5.1 K-Nearest Neighbors for Regression

May 9, 2019

1 Chapter 5. Machine Learning and Regression Models

Prediction problems are ubiquitous in real world applications. For example:

- A real estate agent might want to predict the fair price of a home, using features of the home.
- A sports bettor might want to predict which team will win the game, using information about the teams.
- A historian might want to predict which historical figure wrote an anonymous document, using the words in the document.

In each case, we have two kinds of variables:

- **features** (a.k.a. **predictors**, **inputs**, **independent variables**), such as square footage and number of bedrooms, that are used to predict
- a **label** (a.k.a. **response**, **output**, **dependent variable**), such as house price.

We can formalize the problem mathematically as follows: let $\mathbf{x}$ be the features and y the label; a **predictive model** is a function f that maps $\mathbf{x}$ to y:

$$f : \mathbf{x} \mapsto y.$$

Now suppose we have a new house, with features $\mathbf{x}^*$. A predictive model f predicts the price of this house to be $f(\mathbf{x}^*)$.

How do we come up a predictive model f in the first place? One way is to learn it from existing data, or **training data**. For example, to build a model that predicts the price of a home from the square footage (`Gr Liv Area`), we would need training data like the points shown in black below.

We could then learn a model, f, from this training data. For example, one possible predictive model is the red curve shown in the plot. This model was chosen to fit the points in the training data as tightly as possible. If we wanted to predict the price of a 2700 square foot home using this model, we would simply evaluate $f(2700)$, which comes out to about \$300,000. The key thing to note is that f depends on the training data. If the training data changes, then so does f.

The process of learning predictive models from data is known as **machine learning**. There are many ways to learn a predictive model from data, including *linear regression* (which you may have seen in a statistics course), *decision trees*, and *neural networks*. In this chapter, we will focus on one machine learning algorithm called **k-nearest neighbors** that leverages the distance metrics that you learned about in Chapter 4.

Predictive models are divided into two types, depending on whether the label y is categorical or quantitative. If the label is quantitative, then the prediction problem is a **regression** problem, and the model is called a **regressor**. If the label is categorical, then the prediction problem is a **classification** problem, and the model is called a **classifier**. Chapter 5 covers regression models, while Chapter 6 covers classification models.

2 5.1 K-Nearest Neighbors for Regression

Regressors are predictive models that are employed when the label is quantitative. In this section, we will train a machine learning model that predicts the price of a house from its square footage and other features.

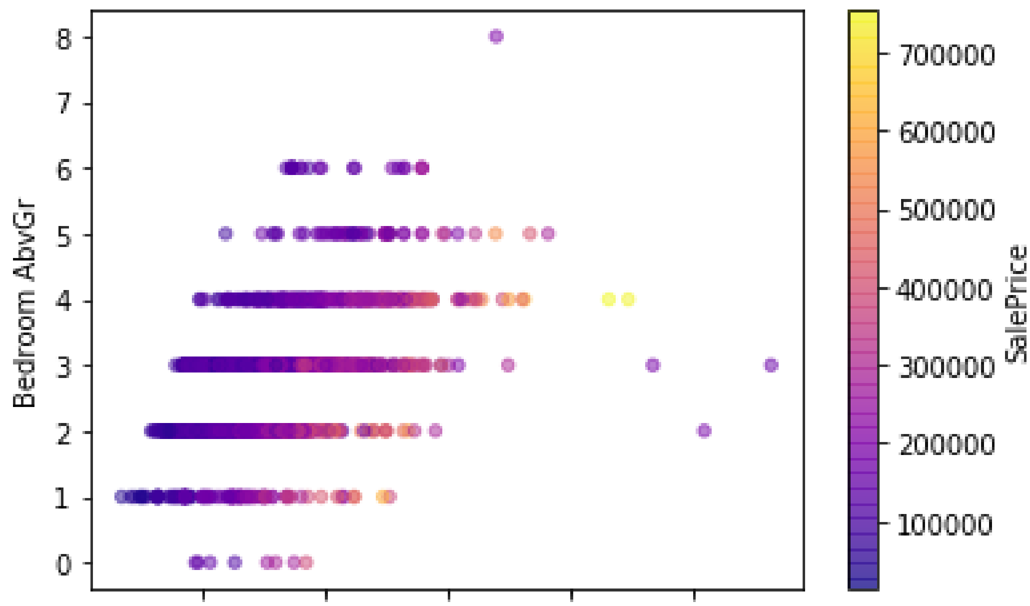
We will use the Ames housing data set as the training data. First, let's read in the data set.

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
pd.options.display.max_rows = 5

housing = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/maste
housing
```

Out[1]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0         1  526301100           20        RL         141.0     31770   Pave
1         2  526350040           20        RH          80.0     11622   Pave
...     ...        ...          ...       ...           ...       ...    ...
2928   2929  924100070           20        RL          77.0     10010   Pave
2929   2930  924151050           60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour    ...     Pool Area Pool QC  Fence  \
0      NaN       IR1          Lvl    ...             0     NaN    NaN
1      NaN       Reg          Lvl    ...             0     NaN  MnPrv
...    ...       ...          ...    ...           ...     ...    ...
2928   NaN       Reg          Lvl    ...             0     NaN    NaN
2929   NaN       Reg          Lvl    ...             0     NaN    NaN

     Misc Feature Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  \
0             NaN        0       5    2010       WD           Normal
1             NaN        0       6    2010       WD           Normal
...           ...      ...     ...     ...       ...             ...
2928          NaN        0       4    2006       WD           Normal
2929          NaN        0      11    2006       WD           Normal

      SalePrice
0        215000
1        105000
...         ...
2928     170000
2929     188000

[2930 rows x 82 columns]
```

Let's focus on just two features for now: square footage (of the dwelling) and the number of bedrooms. Let's plot the training data, using a color gradient to represent the labels. Notice how we can customize the color gradient using the `cmap=` argument. A list of the available colormaps can be found here.

```
In [2]: housing.plot.scatter(x="Gr Liv Area", y="Bedroom AbvGr",
                             c="SalePrice", cmap="plasma", alpha=.5);
```



Notice how points that are close on this plot tend to have similar house prices. This insight is the basis of the *k*-nearest neighbors algorithm for predicting house prices. Suppose that we want to predict the price of a 4000 square foot home with 3 bedrooms, represented by a black circle on the plot below.

We can find the *k* points that are closest to this point in feature space and average their prices as our prediction. For example, the 30-nearest neighbors in the training data to the new home are illustrated in the plot below. We would average the prices of these 30 homes to obtain the predicted price for the new home.

The *k*-nearest neighbors regression algorithm can be summarized as follows:

1. Determine the *k* closest points in the training data to the new point that you want to predict for, based on some distance metric on the features.
2. The predicted label of the new point is the mean (or median) of the labels of the *k* closest points.

Let's see how to implement this in code. First, we extract the training data and scale the features:

```
In [3]: X_train = housing[["Gr Liv Area", "Bedroom AbvGr"]]
        y_train = housing["SalePrice"]
```

```
X_train_mean = X_train.mean()
X_train_std = X_train.std()
X_train_sc = (X_train - X_train_mean) / X_train_std
```

Then, we create a Series for the new house, scaling it in exactly the same way:

```
In [4]: x_new = pd.Series()
        x_new["Gr Liv Area"] = 4000
        x_new["Bedroom AbvGr"] = 3

        x_new_sc = (x_new - X_train_mean) / X_train_std
        x_new_sc
```

```
Out[4]: Gr Liv Area      4.946124
        Bedroom AbvGr    0.176064
        dtype: float64
```

Now we calculate the (Euclidean) distances between this new house and each house in the training data. Then, we sort the distances.

```
In [5]: dists = np.sqrt(((X_train_sc - x_new_sc) ** 2).sum(axis=1))
        dists_sorted = dists.sort_values()
        dists_sorted
```

```
Out[5]: 1306    1.002950
        2181    1.337266
                  ...
        1302    7.507162
        1901    7.644028
        Length: 2930, dtype: float64
```

The first 30 entries of this sorted list are the 30 nearest neighbors. Let's get their indices.

```
In [6]: i_nearest = dists_sorted.index[:30]
        i_nearest
```

```
Out[6]: Int64Index([1306, 2181, 1767, 2445, 2666, 1760, 2450, 1537, 2045,   15,   65,
                    1772,  565, 2214,  422, 2336, 1182, 2570, 1945, 1764, 2500, 2453,
                    2329, 1320, 1497, 2180, 1572, 2737, 2218, 1022],
                   dtype='int64')
```

Now we can look up these indices in the original data.

```
In [7]: housing.loc[i_nearest]
```

```
Out[7]:       Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
        1306   1307  902207220           75        RM          87.0     18386   Pave
        2181   2182  908154205           60        RL         130.0     40094   Pave
        ...     ...        ...          ...       ...           ...       ...    ...
```

```
2218    2219   909281130             70         RL           70.0      10570    Pave
1022    1023   527325070             60         RL            NaN      12227    Pave

     Alley Lot Shape Land Contour    ...      Pool Area Pool QC Fence  \
1306   NaN       Reg          Lvl    ...              0     NaN   NaN
2181   NaN       IR1          Bnk    ...              0     NaN   NaN
...    ...       ...          ...    ...            ...     ...   ...
2218   NaN       Reg          Bnk    ...              0     NaN   NaN
1022   NaN       IR1          Lvl    ...              0     NaN   NaN

     Misc Feature Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  \
1306          NaN        0       5    2008       WD           Normal
2181          NaN        0      10    2007      New          Partial
...           ...      ...     ...     ...      ...              ...
2218          NaN        0      12    2007       WD           Normal
1022          NaN        0       7    2008       WD           Normal

      SalePrice
1306     295000
2181     184750
...         ...
2218     315000
1022     272000

[30 rows x 82 columns]
```

To make a prediction for the price of this new house, we average the sale prices of these 30 nearest neighbors.

```
In [8]: y_train.loc[i_nearest].mean()
```

```
Out[8]: 382429.23333333334
```

So the model predicts that the house is worth $382,429.

2.1 A More Complex Model

The model above only had two features so it was easy to visualize the "nearest neighbors" on the scatterplot. But the magic of k-nearest neighbors is that it still works when there are more features and the data isn't so easy to visualize.

Let's create a model that has 8 features, some of which are categorical.

```
In [9]: # Create a new variable
        housing["Date Sold"] = housing["Yr Sold"] + housing["Mo Sold"] / 12
        features = ["Lot Area", "Gr Liv Area",
                    "Full Bath", "Half Bath",
                    "Bedroom AbvGr",
                    "Year Built", "Date Sold",
                    "Neighborhood"]
```

```python
# Note that "Neighborhood" is a categorical variable.
X_train = pd.get_dummies(housing[features])
y_train = housing["SalePrice"]
```

Suppose an assessor is trying to predict the fair value in 2011 of a 1400-square foot home built in 1980 with 3 bedrooms, 2 full baths, and 1 half bath, on a 9000 square-foot lot in the `OldTown` neighborhood. Let's create the `pandas` `Series` corresponding to this house. Remember that we have dummy variables for each neighborhood. We have to be sure to include these dummy variables in the new `Series` as well. The easiest way to do this is to initialize the index of the `Series` to match the columns of `X_train` above.

```
In [10]: X_train.columns
```

```
Out[10]: Index(['Lot Area', 'Gr Liv Area', 'Full Bath', 'Half Bath', 'Bedroom AbvGr',
               'Year Built', 'Date Sold', 'Neighborhood_Blmngtn',
               'Neighborhood_Blueste', 'Neighborhood_BrDale', 'Neighborhood_BrkSide',
               'Neighborhood_ClearCr', 'Neighborhood_CollgCr', 'Neighborhood_Crawfor',
               'Neighborhood_Edwards', 'Neighborhood_Gilbert', 'Neighborhood_Greens',
               'Neighborhood_GrnHill', 'Neighborhood_IDOTRR', 'Neighborhood_Landmrk',
               'Neighborhood_MeadowV', 'Neighborhood_Mitchel', 'Neighborhood_NAmes',
               'Neighborhood_NPkVill', 'Neighborhood_NWAmes', 'Neighborhood_NoRidge',
               'Neighborhood_NridgHt', 'Neighborhood_OldTown', 'Neighborhood_SWISU',
               'Neighborhood_Sawyer', 'Neighborhood_SawyerW', 'Neighborhood_Somerst',
               'Neighborhood_StoneBr', 'Neighborhood_Timber', 'Neighborhood_Veenker'],
              dtype='object')
```

```
In [11]: # Initialize a Series of NaNs, indexed by the columns of X_train
         x_new = pd.Series(index=X_train.columns)

         # Set the values of the known variables.
         x_new["Lot Area"] = 9000
         x_new["Gr Liv Area"] = 1400
         x_new["Full Bath"] = 2
         x_new["Half Bath"] = 1
         x_new["Bedroom AbvGr"] = 3
         x_new["Year Built"] = 1980
         x_new["Date Sold"] = 2011

         # This house is in Old Town, so its dummy variable has value 1.
         x_new["Neighborhood_OldTown"] = 1
         # The dummy variables for the other neighborhoods all have value 0.
         x_new.fillna(0, inplace=True)

         x_new
```

```
Out[11]: Lot Area                 9000.0
         Gr Liv Area              1400.0
                                    ...
```

```
Neighborhood_Timber        0.0
Neighborhood_Veenker       0.0
Length: 35, dtype: float64
```

Now we can implement k-nearest neighbors much as we did above.

```
In [12]: # Standardize the variables.
         X_train_mean = X_train.mean()
         X_train_std = X_train.std()

         X_train_sc = (X_train - X_train_mean) / X_train_std
         x_new_sc = (x_new - X_train_mean) / X_train_std

         # Find index of 30 nearest neighbors.
         dists = np.sqrt(((X_train_sc - x_new_sc) ** 2).sum(axis=1))
         i_nearest = dists.sort_values()[:30].index

         # Average the labels of these 30 nearest neighbors
         y_train.loc[i_nearest].mean()
```

```
Out[12]: 132343.33333333334
```

So the model predicts that this house is worth $132,343.

2.2 The K-Nearest Neighbors Regression Function

Remember that a predictive model is a function $f : \mathbf{x} \mapsto y$. We can visualize f when $\mathbf{x}$ consists of a single feature, like square footage. We saw a hypothetical predictive model in Figure 5.1 above. What does f look like when the model is a k-nearest neighbors regressor?

First, we extract the training data. There is no need to scale the features in this case because there is only one feature. (The point of scaling is to bring all of the variables to the same scale.

```
In [13]: X_train = housing[["Gr Liv Area"]]
         y_train = housing["SalePrice"]
```

In order to plot f, we need to evaluate the predictive model at a grid of feature values. Since square footage varies from 0 to 6000 square feet in the training data, we create a grid of $\mathbf{x}$ values from 0 to 6000, in increments of 10.

```
In [14]: X_new = pd.DataFrame()
         X_new["Gr Liv Area"] = np.arange(0, 6000, 10)
         X_new
```

```
Out[14]:      Gr Liv Area
         0              0
         1             10
         ..           ...
         598         5980
         599         5990

         [600 rows x 1 columns]
```

Next, we will define a function `get_30NN_prediction` that implements the 30-nearest neighbor algorithm above: given a new observation, it returns the mean label of the 30-nearest neighbors to that observation.

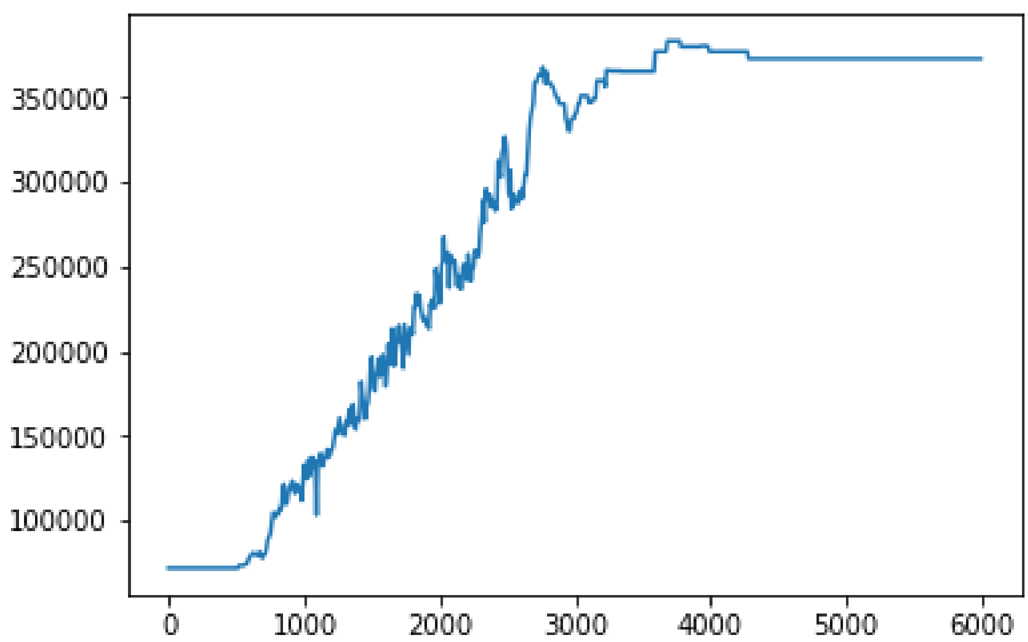
```
In [15]: def get_30NN_prediction(x_new):
             """Given new observation, returns 30-nearest neighbors prediction
             """
             dists = ((X_train - x_new) ** 2).sum(axis=1)
             inds_sorted = dists.sort_values().index[:30]
             return y_train.loc[inds_sorted].mean()
```

We actually have 600 new observations in `X_new`. Let's apply this function to each new observation.

```
In [16]: y_new_pred = X_new.apply(get_30NN_prediction, axis=1)
         y_new_pred

Out[16]: 0       72010.366667
         1       72010.366667
                     ...
         598    372458.466667
         599    372458.466667
         Length: 600, dtype: float64
```

We want to plot these predictions as a curve (`.plot.line()`). pandas will plot the index of the `Series` on the x-axis, so we have to set the index appropriately.

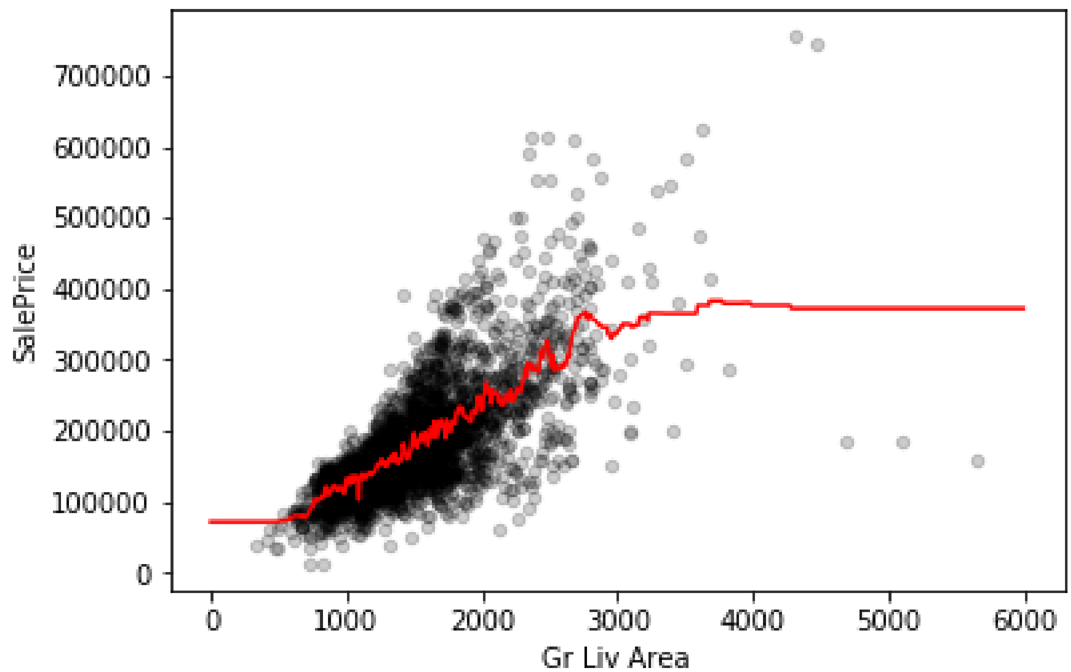
```
In [17]: y_new_pred.index = X_new
         y_new_pred.plot.line()

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa10fd32eb8>
```



Now let's put all the pieces together and overlay this regression function on top of a scatterplot of the training data.

```
In [18]: # Make a scatterplot of the training data
         housing.plot.scatter(x="Gr Liv Area", y="SalePrice", color="black", alpha=.2)

         # Add the predictions as a red line on this scatterplot
         y_new_pred.plot.line(color="red")
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa10ebec7b8>
```



Notice how rough the 30-nearest neighbors regression function looks. In particular, look at the right half of the graph where the training data is sparse. The regression function is a step function in this range. That is because the value of the prediction changes only when the identities of the 30-nearest neighbors change. Houses with a square footage between 4500 and 6000 all have the same 30 nearest neighbors in the training data, so the prediction is constant in that range.

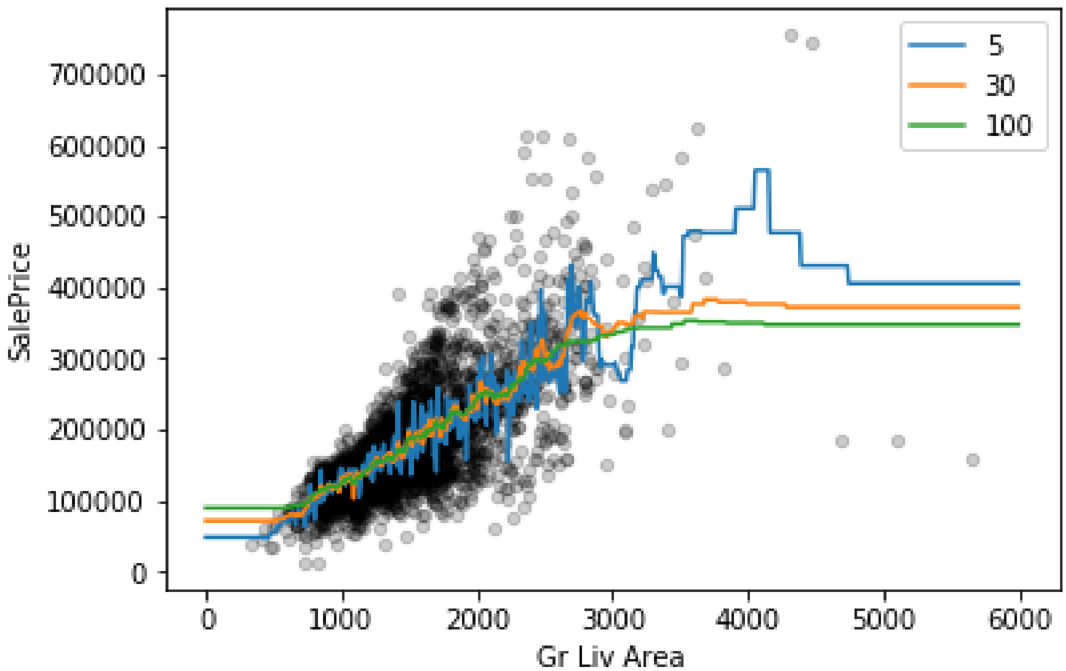
3 Exercises

Exercise 1. Plot the k-nearest neighbors regression function for predicting sale price from just its square footage for $k = 5, 30, 100$. How does the regression function change as k increases?

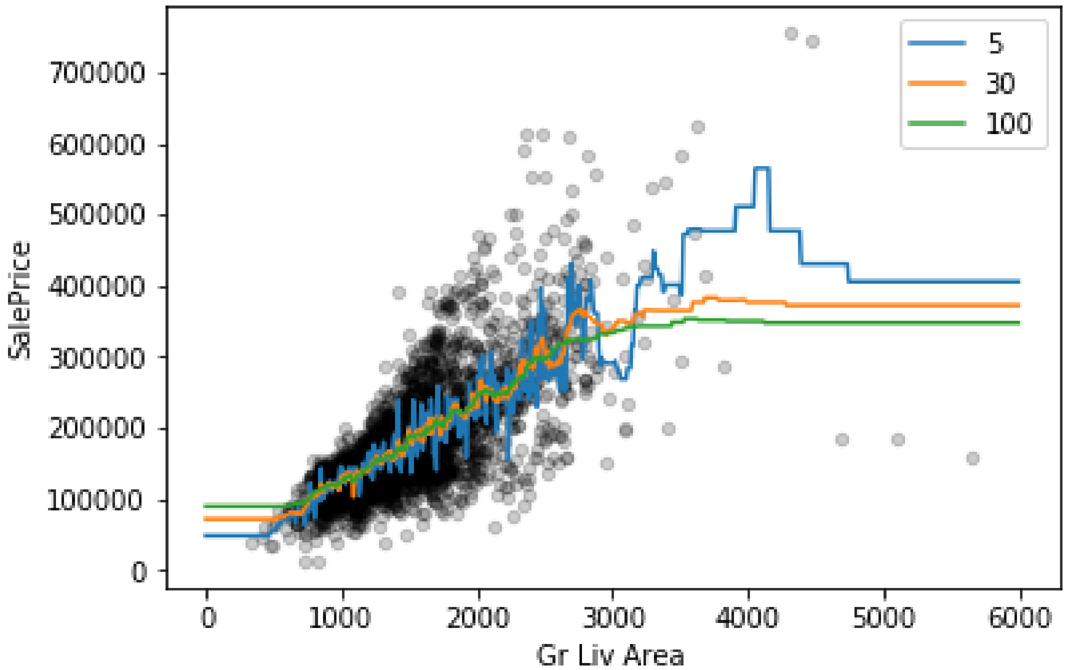
```
In [19]: X_train = housing[["Gr Liv Area"]]
         y_train = housing["SalePrice"]
```

```python
X_new = pd.DataFrame()
X_new["Gr Liv Area"] = np.arange(0, 6000, 10)

def get_5NN_prediction(x_new):
    """Given new observation, returns 30-nearest neighbors prediction
    """
    dists = ((X_train - x_new) ** 2).sum(axis=1)
    inds_sorted = dists.sort_values().index[:5]
    return y_train.loc[inds_sorted].mean()

def get_30NN_prediction(x_new):
    """Given new observation, returns 30-nearest neighbors prediction
    """
    dists = ((X_train - x_new) ** 2).sum(axis=1)
    inds_sorted = dists.sort_values().index[:30]
    return y_train.loc[inds_sorted].mean()

def get_100NN_prediction(x_new):
    """Given new observation, returns 30-nearest neighbors prediction
    """
    dists = ((X_train - x_new) ** 2).sum(axis=1)
    inds_sorted = dists.sort_values().index[:100]
    return y_train.loc[inds_sorted].mean()


y_new_pred_5 = X_new.apply(get_5NN_prediction, axis=1)
y_new_pred_30 = X_new.apply(get_30NN_prediction, axis=1)
y_new_pred_100 = X_new.apply(get_100NN_prediction, axis=1)

y_new_pred_5.index = X_new
y_new_pred_30.index = X_new
y_new_pred_100.index = X_new

housing.plot.scatter(x="Gr Liv Area", y="SalePrice", color="black", alpha=.2)
y_new_pred_5.plot.line(label="5", legend=True)
y_new_pred_30.plot.line(label="30", legend=True)
y_new_pred_100.plot.line(label="100", legend=True)
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa111e79198>
```




```
In [20]: X_train = housing[["Gr Liv Area"]]
         y_train = housing["SalePrice"]

         X_new = pd.DataFrame()
         X_new["Gr Liv Area"] = np.arange(0, 6000, 10)

         def kNN_prediction_factory(k):
             def get_prediction(x_new):
                 dists = ((X_train - x_new) ** 2).sum(axis=1)
                 inds_sorted = dists.sort_values().index[:k]
                 return y_train.loc[inds_sorted].mean()
             return get_prediction

         housing.plot.scatter(x="Gr Liv Area", y="SalePrice", color="black", alpha=.2)
         for k in [5, 30, 100]:
             y_new_pred = X_new.apply(kNN_prediction_factory(k), axis=1)
             y_new_pred.index = X_new
             y_new_pred.plot.line(label=k, legend=True)
```



Exercise 2. You would like to predict how much a male diner will tip on a bill of \$40.00 on a Sunday. Build a *k*-nearest neighbors model to answer this question, using the Tips dataset (`https://raw.githubusercontent.com/dlsun/data-science-book/master/data/tips.csv`) as your training data.

```
In [31]: tips = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/master/c
         tips.head()
```

```
Out[31]:    total_bill   tip     sex smoker  day    time  size
         0       16.99  1.01  Female     No  Sun  Dinner     2
         1       10.34  1.66    Male     No  Sun  Dinner     3
         2       21.01  3.50    Male     No  Sun  Dinner     3
         3       23.68  3.31    Male     No  Sun  Dinner     2
         4       24.59  3.61  Female     No  Sun  Dinner     4
```

```
In [41]: features = ["total_bill", "sex", "day"]
         X_train = pd.get_dummies(tips[features])
         y_train = tips["tip"]
         X_train.columns
```

```
Out[41]: Index(['total_bill', 'sex_Female', 'sex_Male', 'day_Fri', 'day_Sat', 'day_Sun',
                'day_Thur'],
               dtype='object')
```

```
In [34]: # Initialize a Series of NaNs, indexed by the columns of X_train
         x_new = pd.Series(index=X_train.columns)
```

```
# Set the values of the known variables.
x_new["total_bill"] = "40.00"
x_new["sex_Male"] = 1
x_new["day_Sun"] = 1

# This house is in Old Town, so its dummy variable has value 1.

# The dummy variables for the other neighborhoods all have value 0.
x_new.fillna(0, inplace=True)

x_new
```

```
Out[34]: total_bill    40.0
         sex_Female     0.0
                        ...
         day_Sun        1.0
         day_Thur       0.0
         Length: 7, dtype: float64
```

```
In [35]: # Standardize the variables.
         X_train_mean = X_train.mean()
         X_train_std = X_train.std()

         X_train_sc = (X_train - X_train_mean) / X_train_std
         x_new_sc = (x_new - X_train_mean) / X_train_std

         # Find index of 30 nearest neighbors.
         dists = np.sqrt(((X_train_sc - x_new_sc) ** 2).sum(axis=1))
         i_nearest = dists.sort_values()[:30].index

         # Average the labels of these 30 nearest neighbors
         y_train.loc[i_nearest].mean()
```

```
Out[35]: 3.9020000000000001
```

Challenge Exercise. We visualized the k-nearest neighbors regression function above, in the special case where there is only one feature. It is also possible to visualize a regression function in the case where there are two features, using a heat map, where the two axes represent the two features and the color represents the label.

Make a heat map that shows the 30-nearest neighbors regression function when there are two features in the model: square footage (`Gr Liv Area`) and number of bedrooms (`Bedroom AbvGr`).

```
In [ ]: # TYPE YOUR CODE HERE
```