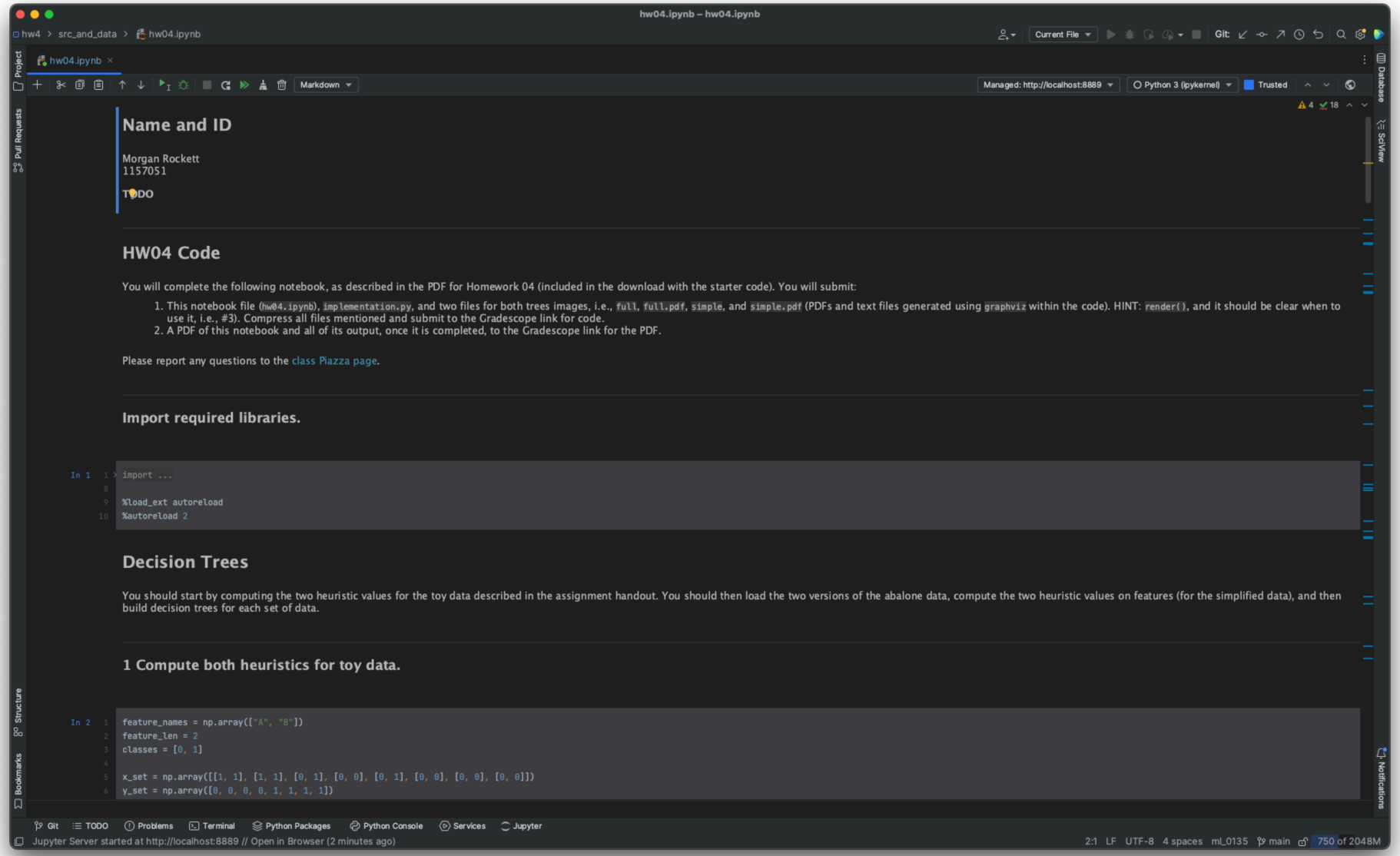
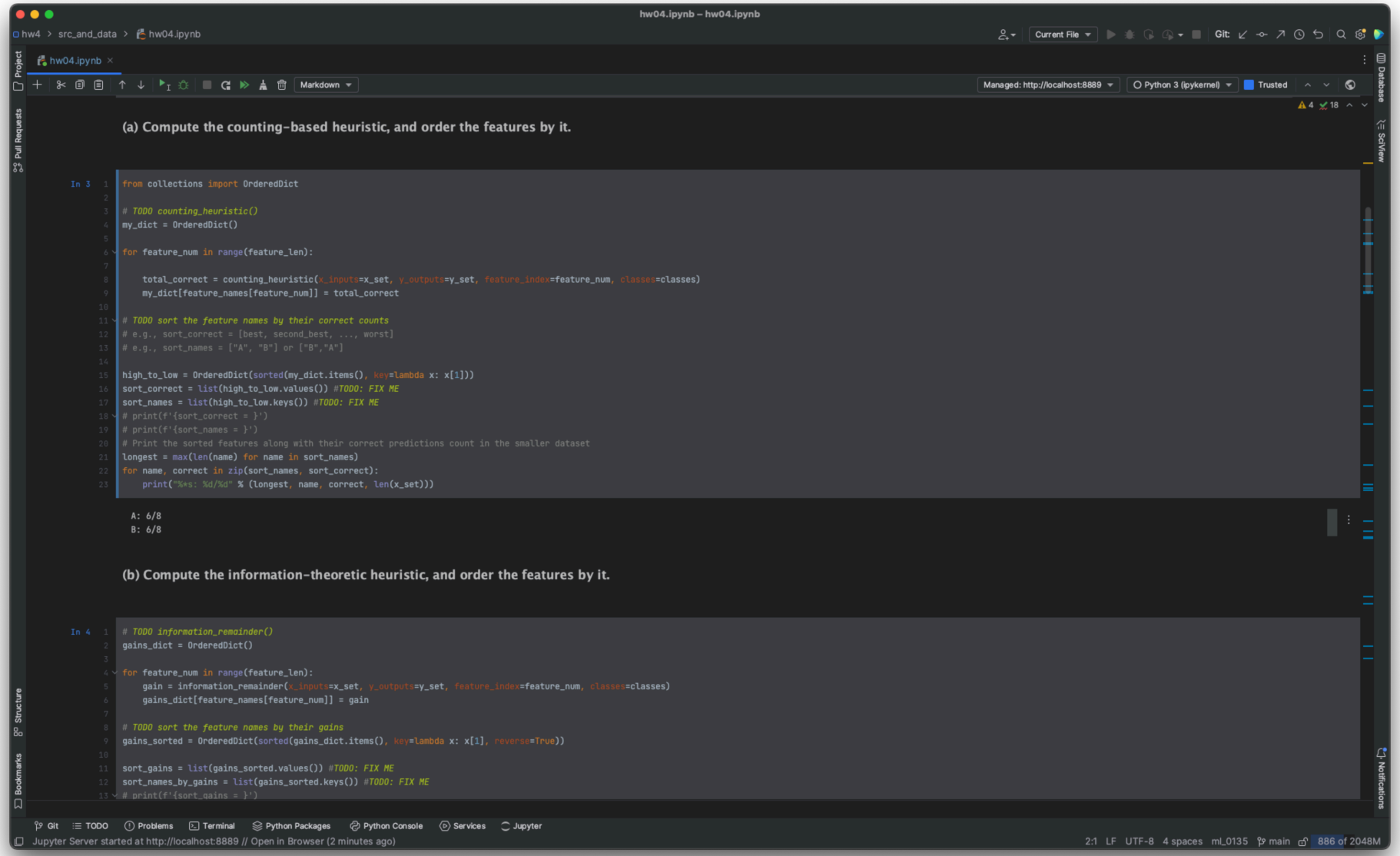
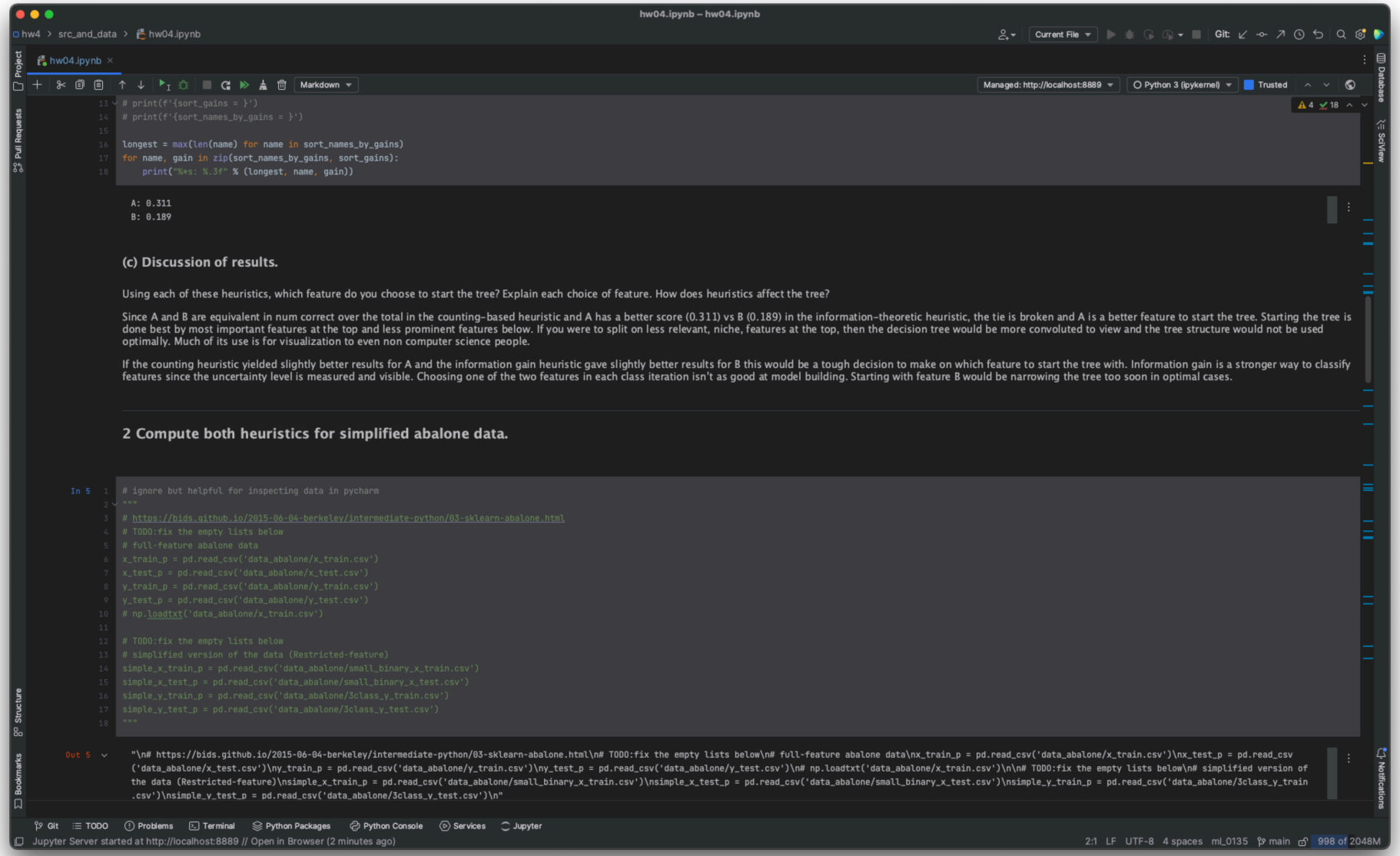
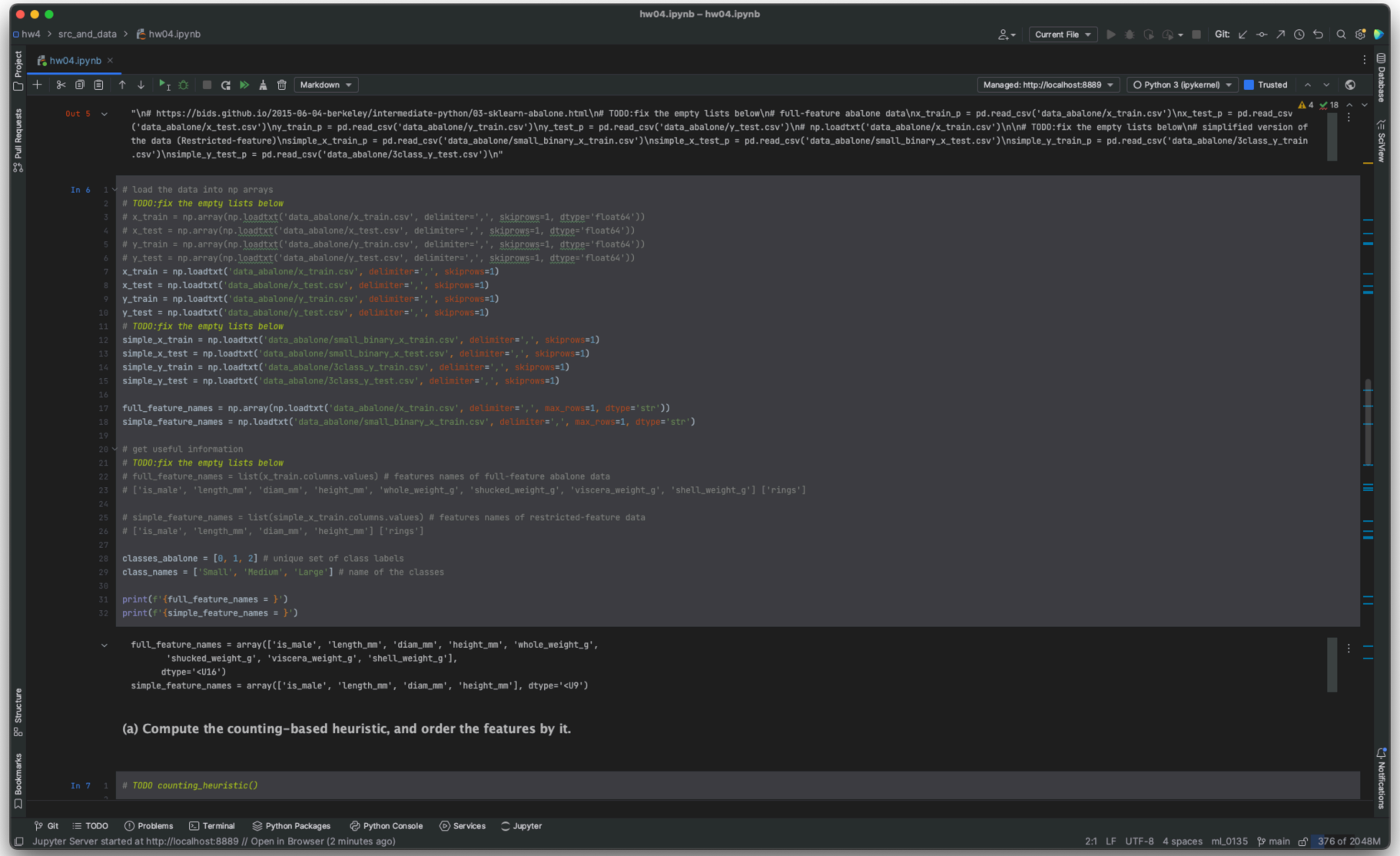
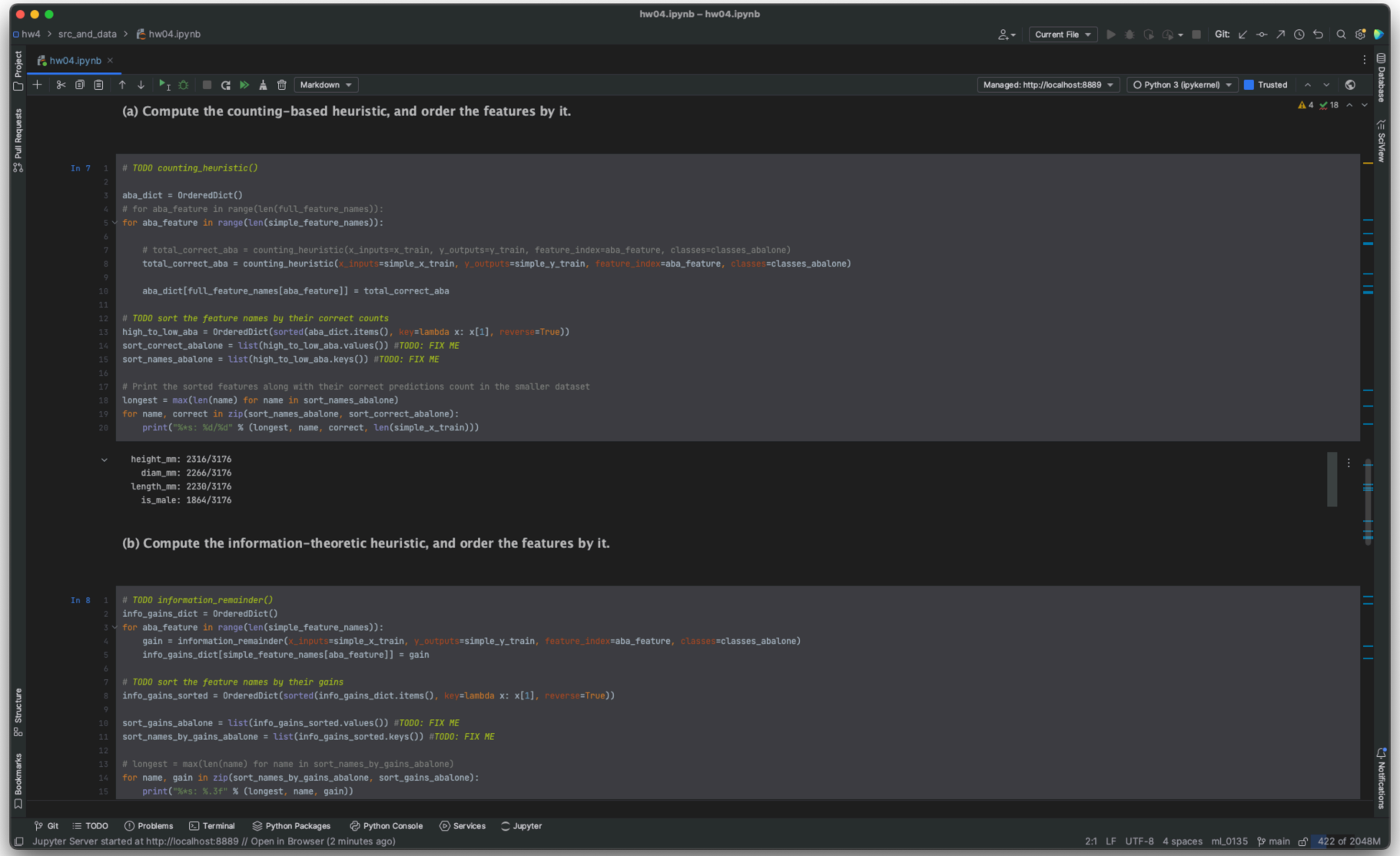

hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
Name and ID
Morgan Rockett
1157051
TODO
HW04 Code
You will complete the following notebook, as described in the PDF for Homework 04 (included in the download with the starter code). You will submit:
1. This notebook file (hw04.ipynb), implementation.py, and two files for both trees images, i.e., full, full.pdf, simple, and simple.pdf (PDFs and text files generated using graphviz within the code). HINT: render(), and it should be clear when to use it, i.e., #3). Compress all files mentioned and submit to the Gradescope link for code.
2. A PDF of this notebook and all of its output, once it is completed, to the Gradescope link for the PDF.
Please report any questions to the class Piazza page.
Import required libraries.
In 1
import ...
%load_ext autoreload
%autoreload 2
Decision Trees
You should start by computing the two heuristic values for the toy data described in the assignment handout. You should then load the two versions of the abalone data, compute the two heuristic values on features (for the simplified data), and then build decision trees for each set of data.
1 Compute both heuristics for toy data.
In 2
feature_names = np.array(["A", "B"])
feature_len = 2
classes = [0, 1]
x_set = np.array([[1, 1], [1, 1], [0, 1], [0, 0], [0, 1], [0, 0], [0, 0], [0, 0]])
y_set = np.array([0, 0, 0, 0, 1, 1, 1, 1])
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main 750 of 2048M


hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
(a) Compute the counting-based heuristic, and order the features by it.
In 3
from collections import OrderedDict
# TODO counting_heuristic()
my_dict = OrderedDict()
for feature_num in range(feature_len):
total_correct = counting_heuristic(x_inputs=x_set, y_outputs=y_set, feature_index=feature_num, classes=classes)
my_dict[feature_names[feature_num]] = total_correct
# TODO sort the feature names by their correct counts
# e.g., sort_correct = [best, second_best, ..., worst]
# e.g., sort_names = ["A", "B"] or ["B","A"]
high_to_low = OrderedDict(sorted(my_dict.items(), key=lambda x: x[1]))
sort_correct = list(high_to_low.values()) #TODO: FIX ME
sort_names = list(high_to_low.keys()) #TODO: FIX ME
# print(f'{sort_correct = }')
# print(f'{sort_names = }')
# Print the sorted features along with their correct predictions count in the smaller dataset
longest = max(len(name) for name in sort_names)
for name, correct in zip(sort_names, sort_correct):
print("%*s: %d/%d" % (longest, name, correct, len(x_set)))
A: 6/8
B: 6/8
(b) Compute the information-theoretic heuristic, and order the features by it.
In 4
# TODO information_remainder()
gains_dict = OrderedDict()
for feature_num in range(feature_len):
gain = information_remainder(x_inputs=x_set, y_outputs=y_set, feature_index=feature_num, classes=classes)
gains_dict[feature_names[feature_num]] = gain
# TODO sort the feature names by their gains
gains_sorted = OrderedDict(sorted(gains_dict.items(), key=lambda x: x[1], reverse=True))
sort_gains = list(gains_sorted.values()) #TODO: FIX ME
sort_names_by_gains = list(gains_sorted.keys()) #TODO: FIX ME
# print(f'{sort_gains = }')
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main
886 of 2048M


hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
# print(f'{sort_gains = }')
# print(f'{sort_names_by_gains = }')
longest = max(len(name) for name in sort_names_by_gains)
for name, gain in zip(sort_names_by_gains, sort_gains):
print("%*s: %.3f" % (longest, name, gain))
A: 0.311
B: 0.189
(c) Discussion of results.
Using each of these heuristics, which feature do you choose to start the tree? Explain each choice of feature. How does heuristics affect the tree?
Since A and B are equivalent in num correct over the total in the counting-based heuristic and A has a better score (0.311) vs B (0.189) in the information-theoretic heuristic, the tie is broken and A is a better feature to start the tree. Starting the tree is done best by most important features at the top and less prominent features below. If you were to split on less relevant, niche, features at the top, then the decision tree would be more convoluted to view and the tree structure would not be used optimally. Much of its use is for visualization to even non computer science people.
If the counting heuristic yielded slightly better results for A and the information gain heuristic gave slightly better results for B this would be a tough decision to make on which feature to start the tree with. Information gain is a stronger way to classify features since the uncertainty level is measured and visible. Choosing one of the two features in each class iteration isn't as good at model building. Starting with feature B would be narrowing the tree too soon in optimal cases.
2 Compute both heuristics for simplified abalone data.
In 5
ignore but helpful for inspecting data in pycharm
"""
# https://bids.github.io/2015-06-04-berkeley/intermediate-python/03-sklearn-abalone.html
# TODO:fix the empty lists below
# full-feature abalone data
x_train_p = pd.read_csv('data_abalone/x_train.csv')
x_test_p = pd.read_csv('data_abalone/x_test.csv')
y_train_p = pd.read_csv('data_abalone/y_train.csv')
y_test_p = pd.read_csv('data_abalone/y_test.csv')
# np.loadtxt('data_abalone/x_train.csv')
# TODO:fix the empty lists below
# simplified version of the data (Restricted-feature)
simple_x_train_p = pd.read_csv('data_abalone/small_binary_x_train.csv')
simple_x_test_p = pd.read_csv('data_abalone/small_binary_x_test.csv')
simple_y_train_p = pd.read_csv('data_abalone/3class_y_train.csv')
simple_y_test_p = pd.read_csv('data_abalone/3class_y_test.csv')
"""
Out 5
"\n# https://bids.github.io/2015-06-04-berkeley/intermediate-python/03-sklearn-abalone.html\n# TODO:fix the empty lists below\n# full-feature abalone data\nx_train_p = pd.read_csv('data_abalone/x_train.csv')\nx_test_p = pd.read_csv('data_abalone/x_test.csv')\ny_train_p = pd.read_csv('data_abalone/y_train.csv')\ny_test_p = pd.read_csv('data_abalone/y_test.csv')\n# np.loadtxt('data_abalone/x_train.csv')\n\n# TODO:fix the empty lists below\n# simplified version of the data (Restricted-feature)\nsimple_x_train_p = pd.read_csv('data_abalone/small_binary_x_train.csv')\nsimple_x_test_p = pd.read_csv('data_abalone/small_binary_x_test.csv')\nsimple_y_train_p = pd.read_csv('data_abalone/3class_y_train.csv')\nsimple_y_test_p = pd.read_csv('data_abalone/3class_y_test.csv')\n"
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main 998 of 2048M


hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
Out 5
"\n# https://bids.github.io/2015-06-04-berkeley/intermediate-python/03-sklearn-abalone.html\n# TODO:fix the empty lists below\n# full-feature abalone data\nx_train_p = pd.read_csv('data_abalone/x_train.csv')\nx_test_p = pd.read_csv('data_abalone/x_test.csv')\ny_train_p = pd.read_csv('data_abalone/y_train.csv')\ny_test_p = pd.read_csv('data_abalone/y_test.csv')\n# np.loadtxt('data_abalone/x_train.csv')\n\n# TODO:fix the empty lists below\n# simplified version of the data (Restricted-feature)\nsimple_x_train_p = pd.read_csv('data_abalone/small_binary_x_train.csv')\nsimple_x_test_p = pd.read_csv('data_abalone/small_binary_x_test.csv')\nsimple_y_train_p = pd.read_csv('data_abalone/3class_y_train.csv')\nsimple_y_test_p = pd.read_csv('data_abalone/3class_y_test.csv')\n"
In 6
# load the data into np arrays
# TODO:fix the empty lists below
# x_train = np.array(np.loadtxt('data_abalone/x_train.csv', delimiter=',', skiprows=1, dtype='float64'))
# x_test = np.array(np.loadtxt('data_abalone/x_test.csv', delimiter=',', skiprows=1, dtype='float64'))
# y_train = np.array(np.loadtxt('data_abalone/y_train.csv', delimiter=',', skiprows=1, dtype='float64'))
# y_test = np.array(np.loadtxt('data_abalone/y_test.csv', delimiter=',', skiprows=1, dtype='float64'))
x_train = np.loadtxt('data_abalone/x_train.csv', delimiter=',', skiprows=1)
x_test = np.loadtxt('data_abalone/x_test.csv', delimiter=',', skiprows=1)
y_train = np.loadtxt('data_abalone/y_train.csv', delimiter=',', skiprows=1)
y_test = np.loadtxt('data_abalone/y_test.csv', delimiter=',', skiprows=1)
# TODO:fix the empty lists below
simple_x_train = np.loadtxt('data_abalone/small_binary_x_train.csv', delimiter=',', skiprows=1)
simple_x_test = np.loadtxt('data_abalone/small_binary_x_test.csv', delimiter=',', skiprows=1)
simple_y_train = np.loadtxt('data_abalone/3class_y_train.csv', delimiter=',', skiprows=1)
simple_y_test = np.loadtxt('data_abalone/3class_y_test.csv', delimiter=',', skiprows=1)
full_feature_names = np.array(np.loadtxt('data_abalone/x_train.csv', delimiter=',', max_rows=1, dtype='str'))
simple_feature_names = np.loadtxt('data_abalone/small_binary_x_train.csv', delimiter=',', max_rows=1, dtype='str')
# get useful information
# TODO:fix the empty lists below
# full_feature_names = list(x_train.columns.values) # features names of full-feature abalone data
# ['is_male', 'length_mm', 'diam_mm', 'height_mm', 'whole_weight_g', 'shucked_weight_g', 'viscera_weight_g', 'shell_weight_g'] ['rings']
# simple_feature_names = list(simple_x_train.columns.values) # features names of restricted-feature data
# ['is_male', 'length_mm', 'diam_mm', 'height_mm'] ['rings']
classes_abalone = [0, 1, 2] # unique set of class labels
class_names = ['Small', 'Medium', 'Large'] # name of the classes
print(f'{full_feature_names = }')
print(f'{simple_feature_names = }')
full_feature_names = array(['is_male', 'length_mm', 'diam_mm', 'height_mm', 'whole_weight_g',
'shucked_weight_g', 'viscera_weight_g', 'shell_weight_g'],
dtype='<U16')
simple_feature_names = array(['is_male', 'length_mm', 'diam_mm', 'height_mm'], dtype='<U9')
(a) Compute the counting-based heuristic, and order the features by it.
In 7
TODO counting_heuristic()
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main 376 of 2048M


hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
(a) Compute the counting-based heuristic, and order the features by it.
In 7
# TODO counting_heuristic()
aba_dict = OrderedDict()
# for aba_feature in range(len(full_feature_names)):
for aba_feature in range(len(simple_feature_names)):
# total_correct_aba = counting_heuristic(x_inputs=x_train, y_outputs=y_train, feature_index=aba_feature, classes=classes_abalone)
total_correct_aba = counting_heuristic(x_inputs=simple_x_train, y_outputs=simple_y_train, feature_index=aba_feature, classes=classes_abalone)
aba_dict[full_feature_names[aba_feature]] = total_correct_aba
# TODO sort the feature names by their correct counts
high_to_low_aba = OrderedDict(sorted(aba_dict.items(), key=lambda x: x[1], reverse=True))
sort_correct_abalone = list(high_to_low_aba.values()) #TODO: FIX ME
sort_names_abalone = list(high_to_low_aba.keys()) #TODO: FIX ME
# Print the sorted features along with their correct predictions count in the smaller dataset
longest = max(len(name) for name in sort_names_abalone)
for name, correct in zip(sort_names_abalone, sort_correct_abalone):
print("%*s: %d/%d" % (longest, name, correct, len(simple_x_train)))
height_mm: 2316/3176
diam_mm: 2266/3176
length_mm: 2230/3176
is_male: 1864/3176
(b) Compute the information-theoretic heuristic, and order the features by it.
In 8
# TODO information_remainder()
info_gains_dict = OrderedDict()
for aba_feature in range(len(simple_feature_names)):
gain = information_remainder(x_inputs=simple_x_train, y_outputs=simple_y_train, feature_index=aba_feature, classes=classes_abalone)
info_gains_dict[simple_feature_names[aba_feature]] = gain
# TODO sort the feature names by their gains
info_gains_sorted = OrderedDict(sorted(info_gains_dict.items(), key=lambda x: x[1], reverse=True))
sort_gains_abalone = list(info_gains_sorted.values()) #TODO: FIX ME
sort_names_by_gains_abalone = list(info_gains_sorted.keys()) #TODO: FIX ME
# longest = max(len(name) for name in sort_names_by_gains_abalone)
for name, gain in zip(sort_names_by_gains_abalone, sort_gains_abalone):
print("%*s: %.3f" % (longest, name, gain))
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main 422 of 2048M

hw04.ipynb – hw04.ipynb

hw4 > src_and_data > hw04.ipynb

Current File | Git:

hw04.ipynb

Markdown | Managed: http://localhost:8889 | Python 3 (ipykernel) | Trusted

4 18

(b) Compute the information-theoretic heuristic, and order the features by it.

In 8

```
# TODO information_remainder()
info_gains_dict = OrderedDict()
for aba_feature in range(len(simple_feature_names)):
    gain = information_remainder(x_inputs=simple_x_train, y_outputs=simple_y_train, feature_index=aba_feature, classes=classes_abalone)
    info_gains_dict[simple_feature_names[aba_feature]] = gain

# TODO sort the feature names by their gains
info_gains_sorted = OrderedDict(sorted(info_gains_dict.items(), key=lambda x: x[1], reverse=True))

sort_gains_abalone = list(info_gains_sorted.values()) #TODO: FIX ME
sort_names_by_gains_abalone = list(info_gains_sorted.keys()) #TODO: FIX ME

# longest = max(len(name) for name in sort_names_by_gains_abalone)
for name, gain in zip(sort_names_by_gains_abalone, sort_gains_abalone):
    print("%*s: %.3f" % (longest, name, gain))
```

```
 height_mm: 0.173
   diam_mm: 0.150
 length_mm: 0.135
   is_male: 0.025
```

3) Generate decision trees (criterion='entropy', random_state=42) for full- and simple-feature data

(a) Train and eval on entire train and test sets. Print accuracy values and generate tree images.

Render the tree diagram, naming it "full." A text file and PDF should be created and saved (i.e., `full` and `full.pdf`) - include both in submission.

In 9

```
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import graphviz

full_model = DecisionTreeClassifier(criterion='entropy', random_state=42)
full_model.fit(X=x_train, y=y_train)

train_accuracy = full_model.score(X=x_train, y=y_train)
test_accuracy = full_model.score(X=x_test, y=y_test)

# TODO calculate accuracies
print(f"Accuracy (train): {train_accuracy:.3f}")
print(f"Accuracy  (test): {test_accuracy:.3f}")
```

Git | TODO | Problems | Terminal | Python Packages | Python Console | Services | Jupyter

Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)

2:1 LF UTF-8 4 spaces ml_0135 main 477 of 2048M

hw04.ipynb – hw04.ipynb

hw4 > src_and_data > hw04.ipynb

Current File | Git:

hw04.ipynb

Markdown | Managed: http://localhost:8889 | Python 3 (ipykernel) | Trusted

3) Generate decision trees (criterion='entropy', random_state=42) for full- and simple-feature data

(a) Train and eval on entire train and test sets. Print accuracy values and generate tree images.

Render the tree diagram, naming it "full." A text file and PDF should be created and saved (i.e., `full` and `full.pdf`) – include both in submission.

In 9

```
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import graphviz

full_model = DecisionTreeClassifier(criterion='entropy', random_state=42)
full_model.fit(X=x_train, y=y_train)

train_accuracy = full_model.score(X=x_train, y=y_train)
test_accuracy = full_model.score(X=x_test, y=y_test)

# TODO calculate accuracies
print(f"Accuracy (train): {train_accuracy:.3f}")
print(f"Accuracy  (test): {test_accuracy:.3f}")

# TODO generate tree image
full_tree = export_graphviz(decision_tree=full_model, feature_names=full_feature_names)
generated_graph_full = graphviz.Source(full_tree)
generated_graph_full.render('full')
```

```
Accuracy (train): 1.000
Accuracy  (test): 0.202
```

Out 9

```
'full.pdf'
```

(b) Restricted-feature (aka simple) data.

Train and eval on simple train and test sets. Same as above, accept this time use the `simple` set. Render the tree diagram, naming it "simple." A text file and PDF should be created and saved (i.e., `simple` and `simple.pdf`) – include both in submission.

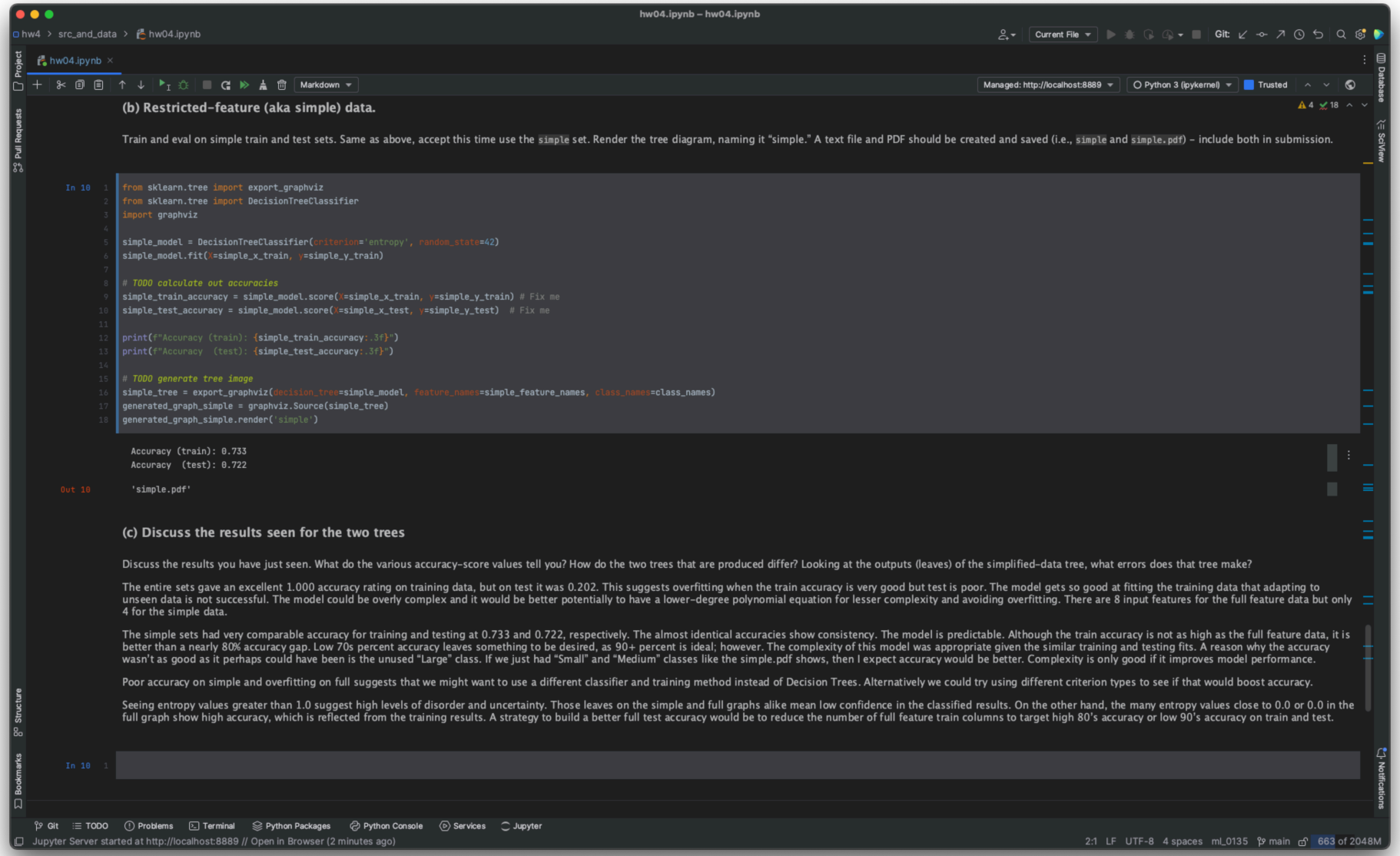
In 10

```
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import graphviz

simple_model = DecisionTreeClassifier(criterion='entropy', random_state=42)
simple_model.fit(X=simple_x_train, y=simple_y_train)

# TODO calculate out accuracies
simple_train_accuracy = simple_model.score(X=simple_x_train, y=simple_y_train) # Fix me
```

Git | TODO | Problems | Terminal | Python Packages | Python Console | Services | Jupyter

Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)

2:1 LF UTF-8 4 spaces ml_0135 main 566 of 2048M


hw04.ipynb – hw04.ipynb
hw4 > src_and_data > hw04.ipynb
Current File
Git:
hw04.ipynb
Markdown
Managed: http://localhost:8889
Python 3 (ipykernel)
Trusted
(b) Restricted-feature (aka simple) data.
Train and eval on simple train and test sets. Same as above, accept this time use the simple set. Render the tree diagram, naming it "simple." A text file and PDF should be created and saved (i.e., simple and simple.pdf) – include both in submission.
In 10
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import graphviz
simple_model = DecisionTreeClassifier(criterion='entropy', random_state=42)
simple_model.fit(X=simple_x_train, y=simple_y_train)
# TODO calculate out accuracies
simple_train_accuracy = simple_model.score(X=simple_x_train, y=simple_y_train) # Fix me
simple_test_accuracy = simple_model.score(X=simple_x_test, y=simple_y_test) # Fix me
print(f"Accuracy (train): {simple_train_accuracy:.3f}")
print(f"Accuracy (test): {simple_test_accuracy:.3f}")
# TODO generate tree image
simple_tree = export_graphviz(decision_tree=simple_model, feature_names=simple_feature_names, class_names=class_names)
generated_graph_simple = graphviz.Source(simple_tree)
generated_graph_simple.render('simple')
Accuracy (train): 0.733
Accuracy (test): 0.722
Out 10
'simple.pdf'
(c) Discuss the results seen for the two trees
Discuss the results you have just seen. What do the various accuracy-score values tell you? How do the two trees that are produced differ? Looking at the outputs (leaves) of the simplified-data tree, what errors does that tree make?
The entire sets gave an excellent 1.000 accuracy rating on training data, but on test it was 0.202. This suggests overfitting when the train accuracy is very good but test is poor. The model gets so good at fitting the training data that adapting to unseen data is not successful. The model could be overly complex and it would be better potentially to have a lower-degree polynomial equation for lesser complexity and avoiding overfitting. There are 8 input features for the full feature data but only 4 for the simple data.
The simple sets had very comparable accuracy for training and testing at 0.733 and 0.722, respectively. The almost identical accuracies show consistency. The model is predictable. Although the train accuracy is not as high as the full feature data, it is better than a nearly 80% accuracy gap. Low 70s percent accuracy leaves something to be desired, as 90+ percent is ideal; however. The complexity of this model was appropriate given the similar training and testing fits. A reason why the accuracy wasn't as good as it perhaps could have been is the unused "Large" class. If we just had "Small" and "Medium" classes like the simple.pdf shows, then I expect accuracy would be better. Complexity is only good if it improves model performance.
Poor accuracy on simple and overfitting on full suggests that we might want to use a different classifier and training method instead of Decision Trees. Alternatively we could try using different criterion types to see if that would boost accuracy.
Seeing entropy values greater than 1.0 suggest high levels of disorder and uncertainty. Those leaves on the simple and full graphs alike mean low confidence in the classified results. On the other hand, the many entropy values close to 0.0 or 0.0 in the full graph show high accuracy, which is reflected from the training results. A strategy to build a better full test accuracy would be to reduce the number of full feature train columns to target high 80's accuracy or low 90's accuracy on train and test.
In 10
Git
TODO
Problems
Terminal
Python Packages
Python Console
Services
Jupyter
Jupyter Server started at http://localhost:8889 // Open in Browser (2 minutes ago)
2:1 LF UTF-8 4 spaces ml_0135 main 663 of 2048M