MushroomClassification

0.0.1 Importing libraries, dataset and exploring data

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import OrdinalEncoder
from sklearn.inspection import permutation_importance
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
warnings.filterwarnings("ignore")
```

Dataset: Mushroom Classification Source: https://www.kaggle.com/datasets/uciml/mushroom-classification The dataset has the purpose of identifying whether mushrooms are safe to eat and which features influence and influence the most the toxicity of the mushrooms.

This is the information needed to understand the letters used in the dataset. It is provided in the source.

Attribute Information: (classes: edible=e, poisonous=p)

```
cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

bruises: bruises=t,no=f

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

gill-attachment: attached=a,descending=d,free=f,notched=n

gill-spacing: close=c,crowded=w,distant=d

gill-size: broad=b,narrow=n
```

gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,wh

stalk-shape: enlarging=e,tapering=t

stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?

stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

veil-type: partial=p,universal=u

veil-color: brown=n,orange=o,white=w,yellow=y

ring-number: none=n,one=o,two=t

ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z

spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y

population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

[2]:
```
df = pd.read_csv('mushrooms.csv', header=1)
df.head()
```

[2]:
```
  class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0     p         x           s         n       t    p               f
1     e         x           s         y       t    a               f
2     e         b           s         w       t    l               f
3     p         x           y         w       t    p               f
4     e         x           s         g       f    n               f

  gill-spacing gill-size gill-color  ... stalk-surface-below-ring  \
0            c         n          k  ...                        s
1            c         b          k  ...                        s
2            c         b          n  ...                        s
3            c         n          n  ...                        s
4            w         b          k  ...                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
```

```
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

[3]:
```
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-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

Data has 23 columns, 8124 rows, no missing values, and is all categorical.

[4]:
```
df['stalk-root'].value_counts()
```

[4]:
```
stalk-root
b    3776
?    2480
e    1120
c     556
r     192
Name: count, dtype: int64
```

As it can be seen, 2480 rows from the feature stalk-root are defined as '?', which represents a category for missing values. The large content of missing values makes it impossible to drop them. Also, they cannot be substituted by numbers, as categorical values. For this reason, they have been treated as a category by the source.

0.0.2 Training data

[5]:
```
y_log = df['class'].replace(['e','p'], [1,0])
X = df.drop('class', axis=1)
X_log = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X_log,y_log, test_size= 0.3,␣
↪random_state=42)
```

0.0.3 Logistic Regression

[6]:
```
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

[6]:
```
LogisticRegression()
```

[7]:
```
predTrain = lr.predict(X_train)
predTest = lr.predict(X_test)
correctpredTrain = predTrain == y_train
correctpredTest= predTest == y_test
```

[8]:
```
AccTrain = sum(correctpredTrain)/len(correctpredTrain)
AccTest = sum(correctpredTest)/len(correctpredTest)
print(AccTrain,AccTest)
```

```
1.0 1.0
```

[9]:
```
result = confusion_matrix(y_test, predTest)
print("Confusion Matrix:")
print(result)
report = classification_report(y_test, predTest)
print("Classification Report:",)
print (report)
```

```
accuracy = accuracy_score(y_test,predTest)
print("Accuracy:",accuracy)
```

```
Confusion Matrix:
[[1181    0]
 [   0 1257]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1181
           1       1.00      1.00      1.00      1257

    accuracy                           1.00      2438
   macro avg       1.00      1.00      1.00      2438
weighted avg       1.00      1.00      1.00      2438

Accuracy: 1.0
```

Through the use of Logistic Regression, the accuracy presents itself as 100% effective. This either means that the method is perfect for this specific analysis, or that there might be something wrong with it that goes much deeper than at first glance.

0.0.4 K Nearest Neighbours

[10]:
```
enc = OrdinalEncoder()
n_df = enc.fit_transform(df)
y = n_df[:,:1]
X = n_df[:,1:]
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3,␣
↪random_state=42)
```

[11]:
```
lr = LogisticRegression()
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[1200   57]
 [  65 1116]]
Classification Report:
              precision    recall  f1-score   support
```

```
         0.0       0.95      0.95      0.95      1257
         1.0       0.95      0.94      0.95      1181

    accuracy                           0.95      2438
   macro avg       0.95      0.95      0.95      2438
weighted avg       0.95      0.95      0.95      2438

Accuracy: 0.9499589827727646
```

Before beginning with KNN, a second process of Logistic Regression is made using the Ordinal Encoder instead of getting dummies for the categorical columns. This appears to work more appropriately, since the accuracy results are more realistic. Even though the results are still very good, the KNN how shown to be a better predictor.

[12]:
```
def knn_tuning(k):
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    return accuracy

knn_list = []
for i in range(1,10):
    knn_list.append(knn_tuning(i))
knn_list
```

[12]:
```
[0.9987694831829368,
 0.9983593109105825,
 0.9963084495488105,
 0.9971287940935193,
 0.9954881050041017,
 0.9958982772764561,
 0.994667760459393,
 0.9938474159146842,
 0.9942575881870386]
```

[13]:
```
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
```

```
[[1252    5]
 [   2 1179]]
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1257
         1.0       1.00      1.00      1.00      1181

    accuracy                           1.00      2438
   macro avg       1.00      1.00      1.00      2438
weighted avg       1.00      1.00      1.00      2438

Accuracy: 0.9971287940935193
```

From the KNN tuning, the K value of 4 was chosen as the ideal, since k=1 or 2,though higher values, are able to get higher accuracy due to their own overfitting. Since the data is not properly divided into neighbours, everything fits to it. With k=3, the accuracy falls a lot, rising again with 4 and then falling once again, making 4 possibly the best match.

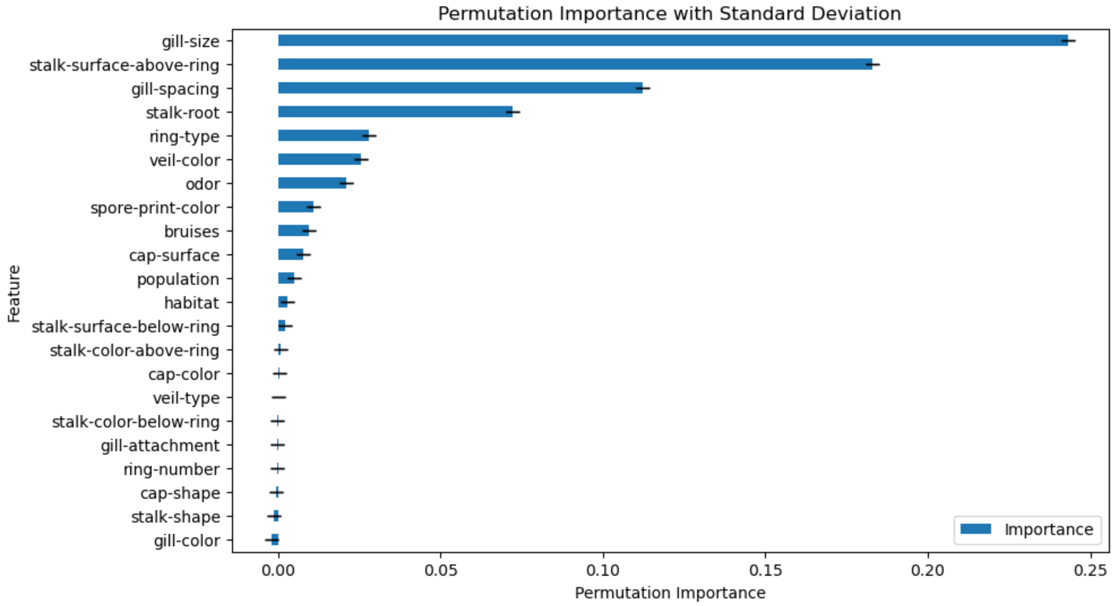
0.0.5 Checking significance of features

[14]:
```python
result = permutation_importance(lr, X_test, y_test, n_repeats=50,
 ↪random_state=42)


feature_importance = pd.DataFrame({'Feature': df.drop('class', axis=1).columns,
                                   'Importance': result.importances_mean,
                                   'Standard Deviation': result.importances_std})
feature_importance = feature_importance.sort_values('Importance', ascending=True)


ax = feature_importance.plot(x='Feature', y='Importance', kind='barh',
 ↪figsize=(10, 6), yerr='Standard Deviation', capsize=4)
ax.set_xlabel('Permutation Importance')
ax.set_title('Permutation Importance with Standard Deviation')
```

[14]:
```
Text(0.5, 1.0, 'Permutation Importance with Standard Deviation')
```



From this analysis, the most influential feature on the classification of mushrooms as poisonous or edible is gill size. The stalk surface above ring is a strong second influence. Gill spacing is still quite meaningful. Others can still be significant, but the three should be able to provide a better model.

[15]:
```
enc = OrdinalEncoder()
df2 = df[['class', 'gill-size','stalk-surface-above-ring','gill-spacing']]
n_df2 = enc.fit_transform(df2)
y = n_df2[:,:1]
X = n_df2[:,1:]
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3,␣
↪random_state=42)
```

[16]:
```
lr2 = LogisticRegression()
lr2.fit(X_train,y_train)
y_pred = lr2.predict(X_test)
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[1205   52]
 [  92 1089]]
Classification Report:
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.94      1257
         1.0       0.95      0.92      0.94      1181

    accuracy                           0.94      2438
   macro avg       0.94      0.94      0.94      2438
weighted avg       0.94      0.94      0.94      2438
```

```
Accuracy: 0.940935192780968
```

As it can be seen, the selection of most significant features in Logistic Regression results in lower values than using all the features. Only the recall for edibles is 0.01 higher.