Gregory Gundersen*, Dec. 10, 2019. https://gregorygundersen.com/blog/2019/12/10/kernel-trick/ (accessed Apr. 25, 2022).

[29] D. Bhalla, “Support Vector Machine Simplified using R,” *ListenData*. https://www.listendata.com/2017/01/support-vector-machine-in-r-tutorial html (accessed Apr. 25, 2022).