LAB #2: MODEL COMPARISON - KNN VS LINEAR REGRESSION

CS 109A, STAT 121A, AC 209A: Data Science

Fall 2016

Harvard University

- We test the code we release to you (in lab and solutions) are tested thoroughly and in multiple environments, but it doesn't mean it's always 100% bug free. If you spot a bug in the code, report it to the help line!

- If you're using our code and experience error. Try to debug it, make sure you're using solution code correctly (i.e. identifiers match, parameters are passed correctly). But if you believe our code, even correctly integrated, cannot run on your system, report it to the help line!

- If you spot typos in slides, what not, report it to the help line!

Reporting bugs in our material doesn't hurt our feelings, it does a service to the community (the whole class).

HOMEWORK GRADING STANDARDS

As Pavlos announced in class:

- **Correctness:** did you perform all tasks in the problem spec correctly? (If a cell was not executed or executed with error, we will not consider it.)
- **Interpretation:** did you interpret your results logically and drew conclusions that are soundly supported by your analyses?
- **Presentation:**
    - Does your code follow Python programming style guidelines?
    - Does your notebook read like a report - text well formatted, good usage of scientific language, data well organized and easy to reference?
    - Is your document well organized - related plots plotted as subplots of a single figure for easy comparison, all axis, subplots, color coding carefully and consistently labeled?
- **Code Design:** (starting with HW3)
    - Are you practicing functional abstraction (if you find yourself copying and pasting similar blocks of code, you need to write a function that handles the task generally)?
    - Are your algorithms efficient (is a quadruply nested `for` loop really the best way to solve your problem)?
    - Does your solution scale (are you spending 2 days computing with a dataset of 100 points)?

After two weeks, we trust in your ability to absorb/adapt our code and to build your own analysis so...

- The lab solutions are no longer in submission ready form (they will be correct and complete, but won't be the most efficient or polished). You need to tweak and adjust them in minor ways.
- We'll build outlines of analyses together as a class during discussion, we will not provide the bullet points for you to fill out.
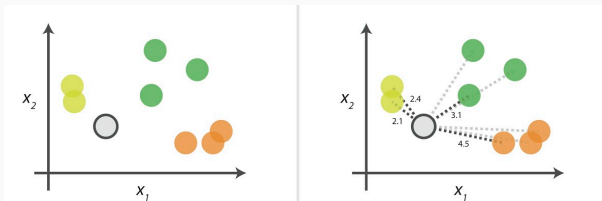
Again, **we trust you** so trust your self and start building up your independence!

REVIEW

Given training data with predictor values X^{train}, response values Y^{train} and testing data with predictor values X^{test} and response values Y^{test}:

- For the i-th **testing** data point with predictor X_i^{test}, find k data points in the **training** set, whose predictors are the most similar (in Euclidean distance) to X_i^{test}.
- Average the values of the response variable of these k points in the training set.
- Set $\hat{Y}_i^{test}$ equal to this average.

Does this remind you of anything from HW1?

INTRODUCTION TO sklearn

The short answer is (almost) everything you need for machine learning/data mining/data analysis, **for this class**.

- Preprocessing data
- Fancy models for classification and regression
- Tools for tuning parameters in models

Question: Why did we ask you to implement KNN (for classification) from scratch?

- We are trying to transition you from user to maker
- You don't truly understand a model/algorithm until you can build it yourself
- You have to understand the details of implementation in order to debug or talk about performance

```python
#import KNN regressor model
from sklearn.neighbors import KNeighborsRegressor as KNN

#creates KNN model for k-nearest neighbor comparison
neighbours = KNN(n_neighbors=k)

#fit() sets the parameters of the model (with training data)
neighbours.fit(x_train, y_train)

#predict test response values given test predictor
predicted_y = neighbours.predict(x_test)

#calculates R^2 of predicted response vs test response
r = neighbours.score(x_test, y_test)
```

Note: x_test, y_test, x_train, y_train are all array(-like) objects. x_test, x_train have shape (n, m), where *n* is the number of points and *m* is the number of features.

LINEAR REGRESSION: sklearn STYLE

```python
#import linear model
from sklearn.linear_model import LinearRegression as Lin_Reg

#create linear model
regression = Lin_Reg()

#fit linear model
regression.fit(x_train y_train)

#predict y-values
predicted_y = regression.predict(x_test)

#score predictions
r = regression.score(x_test, y_test)
```

Note: x_test, y_test, x_train, y_train are all array(-like) objects. x_test, x_train have shape (n, m), where *n* is the number of points and *m* is the number of features.

What are the type of objects that `sklearn` models work with?

We need to turn data tables into arrays and matrices.

	Predictors		Response
	X_1	X_2	Y
obs. 1	3	4	3
obs. 2	2	1	0
obs. 3	5	2	8

$$\longrightarrow X = \begin{matrix} X_1 & X_2 \\ \begin{pmatrix} 3 & 4 \\ 2 & 1 \\ 5 & 2 \end{pmatrix} \end{matrix}, \; Y = \begin{pmatrix} 3 \\ 0 \\ 8 \end{pmatrix}$$

Note that X is a two dimensional array with shape (3, 2) and Y is a one dimensional array with shape (3,) (or (3, 1) as a trivial two dimensional array).

11

What are the type of objects that `sklearn` models work with?

We need to turn models (functional forms) into arrays and matrices.

$$\underbrace{\hat{Y} = f(X_1, X_2) = 2X_1 - X_2}_{Model} \longrightarrow \begin{array}{c} \\ \text{Coef. for } X_1 \\ \text{Coef. for } X_1 \end{array} \overset{M}{\begin{pmatrix} 2 \\ -1 \end{pmatrix}}$$

Note that the model *M* is a one dimensional array with shape (2,) (or (2, 1) as a trivial two dimensional array).

What are the type of objects that `sklearn` models work with?

We need to phrase the prediction task - applying $f(X_1, X_2) = 2X_2 - X_2$ to the data - in terms of matrix operations.

$$
\begin{array}{ccc}
X_1 & X_2 & \\
\end{array}
$$

$$
\begin{pmatrix} 3 & 4 \\ 2 & 1 \\ 5 & 2 \end{pmatrix} * \overset{M}{\begin{pmatrix} 2 \\ -1 \end{pmatrix}} = \overset{\hat{Y}}{\begin{pmatrix} 2*3 & + & -1*4 \\ 2*2 & + & -1*1 \\ 2*5 & + & -1*2 \end{pmatrix}}
$$

Next week, you'll see that turning data and model in to matrices, prediction into matrix multiplication will be very useful to us!

How do we use models to fill in missing data?

How do we use models to fill in missing data?

How do we use models to fill in missing data?   Using **KNN** for $k = 2$?

How do we use models to fill in missing data?   Using **KNN** for $k = 2$?

X	Y
🔴	**1**
🔴	**? = (1 + 0.5) / 2**
🟠	**0.5**
🟠	**0.1**
🟡	**? = (0.1 + 10) / 2**
🟢	**10**
🟢	**0.03**

How do we use models to fill in missing data?

Using **linear regression**?

where *m* and *b* are computing from the dataset excluding the rows with missing values.