

Poisonous Mushroom Classification

Amar Gill, Ruth Yankson, Limor Winter, Claire Saunders

2025-11-18

```
import requests
import zipfile
import pandas as pd
import numpy as np
import altair as alt
alt.renderers.enable("png")  
  
RendererRegistry.enable('png')
```

Summary

(Insert summary after analysis and discussion complete)

Introduction

The Agaricus and Lepiota (Agaricaceae) family of mushrooms, includes many of the common gilled mushrooms found and cultivated in urban and suburban North America (Lincoff, 1981, 500). The family includes the common cultivated mushroom (*Agaricus bisporus*), and some of the best edible mushrooms, but it also includes poisonous mushrooms and a few that are deadly(500). These mushrooms are sometimes found in forests, but many grow on lawns, in the compost, or along roadsides and grassy areas where they are readily accessible for human consumption (500). Accurate assessment of the edibility of these mushrooms is extremely important prior to consumption, however there is no simple rule for determining whether any example is poisonous or edible (500-505).

Here we ask if we can use a supervised machine learning algorithm to predict whether a given mushroom from the Agaricus and Lepiota family of mushrooms is edible or poisonous, based upon its physical characteristics. Answering this question is important because mushroom

poisonings are common and the greatest risk factor for mushroom poisoning is the misidentification of poisonous species as edible, in particular by amateur mushroom foragers and immigrants (Diaz, 2016) If made accessible to the public in North America, accurate prediction of mushrooms in this family could prevent poisoning and support the safe consumption of common edible mushrooms.

Methods

Data

The data set used in this project includes hypothetical mushroom samples drawn from The Audobon Society Field Guide to North American Mushrooms by Gary Lincoff (1981, 500-525). It was sourced from the UCI Machine Learning Repository, and can be found [here](#), specifically the file *agarius-lepiota.data*. It includes 8124 descriptions of hypothetical mushroom samples that correspond to 23 species of mushrooms in the Agaricus and Lepiota Family (Mushroom, 1981). Each row includes 22 categorical variables that are physical characteristics of the mushroom sample, including the odor, gill-spacing, gill-color, gill-size, population, habitat, etc, and is labelled as edible or poisonous.

Analysis

```
#adding the downloaded data to the data directory by code
# download data as zip and extract
url = "https://archive.ics.uci.edu/static/public/73/mushroom.zip"

response = requests.get(url)
with open("../data/raw/mushroom.zip", 'wb') as f:
    f.write(response.content)

with zipfile.ZipFile("../data/raw/mushroom.zip", 'r') as zip_ref:
    zip_ref.extractall("../data/raw")
```

Data cleaning and wrangling

```
col_names = [
    "class", "cap_shape", "cap_surface", "cap_color", "bruises", "odor",
    "gill_attachment", "gill_spacing", "gill_size", "gill_color",
```

```

    "stalk_shape", "stalk_root",
    "stalk_surface_above_ring", "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"
]

df = pd.read_csv("../data/raw/agaricus-lepiota.data", names=col_names)

# Changing the target to numeric values: poisonous=1, edible=0
df["class_label"] = df["class"].map({"p": 1, "e": 0})

# Drop rows with na in the target column
df = df.dropna(subset=["class_label"])
df.head()

```

	class	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size
0	p	x	s	n	t	p	f	c	n
1	e	x	s	y	t	a	f	c	b
2	e	b	s	w	t	l	f	c	b
3	p	x	y	w	t	p	f	c	n
4	e	x	s	g	f	n	f	w	b

```
df.isna().sum()
```

class	0
cap_shape	0
cap_surface	0
cap_color	0
bruises	0
odor	0
gill_attachment	0
gill_spacing	0
gill_size	0
gill_color	0
stalk_shape	0
stalk_root	0
stalk_surface_above_ring	0
stalk_surface_below_ring	0

```
stalk_color_above_ring      0
stalk_color_below_ring      0
veil_type                   0
veil_color                  0
ring_number                 0
ring_type                   0
spore_print_color           0
population                  0
habitat                     0
class_label                 0
dtype: int64
```

```
# Encode "missing" (?) as NA
(df == '?').sum()
```

```
class                      0
cap_shape                  0
cap_surface                0
cap_color                  0
bruises                     0
odor                       0
gill_attachment             0
gill_spacing                0
gill_size                   0
gill_color                  0
stalk_shape                 0
stalk_root                  2480
stalk_surface_above_ring    0
stalk_surface_below_ring     0
stalk_color_above_ring       0
stalk_color_below_ring       0
veil_type                   0
veil_color                  0
ring_number                 0
ring_type                   0
spore_print_color           0
population                  0
habitat                     0
class_label                 0
dtype: int64
```

```
# Drop stalk_root column
df.drop(columns=['stalk_root'], inplace=True)
```

```
df.isna().sum()
```

```
class                      0
cap_shape                  0
cap_surface                0
cap_color                  0
bruises                     0
odor                       0
gill_attachment             0
gill_spacing                0
gill_size                   0
gill_color                  0
stalk_shape                 0
stalk_surface_above_ring    0
stalk_surface_below_ring     0
stalk_color_above_ring       0
stalk_color_below_ring       0
veil_type                   0
veil_color                  0
ring_number                 0
ring_type                   0
spore_print_color            0
population                  0
habitat                     0
class_label                 0
dtype: int64
```

```
# rename class_label to is_poisonous for target clarity
df.rename(columns={'class_label': 'is_poisonous'}, inplace=True)
```

EDA summary

```
# Basic information
df.head()
```

	class	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size
0	p	x	s	n	t	p	f	c	n
1	e	x	s	y	t	a	f	c	b
2	e	b	s	w	t	l	f	c	b
3	p	x	y	w	t	p	f	c	n
4	e	x	s	g	f	n	f	w	b

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   class            8124 non-null    object 
 1   cap_shape        8124 non-null    object 
 2   cap_surface      8124 non-null    object 
 3   cap_color        8124 non-null    object 
 4   bruises          8124 non-null    object 
 5   odor             8124 non-null    object 
 6   gill_attachment  8124 non-null    object 
 7   gill_spacing     8124 non-null    object 
 8   gill_size        8124 non-null    object 
 9   gill_color       8124 non-null    object 
 10  stalk_shape      8124 non-null    object 
 11  stalk_surface_above_ring 8124 non-null    object 
 12  stalk_surface_below_ring 8124 non-null    object 
 13  stalk_color_above_ring 8124 non-null    object 
 14  stalk_color_below_ring 8124 non-null    object 
 15  veil_type        8124 non-null    object 
 16  veil_color       8124 non-null    object 
 17  ring_number      8124 non-null    object 
 18  ring_type         8124 non-null    object 
 19  spore_print_color 8124 non-null    object 
 20  population        8124 non-null    object 
 21  habitat           8124 non-null    object 
 22  is_poisonous      8124 non-null    int64 

dtypes: int64(1), object(22)
memory usage: 1.4+ MB
```

```
df.shape
```

```
(8124, 23)
```

```
# Inspect target variable: e = edible = 0, p = poisonous = 1
df['is_poisonous'].value_counts()
```

```
is_poisonous
0    4208
1    3916
Name: count, dtype: int64
```

```
# Display the proportion of each mushroom class (edible vs poisonous)
df['is_poisonous'].value_counts(normalize=True)
```

```
is_poisonous
0    0.517971
1    0.482029
Name: proportion, dtype: float64
```

```
# Get current col_names
col_names = list(df.columns)
col_names
```

```
['class',
 'cap_shape',
 'cap_surface',
 'cap_color',
 'bruises',
 'odor',
 'gill_attachment',
 'gill_spacing',
 'gill_size',
 'gill_color',
 'stalk_shape',
 'stalk_surface_above_ring',
 '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',
'is_poisonous']

# Key mushroom feature summary
summary_rows = []
for col in col_names:
    vc = df[col].value_counts()
    summary_rows.append({
        "feature": col,
        "n_categories": vc.shape[0],
        "most_frequent_category": vc.index[0],
        "most_frequent_count": int(vc.iloc[0])
    })

feature_summary = (
    pd.DataFrame(summary_rows)
    .sort_values("n_categories", ascending=False)
    .reset_index(drop=True)
)
feature_summary.index = feature_summary.index + 1
feature_summary

```

Table 3: Summary of mushroom feature characteristics

	feature	n_categories	most_frequent_category	most_frequent_count
1	gill_color	12	b	1728
2	cap_color	10	n	2284
3	odor	9	n	3528
4	spore_print_color	9	w	2388
5	stalk_color_above_ring	9	w	4464
6	stalk_color_below_ring	9	w	4384
7	habitat	7	d	3148
8	cap_shape	6	x	3656
9	population	6	v	4040
10	ring_type	5	p	3968

Table 3: Summary of mushroom feature characteristics

feature	n_categories	most_frequent_category	most_frequent_count
11 cap_surface	4	y	3244
12 veil_color	4	w	7924
13 stalk_surface_above_ring	4	s	5176
14 stalk_surface_below_ring	4	s	4936
15 ring_number	3	o	7488
16 gill_attachment	2	f	7914
17 bruises	2	f	4748
18 class	2	e	4208
19 stalk_shape	2	t	4608
20 gill_size	2	b	5612
21 gill_spacing	2	c	6812
22 is_poisonous	2	0	4208
23 veil_type	1	p	8124

```

def get_poison_rate_by(feature):
    """
    Compute the proportion of poisonous and edible mushrooms for each category
    of a given feature.

    Parameters
    -----
    feature : str
        The name of the categorical column in `df` for which the poison rates
        should be calculated.

    Returns
    -----
    pandas.DataFrame
        A DataFrame indexed by the feature's categories with two columns:
        `edible_frac` and `poisonous_frac`, sorted in descending order of
        `poisonous_frac`.
    """
    category_poisonous = pd.crosstab(df[feature], df['is_poisonous'], normalize='index')
    category_poisonous.columns = ['edible_frac', 'poisonous_frac']
    return category_poisonous.sort_values('poisonous_frac', ascending=False)

```

```
# Feature category maps
odor_map = {
```

```
        "a": "almond",
        "c": "creosote",
        "f": "foul",
        "l": "anise",
        "m": "musty",
        "n": "none",
        "p": "pungent",
        "s": "spicy",
        "y": "fishy",
    }

habitat_map = {
    "g": "grasses",
    "l": "leaves",
    "m": "meadows",
    "p": "paths",
    "u": "urban",
    "w": "waste",
    "d": "woods",
}

gill_size_map = {
    "b": "broad",
    "n": "narrow",
}

bruises_map = {
    "t": "bruises",
    "f": "no bruises",
}

population_map = {
    "a": "abundant",
    "c": "clustered",
    "n": "numerous",
    "s": "scattered",
    "v": "several",
    "y": "solitary",
}

ring_type_map = {
    "c": "cobwebby",
```

```
"e": "evanescent",
"f": "flaring",
"l": "large",
"n": "none",
"p": "pendant",
"s": "sheathing",
"z": "zone",
}
```

```
def stacked_poison_chart(
    figure_number: int,
    feature: str,
    feature_label: str | None = None,
    category_labels: dict[str, str] | None = None,
):
    """
    Create a stacked bar chart showing edible vs poisonous fractions
    for each category of a given feature.

    Parameters
    -----
    feature : str
        Column name in df (e.g., "odor", "gill_size", "habitat").
    feature_label : str, optional
        Pretty label for axes / title. If None, derived from the column name.
    category_labels : dict[str, str], optional
        Optional mapping from raw category codes to human-readable labels.
        For example, for odor: {"a": "almond", "n": "none", ...}.
        If provided, these labels will be used on the y-axis and in tooltips.

    Returns
    -----
    alt.Chart
        Altair chart object with stacked bars.
    """
    # Use a nicer label if not provided
    if feature_label is None:
        feature_label = feature.replace("_", " ").title()

    # Table of edible / poisonous fractions by category
    ct = get_poison_rate_by(feature).reset_index()
```

```

# If a category label map is provided, add a display column
if category_labels is not None:
    display_col = f"{feature}_label"
    ct[display_col] = ct[feature].map(category_labels).fillna(ct[feature])
else:
    display_col = feature # use the raw feature values

# Long format for stacked bars
long_df = ct.melt(
    id_vars=display_col,
    value_vars=["edible_frac", "poisonous_frac"],
    var_name="class",
    value_name="fraction",
)
# Human-readable class labels for legend
long_df["class_label"] = long_df["class"].map(
    {"edible_frac": "Edible", "poisonous_frac": "Poisonous"}
)

# Build stacked bar chart
chart = (
    alt.Chart(long_df)
    .mark_bar()
    .encode(
        y=alt.Y(f"{display_col}:N", title=feature_label, sort="-x"),
        x=alt.X("fraction:Q", title="Fraction", axis=alt.Axis(format=".2f")),
        color=alt.Color(
            "class_label:N",
            title="Mushroom class",
            scale=alt.Scale(
                domain=["Edible", "Poisonous"],
                range=["#17BECF", "#7E1E9C"],
            ),
            legend=alt.Legend(title="Mushroom class"),
        ),
        tooltip=[
            alt.Tooltip(f"{display_col}:N", title=feature_label),
            alt.Tooltip("class_label:N", title="Class"),
            alt.Tooltip("fraction:Q", title="Fraction", format=".2f"),
        ],
    )
)

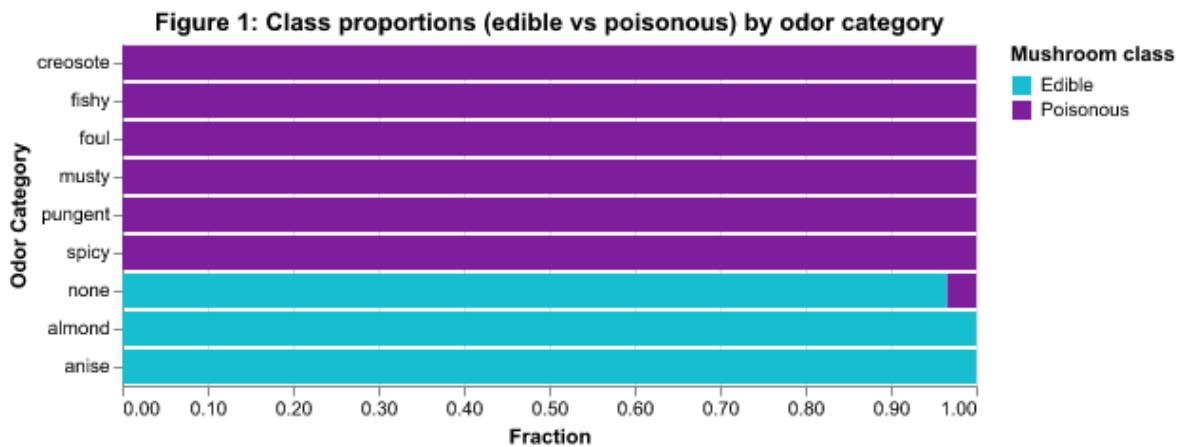
```

```

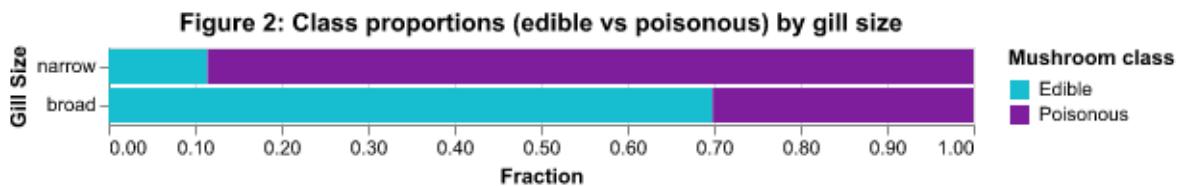
    .properties(
        title=f"Figure {figure_number}: Class proportions (edible vs poisonous) by {feature_name}",
        width=450,
    )
)
return chart

```

```
# Visualize the edible vs poisonous class proportions for each odor category
stacked_poison_chart(1, "odor", "Odor Category", odor_map)
```

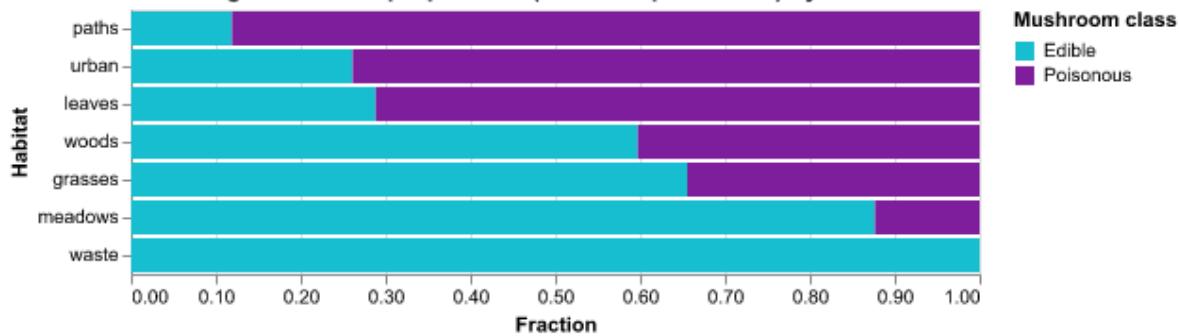


```
# Visualize the edible vs poisonous class proportions for each gill size
stacked_poison_chart(2, "gill_size", "Gill Size", gill_size_map)
```



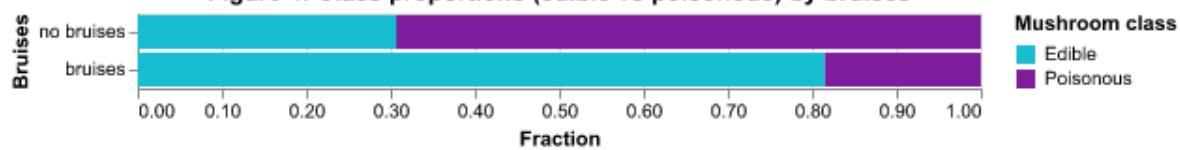
```
# Visualize the edible vs poisonous class proportions for each habitat
stacked_poison_chart(3, "habitat", "Habitat", habitat_map)
```

Figure 3: Class proportions (edible vs poisonous) by habitat



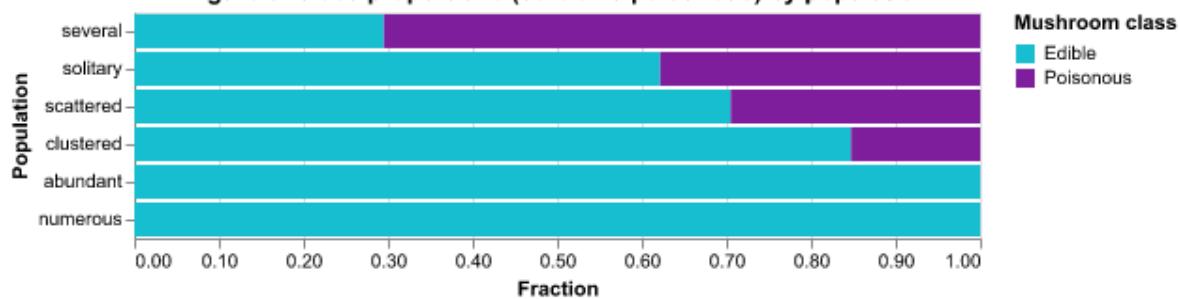
```
# Visualize the edible vs poisonous class proportions for each bruise category
stacked_poison_chart(4, "bruises", "Bruises", bruises_map)
```

Figure 4: Class proportions (edible vs poisonous) by bruises



```
# Visualize the edible vs poisonous class proportions for each population type
stacked_poison_chart(5, "population", "Population", population_map)
```

Figure 5: Class proportions (edible vs poisonous) by population



```
# Column names without veil_type, or target column
exclude = ["veil_type", "class", "is_poisonous"] # add more if needed
feature_cols = [col for col in df.columns if col not in exclude]
```

```
# Compute the variance in poisonous fraction across the categories of each feature
# Show strongly the feature predicts toxicity
```

```

feature_scores = []

for col in feature_cols:
    ct = get_poison_rate_by(col)
    score = ct['poisonous_frac'].var()
    feature_scores.append({'feature': col, 'poison_variance': round(score, 2)})

feature_importance_eda = pd.DataFrame(feature_scores).sort_values(
    'poison_variance', ascending=False
).reset_index(drop=True)

feature_importance_eda.index = feature_importance_eda.index + 1
feature_importance_eda

```

Table 4: Ranking of features by how well they distinguish poisonous mushrooms

	feature	poison_variance
1	odor	0.24
2	veil_color	0.23
3	stalk_color_above_ring	0.22
4	stalk_color_below_ring	0.21
5	ring_type	0.21
6	spore_print_color	0.20
7	ring_number	0.19
8	gill_size	0.17
9	gill_color	0.14
10	cap_shape	0.14
11	bruises	0.13
12	gill_spacing	0.11
13	habitat	0.11
14	stalk_surface_below_ring	0.11
15	stalk_surface_above_ring	0.10
16	cap_surface	0.08
17	gill_attachment	0.08
18	population	0.07
19	cap_color	0.06
20	stalk_shape	0.01

We are trying to classify whether a mushroom is edible (0) or poisonous (1). Going to do a 70/30 train/test split due to the large and roughly equal proportions of edible and poisonous mushrooms.

A random state of 123 will ensure reproducibility when ran by subsequent users.

Before we continue, we must preprocess our data such that the features work with our models. Since all of our features are strings, we can opt to use one-hot encoding for them all.

```
from sklearn.preprocessing import OneHotEncoder

preprocessor = OneHotEncoder()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    df.loc[:, 'class':'habitat'], df['is_poisonous'], test_size=0.3, random_state=123
)
```

We fit a `DummyClassifier` to our data as a baseline to compare our actual model with. Random state used again for reproducibility.

```
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_validate
from sklearn.pipeline import make_pipeline

dc_pipe = make_pipeline(
    preprocessor,
    DummyClassifier(random_state=123)
)

dc_pipe.fit(X_train, y_train)

cross_val_dc = pd.DataFrame(
    cross_validate(
        dc_pipe, X_train, y_train, cv=10, n_jobs=-1, return_train_score=True
    ).mean().to_frame().rename(columns={0: "mean_value"})
)

cross_val_dc
```

	mean_value
fit_time	0.016891
score_time	0.004274
test_score	0.516532
train_score	0.516532

In our classification problem, we opt to use SVC (Buitinck, L., Louppe, G., Blondel, M., Pedregosa, Fabian, Mueller, A., Grisel, O., ... Ga"el Varoquaux. (2013)) as our model of choice for its effective usage in this scenario. We do not have an extreme amount of data (< 10000 observations) on top of only being a binary classification task.

```
from sklearn.svm import SVC

svc_pipe = make_pipeline(
    preprocessor,
    SVC(random_state=123)
)

svc_pipe.fit(X_train, y_train)

cross_val_svc = pd.DataFrame(
    cross_validate(
        svc_pipe, X_train, y_train, cv=10, n_jobs=-1, return_train_score=True
    ).mean().to_frame().rename(columns={0: "mean_value"}))

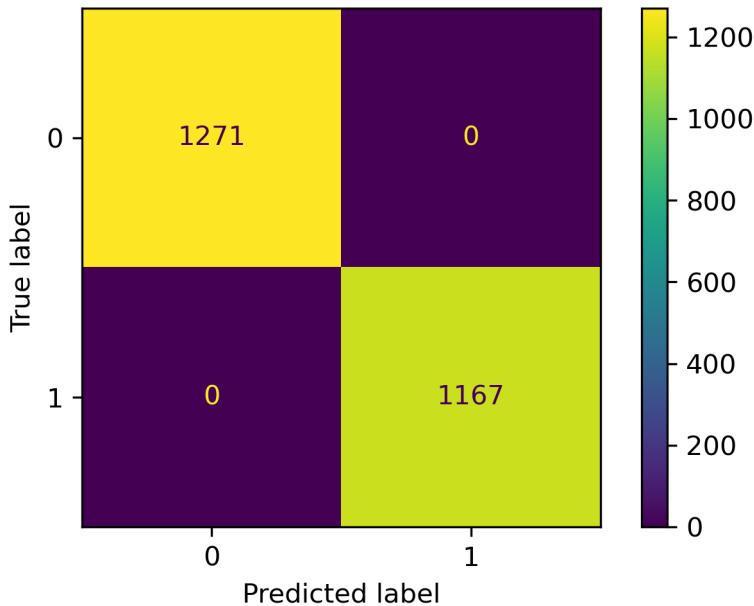
cross_val_svc
```

	mean_value
fit_time	0.302948
score_time	0.033286
test_score	1.000000
train_score	1.000000

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = svc_pipe.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm).plot()
```



Results & Discussion

Our prediction model performed excellently on test data, classifying all of the mushroom examples correctly as poisonous or edible, with no error. Given that our model predicted perfectly, no optimization was required. This performance is identical to the results published on the [UCI website](#) from which the data was sourced (Mushroom).

As shown in the confusion matrix, the model has a recall and precision of 1, due to no false positives or negatives. The model also shows robustness, with no difference in the accuracy of training and test scores, so it is promising that this model could be deployed effectively to protect mushroom consumers from eating poisonous mushrooms. That said, there are some practical concerns for public implementation, namely the model is fit on 20 different features that correspond to specific categorical physical characteristics of a mushroom, and thus accuracy requires accurate identification of physical characteristics.

Hence, if an amateur mushroom enthusiast aimed to utilize the model to predict edibility but misidentified the physical characteristics of the mushroom they were attempting to classify, then the accuracy of the models predictions would be effectively moot as it would not be returning a prediction for the true foraged mushroom.

One might consider creating a similar predictor with a smaller number of features to increase the practical efficacy of the model, though they would need to consider balancing the models accuracy with efficacy, because the risk of falsely predicting a mushroom as edible could be fatal.

It is also worth noting that the hypothetical mushroom observations in this model are based upon a book from 1981 (Lincoff), so it would be wise to consider whether the data could be updated to be sourced from a more recent source of mushroom science.

References

- Lincoff, G. H. (1981). The Audubon Society Field Guide to North American Mushrooms (1981). New York: Alfred A. Knopf
- Mushroom. (1981). UCI Machine Learning Repository. <https://doi.org/10.24432/C5959T>
- Diaz, J. H. (2016). Mistaken mushroom poisonings. *Wilderness & Environmental Medicine*, 27(2), 330-335.
- Buitinck, L., Louppe, G., Blondel, M., Pedregosa, Fabian, Mueller, A., Grisel, O., ... Ga”el Varoquaux. (2013). API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning (pp. 108–122).