Edible Mushroom Classification

Zeynep Ferah Akkurt
150119824

Merve Rana Kızıl
150119825

***Abstract*—The aim of the project is to estimate if a mushroom is safe to eat or poisoned. We perform mushroom classification depend on its cap, gill and stalk features i.e., color, surface, size with several machine learning algorithms.**

Keywords—Mushroom, Machine Learning, Classification, Logistic Regression, k-NN, decision trees, naive Bayes, support vector machines, neural networks

I. INTRODUCTION

Mushrooms are used in the meals due to their tastiness. However, not all the mushrooms are edible and there are still various mushroom species which their edibility has not been defined. Therefore, the automatic classification of mushrooms is a valuable thing. Many people die because of poisonous mushrooms every year. Automatic mushroom classification may play an important role to decrease the food poisoning, hence the aforesaid mortality. [1]

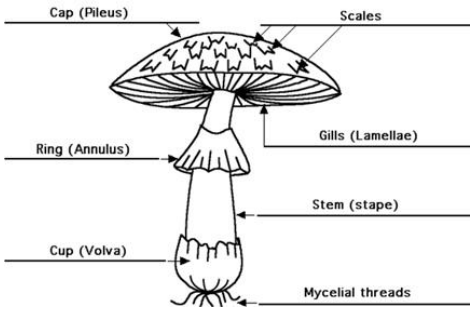

Fig. 1. Parts of a mushroom.[2]

Appearance features e.g., cap shape, stalk, color of mushrooms (Fig. 1), are used to determine whether a mushroom is edible or not. [1]

II. TASKS

Section two defines tasks of the process for this project and the steps that needs to be followed to accomplish. Each step is substantial to reach a satisfied success.

The first step begins with a deep analysis for how to proceed in this project by searching the methods, understanding the dataset and how to use that dataset properly etc. The second step involves the design part which will be about the algorithms and how to use them for the solution of mushroom classification problem. After these two steps, coding and testing will take place and at the end, a conclusion will be made. Hopefully all these steps will be accomplished in a sufficient way.

A. Workflow

The workflow in this project goes as follows:

1. Define the problem (project subject)
2. Initial literature search
3. Examine and analysis of dataset
4. Design
5. Pre-Processing
   a. One-hot-encoder
   b. Label encoding
6. Implementation of different algorithms
   a. k-NN
   b. Random Forest
   c. Naive Bayes
   d. Support Vector Machines
   e. Neural Network
7. Visualization of classification results
8. Comparison of methods
9. Conclusion

First three subtask's aim is to be familiar with the problem and its concepts. After examining the dataset for classification of mushrooms (explained in detail in section 3 as *Dataset and features*), we plan to use five different algorithms (defined in part 5 as *Methods*) to achieve sufficient results with good accuracy. Each task is accomplished with contributions of each member equally.

III. DATASET AND FEATURES

This project uses the dataset which is obtained from Kaggle.com website as Mushroom Data set (UCI Machine Learning Repository). It includes descriptions of hypothetical samples corresponding to 23 species and each species is identified as edible, poisonous, or of unknown edibility and not recommended. This unknown type is also considered and combined with the class of poisonous one. However, there is no simple rule for determining the edibility of a mushroom.

The total number of data is as much as 8124, which contains each attribute's initials. Representative data can be seen in Table I and an explanation for some of them in Table II.

class	***cap-shape***	***cap-surface***	***cap-color***	***bruises***	***odor***
p	x	s	n	t	p
e	x	s	y	t	a
e	b	s	w	t	L
p	x	y	w	t	p
e	x	s	g	f	n

Table I. Mushroom Dataset

Attribute	*Detail*
Cap-shape	The shape of most upper part of mushrooms
Cap-color	Color of upper part
Odor	How the mushrooms smell
Gill-color	The color of under the cap
Gill-size	the size of under the cap
Habitat	The place where mushrooms grow

Table II. Most Common Mushroom Attributes

For further information:
https://archive.ics.uci.edu/ml/datasets/Mushroom

IV. Examination and Visualization of Dataset

In this section, we will discuss the distribution of features and their corresponding values (edible or poisonous), the correlation of the features, and how to use that dataset to train.

A general information about the dataset is printed on a text file. The dataset contains 23 features for the mushrooms like class, cap-shape, cap-surface, cap-color, bruises etc. The number of unique values, top values and their frequencies are found for each feature (shown in Table III.). There are 8124 sample mushrooms in the dataset.

	count	unique	top	freq
class	8124	2	e	4208
cap-shape	8124	6	x	3656
cap-surface	8124	4	y	3244
cap-color	8124	10	n	2284
bruises	8124	2	f	4748

Table III. Most Common Mushroom Attributes

Samples are checked if there are any null values, and no null values are observed in the dataset. The distribution of the mushroom class label is found as 48% for the poisonous, and 52% edible approximately.

```
------------class------------
Feature number:1
unique feature values: ['p' 'e']
Percentage of feature values:
p    0.482029
e    0.517971
Name: class, dtype: float64
```

Fig. 3. Example output information of one feature

This information (fig.3) about the dataset is plotted into charts that are shown below (fig. 4 and fig. 5).

Before starting the training and testing steps the dataset is split into two parts. 80% of the dataset is used for training, and the rest, 20%, is used for testing. The encoding operation is applied in order to transform categorical variables into numerical values. The dataset is partitioned mushroom class as Y label and the remaining features as X labels. One hot encoding is used for X labels because there is no numerical correlation over the X labels. These steps are discussed in more details in section VI as *Implementation*.

- *Correlation Matrix*

A correlation matrix is a table (each cell) showing correlation coefficients between variables (in this case features of a mushroom i.e., class, cap shape, cap surface etc.).

Correlation matrix is given below as figure 5. It is created to show which variable is having a high or low correlation in respect to another variable. To be able to show that, pre-processing (it will be explained in section VI as *Implementation* and values shown in Table IV.) is applied to dataset, this means each feature is represented numerically and based on those values the coefficients is calculated. The lighter the color is, the correlation is higher. For each variable, there is a correlation that is 1 which is the highest value, if two of the variables is the same (correlation with itself). This explains the diagonal part of the matrix (fig.6).

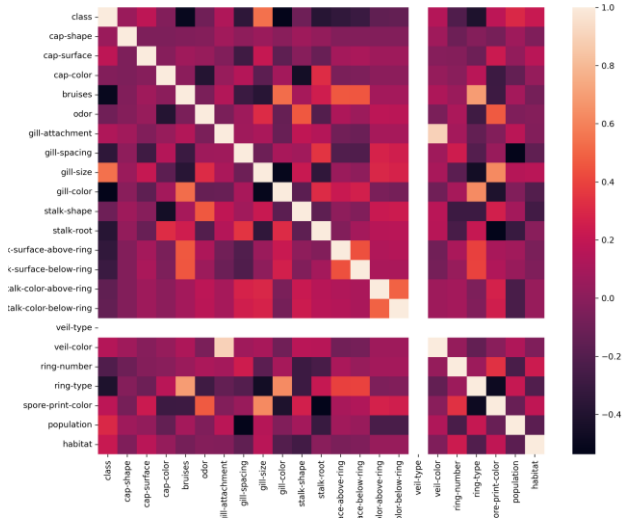

Fig.6. Correlation Matrix

From the above output of correlation matrix, it is seen that it is symmetrical (below and above diagonal) i.e., the bottom left is same as the top right. It is also observed that some of the variables are positively and negatively correlated with each other. Since veil-type has only one variable, it is not in the matrix.

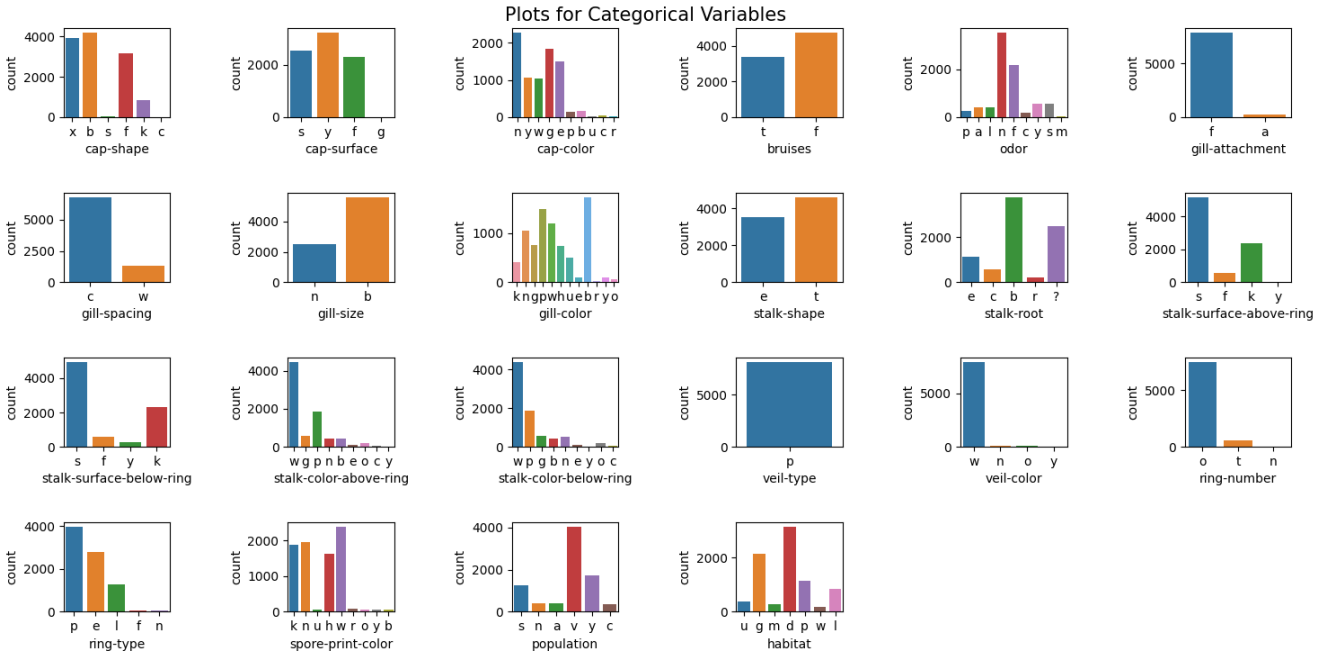

Fig. 4. Plots for categorical variables

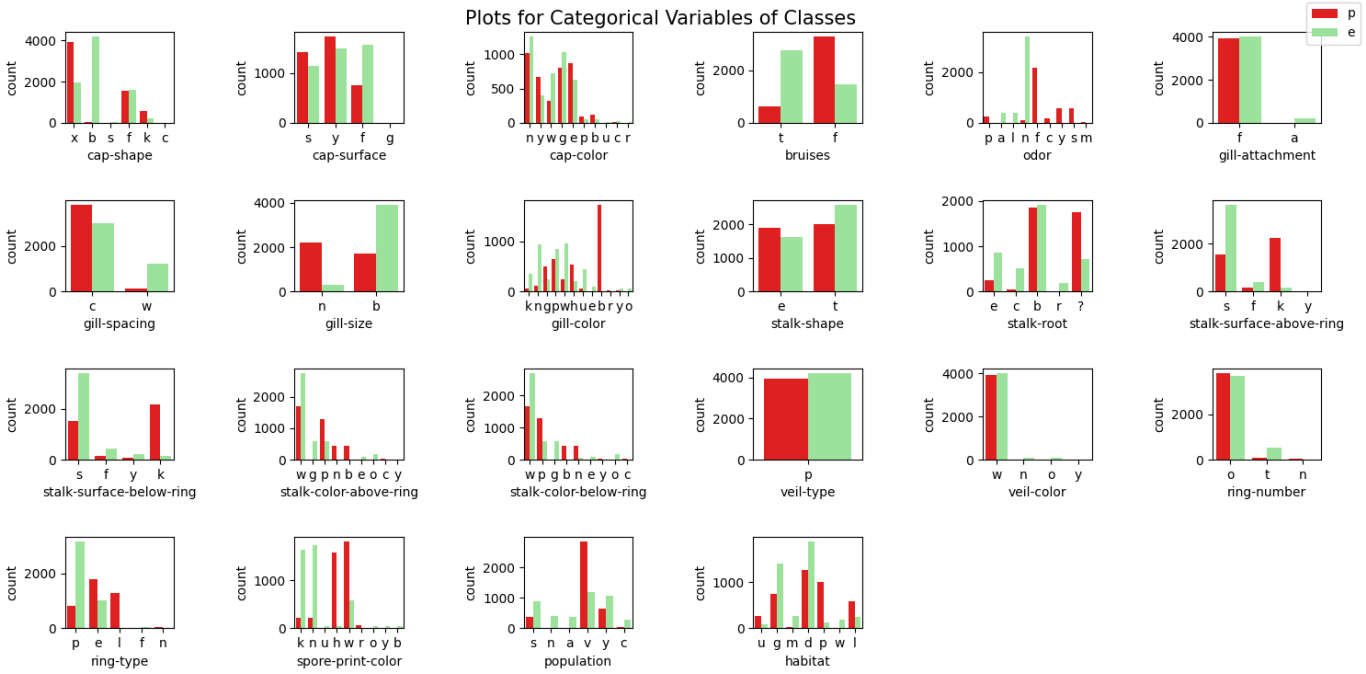

Fig 5. Plots for categorical variables of classes

V. METHODS

This project aims to find a feasible classification algorithm for edible mushroom with the best accuracy and easy to implement. A simple definition of each classification algorithm planned to use in this project is given below.

A. *k-NN*

k- nearest neighbor algorithm is a classification that uses a method to find the data that is closest to the object. [3] Therefore, its performance depends on the distance metric that is used to identify nearest neighbors. Euclidean distance is used by most k-NN classifiers, and it is defined as in equation (1).[2]

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \qquad (1)$$

where $d(p,q)$ is the distance between *p* and *q* data.[3]

B. *Random Forest*

Random forests are an ensemble learning method for classification and they consist of many decision trees which are constructed during training. Random forests are used to measure the importance of variables.

Bagging sampling methods are used to choose a training set for the decision tree from the original training set. Then, for each node of the tree, random variables are chosen, and the best split id calculated using the variables in the training set.[4]

C. *Naive Bayes*

Naïve Bayes (NB) is a probabilistic classifier based on applying Bayes' theorem with an assumption of independence among predictors. [5] Naïve Bayes uses a method that uses the information in sample data to estimate the previous probability P(y | x) of each class y given an object x. These estimations are used for classification.[6]

D. *Support Vector Machine*

The support vector machine (SVM) has developed rapidly in the past few decades, and it is based on structural risk minimization (SRM) principle. It is used for both classification and regression problems. SVM constructs a hyperplane which can be used for classification and regression. Then, it separated the samples based on the maximum margin approach.[7]

E. *Neural Network*

Artificial Neural Networks (ANN) are multi-layer fully connected neural nets. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. It is possible to make the network deeper by increasing the number of hidden layers to conclude better solutions. However, the more layers we have, it becomes more complex.

VI. IMPLEMENTATION

Naive Bayes and SVM algorithms are implemented. And both follows the same steps:

1. Examine and understand data
2. Preprocessing (Normalizing label)
3. Split the dataset as training and testing
4. Improve the model and repeat the process

A. Examine and understand data

This part is accomplished and explained in detail in one of the previous sections *IV. Examination and Visualization of dataset.*

B. Preprocessing (Normalizing label)

The aim is to encode all labels into numerical format to fit into our models. Two different preprocessing methods considered for preprocessing. These methods are label encoding and one hot encoding. Label encoding is a preprocessing method that is used for ordinal categorical types. This means there should be an order between the categories in the dataset. One hot encoding is used for nominal categorical types.

One hot encoding is a better method for preprocessing since there is no relationship between the categories of the dataset. Additional columns, which contain ones and zeros, are added for one hot encoding using the get_dummies() function of Pandas library. As seen in the screenshot below, a column is added for every category's feature.

	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s
0	0	0	0	0	0
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
...	...	...	...	...	...
8119	0	0	0	1	0
8120	0	0	0	0	0
8121	0	0	1	0	0
8122	0	0	0	1	0
8123	0	0	0	0	0

Table V. Additional columns for one hot encoding

In supervised learning, the key for learning is the dataset. If the dataset is clean and if the number of instances is enough, almost every algorithm (different models) can be used and have a good accuracy. In a dataset, a lot of things can cause a problem, i.e., missing values (nulls). Or if the set consist of images, it must be cleaned from unnecessary parts so that it can be read. So, it is very essential to prepare the dataset. That is why, before implementation of any learning algorithm, the examination and visualization of data must be completed in order to create a good model.

The next step is preparing the data so that any algorithm can understand. This means of course a mathematical expression of the data. It must be translated as numeric values. This is called pre-processing. In this project each label is represented as “e” or “p” (edible or poisoned). For some of the other feature representation is shown in table IV. For example, after pre-processing and encoding the data, these two labels will be represented as 1 or 0. For each feature, this must be accomplished.

	class	cap-shape	cap-surface	...	spore-print-color	population	habitat
0	p	x	s	...	k	s	u
1	e	x	s	...	n	n	g
2	e	b	s	...	n	n	m
3	p	x	y	...	k	s	u
4	e	x	s	...	n	a	g

Table V. Dataset before encoding

For later explanations, lets define a X and a y vector. X is every feature (every column in the dataset) except labels. And y contains the labels which is the class of a mushroom as edible or poison. They should be encoded. It is shown in table V.

For transformation of X and y numerically there are several libraries. For example, scikit-learn provides a preprocessing with label encoder, or Pandas library has a function called get_dummies() that does the same thing.

	class	cap-shape	cap-surface	...	spore-print-color	population	habitat
0	1	5	2	...	2	3	5
1	0	5	2	...	3	2	1
2	0	0	2	...	3	2	3
3	1	5	3	...	2	3	5
4	0	5	2	...	3	0	1

Table VI. Dataset after encoding

These values are used for correlation matrix in section *IV. Examination and Visualization of dataset.*

C. Split the dataset as training and testing

This part is about how to split the dataset as training and testing after preparation is completed and why it needs to be split.

In machine learning, there must be two separate data. One for learning and the other one is for to understand how well the learning is accomplished. For this case, unless a separate dataset for training and testing is provided, the set must be split. Generally, %70-80 of the data is used in training and the rest of it is for testing. Meaning of testing is try the model with a set of data that the model has not seen before. And if it predicts correctly, only this way, it can be concluded that a model has learnt well or not. In this project dataset divided as %80 for training and %20 for testing for the algorithms except ANN. For ANN, the dataset for training which is the %80 part, is divided again as %64 for training and %16 for validation.

So, previously defined X and y will be split as X_train, X_test and y_train, y_test. Now the model can be trained and tested.

D. Improve the model and repeat the process

A model can be improved if expected accuracy is not achieved.

In this project that kind of problem has not occurred. But in another experiment, it is concluded that, if a pre-processing of the data is not accomplished well or with another way of methods, the accuracy might decrease or increase.

VII. Results

The model that is created for ANN is a very basic and simple one. Even though it has only three layers, it has a very high accuracy. For ANN, as it is mentioned before, the dataset is split into three parts. %64 for training (around 6000) and %16 for validation and at the end%20 for testing. The results for training can be seen in fig.6.

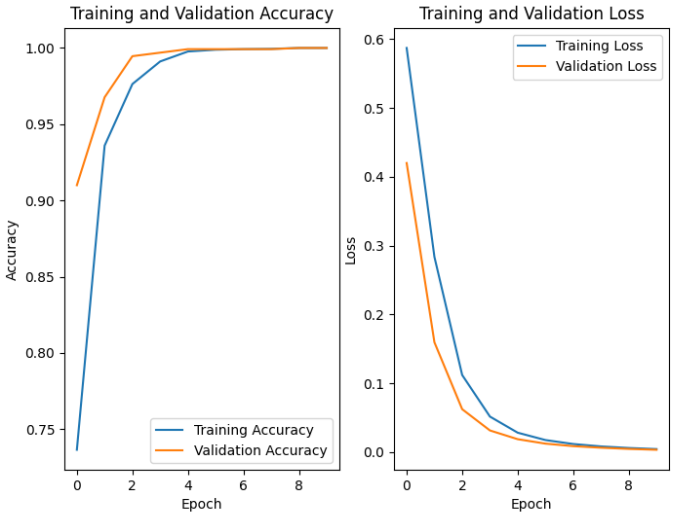

Fig. 6. ANN Training vs validation results for 15 epochs

From figure 6, it is seen that model did a good job because training and validation accuracies increased together. This means it is a good model, and probably will give good testing result. It is also observed that the validation accuracy starts from approximately %90 which is very high.

Even though number of epochs is not that much, the accuracy reaches the maximum that it can achieve. After a while it converges to %100.

Testing results for 15 epochs:
Test Accuracy of ANN: 99.93846416473389%
Loss of ANN: 0.022220865357667208%

This high accuracy is discussed in section *Discussions.*

Other algorithms have their own way to make prediction and by using confusion matrices, the number of misclassified mushrooms can be seen (in Figure 6). From those matrices, it can be seen that k-NN, and random forest has no misclassified mushroom, which means they have a 100% accuracy. it should be pointed out that these accuracies found by using the testing dataset which contains the data that the model has not seen before.

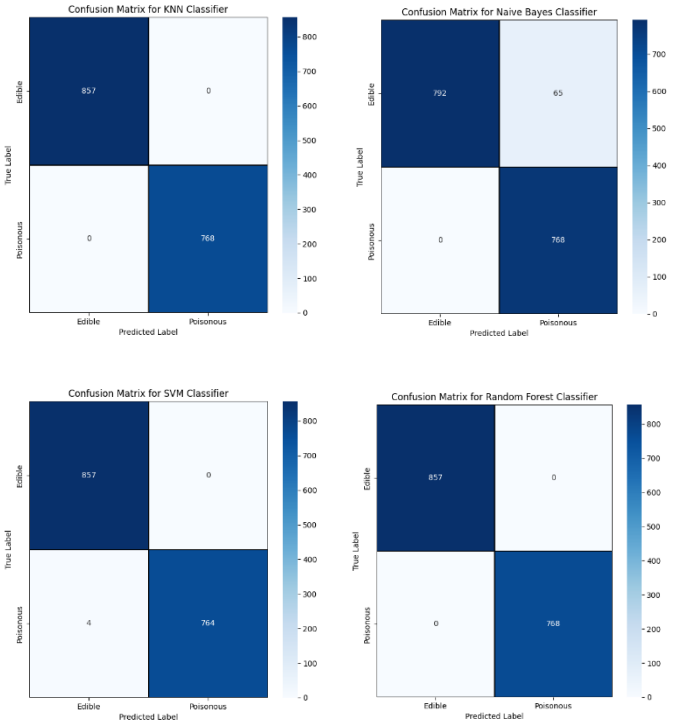

Fig. 6. Confusion matrices by using one-hot-encoding for preprocessing of NB, SVM, k-NN and random.

With different pre-processing methods, we can finally compare the accuracies. Even though one hot encoder is a better approach since there is no relationship between features for this project, label-encoder is also implemented just to prove this approach. The results can be seen in table VI.

	One-Hot-Encoding	Label-Encoding
SVM	%99.75	%100
Naive Bayes	%95.26	%91.75
K-NN	%100	%99.88
Random Forest	%100	%100
ANN	%99.94	%96.74

Table VI. Accuracy results.

VI. Conclusion & Future Works

ANN, k-NN and random forest has the highest accuracies. But random forest can be chosen as the best one because it has the highest acc in both preprocessing methods.
Since ANN takes a little bit more time than the others and more complicated, it is not really the perfect model for this problem.
But results are way to good that it creates a question mark.

- Why the accuracy of every algorithm is very high even though dataset is not very large (8124 instances)?

There are 2 theories for this discussion.

- The reason for the perfect score might be that there are several features in the data set that have an imbalance distribution on the target. It can be seen from Fig.5. For instance, cap-shape with a value of ‘b’, almost all of them is edible.

- Another reason could be training and testing datasets may be very much alike. So that the model does not confuse. It is like it has seen same situation before. As a solution to this problem can be n-fold cross-validation. If testing sets can be in different combination, the real accuracy can be seen. It may also be a good choice for testing. As a second solution, the dataset could be extended.

REFERENCES

[1] Y. Wang, J. Du, H. Zhang, and X. Yang, "Mushroom Toxicity Recognition Based on Multigrained Cascade Forest," *Scientific Programming*, vol. 2020, 2020, doi: 10.1155/2020/8849011.

[2] N. Chumuang *et al.*, "Mushroom Classification by Physical Characteristics by Technique of k-Nearest Neighbor," Nov. 2020. doi: 10.1109/iSAI-NLP51646.2020.9376820.

[3] N. Chitayae and A. Sunyoto, "Performance Comparison of Mushroom Types Classification Using K-Nearest Neighbor Method and Decision Tree Method," in *2020 3rd International Conference on Information and Communications Technology, ICOIACT 2020*, Nov. 2020, pp. 308–313. doi: 10.1109/ICOIACT50329.2020.9332148.

[4] P. T. R, "A Comparative Study on Decision Tree and Random Forest Using R Tool," *IJARCCE*, pp. 196–199, Jan. 2015, doi: 10.17148/ijarcce.2015.4142.

[5] P. Bermejo, J. A. Gámez, and J. M. Puerta, "Speeding up incremental wrapper feature subset selection with Naive Bayes classifier," *Knowledge-Based Systems*, vol. 55, pp. 140–147, Jan. 2014, doi: 10.1016/j.knosys.2013.10.016.

[6] G. I. Webb, "Naïve Bayes," in *Encyclopedia of Machine Learning and Data Mining*, Springer US, 2016, pp. 1–2. doi: 10.1007/978-1-4899-7502-7_581-1.

[7] B. Kumar and D. Gupta, "Universum based Lagrangian twin bounded support vector machine to classify EEG signals," *Computer Methods and Programs in Biomedicine*, vol. 208, Sep. 2021, doi: 10.1016/j.cmpb.2021.106244.

Algorithms.py

```
import os

import matplotlib.pyplot as plt
import seaborn
import tensorflow as tf
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

tensorflow_version = float(tf.__version__[0:3])
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'


class Algorithm:
    def __init__(self, X, y):
        self.X = X
        self.y = y

        # split the dataset for %80 training and %20 testing
        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(X, y, test_size=0.2,shuffle=True)

        self.y_pred = 0
        self.classification_method_str = ""

        # accuracies of each algorithm
        self.svm_acc = 0
        self.nb_Acc = 0
        self.knn_acc = 0
        self.rf_acc = 0
        self.ann_acc = 0
        self.ann_loss = 0

    def supportVectorMachines(self):
        self.classification_method_str = "SVM"

        svm = SVC(random_state=45, gamma="auto")
        svm.fit(self.x_train, self.y_train)
        self.svm_acc = round(svm.score(self.x_test, self.y_test) * 100, 2)
        self.y_pred = svm.predict(self.x_test)

        print("Test Accuracy of SVM: {}%".format(self.svm_acc))

        self.plot_confusion_matrix()

    def naiveBayes(self):
        self.classification_method_str = "Naive Bayes"
        nb = GaussianNB()
        nb.fit(self.x_train, self.y_train)
        self.nb_Acc = round(nb.score(self.x_test, self.y_test) * 100, 2)
        self.y_pred = nb.predict(self.x_test)
```

```
    print("Test Accuracy of NB: {}%".format(self.nb_Acc))
    self.plot_confusion_matrix()

def k_NearestNeighbor(self):
    self.classification_method_str = "KNN"
    knn = KNeighborsClassifier()
    knn_model = knn.fit(self.x_train, self.y_train)
    # knn_model
    self.y_pred = knn_model.predict(self.x_test)
    self.knn_acc = round(metrics.accuracy_score(self.y_test, self.y_pred) * 100, 2)
    print("Test Accuracy of kNN: {}%".format(self.knn_acc))
    self.plot_confusion_matrix()

def randomForest(self):
    self.classification_method_str = "Random Forest"
    rf_model = RandomForestClassifier().fit(self.x_train, self.y_train)
    self.y_pred = rf_model.predict(self.x_test)
    self.rf_acc = round(metrics.accuracy_score(self.y_test, self.y_pred) * 100, 2)
    print("Test Accuracy of random forest: {}%".format(self.rf_acc))
    self.plot_confusion_matrix()

def neuralNetwork(self):
    self.classification_method_str = "ANN"
    # split training set as training and validation
    X_train, X_val, y_train, y_val = train_test_split(self.x_train, self.y_train, test_size=0.2, shuffle=True)

    # model
    model = Sequential([
        layers.Dense(32, input_dim=X_train.shape[1], activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(1, activation='sigmoid'),
    ])

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()

    # Train the model
    # early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=5, restore_best_weights=True)
    epochs = 15
    history = model.fit(x=X_train,
                        y=y_train,
                        epochs=epochs,
                        batch_size=200,
                        validation_data=(X_val, y_val),
                        # callbacks=[early_stop]
                        )

    # results of training
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    # plotting training vs validation for accuracy
    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
```

```
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.title('Training and Validation Accuracy')

        # plotting training vs validation for loss
        plt.subplot(1, 2, 2)
        plt.plot(epochs_range, loss, label='Training Loss')
        plt.plot(epochs_range, val_loss, label='Validation Loss')
        plt.legend(loc='upper right')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.title('Training and Validation Loss')
        plt.savefig(os.getcwd() + '/training&validation.png')

        # Test the model
        self.ann_loss, self.ann_acc = model.evaluate(self.x_test, self.y_test)
        self.ann_loss, self.ann_acc = self.ann_loss * 10, self.ann_acc * 100
        print("Test Accuracy of ANN: {}%".format(self.ann_acc), ", Loss of ANN: {}%".format(self.ann_loss))

        # save the model
        model.save("model artifact")

    # plotting confusion matrix for the algorithm
    def plot_confusion_matrix(self):
        conf_matrix = confusion_matrix(self.y_test, self.y_pred)

        x_axis_labels = ["Edible", "Poisonous"]
        y_axis_labels = ["Edible", "Poisonous"]

        f, ax = plt.subplots(figsize=(7, 7))
        seaborn.heatmap(conf_matrix, annot=True, linewidths=0.2, linecolor="black", fmt=".0f", ax=ax, cmap="Blues",
                        xticklabels=x_axis_labels, yticklabels=y_axis_labels)
        plt.xlabel("Predicted Label")
        plt.ylabel("True Label")
        plt.title("Confusion Matrix for " + self.classification_method_str + " Classifier")
        plt.show()
```

DatasetPlot.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class DatasetPlot:
    def __init__(self, MushroomData):
        self.df = MushroomData

    # saving the plot in feature_figures folder
    def savePlot(self, feature):
        mainPath = os.getcwd()
        folderName = mainPath + '/feature_figures'
        if not os.path.exists(folderName):
            os.makedirs(folderName)
        filename = folderName + '/' + str(feature) + ".png"

        plt.savefig(filename, dpi=300)

    # plotting one feature
    def plotDataFeatures(self, feature):
        a = plt.figure()
        # plot the dataframe
```

```
    plt.xlabel(feature)
    plt.ylabel("Number of instances")
    title = "Distribution of feature values"
    plt.title(title)

    self.df[feature].value_counts(normalize=False, ascending=False).plot(kind='bar', color=plt.cm.Paired(
        np.arange(len(feature))), rot=0)

    plt.tight_layout()
    self.savePlot(feature)
    plt.close(a)

# general look on dataset and gives an output as datasetInfo.txt
def examineDataset(self):

    with open('datasetInfo.txt', 'w') as f:
        f.write("General look on dataset: \n")
        f.write(str(self.df.head(5)))
        m = '\n' + 'Number of rows:' + str(self.df.shape[0]) + 'Number of columns: ' + str(self.df.shape[1]) + '\n'
        f.write(m)
        # number of null values for each column
        m2 = str(self.df.isnull().values.any()) + ":There is no null value" + '\n'
        f.write(m2)
        # total columns description
        f.write(str(self.df.describe().T))

        fno = 0
        for feature in self.df.columns:
            m = '\n' + "----------" + str(feature) + "----------" + '\n' + "Feature number:" + str(fno + 1)
            f.write(m)
            # number of classes
            f.write("\nunique feature values: ")
            f.write(str(self.df[feature].unique()))
            f.write("\nPercentage of feature values: \n")
            values = self.df[feature].value_counts(normalize=True, ascending=True)
            f.write(str(values))
            f.write("\n")

            self.plotDataFeatures(feature)
            fno += 1

# plots all feature as one figure for general stats, one for based on their class
def plotAll(self):
    p, axes = plt.subplots(nrows=4, ncols=6, figsize=(6, 4))
    p2, axes2 = plt.subplots(nrows=4, ncols=6, figsize=(6, 4))
    p.tight_layout()
    p.suptitle('Plots for Categorical Variables of Classes', fontsize=15)
    p2.tight_layout()
    p2.suptitle('Plots for Categorical Variables', fontsize=15)

    x_axes = 0
    y_axes = 0
    color = ['red', 'lightgreen']

    fno = 0
    for feature in self.df.columns:
        a = sns.countplot(data=self.df, x=feature, hue='class', palette=color, ax=axes[x_axes, y_axes])
        a.legend([], [], frameon=False)
        sns.countplot(data=self.df, x=feature, ax=axes2[x_axes, y_axes])

        x_axes = int(fno / 6)
        y_axes = int(fno % 6)
```

```
            fno += 1

        p.legend(['p', 'e'], loc='upper right')
        plt.show()

    # plots correlation matrix of features
    def correlationMatrix(self, df):
        with open('datasetInfo.txt', 'a') as f:
            f.write("\nCorrelation:\n")
            f.write(str(df.head()))
            f.write(str(df.corr().T[:1]))

            corrPlot = plt.figure(figsize=(12, 10))
            sns.heatmap(df.corr())
            self.savePlot("correlationMatrix")
            plt.close(corrPlot)
```

main.py

```
# CSE4088 - Introduction to Machine Learning
# Term Project
# Part 1
# 150119824 - Zeynep Ferah Akkurt
# 150119825 - Merve Rana Kızıl
# main.py
import pandas
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from DatasetPlot import *
from Algorithm import *
import os


# using a copy and changing the all dataframe as numerically, then extract X and y from it
def labelEncoder(df):
    for col in df.columns:
        le = LabelEncoder()
        le.fit(df[str(col)])
        df[str(col)] = le.transform(df[str(col)])

    X = df.iloc[:, 1:23]  # all rows, all the features and no labels
    y = df.iloc[:, 0]

    return X, y


# transformation of X and y numerically and returns directly (not changing the dataframe)
def oneHotEncoder(df):
    # categorized --> numeric values
    X = df.drop(["class"], axis=1)
    y = df["class"]

    X = pd.get_dummies(X)
    # we need to encode our y-labels numerically
    le = LabelEncoder()
    y = le.fit_transform(y)

    return X, y


def main():
    path = os.getcwd()
```

```
    fileName = path + '\mushrooms.csv'
    # load the data
    data = pd.read_csv(fileName)

    print("Examine the dataset:")
    # examine and plot the dataset
    dataplots = DatasetPlot(data)
    dataplots.examineDataset()
    dataplots.plotAll()

    # check correlation
    corr_df = dataplots.df.copy()
    labelEncoder(corr_df)
    dataplots.correlationMatrix(corr_df)

    # encode the dataset as numerically to train and test
    #X, y = labelEncoder(data)
    X, y = oneHotEncoder(data)


    # training and testing with different algorithms
    algorithm = Algorithm(X, y)
    algorithm.supportVectorMachines()
    algorithm.naiveBayes()
    algorithm.k_NearestNeighbor()
    algorithm.randomForest()
    algorithm.neuralNetwork()


if __name__ == "__main__":
    main()
```