jccoulson_6

March 19, 2024

1 Homework 6

1.0.1 CSCI 611, Spring 2024

Jesse Coulson

1.1 Problem Statement

We are trying to classify mushrooms into edible or poisonous based on categorical features.

Subtasks: - Transform categorical attributes to numeric attributes - build a decision tree classifier and a multi tree classifier - Create visualizations of results from classifiers with metrics - generate graphical form of the tree diagram from best decision tree - analyze the experience of making a decision tree clasifier from a categorical dataset

[1]:
```
#libraries needed for this ipynb
import seaborn as sns
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
import csv
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import GridSearchCV
```

1.2 Reading in Data and Checking Format

- Reading in data with read_csv from csv downloaded from assignment page
- Printing out head of dataframe to see sample of vars
- Examine info on data types of dataframe

[2]:
```
#read in df
df= pd.read_csv("mushroom.csv")

#print data to see data and format
print(df.head(), "\n")
```

```
#verify all types are categorical
df.info()
```

```
  cap-shape cap-surface          cap-color bruises?    odor gill-attachment  \
0    convex               smooth     white  bruises  almond            free
1    convex               smooth     white  bruises  almond            free
2    convex               smooth     white  bruises  almond            free
3    convex               smooth     white  bruises  almond            free
4    convex               smooth     white  bruises  almond            free

  gill-spacing gill-size gill-color stalk-shape  … stalk-color-above-ring  \
0      crowded    narrow      white    tapering  …                  white
1      crowded    narrow      white    tapering  …                  white
2      crowded    narrow       pink    tapering  …                  white
3      crowded    narrow       pink    tapering  …                  white
4      crowded    narrow      brown    tapering  …                  white

  stalk-surface-color-ring veil-type veil-color ring-number ring-type  \
0                    white   partial      white         one   pendant
1                    white   partial      white         one   pendant
2                    white   partial      white         one   pendant
3                    white   partial      white         one   pendant
4                    white   partial      white         one   pendant

  spore-print-color    population habitat edibleness
0               purple    several   woods     edible
1                brown    several   woods     edible
2               purple    several   woods     edible
3                brown    several   woods     edible
4               purple    several   woods     edible

[5 rows x 23 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8416 entries, 0 to 8415
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   cap-shape                 8416 non-null   object
 1   cap-surface               8416 non-null   object
 2   cap-color                 8416 non-null   object
 3   bruises?                  8416 non-null   object
 4   odor                      8416 non-null   object
 5   gill-attachment           8416 non-null   object
 6   gill-spacing              8416 non-null   object
 7   gill-size                 8416 non-null   object
 8   gill-color                8416 non-null   object
```

```
 9   stalk-shape               8416 non-null   object
 10  stalk-root                5936 non-null   object
 11  stalk-surface-above-ring  8416 non-null   object
 12  stalk-surface-below-ring  8416 non-null   object
 13  stalk-color-above-ring    8416 non-null   object
 14  stalk-surface-color-ring  8416 non-null   object
 15  veil-type                 8416 non-null   object
 16  veil-color                8416 non-null   object
 17  ring-number               8416 non-null   object
 18  ring-type                 8416 non-null   object
 19  spore-print-color         8416 non-null   object
 20  population                8416 non-null   object
 21  habitat                   8416 non-null   object
 22  edibleness                8416 non-null   object
dtypes: object(23)
memory usage: 1.5+ MB
```

1.3 Categorical Variable transformation

- using ColumnTransformer to turn all features numeric
- will use one hot encoding for some and ordinal for others where there is hierarchy
- encoding binary variables before using column transformer

[3]:
```
#the column names have whitespace need to strip out
df.columns = df.columns.str.strip()

#veil type only has one type so doesnt make sense to use, so will remove
print(df['veil-type'].unique())
df = df.drop(columns=['veil-type'])

bin_var = {'edibleness': {'edible':1, 'poisonous':0}}
df = df.replace(bin_var)

bin_var = {'bruises?': {'no':0, 'bruises':1}}
df = df.replace(bin_var)

ordinal_vars = ['gill-size', 'ring-number']

onehot_vars = ['cap-shape',
'cap-surface',
'cap-color',
'odor',
'gill-attachment',
'gill-spacing',
'gill-color',
'stalk-shape',
'stalk-root',
'stalk-surface-above-ring',
```

```
'stalk-surface-below-ring',
'stalk-color-above-ring',
'stalk-surface-color-ring',
'veil-color',
'ring-type',
'spore-print-color',
'population',
'habitat']
```

```
['partial']
```

[4]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

#column transform to tansform all features at once
column_transformer = ColumnTransformer(transformers=[
    ('onehot', OneHotEncoder(), onehot_vars),
    ('ordinal', OrdinalEncoder(), ordinal_vars),
    ],
    remainder='passthrough' #for the already encoded binary vars
    )
```

[5]:
```
trans_data = column_transformer.fit_transform(df)

#names for onehot encoded vars
onehot_feature_names = column_transformer.named_transformers_['onehot'].
 ↪get_feature_names_out()

#combining all the feature names for the df
all_feature_names = np.concatenate([onehot_feature_names, ordinal_vars,␣
 ↪['bruises?','edibleness']]) #putting binary encoded vars in correct order

trans_df = pd.DataFrame(trans_data.toarray(), columns=all_feature_names)
trans_df.head()
```

[5]:

	cap-shape_bell	cap-shape_conical	cap-shape_convex	cap-shape_flat	\
0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	
2	0.0	0.0	1.0	0.0	
3	0.0	0.0	1.0	0.0	
4	0.0	0.0	1.0	0.0	

	cap-shape_knobbed	cap-shape_sunken	cap-surface_fibrous	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	

```
4                0.0               0.0                  0.0

   cap-surface_grooves  cap-surface_scaly  cap-surface_smooth  …  \
0                  0.0                0.0                 1.0  …
1                  0.0                0.0                 1.0  …
2                  0.0                0.0                 1.0  …
3                  0.0                0.0                 1.0  …
4                  0.0                0.0                 1.0  …

   habitat_leaves  habitat_meadows  habitat_paths  habitat_urban  \
0             0.0              0.0            0.0            0.0
1             0.0              0.0            0.0            0.0
2             0.0              0.0            0.0            0.0
3             0.0              0.0            0.0            0.0
4             0.0              0.0            0.0            0.0

   habitat_waste  habitat_woods  gill-size  ring-number  bruises?  edibleness
0            0.0            1.0        1.0          1.0       1.0         1.0
1            0.0            1.0        1.0          1.0       1.0         1.0
2            0.0            1.0        1.0          1.0       1.0         1.0
3            0.0            1.0        1.0          1.0       1.0         1.0
4            0.0            1.0        1.0          1.0       1.0         1.0

[5 rows x 113 columns]
```

1.3.1 Categorical columns transformation conclusion

Overall I ended up 18 variables with one hot encoding, two variables with ordinal encoding, and 1 feature with 1 target for binary encoding. For binary encoding I had to manually map the values to 0 and 1 because binary encoder would have required a lot more work to add to the column transformer pipeline. Now after all the transformations there are many columns, over 100. There has to be a binary column for each category in the features that were one hot encoded

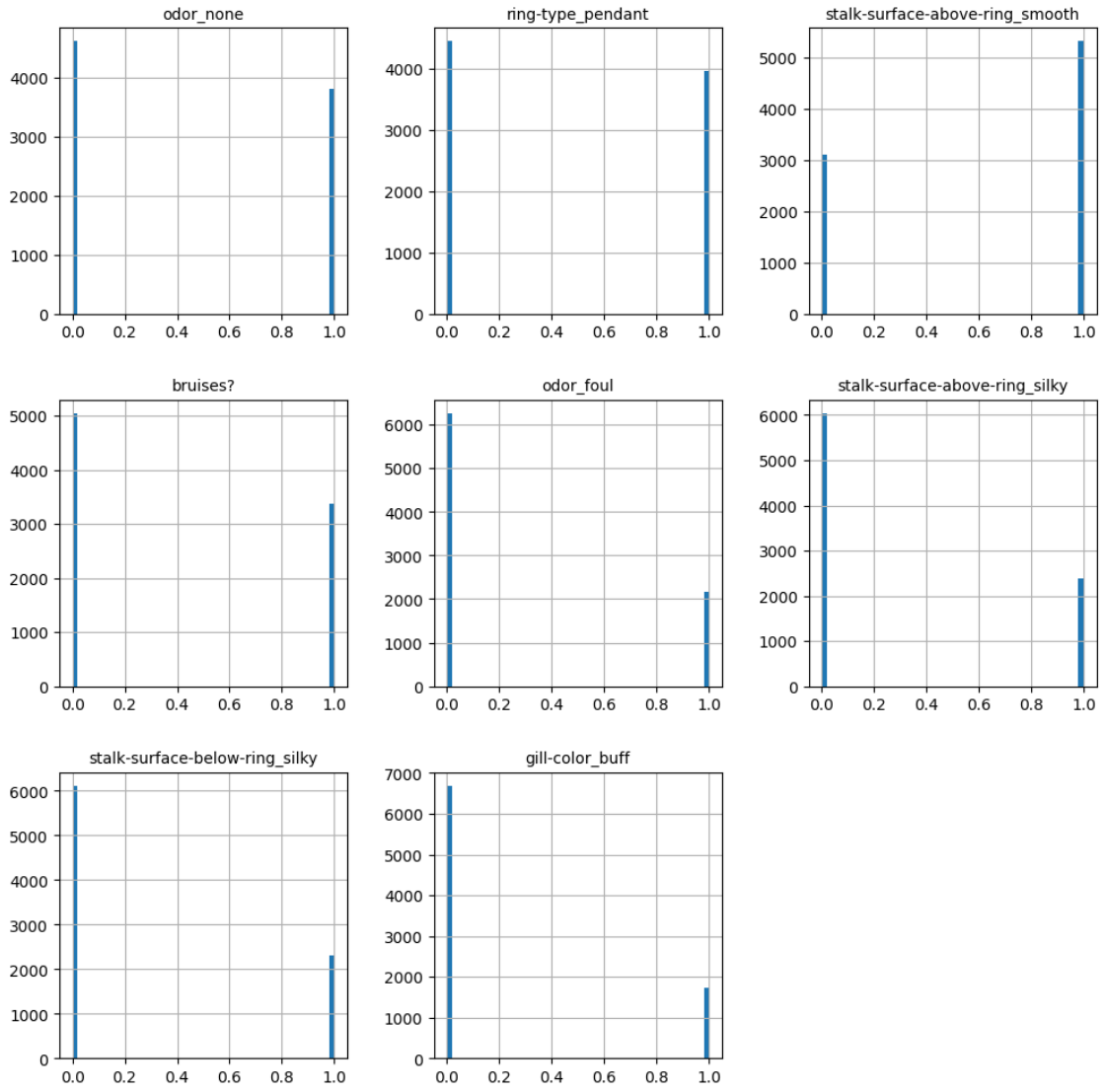
1.4 Finding promosing features and making visualizations

- Using correlation matrix to find promosing features correlated with edibleness
- Choosing the top 4 positively correlated vars and the top 4 negatively correlated vars
- vars are odor_none, ring-type_pendant, stalk-surface-above-ring_smooth, bruises?, odor_foul, stalk-surface-above-ring_silky, stalk-surface-below-ring_silky, gill-color_buff
- will use heatmaps to try and see correlation with features and target, will only use 4 heatmaps for brevity

[6]:
```
#seeing correlation of all vars
corr_matrix = trans_df.corr(numeric_only= True)
corr_matrix["edibleness"].sort_values(ascending=False)
```

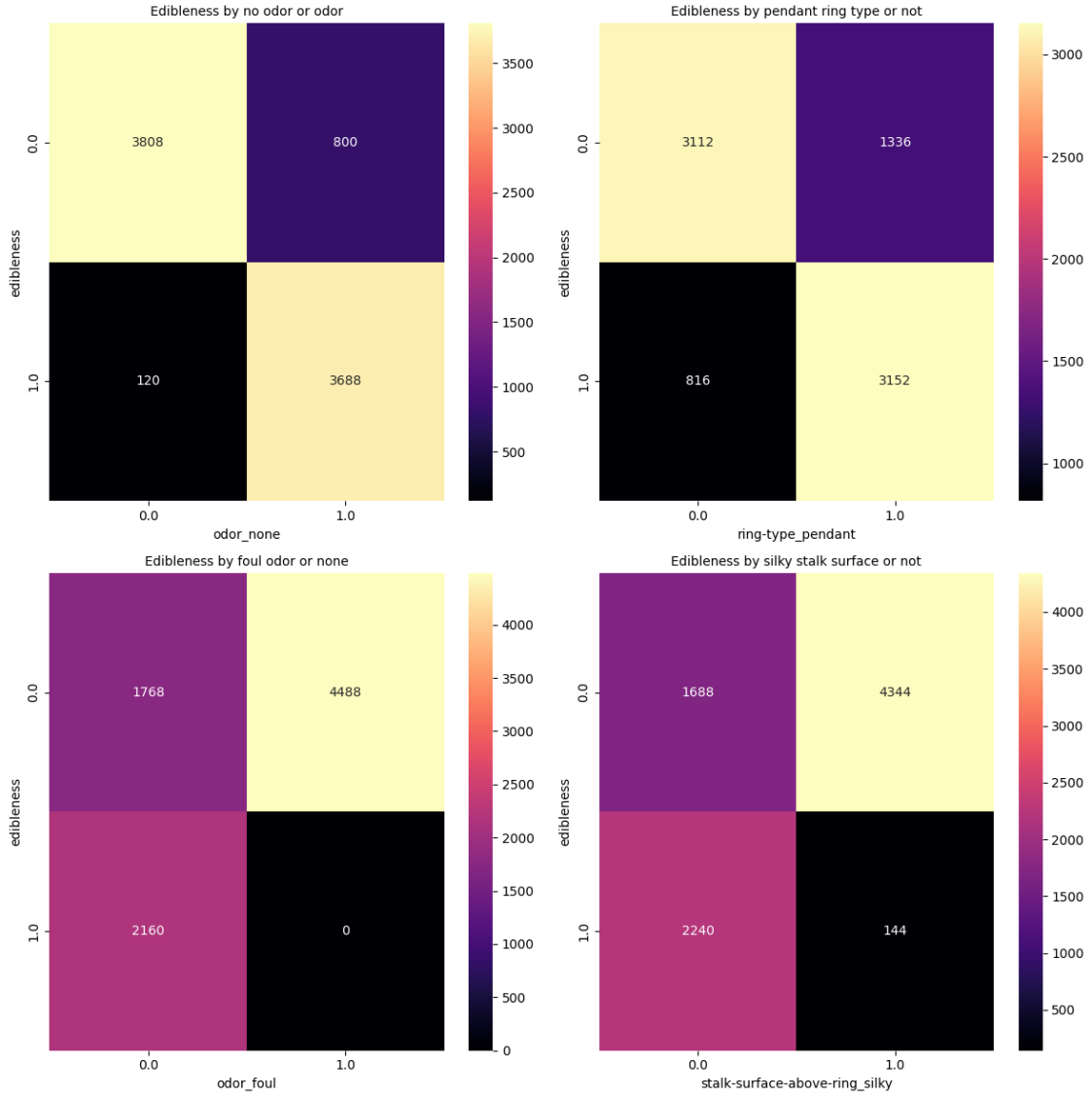
[6]:
```
edibleness                          1.000000
odor_none                           0.793034
ring-type_pendant                   0.494286
stalk-surface-above-ring_smooth     0.466675
bruises?                            0.462454
                                       …
gill-size                          -0.540032
gill-color_buff                    -0.543331
stalk-surface-below-ring_silky     -0.579364
stalk-surface-above-ring_silky     -0.595875
odor_foul                          -0.628086
Name: edibleness, Length: 113, dtype: float64
```

[7]:
```
#columns to view histograms of
cols = ['odor_none', 'ring-type_pendant', 'stalk-surface-above-ring_smooth',␣
 ↪'bruises?', 'odor_foul', 'stalk-surface-above-ring_silky',␣
 ↪'stalk-surface-below-ring_silky', 'gill-color_buff']

#histograms of the variables
plt.rc('font', size=10)
plt.rc('axes', labelsize=10, titlesize=10)
plt.rc('legend', fontsize=10)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)
#small plots side by side
trans_df[cols].hist(bins=50, figsize=(12, 12))
#title for histograms
plt.suptitle('Histogram of Each Individual Variable', fontsize=17, fontweight =␣
 ↪"bold",verticalalignment='top', horizontalalignment='center')
plt.show()
```



[8]:
```
#heatmaps of 4 vars

#creating crosstab of the 4 features
col_freq_1 = pd.crosstab(trans_df['odor_none'], trans_df['edibleness'])
col_freq_2 = pd.crosstab(trans_df['ring-type_pendant'], trans_df['edibleness'])
col_freq_3 = pd.crosstab(trans_df['odor_foul'], trans_df['edibleness'])
col_freq_4 = pd.crosstab(trans_df['stalk-surface-above-ring_silky'],␣
 ↪trans_df['edibleness'])

#setting up plot in 2x2 grid
```

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))

#heatmap for first feature
sns.heatmap(col_freq_1, annot=True, fmt='d', cmap='magma', ax=axes[0, 0])
axes[0, 0].set_title('Edibleness by no odor or odor')
axes[0, 0].set_xlabel('odor_none')
axes[0, 0].set_ylabel('edibleness')

#2nd feature
sns.heatmap(col_freq_2, annot=True, fmt='d', cmap='magma', ax=axes[0, 1])
axes[0, 1].set_title('Edibleness by pendant ring type or not')
axes[0, 1].set_xlabel('ring-type_pendant')
axes[0, 1].set_ylabel('edibleness')

#3rd feature
sns.heatmap(col_freq_3, annot=True, fmt='d', cmap='magma', ax=axes[1, 0])
axes[1, 0].set_title('Edibleness by foul odor or none')
axes[1, 0].set_xlabel('odor_foul')
axes[1, 0].set_ylabel('edibleness')

#4th feature
sns.heatmap(col_freq_4, annot=True, fmt='d', cmap='magma', ax=axes[1, 1])
axes[1, 1].set_title('Edibleness by silky stalk surface or not')
axes[1, 1].set_xlabel('stalk-surface-above-ring_silky')
axes[1, 1].set_ylabel('edibleness')

#tight layout
plt.tight_layout()
plt.show()
```



1.4.1 Visualizations of important features conclusion

All the top features are binary, from the histograms most of the variables have an uneven distributuion of having the category or not. This is fine since its highly correlated and nothing too skewed to being only one category. From the heatmaps the features match up closely to edibleness. For example when the odor is not foul there is over 2160 samples that are edible and when it is foul there are non that are edible.

1.5 Splitting data into training and testing

- Spliting data into 80% training, 20% testing
- using the 8 features I identified from eda

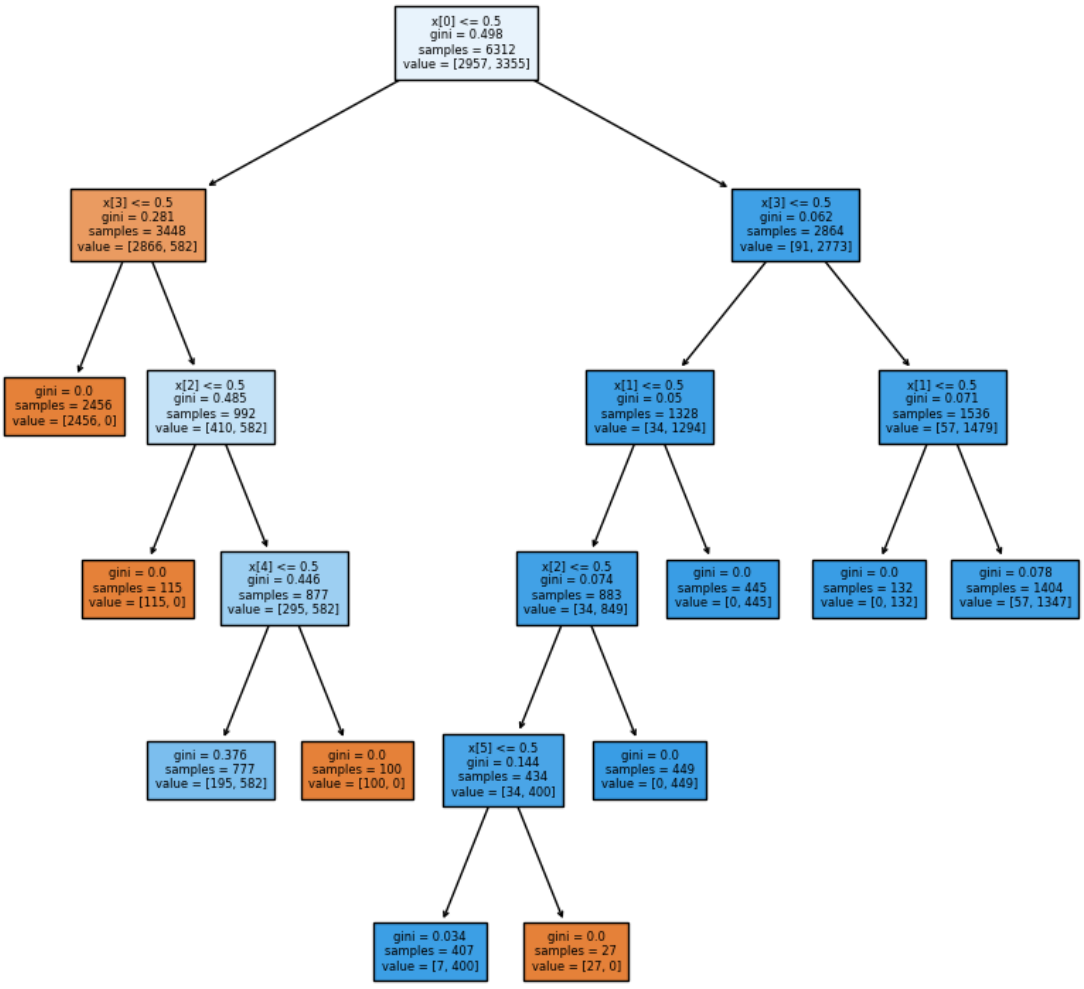
[9]:
```
X = trans_df[['odor_none', 'ring-type_pendant',␣
 ↪'stalk-surface-above-ring_smooth', 'bruises?', 'odor_foul',␣
 ↪'stalk-surface-above-ring_silky', 'stalk-surface-below-ring_silky',␣
 ↪'gill-color_buff']]
y = trans_df['edibleness'] #get target feature

#splitting into training and testing
X_train, X_test, y_train, y_test = train_test_split(
   X, y, random_state=42)
```

1.6 Creating models with grid search and visualizing tree

- finding optimal hyperparameters using grid search with 5 fold cross validation
- using decisiontreeclassifier and randomforestclassifier
- using hyperparameters criterion, max_depth, min_samples_split, and max_features for both classifiers
- Adding another hyperparameter for bootstrap for randomforest
- visualizing best decision tree with plot_tree

[10]:
```
#using grdsearch with 5 fold cross validation
#these hyperparameters were chosen because they are very important to decision␣
 ↪trees
decision_tree_grid= GridSearchCV(estimator=DecisionTreeClassifier(random_state␣
 ↪=42),
             param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [None,␣
 ↪5, 10], 'min_samples_split': [2,5,10],
                         'max_features':['sqrt', 'log2']
                         }, cv=5)

#fit on train data
decision_tree_grid.fit(X_train, y_train)

#printing the optimal hyperparametrs and accuracy of the model
print("Hypereparams of best decision tree model:", decision_tree_grid.
 ↪best_params_)
print("Accuracy score of best decision tree model:", decision_tree_grid.
 ↪best_score_)

#visualizing best tree
plt.figure(figsize=(10, 10))  # Set the figure size (optional)
plot_tree(decision_tree_grid.best_estimator_, filled=True, fontsize=6) #need␣
 ↪filled=True for color
plt.show()
```

```
Hypereparams of best decision tree model: {'criterion': 'gini', 'max_depth':
None, 'max_features': 'sqrt', 'min_samples_split': 2}
Accuracy score of best decision tree model: 0.9589658361283538
```



1.6.1 Decision tree visual observations

You can see that the decision tree is very good at classifying the the training data. It ends with many pure leaf nodes. The tree has a depth of 5 and each split seems to have a lot of information gain

[11]:
```
#only extra hyperparameter added is bootstrap, leaving n_estimators at default␣
↪100
random_forest_grid= GridSearchCV(estimator=RandomForestClassifier(),
            param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [None,␣
↪5, 10], 'min_samples_split': [2,5,10],
                        'max_features':['sqrt', 'log2'], 'bootstrap': [True,␣
↪False]
```

```
                          }, cv=5)

random_forest_grid.fit(X_train, y_train)

#printing the optimal hyperparametrs and accuracy of the model
print("Hypereparams of best random forest model:", random_forest_grid.
 ↪best_params_)
print("Accuracy score of best random forest model:", random_forest_grid.
 ↪best_score_)
```

```
Hypereparams of best random forest model: {'bootstrap': True, 'criterion':
'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_split': 2}
Accuracy score of best random forest model: 0.9589658361283538
```

1.6.2 Models created from gridsearch conclusion

Both the models have very high accuracy over 95%. The optimal hyperparameters found were 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_split': 2. This makes sense for getting the maximum score. Having no max depth will let it go for maximum accuracy, min samples split being 2 allows internal nodes to split easier. All the hyperparams help optimize score. For random forest all the same hyperparameters were selected, bootstrap True was selected for the randomforest only hyperparam. It makes sense because bootstrap allows each tree in the randomforest to see different data so that should help increase robustness of model

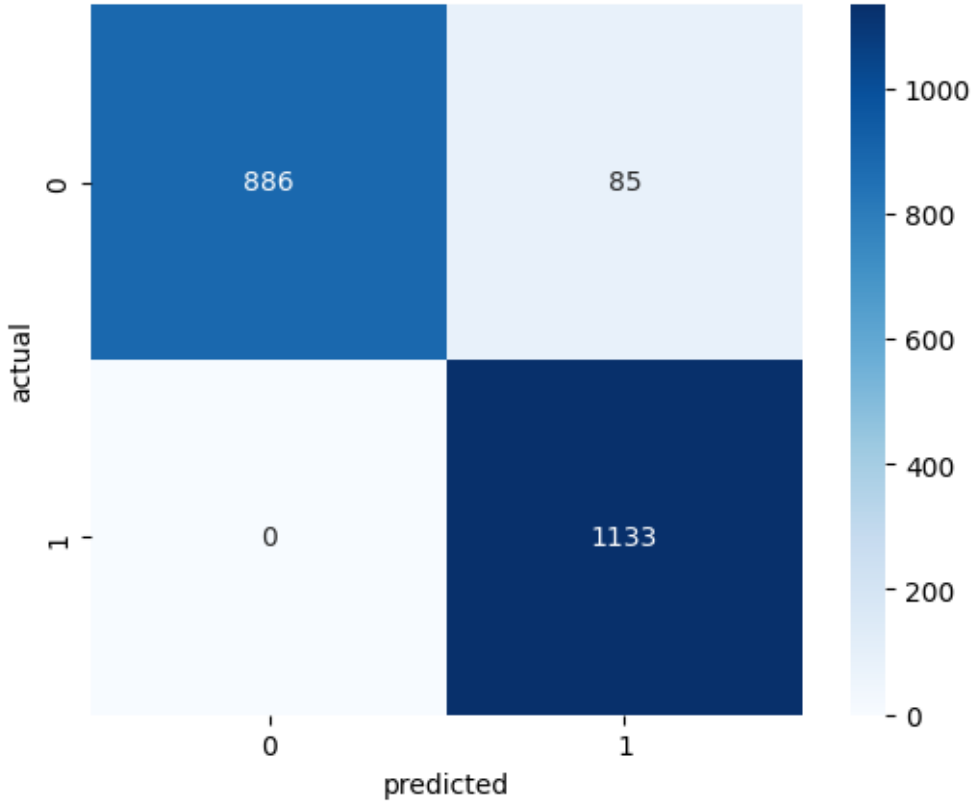
1.7 Final test metrics for both models

- Will predict with test data to find metrics and display confusion metrics for both models
- Metrics I will be analyzing are accuracy, precision, recall, and f1 score

[12]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
y_pred = decision_tree_grid.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy on test set: ",acc)

#proportion of positives that were correctly identifed out of all actual␣
 ↪positives
precision = precision_score(y_test, y_pred, average='macro')
print("Precision: ", precision)

#how many positives were correctly identified
recall = recall_score(y_test, y_pred, average='macro')
print("Recall: ", recall)

#metric that combines precision and recall
f1 = f1_score(y_test, y_pred, average='macro')
print("F1 score: ", f1)
```

```
#creating confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, square=True, annot=True, fmt='d', cmap="Blues",␣
 ↪cbar=True)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.show()
plt.clf()
```

```
Accuracy on test set:  0.9596007604562737
Precision:  0.9651067323481117
Recall:  0.9562306900102986
F1 score:  0.9590362102584928
```

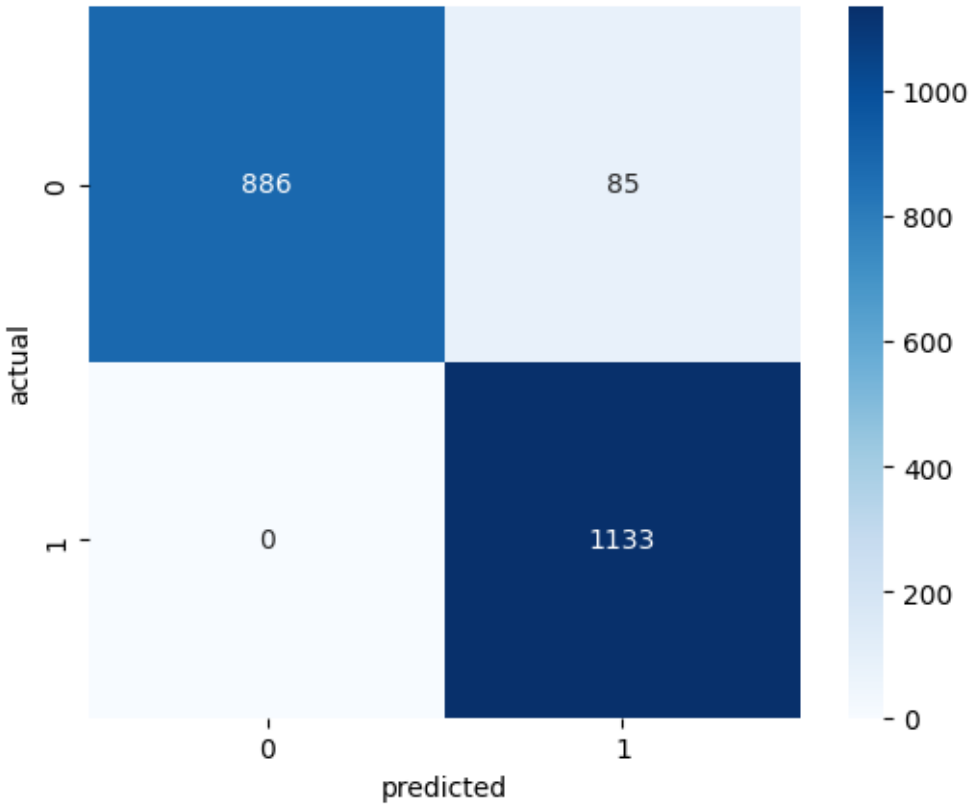



```
<Figure size 640x480 with 0 Axes>
```

[13]:
```
y_pred = random_forest_grid.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy on test set: ",acc)
```

```
#proportion of positives that were correctly identifed out of all actual␣
 ↪positives
precision = precision_score(y_test, y_pred, average='macro')
print("Precision: ", precision)

#how many positives were correctly identified
recall = recall_score(y_test, y_pred, average='macro')
print("Recall: ", recall)

#metric that combines precision and recall
f1 = f1_score(y_test, y_pred, average='macro')
print("F1 score: ", f1)


#creating confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, square=True, annot=True, fmt='d', cmap="Blues",␣
 ↪cbar=True)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.show()
```

```
Accuracy on test set:  0.9596007604562737
Precision:  0.9651067323481117
Recall:  0.9562306900102986
F1 score:  0.9590362102584928
```



1.7.1 Final Metrics Conclusion

Both of thes models have the exact same metrics. At first I was puzzled by this and spent a long time checking for errors or why this could possibly be. I changed around the hyperparamaters and even tried other ensemble methods like adaboost but I still go the same results. From research online I blieve they have the exact same metrics because the data is all binary features. This makes them all go downa similar path and predict the same outcomes. To addres the actual metrics, they are very good! The accuracy is about 96% which is very high. The precision, recall, and f1 score are all similar which means there is not a huge variance in false psoitives vs flase negatives. From the confusion matrix it did not misclassify any of the plants it predicted as edible. For the prediction of non edible it misclassified 85 that should have been edible. Overall very good performance for the models

2 Conclusion of tree building and experience with categorical dataset

Overal tree building was good, the models were very accurate and the added visual component is very compelling of creating trees. Using a categorical only dataset on the otherhand can get unweildy. When tehre is no heirarhy between the variables and we have to onehot encode we end up having to create many columns that corespond to each category of each variable. All thes variables end up binary so they do not give as much information as other types of features. These

categorical vars also caused me the issue of same performance on botht he ensemble and single decision tree which would not have occured from a dataset that was only categorical. Overall the creation of the decision trees and visualizations were positive while dealing with only non heirarchal categorical variables was negative