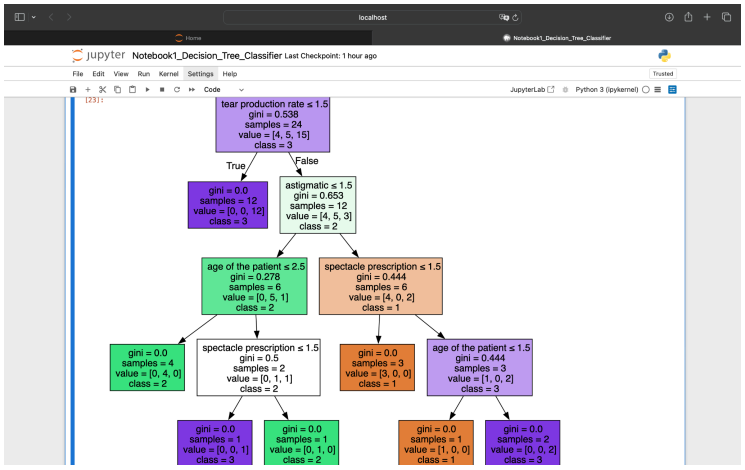
Ashley Chen
24 February 2025

DSCI549 - Homework 3

1. a. 

b. The accuracy of the classifier is 0.83 or 83% (±0.31 or 31%).

[26]:

```
print("Scores:")
[print(score) for score in scores]
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Scores:
1.0
0.875
0.625
Accuracy: 0.83 (+/- 0.31)
```

c. I input the features [young, myope, astigmatic, reduced] → encoded as 0,0,1,0, and the trained classifier predicted 3, corresponding to hard lenses.

[50]:

```
testset=input('Please Enter Your Prediction Set:')
testset=testset.strip().split(",")
temp=[]
for i in range(len(testset)):
    temp.append(float(testset[i]))
testset=np.array(temp).reshape((1,len(temp)))
predictions=clf.predict(testset)
```

```
Please Enter Your Prediction Set: 0,0,1,0
```

[51]:

```
print(predictions)
```

```
['3']
```

d. The warning says that one of my classes only has 4 examples, which is less than the 7 folds I inputted. This indicates that some of the folds won't get any examples of that smaller class at all.

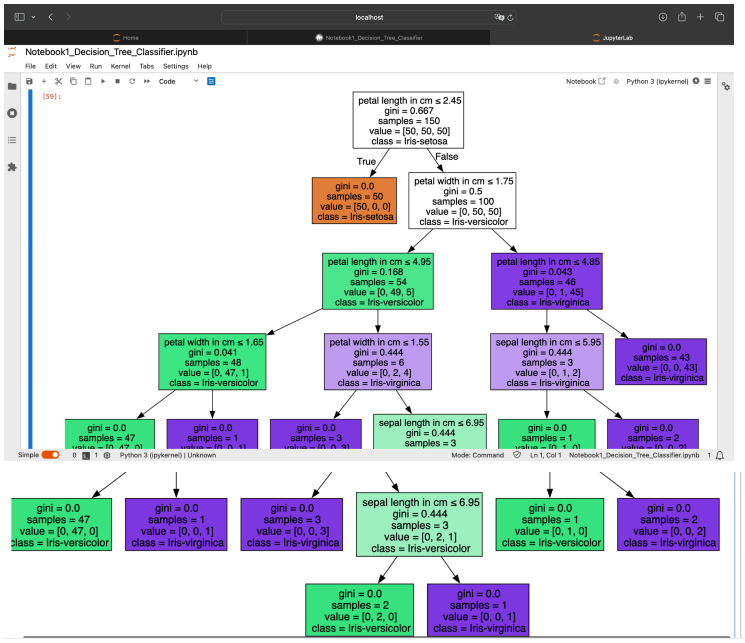
As a result, the classifier doesn't see and can't test on those examples in certain folds, making the evaluation less reliable.

[52]:

```
n_foldCV=int(input("Please Enter the Number of Folds:"))
attributes,instances,labels=loadDataSet(dataset)
clf = tree.DecisionTreeClassifier()
clf = clf.fit(instances,labels)
scores = cross_val_score(clf, instances, labels, cv=n_foldCV)
```

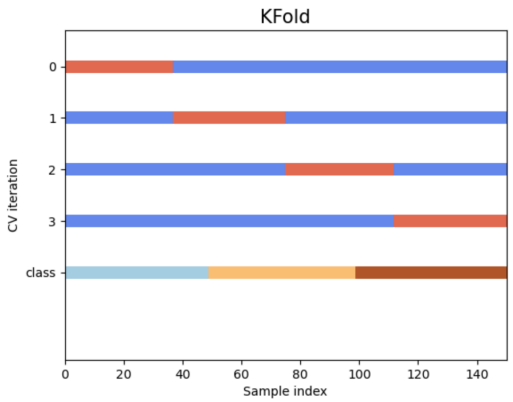
```
Please Enter the Number of Folds: 7
/opt/anaconda3/lib/python3.12/site-packages/sklearn/model_selection/_split.py:776: UserWarning: The least populated class in y has only 4 members, which is less than n_splits=7.
  warnings.warn(
```

e. iris.csv:



e. I used 4 folds for this dataset and got an accuracy of 0.96 or 96% (±0.03 or 3%).

[9]: `<Axes: title={'center': 'KFold'}, xlabel='Sample index', ylabel='CV iteration'>`



[10]:

```
print("Scores:")
[print(score) for score in scores]
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Scores:
0.9736842105263158
0.9473684210526315
0.9459459459459459
0.972972972972973
Accuracy: 0.96 (+/- 0.03)
```

2. a. The Gaussian Naive Bayes Classifier is best for data with numeric continuous features that follow a normal distribution. That's why the Iris dataset is better for this classifier because it has real-valued measurements like sepal, petal lengths, and widths. On the other hand, the lenses dataset is best for the Multinomial Naive Bayes dataset because this classifier is best for categorical or count-based features. The lenses dataset includes features like age group and prescription type.
   b. Using the Iris dataset for the Gaussian Naive Bayes Classifier with 3 folds, the accuracy came out to be 0.94 or 94% ($\pm$0.03 or 3%). Using the lenses dataset for the Multinomial Naive Bayes Classifier with 4 folds, the accuracy came out to be 0.62 or 62% ($\pm$0.14 or 14%).

[13]:
```
dataset=input('Please Enter Your Dataset:')
attributes,instances,labels=loadDataSet(dataset)
clf_G = GaussianNB()
clf_G.fit(instances, labels)
print("Gaussian Naïve Bayes is used.")
```
```
Please Enter Your Dataset: iris.csv
Gaussian Naïve Bayes is used.
```

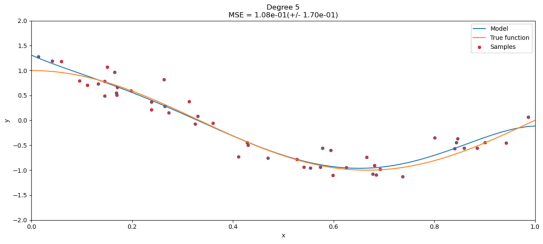
[14]:
```
n_foldCV=int(input("Please Enter the Number of Folds:"))
attributes,instances,labels=loadDataSet(dataset)
clf_G = GaussianNB()
scores = cross_val_score(clf_G, instances, labels, cv=n_foldCV)
print("======GaussianNB======")
print(np.array2string(scores,separator=","))
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```
```
Please Enter the Number of Folds: 3
======GaussianNB======
[0.92,0.94,0.96]
Accuracy: 0.94 (+/- 0.03)
```

[16]:
```
dataset=input('Please Enter Your Dataset:')
attributes,instances,labels=loadDataSet(dataset)
clf_M = MultinomialNB()
clf_M.fit(instances, labels)
print("Multinomial Naïve Bayes is used.")
```
```
Please Enter Your Dataset: lenses.csv
Multinomial Naïve Bayes is used.
```

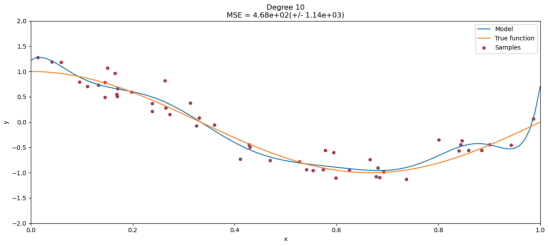
[19]:
```
n_foldCV=int(input("Please Enter the Number of Folds:"))
attributes,instances,labels=loadDataSet(dataset)
clf_M = MultinomialNB()
scores = cross_val_score(clf_M, instances, labels, cv=n_foldCV)
print("======MultinomialNB======")
print(np.array2string(scores,separator=","))
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```
```
Please Enter the Number of Folds: 4
======MultinomialNB======
[0.66666667,0.66666667,0.66666667,0.5       ]
Accuracy: 0.62 (+/- 0.14)
```

3. a. The degrees of polynomials I used were 5, 10, 15, and 20, respectively.

[9]:
```
plt.figure(figsize=(15, 6))

scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                         scoring="neg_mean_squared_error", cv=10)

X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
plt.plot(X_test, true_fun(X_test), label="True function")
plt.scatter(X, y, edgecolor='r', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
    degrees, -scores.mean(), scores.std()))
plt.show()
```

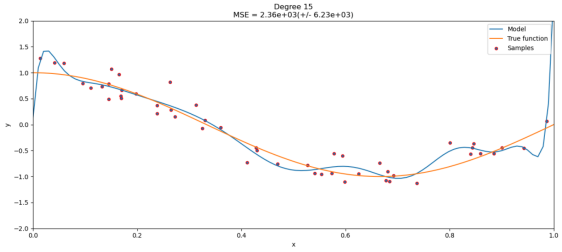


[11]:
```
plt.figure(figsize=(15, 6))

scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                         scoring="neg_mean_squared_error", cv=10)

X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
plt.plot(X_test, true_fun(X_test), label="True function")
plt.scatter(X, y, edgecolor='r', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
    degrees, -scores.mean(), scores.std()))
plt.show()
```



[13]:

```
plt.figure(figsize=(15, 6))

scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                         scoring="neg_mean_squared_error", cv=10)

X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
plt.plot(X_test, true_fun(X_test), label="True function")
plt.scatter(X, y, edgecolor='r', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
    degrees, -scores.mean(), scores.std()))
plt.show()
```



[15]:

```
plt.figure(figsize=(15, 6))

scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                         scoring="neg_mean_squared_error", cv=10)

X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
plt.plot(X_test, true_fun(X_test), label="True function")
plt.scatter(X, y, edgecolor='r', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
    degrees, -scores.mean(), scores.std()))
plt.show()
```