Predicting Income with Random Forests

In this project, we will be using a dataset containing census information from UCI's Machine Learning Repository.

By using this census data with a random forest, we will try to predict whether or not a person makes more than $50,000.

For more help click here

Let's get started!

Tasks
17/17Complete

Mark the tasks as complete by checking them off

Investigate The Data

1.
Let's begin by investigating the data available to us. Click on the file `income.csv` and take a look. Notice the first row of the dataset contains the names of our columns. What columns do you think might be helpful in predicting a person's income?

You can find more detailed descriptions of the columns on UCI's Machine Learning Repository.

2.
Go back to the `income.py` file. We want to get all of that data into a Pandas DataFrame. Use the `pd.read_csv()` function using `"income.csv"` as a parameter and store the result in a variable named `income_data`. Since the first row of our file contains the names of the columns, we also want to add the argument `header = 0`.

3.
Take a look at one of the rows of the data we've imported. Print `income_data.iloc[0]` to see the first row in its entirety. Did this person make more than $50,000? What is the name of the column that contains that information?

4.
There's a small problem with our data that is a little hard to catch — every string has an extra space at the start. For example, the first row's `native-country` is `" United-States"`, but we want it to be `"United-States"`. This is happening because in `income.csv` there are spaces after the commas. To fix this, we can add the parameter `delimiter = ", "` to our `read_csv()` function.
Stuck? Get a hint

Format The Data For Scikit-learn

5.
Now that we have our data imported into a DataFrame, we can begin putting it in a format that our Random Forest can work with. To do this, we need to separate the labels from the rest of the data.

For this project, the labels are in the column called `"income"`. We want to grab only this column. You can get a single column from a DataFrame using this syntax:

```
one_column = data_frame_name[["column_name"]]
```

Create a variable named `labels` that contains only the column `"income"` from the `income_data` DataFrame.

6.
We'll also want to pick which columns to use when trying to predict income. For now, let's select `"age"`, `"capital-gain"`, `"capital-loss"`, `"hours-per-week"`, and `"sex"`. Create a new variable named `data` that contains only those columns. The syntax for this is very similar to selecting only one column:

```
many_columns = data_frame_name[["a", "b", "c"]]
```

In this example, `many_columns` now contains the columns `"a"`, `"b"`, and `"c"` from `data_frame_name`.

7.
Finally, we want to split our data and labels into a training set and a test set. We'll use the training set to build the random forest, and the test set to see how accurate it is. Use the `train_test_split()` function to do this.

`train_test_split()` returns four values — name them `train_data`, `test_data`, `train_labels`, and `test_labels`. When calling `train_test_split()`, it should take three arguments — `data`, `labels` and `random_state = 1`.

Create The Random Forest

8.
We're now ready to use this data to build and test our random forest. First, create a `RandomForestClassifier` and name it `forest`. When you create the random forest, use the parameter `random_state = 1`.

9.
Next, we need to fit the model. We want to use the training data and training labels to train the random forest.

Call `forest`'s `.fit()` method using `train_data` and `train_labels` as parameters. When you run your code, there should be an error!

10.
There seems to be a problem with using the column `"sex"` when training the random forest.

In that column, there are values like `"Male"` and `"Female"`. Random forests can't use columns that contain Strings — they have to be continuous values like integers or floats. We'll solve this problem soon, but for now, let's remove the `"sex"` column when creating `data`.

11.
Now that our training set doesn't have a column containing strings, we have successfully fit our random forest.

We can now test its accuracy. Call `forest`'s `.score()` method using `test_data` and `test_labels` as parameters. Print the result.

Changing Column Types

12.
Now that we know the random forest works, let's go back and try to add the `"sex"` column.

Recall that the problem was that this column contained strings. If we transformed those strings into integers, we could use this data!

If we take every row and make every `"Male"` a `0` and every `"Female"` a `1`, we could then use the column in our random forest. Before creating the `data` variable, use this line of code:

```
income_data["sex-int"] = income_data["sex"].apply(lambda row: 0 if row == "Male" else 1)
```

This creates a new column called `"sex-int"` in the `income_data` DataFrame. Every row in that new column contains a `0` if the row's `"sex"` column contained `"Male"` and a `1` otherwise.

13.
Add `"sex-int"` to your list of columns included in `data`. What is your accuracy now?

14.
There are a couple of other columns that use strings that might be useful to use. Let's try transforming the values in the `"native-country"` column.

We should first take a look at the different values that exist in the column. Print `income_data["native-country"].value_counts()`.

15.
Since the majority of the data comes from `"United-States"`, it might make sense to make a column where every row that contains `"United-States"` becomes a `0` and any other country becomes a `1`. Use the syntax from creating the `"sex-int"` column to create a `"country-int"` column.

When mapping Strings to numbers like this, it is important to make sure that continuous numbers make sense. For example, it wouldn't make much sense to map `"United-States"` to `0`, `"Germany"` to `1`, and `"Mexico"` to `2`. If we did this, we're saying that Mexico is more similar to Germany than it is to the United States.

However, if you had values in a column like `"low"`, `"medium"`, and `"high"` mapping those values to `0`, `1`, and `2` would make sense because their representation as Strings is also continuous.
Hint

```
income_data["_____"] = income_data["_____"].apply(________)
```

16.
Add `"country-int"` to the columns used when creating `data`. How does this change the accuracy of your model?

Explore On Your Own

17.
Now that you've gotten the hang of transforming, adding, and removing columns from your training data, it's time to explore on your own to try to make the best classifier possible.

As you play around with the data, here are some ideas that you might want to try:

- Create a `tree.DecisionTreeClassifier`, train it, test is using the same data, and compare the results to the random forest. When does the random forest do better than the single tree? When does a single tree do just as well as the forest?
- After calling `.fit()` on the forest, print `forest.feature_importances_`. This will show you a list of numbers where each number corresponds to the relevance of a column from the training data. Which features tend to be more relevant?
- Use some of the other columns that use continuous variables, or transform columns that use strings!

```python
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

# 2
income_data = pd.read_csv('income.csv', header = 0)
print(income_data.head())

# 3
print(income_data.iloc[0])

# 4
income_df = pd.read_csv("income.csv", header = 0, delimiter = ", ")
print(income_df.iloc[0])

# 12
income_df["sex-int"] = income_df["sex"].apply(lambda row: 0 if row == "Male" else 1)
```

```
# 14
print(income_df["native-country"].value_counts())

# 15
income_df["country-int"] = income_df["native-country"].apply(lambda row: 0 if row == "United-
States" else 1)

# 14
print(income_df["education"].value_counts())

# 15
income_df["education-int"] = income_df["education"].apply(lambda row: 0 if row == "HS-
grad" else 1)

#5
labels = income_df[['income']]

# 6, 10
data = income_df[['age', 'capital-gain', 'capital-loss', 'hours-per-week', 'sex-
int', 'country-int', 'education-int']]

# 7
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, random_stat
e = 1)

# 8
forest = RandomForestClassifier (random_state = 1)

# 9
forest.fit(train_data, train_labels)

# 11
print(forest.score(test_data, test_labels))
```