Predicting house prices using k-nearest neighbors regression

In this notebook, you will implement k-nearest neighbors regression. You will:

- Find the k-nearest neighbors of a given query input
- Predict the output for the query input using the k-nearest neighbors
- Choose the best value of k using a validation set

If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided to you below for this quiz. This notebook contains the instructions, quiz questions and partially-completed code for you to use, as well as some cells to test your code.

What you need to download

If you are using GraphLab Create

- Download the King County House Sales data in SFrame format: kc_house_data_small.gl.zip. **Notice the _small postfix.**
- Download the companion IPython Notebook: week-6-local-regression-assignment-blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

If you are not using GraphLab Create

- Download the King County House Sales data csv file: kc_house_data_small.csv
- Download the King County House Sales training data csv file: kc_house_data_small_train.csv
- Download the King County House Sales testing data csv file: kc_house_data_small_test.csv
- Download the King County House Sales validation data csv file: kc_house_data_small_validation.csv
- **IMPORTANT: use the following types for columns when importing the csv files. Otherwise, they may not be imported correctly: [str, str, float, float, float, float, int, float, int, int, int, int, int, int, int, int, int, str, float, float, float, float]. If your tool of choice requires a dictionary of types for importing csv files (e.g. Pandas), use:**

```
dtype_dict = {'bathrooms':float, 'waterfront':int, 'sqft_above':int, 'sqft_living15':floa
t, 'grade':int, 'yr_renovated':int, 'price':float, 'bedrooms':float, 'zipcode':str, 'long
':float, 'sqft_lot15':float, 'sqft_living':float, 'floors':float, 'condition':int, 'lat':
float, 'date':str, 'sqft_basement':int, 'yr_built':int, 'id':str, 'sqft_lot':int, 'view':
int}
```

Useful resources

You may need to install the software tools or use the free Amazon EC2 machine. Instructions for both options are provided in the reading for Module 1 (Simple Regression).

If you are following the IPython Notebook and/or are new to numpy, then you might find the following tutorial helpful: numpy-tutorial.ipynb

If you are using GraphLab Create and the companion IPython Notebook

Open the companion IPython notebook and follow the instructions in the notebook.

If instead you are using other tools to do your homework

You are welcome to write your own code and use any other libraries, like Pandas or R, to help you in the process. If you would like to take this path, follow the instructions below.

1. If you're using SFrame, import GraphLab Create and load in the house data as follows:

```
sales = graphlab.SFrame('kc_house_data_small.gl/')
```

Split the data into training, test, and validation sets as follows:

```
(train_and_validation, test) = sales.random_split(.8, seed=1) (train, validation) = train
_and_validation.random_split(.8, seed=1)
```

If you are not using SFrame, load all three csv files listed in "What you need to download"..

2. If you're using Python: To do the matrix operations required to perform k nearest neighbors, we will be using the popular python library 'numpy' which is a computational library specialized for operations on arrays. For students unfamiliar with numpy we have created a numpy tutorial (see "Useful resources"). It is common to import numpy under the name 'np' for short. To do this, execute:

```
import numpy as np
```

3. To efficiently compute pairwise distances among data points, we will convert the SFrame (or dataframe) into a 2D Numpy array. First import the numpy library and then copy and paste get_numpy_data() (or equivalent). The function takes a dataset, a list of features (e.g. ['sqft_living', 'bedrooms']) to be used as inputs, and a name of the output (e.g. 'price'). It returns a 'features_matrix' (2D array) consisting of a column of ones followed by columns containing the values of the input features in the data set in the same order as the input list. It also returns an 'output_array', which is an array of the values of the output in the dataset (e.g. 'price').

e.g. in Python:

```
def get_numpy_data(data_sframe, features, output):     ...     return (feature_matrix, ou
tput_array)
```

4. Similarly, copy and paste the normalize_features function (or equivalent) from Module 5 (Ridge Regression). Given a feature matrix, each column is divided (element-wise) by its 2-norm. The function returns two items: (i) a feature matrix with normalized columns and (ii) the norms of the original columns.

e.g. in Python:

```
def normalize_features(features):     ...     return (normalized_features, norms)
```

5. Using get_numpy_data (or equivalent), extract numpy arrays of the training, test, and validation sets.

6. In computing distances, it is crucial to normalize features. Otherwise, for example, the 'sqft_living' feature (typically on the order of thousands) would exert a much larger influence on distance than the 'bedrooms' feature (typically on the order of ones). We divide each column of the training feature matrix by its 2-norm, so that the transformed column has unit norm.

IMPORTANT: Make sure to store the norms of the features in the training set. The features in the test and validation sets must be divided by these same norms, so that the training, test, and validation sets are normalized consistently.

e.g. in Python:

```
features_train, norms = normalize_features(features_train) features_test = features_test / norms features_valid = features_valid / norms
```

Compute a single distance

7. To start, let's just explore computing the "distance" between two given houses. We will take our query house to be the first house of the test set and look at the distance between this house and the 10th house of the training set.

To see the features associated with the query house, print the first row (index 0) of the test feature matrix. You should get an 18-dimensional vector whose components are between 0 and 1. Similarly, print the 10th row (index 9) of the training feature matrix.

e.g. in Python:

```
print features_test[0] print features_train[9]
```

8. ***Quiz Question: What is the Euclidean distance between the query house and the 10th house of the training set?***

Note: Do not use the 'np.linalg.norm' function; use 'np.sqrt', 'np.sum', and the power operator (**) instead. The latter approach is more easily adapted to computing multiple distances at once.

9. Of course, to do nearest neighbor regression, we need to compute the distance between our query house and*all* houses in the training set.

To visualize this nearest-neighbor search, let's first compute the distance from our query house (features_test[0]) to the first 10 houses of the training set (features_train[0:10]) and then search for the nearest neighbor within this small set of houses. Through restricting ourselves to a small set of houses to begin with, we can visually scan the list of 10 distances to verify that our code for finding the nearest neighbor is working.

Write a loop to compute the Euclidean distance from the query house to each of the first 10 houses in the training set.

10. ***Quiz Question: Among the first 10 training houses, which house is the closest to the query house?***

11. It is computationally inefficient to loop over computing distances to all houses in our training dataset. Fortunately, many of the numpy functions can be vectorized, applying the same operation over multiple values or vectors. We now walk through this process. (The material up to #13 is specific to numpy; if you are using other languages such as R or Matlab, consult relevant manuals on vectorization.)

Consider the following loop that computes the element-wise difference between the features of the query house (features_test[0]) and the first 3 training houses (features_train[0:3]):

```
for i in xrange(3):     print features_train[i]-features_test[0]     # should print 3 vec
tors of length 18
```

The subtraction operator (-) in numpy is vectorized as follows:

```
print features_train[0:3] - features_test[0]
```

Note that the output of this vectorized operation is identical to that of the loop above, which can be verified below:

```
# verify that vectorization works results = features_train[0:3] - features_test[0] print
results[0] - (features_train[0]-features_test[0]) # should print all 0's if results[0] ==
 (features_train[0]-features_test[0]) print results[1] - (features_train[1]-features_test
[0]) # should print all 0's if results[1] == (features_train[1]-features_test[0]) print r
esults[2] - (features_train[2]-features_test[0]) # should print all 0's if results[2] ==
(features_train[2]-features_test[0])
```

Aside: it is a good idea to write tests like this snippet whenever you are vectorizing a complicated operation.

Perform 1-nearest neighbor regression

12. Now that we have the element-wise differences, it is not too hard to compute the Euclidean distances between our query house and all of the training houses. First, write a single-line expression to define a variable ‘diff’ such that ‘diff[i]’ gives the element-wise difference between the features of the query house and the i-th training house.

To test your code, print diff[-1].sum(), which should be -0.0934339605842.

13. The next step in computing the Euclidean distances is to take these feature-by-feature differences in ‘diff’, square each, and take the sum over feature indices. That is, compute the sum of squared feature differences for each training house (row in ‘diff’).

By default, ‘np.sum’ sums up everything in the matrix and returns a single number. To instead sum only over a row or column, we need to specifiy the ‘axis’ parameter described in the np.sum documentation. In particular, ‘axis=1’ computes the sum across each row.

So

```
np.sum(diff**2, axis=1)
```

computes this sum of squared feature differences for all training houses. Verify that the two expressions

```
np.sum(diff**2, axis=1)[15]
```

and

```
np.sum(diff[15]**2)
```

yield the same results. That is, the output for the 16th house in the training set is equivalent to having examined only the 16th row of 'diff' and computing the sum of squares on that row alone.

14. With this result in mind, write a single-line expression to compute the Euclidean distances from the query to all the instances. Assign the result to variable distances.

Hint: don't forget to take the square root of the sum of squares.

Hint: distances[100] should contain 0.0237082324496.

15. Now you are ready to write a function that computes the distances from a query house to all training houses. The function should take two parameters: (i) the matrix of training features and (ii) the single feature vector associated with the query.

e.g. in Python:

```
def compute_distances(features_instances, features_query):     ...     return distances
```

16. ***Quiz Question: Take the query house to be third house of the test set (features_test[2]). What is the index of the house in the training set that is closest to this query house?***

17. ***Quiz Question: What is the predicted value of the query house based on 1-nearest neighbor regression?***

Perform k-nearest neighbor regression

18. Using the functions above, implement a function that takes in

- the value of k;
- the feature matrix for the instances; and
- the feature of the query

and returns the indices of the k closest training houses. For instance, with 2-nearest neighbor, a return value of [5, 10] would indicate that the 6th and 11th training houses are closest to the query house.

e.g. in Python:

```
def k_nearest_neighbors(k, feature_train, features_query):     ...     return neighbors
```

Hint: look at the documentation for np.argsort.

19. ***Quiz Question: Take the query house to be third house of the test set (features_test[2]). What are the indices of the 4 training houses closest to the query house?***

20. Now that we know how to find the k-nearest neighbors, write a function that predicts the value of a given query house. **For simplicity, take the average of the prices of the k nearest neighbors in the training set.** The function should have the following parameters:

- the value of k;
- the feature matrix for the instances;
- the output values (prices) of the instances; and
- the feature of the query, whose price we're predicting.

The function should return a predicted value of the query house.

e.g. in Python:

```
def predict_output_of_query(k, features_train, output_train, features_query):     ...
 return prediction
```

Hint: you can extract multiple items from a numpy array using a list of indices. For instance, output_train[[6,10]] returns the output values (prices) of the 7th and 11th instances.

21. ***Quiz Question: Again taking the query house to be third house of the test set (features_test[2]), predict the value of the query house using k-nearest neighbors with k=4 and the simple averaging method described and implemented above.***

22. Finally, write a function to predict the value of each and every house in a query set. (The query set can be any subset of the dataset, be it the test set or validation set.) The idea is to have a loop where we take each house in the query set as the query house and make a prediction for that specific house. The new function should take the following parameters:

- the value of k;
- the feature matrix for the training set;
- the output values (prices) of the training houses; and
- the feature matrix for the query set.

The function should return a set of predicted values, one for each house in the query set.

e.g. in Python:

```
def predict_output(k, features_train, output_train, features_query):     ...     return p
redictions
```

Hint: to get the number of houses in the test set, use the .shape field of the feature matrix. See the documentation.

23. ***Quiz Question: Make predictions for the first 10 houses in the test set, using k=10. What is the index of the house in this query set that has the lowest predicted value? What is the predicted value of this house?***

Choosing the best value of k using a validation set

24. There remains a question of choosing the value of k to use in making predictions. Here, we use a validation set to choose this value. Write a loop that does the following:

For k in [1, 2, … 15]:

- Make predictions for the VALIDATION data using the k-nearest neighbors from the TRAINING data.
- Compute the RSS on VALIDATION data

Report which k produced the lowest RSS on validation data.

25. ***Quiz Question: What is the RSS on the TEST data using the value of k found above? To be clear, sum over all houses in the TEST set.***