1 Introduction

Mushrooms are a diverse group that contains both edible and poisonous ones and it is sometimes hard to identify if the mushroom is suitable for eating. Thus, classifying mushrooms based on their characteristics is an important application in biological field and machine learning can help to identify these characteristics. Automation can make the process faster and help biological experts to recognize mushrooms. However, the model can have wrong results, so one should never eat an unrecognized mushroom although the model says it's safe.

The structure of the report is as follows: Section 2 formulates the problem and discusses features and more about the data. Section 3 contains data preprocessing and discussion about the model selection. Section 4 tells results and how the models were performed. Section 5 contains conclusions.

2 Problem formulation

In this project we are trying to predict whether the mushroom is poisonous or not based on many different features of the mushroom. We are using Mushroom dataset from UCI Machine Learning repository which contains different mushrooms and their characteristics. The dataset was found on Kaggle and is based on 23 species from book The Audubon Society Field Guide to North American Mushrooms (1981) [1]. It contains categorical data and each data point represents one mushroom. One mushroom has 22 features, for example color and cap-shape. The label is the binary outcome whether the mushroom is poisonous or not and the problem is supervised classification problem.

3 Methods

3.1 Data preprosessing

The dataset has 8124 datapoints each corresponding the different characteristics of the mushroom [Figure 1 and Figure 3]. The column stalk-root contained 2480 missing values but about 1800 of them were poisonous, so we decided to leave them in the dataset. If we had removed the corresponding rows, almost half of the poisonous mushrooms would have been removed. This would have skewed the distribution between edible and poisonous.

class	8124
cap-shape	8124
cap-surface	8124
cap-color	8124
bruises	8124
odor	8124
gill-attachment	8124
gill-spacing	8124
gill-size	8124
gill-color	8124
stalk-shape	8124
stalk-root	8124
stalk-surface-above-ring	8124
stalk-surface-below-ring	8124
stalk-color-above-ring	8124
stalk-color-below-ring	8124
veil-type	8124
veil-color	8124
ring-number	8124
ring-type	8124
spore-print-color	8124
population	8124
habitat	8124

dtype: int64

Figure 1. Column names and their count

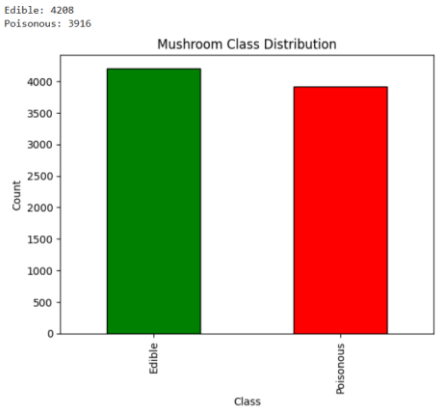

Figure 2. Distribution of mushrooms edibility

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g

5 rows × 23 columns

Figure 3: Example of the original dataset

Because the values were categorical, we had to change the values in the columns to binary values. We used one-hot encoding to change them. For example, the column color is now divided into new columns like color_g and color_b each column consisting of values 0 and 1. The encoding also created a new column called Missing for the missing values in the column stalk-root.

After that, we removed the duplicate columns that had originally two values. For example, the column class had originally values edible and poisonous (true/false) and the encoding divided it into two new columns (class_Edible /class_Poisonous), so we removed the column class_Edible. Class_Poisonous is enough to present whether the mushroom is poisonous or not.

3.2 Models

We decided to choose a Logistic Regression as our primary model. It predicts the probability of the binary outcome based on the linear combination of the input features[3]. It maps the features to a probability of being poisnous or not. Since our data describes binary classification problem, logistic regression is the best model to use. It is also a simple and interpretable model.

As a loss function, we used the logistic loss. It measures how the predicted probabilities and the true labels are matched. The logistic loss was selected because it functions well with the logistic regression model and it suits perfectly to the binary classification problem. It also has a ready library available so it is easy to use.

As a second model, we decided to use Random forests classifier. It is a supervised machine learning model that combines multiple decision trees where each decision tree is trained with random sample of data[2]. Random forests suits our problem because we have a small dataset and it can find complex non-linear dependencies that logistic regression can't. It also reduces overfitting using many decision trees. As a loss in training, we have Gini impurity, which tells the likelihood of misclassification of randomly chosen element. We use it because it is simple to calculate and it is commonly used with decision trees. For the result validation we use 0/1 loss because we can directly see how well the model classified the mushrooms.

We decided to split our 8124 datapoints so that 80% of data (6499 datapoints) is used for training and the rest 20% (1625 datapoints) is used for testing. To make the model as accurate as possible, we use k-fold cross validation method. We decided to use k=5 value which means that from those 6499 training datapoints, we use 5200 datapoints to train the model and the rest 1299 datapoints to validate it. This process is run 5 rounds with different validation set. Finally, we estimate the model's performance on the training data by calculating the mean accuracy of each round.

4.1 Results

The Logistic regression got us very accurate training and validation results as shown in figure 4 and figure 5. Training error was only 0.00003 and validation error 0.0004. The errors of each fold are so low in both training and validation, that it is difficult to even see the difference between them [Figure 4 and 5]. One reason for this accuracy can be that, if the data has a lot of mushrooms that have some very predictable character, like odor for example, the model can make very accurate predictions just based on this one character. The accuracy can also be seen in the logistic loss. Mean loss is only 0.007.

The Random forests got also very accurate results. Both training error mean and validation error mean were 0, so the model predicted perfectly poisonous mushrooms. The model was able to learn the data well for the same reason as logistic regression: some characteristics were highly correleted with poisonous. The random forest could learn these characteristics better than logistic regression because it can find more complex patterns from data and predict better non-linear dependencies. The test error for random forest was 0 so it performed well also with the test data. However, that doesn't mean that the model would always perform so well. In our case the data contained realistic samples from the book but it doesn't mean that random forest

would perform same way with the real word samples. The data we had was really optimal to random forest.

Both of the models performed very accurately on our data. The LogisticRegression predicted only couple of the labels wrong, which in our decently big dataset is not a major error. RandomForest was so accurate it predicted correctly every 6499 labels. The final test error for LogisticRegression was 0.0006 and 0 for the RandomForest. The reason for both methods being so accurate must rely on our dataset which is easily predictable. Overfitting is not the issue since both methods performed evenly accurate for training, validating and testing data. Both of the methods performed well but because the RandomForest performed perfectly, we decided to select it as our final method.

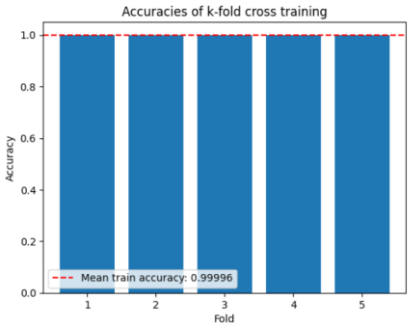

Figure 4. K-fold cross training accuracy

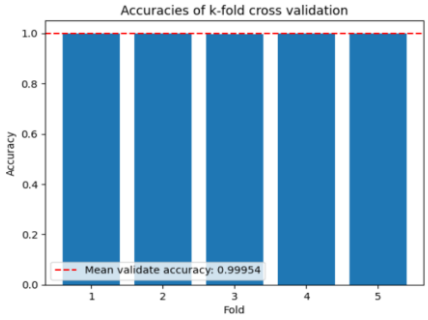

Figure 5. K-fold cross validation accuracy

5 Conclusion

As a conclusion, both methods learned the data well but the random forest was slightly better as described in the results. The dataset was optimal and easy to learn. There is still much to learn about mushrooms and their characteristics. Even if our models performed well, they could give different results with the datasets that consist of mushrooms that don't have easily separatable characteristics. Models can struggle more with real word samples and more diverse data. Thus, potential improvements would be testing the data with more diverse dataset and trying to generalize it better to real word samples. One can neither try to predict Finnish mushrooms with our model because species in our data were from America.

Finally, one must never eat an unrecognized mushroom even if the model says it is edible because even the best model can always make a mistake and eating poisonous mushroom can lead to death.

References:

[1] https://www.kaggle.com/datasets/uciml/mushroom-classification/data

[2]https://www.sciencedirect.com/topics/computer-science/random-forest-classifier

[3]https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

Appendices:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# The raw dataframe
df = pd.read_csv('mushrooms.csv')

# Dataset size
size = df.count()
print(size)

# Display first 5 rows
df.head()
```

```
# Histogram to show the distribution between Poisonous and Edible mushrooms in our dataset
counts = df['class'].value_counts()
count_edible = counts['Edible']
count_poisonous = counts['Poisonous']

counts.plot(kind='bar', color=['green', 'red'], edgecolor='black')
plt.title('Mushroom Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

print(f'Edible: {count_edible}')
print(f'Poisonous: {count_poisonous}')
plt.show()


# One-hot encoding
df_encoded = pd.get_dummies(df).astype(int)

# Deleting useless columns that .get_dummies function created
# For example we have already column 'class_Poisonous' so we dont need 'class_Edible' which is just opposite of poisonous
df_encoded.drop(['class_Edible', 'bruises_No', 'gill-size_Broad', 'stalk-shape_Enlarging', 'veil-type_Partial'], axis=1, inplace=True)

df_encoded.head()
```

```
# Labels for LogisticRegression
y = df_encoded['class_Poisonous'].to_numpy()
# Features for LogisticRegression
X = df_encoded.drop('class_Poisonous', axis=1).to_numpy()

# Logistic Regression
lg = LogisticRegression()

# Splitting the data to training (80%) and validation (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=27)

# Calculate k-fold cross validation accuracy and training accuracy
cross_val_scores = cross_validate(lg, X_train, y_train, cv=5, scoring='accuracy', return_train_score=True)

# Print results of training scores
mean_train = cross_val_scores['train_score'].mean()
print(f"Scores of k-fold cross train: {cross_val_scores['train_score']}")
print(f"Mean of train accuracy: {mean_train}")

# Visualize k-fold cross train scores
plt.bar(range(1, len(cross_val_scores['train_score']) + 1), cross_val_scores['train_score'])
plt.ylim(0, 1.05)
plt.xticks(range(1, len(cross_val_scores['train_score']) + 1))
plt.axhline(mean_train, color='red', linestyle='--', label=f"Mean train accuracy: {mean_train:.5f}")
plt.title("Accuracies of k-fold cross training")
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
# Print results of validation scores
mean_validate = cross_val_scores['test_score'].mean()
print(f"Scores of k-fold cross test: {cross_val_scores['test_score']}")
print(f"Mean of test accuracy: {mean_validate}")

# Visualize k-fold cross validation scores
plt.bar(range(1, len(cross_val_scores['test_score']) + 1), cross_val_scores['test_score'])
plt.ylim(0, 1.05)
plt.xticks(range(1, len(cross_val_scores['test_score']) + 1))
plt.axhline(mean_validate, color='red', linestyle='--', label=f"Mean validate accuracy: {mean_validate:.5f}")
plt.title("Accuracies of k-fold cross validation")
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.legend(loc='lower left')
plt.show()

# Calculate Logistic loss for k-fold cross validation
logistic_loss = -cross_val_score(lg, X_train, y_train, cv=5, scoring='neg_log_loss')
mean_loss = logistic_loss.mean()
print(f"Scores of k-fold cross validation logistic loss: {logistic_loss}")
print(f"Mean logistic loss: {mean_loss}")

# Visualize logistic losses of k-fold cross validation
plt.bar(range(1, len(logistic_loss) + 1), logistic_loss)
plt.ylim(0, 0.1)
plt.xticks(range(1, len(logistic_loss) + 1))
plt.axhline(mean_loss, color='red', linestyle='--', label=f"Mean logistic loss: {mean_loss:.5f}")
plt.title("Logistic losses of k-fold cross validation")
plt.xlabel("Fold")
plt.ylabel("Loss")
plt.legend(loc='upper left')
plt.show()



# RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=27, n_jobs=1)

# RandomForest k-cross validation
cross_val_scores_rf = cross_validate(rf, X_train, y_train, cv=5, scoring='accuracy', return_train_score=True)

# Training scores for each fold
train_scores = cross_val_scores_rf['train_score']

# Validation scores for each fold
val_scores = cross_val_scores_rf['test_score']

#Training and validation errors
train_error_rf = 1-train_scores.mean()
val_error_rf = 1-val_scores.mean()
```

```
#Logistic regression training data fit
lg.fit(X_train, y_train)

#RandomForest training data fit
rf.fit(X_train, y_train)

#Predicting results
y_pred_lg = lg.predict(X_test)
y_pred_rf = rf.predict(X_test)

#Test error
lg_test_error = 1-accuracy_score(y_test, y_pred_lg)
rf_test_error = 1-accuracy_score(y_test, y_pred_rf)

print("Random forest training scores:", train_scores)
print("Random forest validation scores:", val_scores)
print("Random forest training error mean:", train_error_rf)
print("Random forest validation error mean:", val_error_rf)

print("Logistic regression test error:", lg_test_error)
print("Random forests test error:", rf_test_error)
```