k Nearest Neighbors

Justin Post

Recap

- MLR, Penalized MLR, & Regression Trees
  - Commonly used model with a numeric response
- Logistic Regression, Penalized Logistic Regression, & Classification Trees
  - Commonly used model with a binary response
- MLR & Logistic regression are more structured (linear)
- Trees easier to read but more variable (non-linear)
- Ensemble trees can greatly improve predictions in some cases (but you lose interpretability)

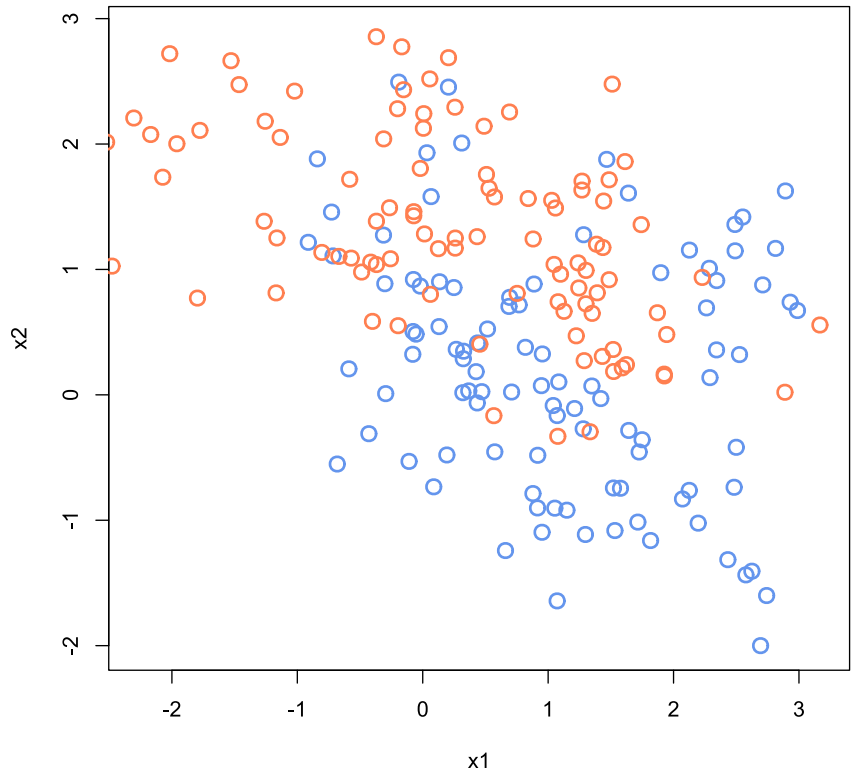
Recap

- MLR, Penalized MLR, & Regression Trees
  - Commonly used model with a numeric response
- Logistic Regression, Penalized Logistic Regression, & Classification Trees
  - Commonly used model with a binary response
- MLR & Logistic regression are more structured (linear)
- Trees easier to read but more variable (non-linear)
- Ensemble trees can greatly improve predictions in some cases (but you lose interpretability)

Now: k Nearest Neighbors (kNN) - another non-linear method for prediction/classification

kNN (Classification)

Suppose you have two **numeric** predictors and a categorical response (red or blue)



kNN (Classification)

Want to predict class membership (red or blue) based on (x1, x2) combination

kNN algorithm:

- Use "closest" k observations from training set to predict class
- Often Euclidean distance used: $ob_1 = (x_{11}, x_{21})$, $ob_2 = (x_{12}, x_{22})$ then $d(ob_1, ob_2) = \sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2)}$

kNN (Classification)

Want to predict class membership (red or blue) based on (x1, x2) combination

kNN algorithm:

- Use "closest" k observations from training set to predict class
- Often Euclidean distance used: $ob_1 = (x_{11}, x_{21})$, $ob_2 = (x_{12}, x_{22})$ then $d(ob_1, ob_2) = \sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2)}$
- Find estimates:

$$P(red|x1, x2) = \text{proportion of k closest values that are red}$$

$$P(blue|x1, x2) = \text{proportion of k closest values that are blue}$$

kNN (Classification)

Want to predict class membership (red or blue) based on (x1, x2) combination

kNN algorithm:

- Use "closest" k observations from training set to predict class
- Often Euclidean distance used: $ob_1 = (x_{11}, x_{21})$, $ob_2 = (x_{12}, x_{22})$ then $d(ob_1, ob_2) = \sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2)}$
- Find estimates:

$$P(red|x1, x2) = \text{proportion of k closest values that are red}$$

$$P(blue|x1, x2) = \text{proportion of k closest values that are blue}$$

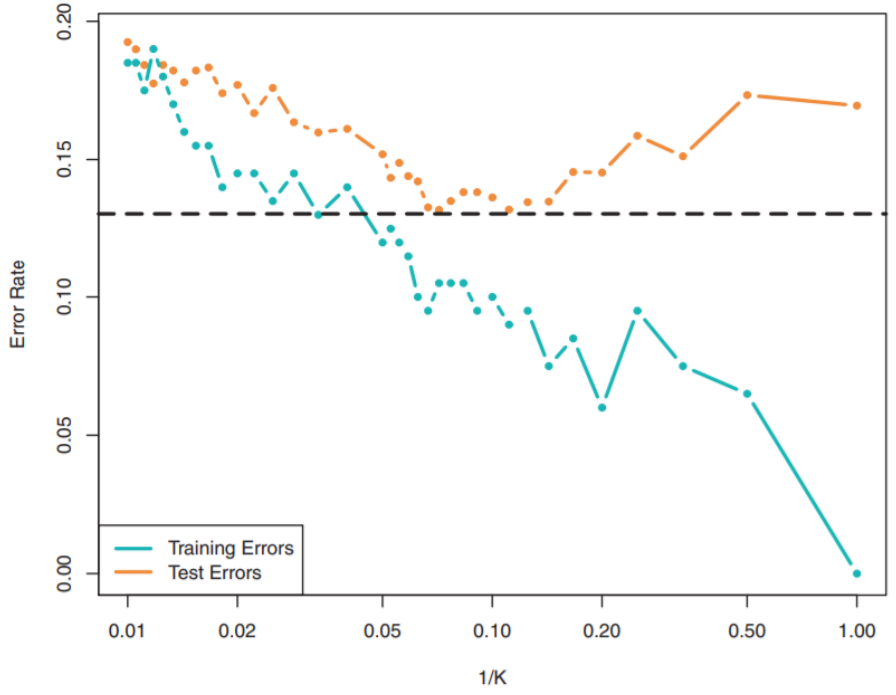
- Classify (predict) to class with highest probability
- App here: https://shiny.stat.ncsu.edu/jbpost2/knn/

kNN k value

- Small k implies flexible (possibly overfit, higher variance)
  - Training error will be small, may not extend to testing error
- Large k implies more rigid (possibly underfit, lower variance)

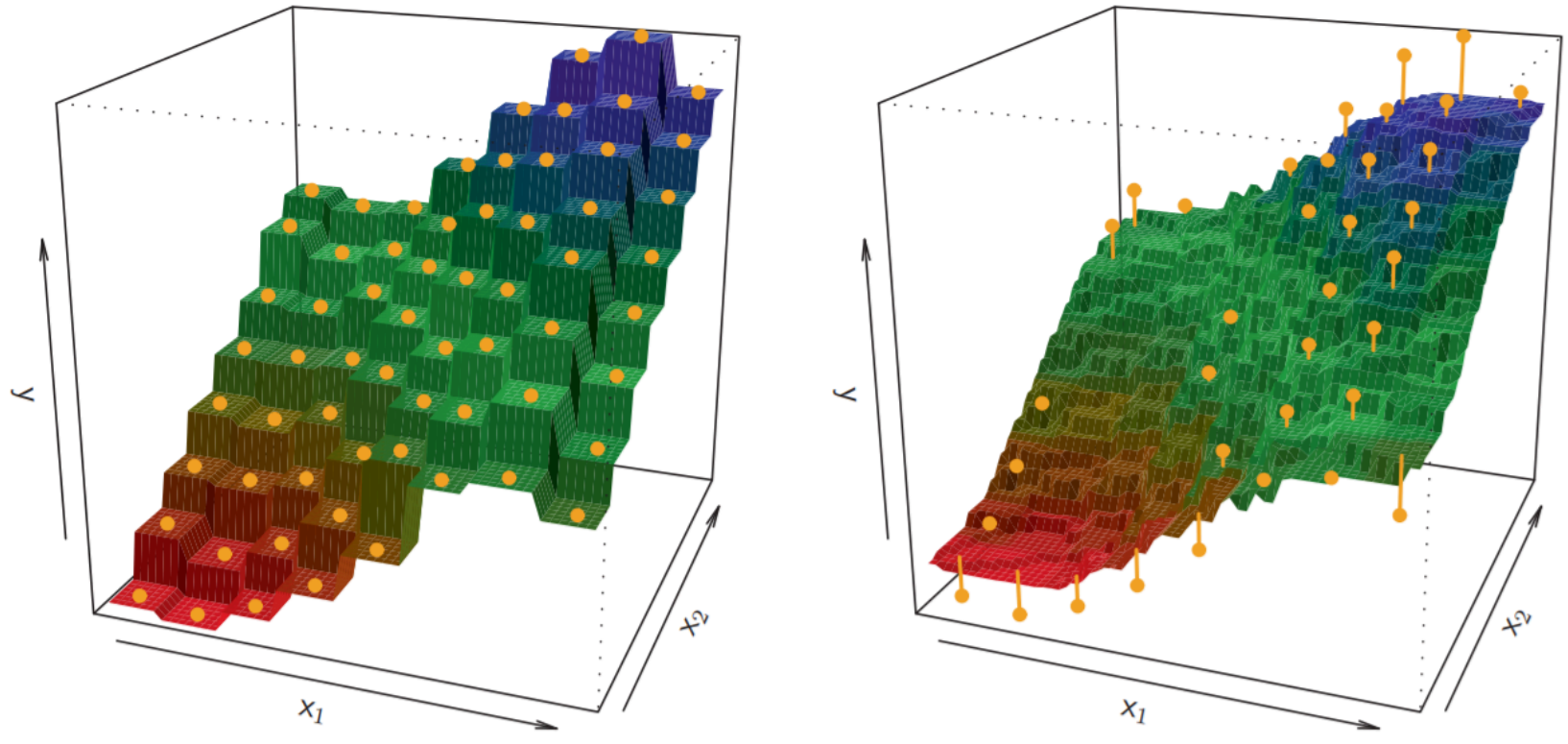
kNN k value

- Small k implies flexible (possibly overfit, higher variance)
  - Training error will be small, may not extend to testing error
- Large k implies more rigid (possibly underfit, lower variance)



kNN for Regression

- Same idea!
  - Use average of responses of "closest" k observations in training set as prediction
  - Closest again often Euclidean distance
- Note: Should usually standardize predictors (center/scale) any time you use 'distance' as scale becomes important



From: Introduction to Statistical Learning
k = 1 on the left, k = 9 on the right

More than Two Predictors

- Must all be numeric unless you develop or use a 'distance' measure that is appropriate for categorical data

More than Two Predictors

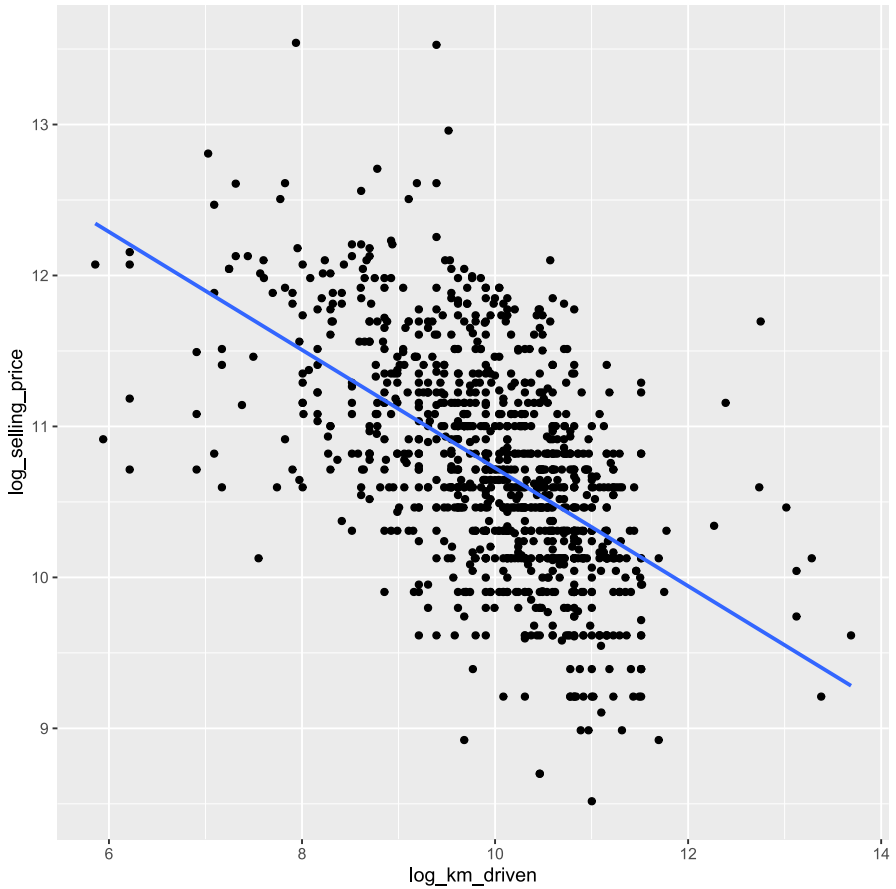
- Must all be numeric unless you develop or use a 'distance' measure that is appropriate for categorical data
- For all numeric data, Euclidean distance extends easily and is the default!

$$ob_1 = (x_{11}, x_{21}, \ldots, x_{p1}), ob_2 = (x_{12}, x_{22}, \ldots, x_{p2})$$

$$D(ob_1, ob_2) = \sqrt{\sum_{i=1}^{p} (x_{i1} - x_{i2})^2}$$

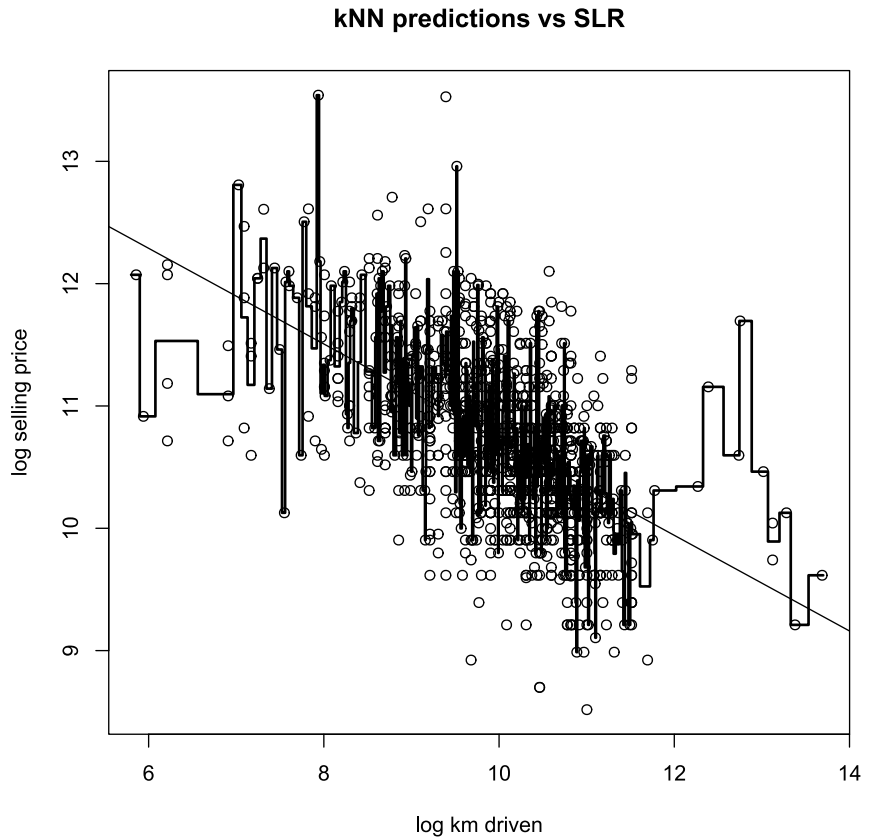
Visualize Fit vs SLR

- Consider `bike_data` we've used and `ex_showroom_price` as a predictor of `selling_price`



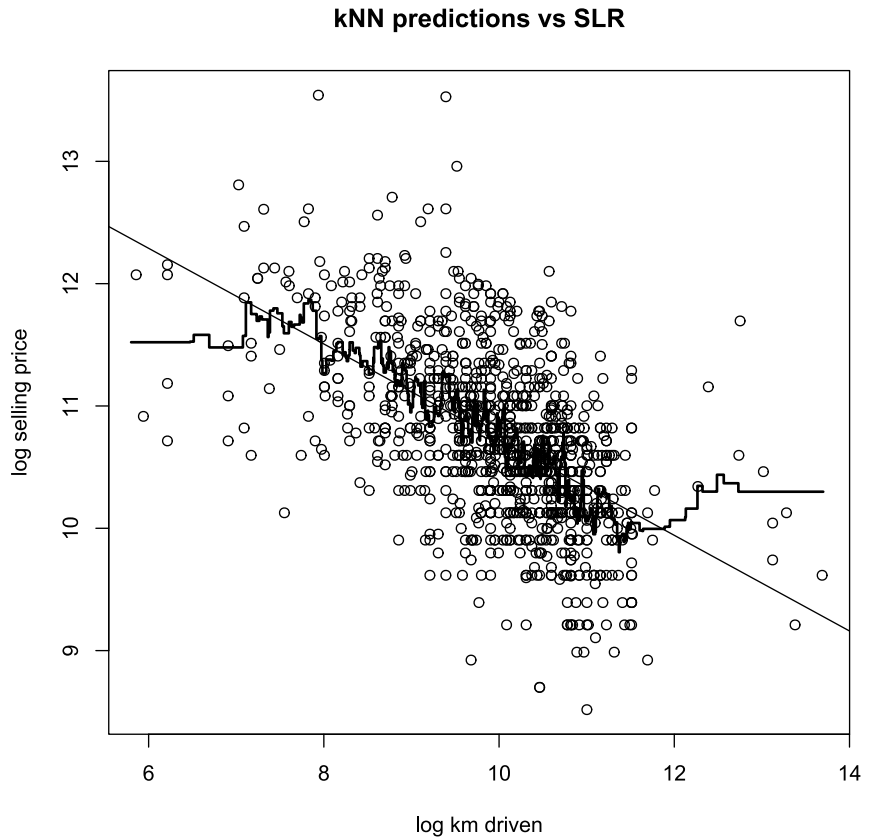
Visualize Fit vs SLR

- SLR vs kNN with $k = 1$



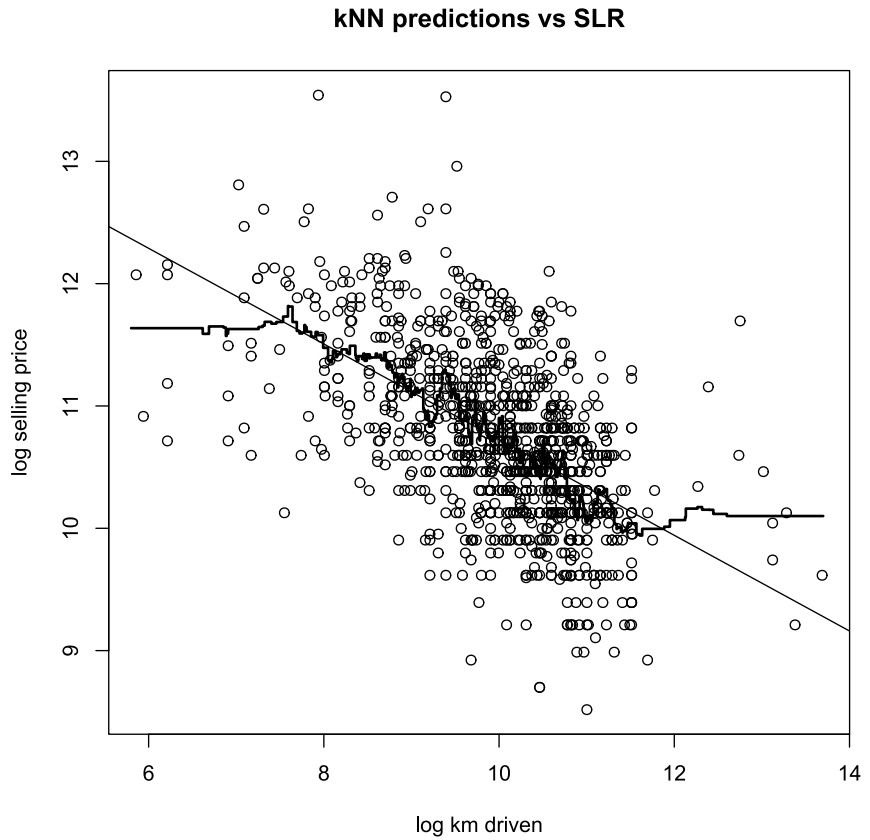
Visualize Fit vs SLR

- SLR vs kNN with $k = 10$



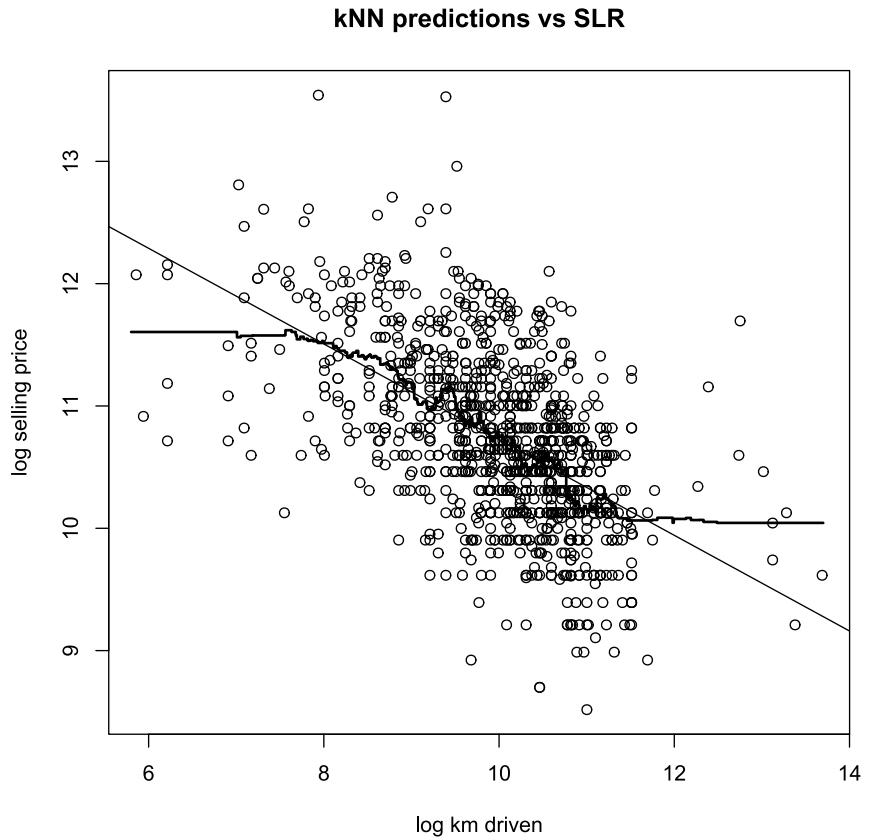
Visualize Fit vs SLR

- SLR vs kNN with $k = 20$



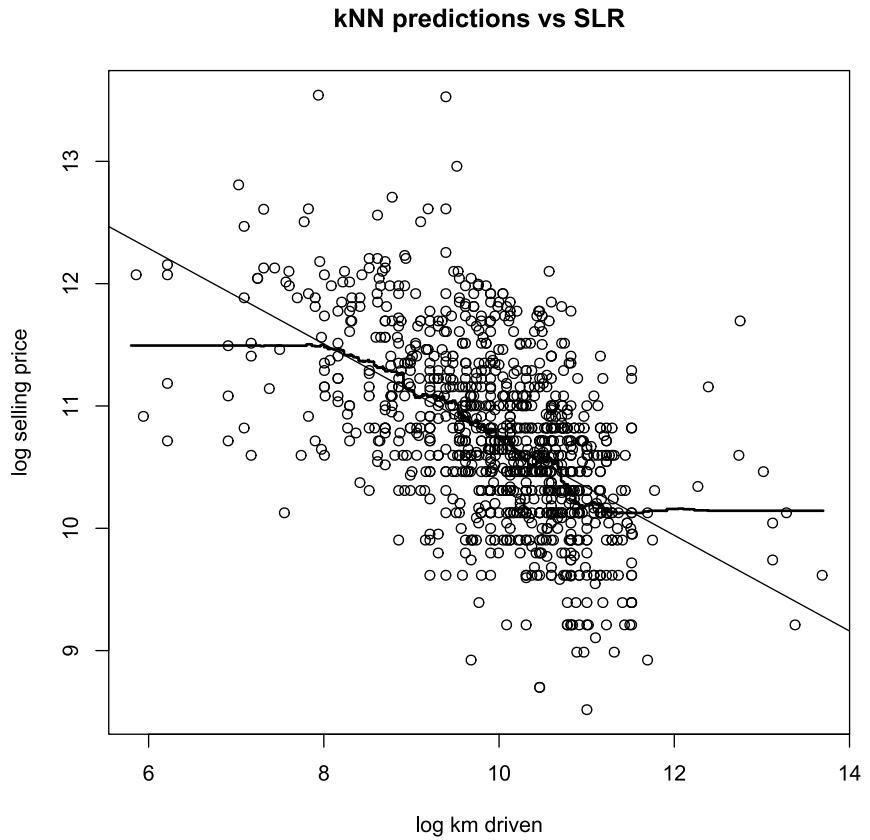
Visualize Fit vs SLR

- SLR vs kNN with $k = 50$



Visualize Fit vs SLR

- SLR vs kNN with $k = 100$



Fitting kNN with `sklearn`

- Same process as other models
  - Create an instance of the model
  - Use the `.fit()` method
  - Predict with `.predict()`
- Of course we likely want to use CV
  - Use `GridSearchCV()`

Fitting kNN with `sklearn`

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
bike_data = pd.read_csv("data/bikeDetails.csv")
#create response and new predictor
bike_data['log_selling_price'] = np.log(bike_data['selling_price'])
bike_data['log_km_driven'] = np.log(bike_data['km_driven'])
```

Fitting kNN with `sklearn`

- Fit the model with $k = 3$

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors = 3)
neigh.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

Fitting kNN with `sklearn`

- Fit the model with $k = 3$

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors = 3)
neigh.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

- Compare predictions with the Bagged tree model

```
from sklearn.ensemble import RandomForestRegressor
bag_tree = RandomForestRegressor(max_features = None, n_estimators = 500)
bag_tree.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

Fitting kNN with `sklearn`

- Fit the model with $k = 3$

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors = 3)
neigh.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

- Compare predictions with the Bagged tree model

```
from sklearn.ensemble import RandomForestRegressor
bag_tree = RandomForestRegressor(max_features = None, n_estimators = 500)
bag_tree.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

```
##    log_km_driven  year  bagged_preds     kNN_preds
## 0            9.5  1990  41952.680319  24986.659549
## 1            9.5  2015  46689.655309  73986.362230
## 2           10.6  1990  16216.796741  24986.659549
## 3           10.6  2015  34235.698665  34552.116150
```

`GridSearchCV`

- No 'built-in' CV function
- Use `GridSearchCV()`

GridSearchCV

- No 'built-in' CV function
- Use `GridSearchCV()`

```
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 100))
param_grid = dict(n_neighbors=k_range)
  # defining parameter range
grid = GridSearchCV(KNeighborsRegressor(),
                    param_grid,
                    cv=5,
                    scoring='neg_root_mean_squared_error')
```

GridSearchCV

- No 'built-in' CV function
- Use `GridSearchCV()`

```
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 100))
param_grid = dict(n_neighbors=k_range)
  # defining parameter range
grid = GridSearchCV(KNeighborsRegressor(),
                    param_grid,
                    cv=5,
                    scoring='neg_root_mean_squared_error')
```

```
grid.fit(bike_data[['log_km_driven', 'year']],
             bike_data['log_selling_price'])
```

```
print(grid.best_params_)
```

```
## {'n_neighbors': 49}
```

Recap

- kNN uses k closest observations from the training set for prediction
- Very flexible to not flexible!
- Can be used for both regression and classification problems
  - `KNeighborsRegressor()` or `KNeighborsClassifier()`
- CV easily done with `GridSearchCV()`