ISTA 331 HW 6: TREES FOR CLASSIFICATION AND REGRESSION

1. Introduction

In this assignment, you'll train and evaluate some decision trees and random forests for classification and regression tasks.

2. Instructions

2.1. **Code and submission.** Create a Python module called `hw6.py` and upload it to the assignment folder on D2L. You will need the following import statements:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
```

Then implement the 9 functions described in the specifications below.

2.2. **Documentation.** Your script must contain a header docstring containing your name, ISTA 331, the date, and a brief description of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

2.3. **Testing and required files.** You'll need the `test_hw6.py` test script, and the following data files: `training.csv`, `testing.csv`, `bikes.csv`.

2.4. **Grading.** Your submission will be graded by running it through the test script, examination of the plots, and examination of your code. Code should be clear and concise, but you will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

The test script grades the returned values of all functions below except for `plot_confusion_matrix` (which doesn't return anything). These are, collectively, worth 80% of your grade. The remaining 20% is based on the plots.

2.5. **Collaboration.** Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

3. Function Specifications

3.1. Classification trees.

- ~~`get_classification_frames`~~: this function takes no arguments. It creates a `DataFrame` from the `training.csv` and `testing.csv` file. Use only the first 10 columns. The data frame should look like this:

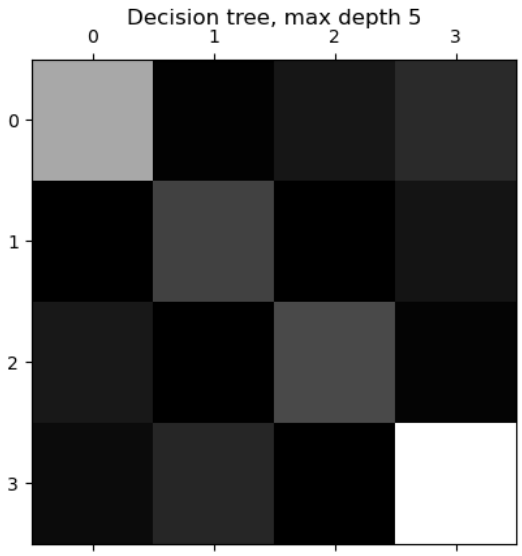
```
In [21]: forest.head()
```

Out[21]:

	class	b1	b2	b3	b4	b5	b6	b7	b8	b9
0	d	39	36	57	91	59	101	93	27	60
1	h	84	30	57	112	51	98	92	26	62
2	s	53	25	49	99	51	93	84	26	58
3	s	59	26	49	103	47	92	82	25	56
4	d	57	49	66	103	64	106	114	28	59

  This data is satellite-based spectroscopy data, attempting to classify four types of land in Japan. The classes are `'s'` (sugi forest), `'h'` (hinoki forest), `'d'` (mixed deciduous forest), `'o'` (non-forest land).[1]
- ~~`get_X_and_y`~~: this function takes a frame and returns a DataFrame of predictors and a vector (Series) of class labels.
- `make_and_test_tree`: this function takes 5 arguments: a training `X`, a training `y`, a testing `X`, a testing `y`, and a maximum depth. Initialize and fit a `DecisionTreeClassifier` on the training data with the given maximum depth. Return a confusion matrix measuring the accuracy of the model on the testing data.
- `plot_confusion_matrix`: this function takes the same 5 arguments as the previous function. Get a confusion matrix from the previous function and display it using `plt.matshow`. Pass the parameter `cmap = plt.cm.gray`. Don't call `plt.show()` in this function; you'll call it later in `main`.

  The resulting confusion matrix plot will look something like this:



[1] Suki and hinoki are types of trees. Not our kind of trees.

3.2. **Regression trees.**

- `get_regression_frame`: this function takes no arguments and creates a `DataFrame` from the `bikes.csv` file. The data frame should look like this:

```
In [19]: data.head()
```

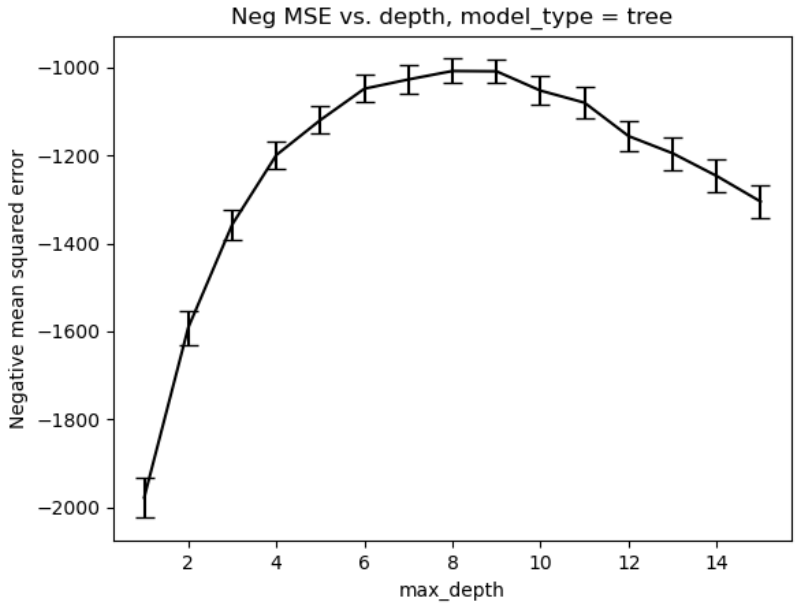
Out[19]:

	datetime	season	holiday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	2011-01-01 00:00:00	1	0	0	1	24.0	28.79	81.0	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	22.0	27.27	80.0	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	22.0	27.27	80.0	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	24.0	28.79	75.0	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	24.0	28.79	75.0	0.0	0	1	1

  This is data from a bike sharing program in London. Our target variable is `casual`, the daily number of non-registered riders.
- `get_regression_X_and_y`: this function takes the frame created by `get_regression_frame` and splits it into training and testing `X` and `y`. Use `np.random.choice` to select a random sample of 15000 instances to be the training set. Take the rest to be the testing set. For `X`, include the predictors `['season', 'holiday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed']`; for `y`, use the target variable `'casual'`.

  Return training `X`, testing `X`, training `y`, testing `y` in that order.
- `make_depth_plot:` this function takes four arguments: a `X` and a `y`, a maximum depth n, and a keyword representing the model type, either `'tree'` or `'forest'`. For each integer `i` between 1 and n inclusive, initalize a model with `max_depth = i`. Make a `DecisionTreeRegressor` if the keyword is `tree`, or a `RandomForestRegressor` if the keyword is `'forest'`. Use `cross_val_score` with `cv = 5` (five-fold-cross-validation) and `scoring = 'neg_mean_squared_error'` to evaluate the model on the training data. This uses negative MSE to evaluate the model (negative to make it a 'score' instead of a 'cost'). Note the mean and standard deviation of the 5 scores.

  When you have evaluated models for all `i` from 1 to n, plot the average score against `i`. Don't call `plt.show` here; you'll call it later in `main`. Put error bars on the plot using the standard error of the scores (standard deviation divided by square root of # of scores). A plot should look something like this:



  Return the value of `i` that produced the best score.

- `compare_regressors`: This function takes a training `X`, a training `y`, testing `X`, a testing `y`, and a list containing a `DecisionTreeRegressor` and a `RandomForestRegressor` (already fit to the training data). For each model, compute its MSE on the training set. Then compute its R^2 using the formula

$$R^2 = 1 - \frac{MSE}{Var(y)}$$

  Var means variance, which you can calculate with `np.var`.

  Next, calculate the model's `RMSE` on the *testing* set. Print the results in the following format:

```
-----------------------------------
Model type:   DecisionTreeRegressor
Depth:        xxxxx
R^2:          xxxxx
Testing RMSE: xxxxx
-----------------------------------
Model type:   RandomForestRegressor
Depth:        xxxxx
R^2:          xxxxx
Testing RMSE: xxxxx
```

  where the `x`s are replaced by the actual values. Round the floats to 4 decimal places.

3.3. **Main function.**

- `main`: get your frames, split them into training and testing data.

  Plot confusion matrices for the classification task with max depth of 1, then max depth of 5. Call `plt.show()` after each call to `plot_confusion_matrix`.

  Make the depth plot for the regression data for each type of model (tree and forest), testing depths from 1 to 15. Call `plt.show()` after each call to `make_depth_plot`.

  Finally, create and fit a `DecisionTreeRegressor` and a `RandomForestRegressor` on the training data using the best value of `max_depth` found by the depth plots. Call `compare_regressors` to compare the two models.