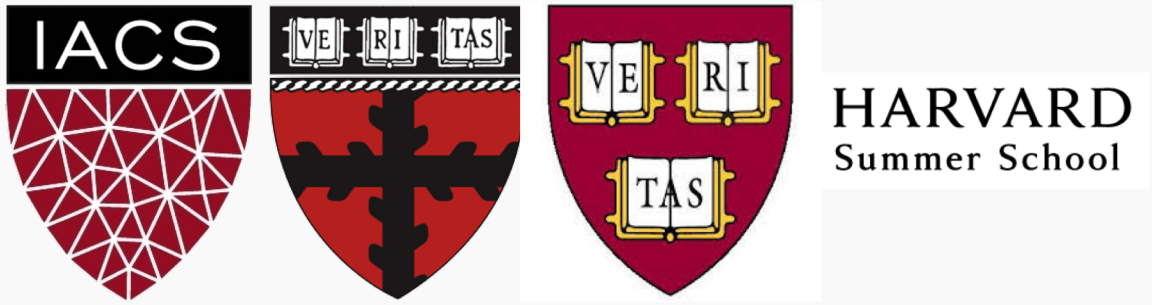
Lecture #7: *k*-NN Classification, PCA, and Imputation

CS-S109A: Introduction to Data Science

Kevin Rader



ANNOUNCEMENTS

- **HW3** is due tomorrow (Tues, July 14) at 11:59pm
  - **Q5.6** asks about comparing regularization effects on 'temp' and 'atemp' but this is a mistake as it was dropped earlier in the HW."
  - **HW Submission** (in general): please (1) don't submit zip files (2) rerun you notebook (3) and make sure your answers are prominent (i.e., not hidden in a comment)
- Go by the **course schedule** on Ed to avoid confusion

Lecture Outline

- *k*-NN review
- *k*-NN for Classification
- Use of Interaction Terms
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- Dealing with Missingness
  - Types of Missingness
  - Imputation Methods

k-Nearest Neighbors

We've already seen the *k*-NN method for predicting a quantitative response (it was the very first method we introduced). How was *k*-NN implemented in the Regression setting (quantitative response)?

The approach was simple: to predict an observation's response, use the **other** available observations that are most similar to it.

For a specified value of *k*, each observation's outcome is predicted to be the **average** of the *k*-closest observations as measured by some distance of the predictor(s).

With one predictor, the method was easily implemented.

Review: Choice of k

How well the predictions perform is related to the choice of *k*.

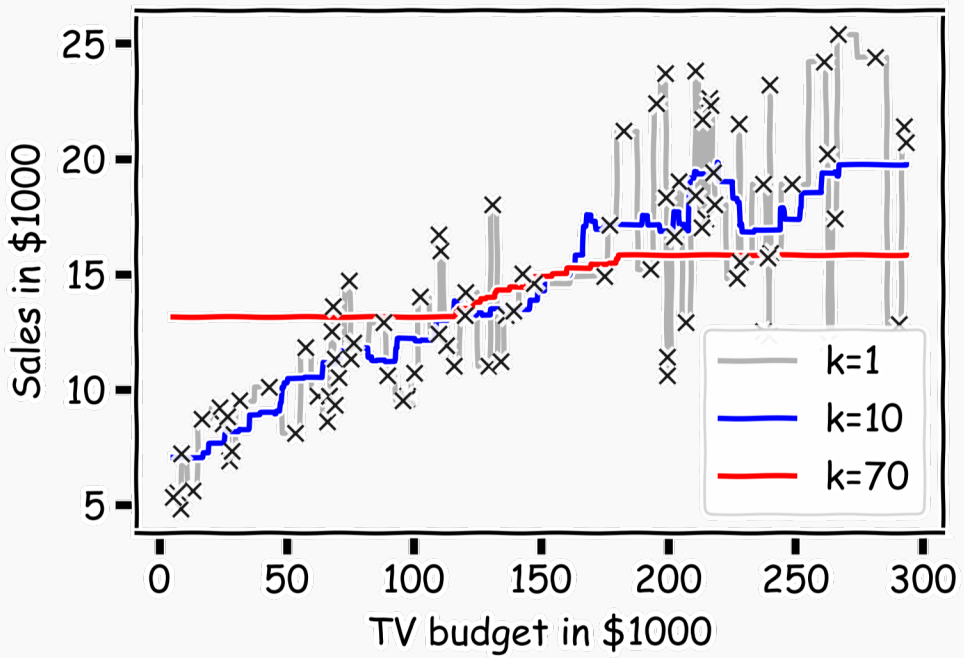
What will the predictions look like if *k* is very small? What if it is very large?

More specifically, what will the predictions be for new observations if $k = n$?

$$\bar{Y}$$

A picture is worth a thousand words...

Choice of *k* matters



k-NN for Classification

k-NN for Classification

How can we modify the *k*-NN approach for classification?

The approach here is the same as for *k*-NN regression: use the other available observations that are most similar to the observation we are trying to predict (classify into a group) based on the predictors at hand.

How do we classify which category a specific observation should be in based on its nearest neighbors?

The category that shows up the most among the nearest neighbors.

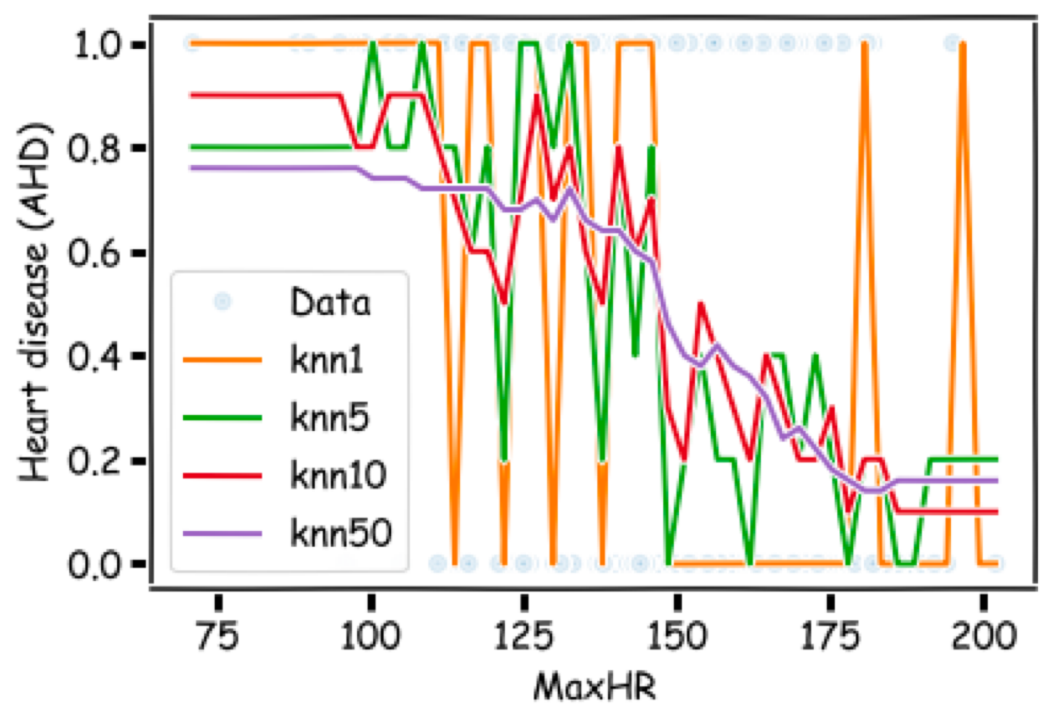
k-NN for Classification: formal definition

The *k*-NN classifier first identifies the *k* points in the training data that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates the conditional probability for class *j* as the fraction of points in $\mathcal{N}_0$ whose response values equal *j*:

$$P(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

Then, the *k*-NN classifier applies Bayes rule and classifies the test observation, x_0, to the class with largest estimated probability.

Estimated Probabilities in *k*-NN Classification



k-NN for Classification (cont.)

There are some issues that may arise:

- How can we handle a tie?
- What could be a major problem with always classifying to the most common group amongst the neighbors?

- How can we handle this?

k-NN with Multiple Predictors

How could we extend *k*-NN (both regression and classification) when there are multiple predictors?

We would need to define a measure of distance for observations in order to which are the most similar to the observation we are trying to predict.

Euclidean distance is a good option. To measure the distance of a new observation, $\boldsymbol{x}_0$ from each observation in the data set, $\boldsymbol{x}_i$:

$$D^2(\boldsymbol{x}_i, \boldsymbol{x}_0) = \sum_{j=1}^{P} \left(x_{i,j} - x_{0,j}\right)^2$$

k-NN with Multiple Predictors (cont.)

But what must we be careful about when measuring distance?

1. Differences in variability in our predictors!
2. Having a mixture of quantitative and categorical predictors.

So what should be good practice? To determine closest neighbors when $p > 1$, you should first standardize the predictors! And you can even standardize the binaries if you want to include them.

How else could we determine closeness in this multi-dimensional setting?

k-NN Classification in Python

Performing kNN classification in python is done via **`KNeighborsClassifier`** in **`sklearn.neighbors.`**

An example:

```
#two predictors
from sklearn import neighbors

knn1 = neighbors.KNeighborsClassifier(n_neighbors=1)
knn5 = neighbors.KNeighborsClassifier(n_neighbors=5)
knn10 = neighbors.KNeighborsClassifier(n_neighbors=10)
knn50 = neighbors.KNeighborsClassifier(n_neighbors=50)

data_x = df_heart[['MaxHR','RestBP']]
data_y = df_heart.AHD.map(lambda x: 0 if x=='No' else 1)

knn1.fit(data_x, data_y);
knn5.fit(data_x, data_y);
knn10.fit(data_x, data_y);
knn50.fit(data_x, data_y);

print(knn1.score(data_x, data_y))
print(knn5.score(data_x, data_y))
print(knn10.score(data_x, data_y))
print(knn50.score(data_x, data_y))
```

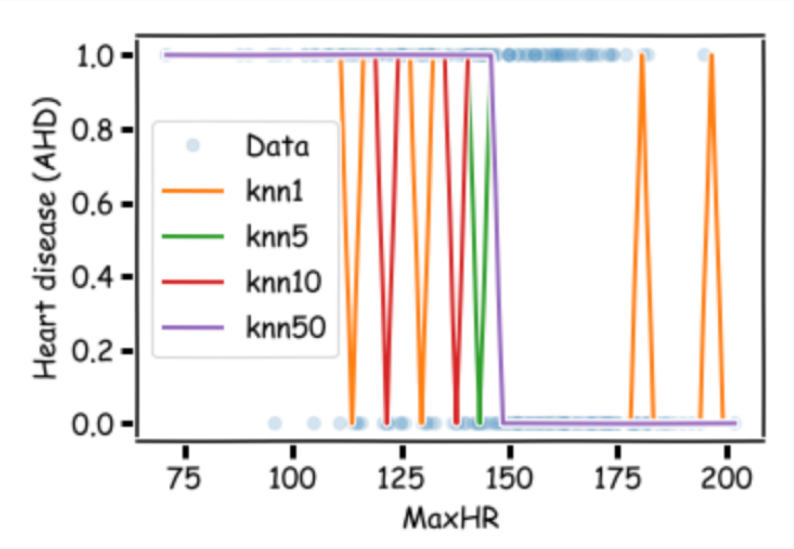
```
0.960396039604
0.712871287129
0.716171617162
0.706270627063
```

Classification Boundaries in *k*-NN Classification

What will the classification boundaries look line in *k*-NN classification? With one predictors? With 2 predictors? With 3+ predictors?

How can we visualize these? In 1D? In 2D? In 3D+?

How do they compare to Logistic Regression classification boundaries?



Interaction Terms and Unique Parameterizations

NYC Taxi vs. Uber

We'd like to compare Taxi and Uber rides in NYC (for example, how much the fare costs based on length of trip, time of day, location, etc.).

A public dataset has 1.9 million Taxi and Uber trips. Each trip is described by $p = 23$ useable predictors (and 1 response variable).

```
In [11]: print(nyc_cab_df.shape)
         nyc_cab_df.head()

         (1873671, 30)
```

Out[11]:

	AWND	Base	Day	Dropoff_latitude	Dropoff_longitude	Ehail_fee	Extra	Fare_amount	Lpep_dropoff_datetime	MTA_tax	...	TMIN	Tip_amount	Tolls_amou
0	4.7	B02512	1	NaN	NaN	NaN	NaN	33.863498	2014-04-01 00:24:00	NaN	...	39	NaN	Na
1	4.7	B02512	1	NaN	NaN	NaN	NaN	19.022892	2014-04-01 00:29:00	NaN	...	39	NaN	Na
2	4.7	B02512	1	NaN	NaN	NaN	NaN	25.498981	2014-04-01 00:34:00	NaN	...	39	NaN	Na
3	4.7	B02512	1	NaN	NaN	NaN	NaN	28.024628	2014-04-01 00:39:00	NaN	...	39	NaN	Na
4	4.7	B02512	1	NaN	NaN	NaN	NaN	12.083589	2014-04-01 00:40:00	NaN	...	39	NaN	Na

5 rows × 30 columns

Interaction Terms: A Review

Recall that an interaction term between predictors X_1 and X_2 can be incorporated into a regression model by including the multiplicative (i.e. cross) term in the model, for example

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 + X_2) + \varepsilon$$

Suppose X_1 is a binary predictor indicating whether a NYC ride pickup is a taxi or an Uber, X_2 is the length of the trip, and Y is the fare for the ride.

What is the interpretation of β_3?

Including Interaction Terms in Models

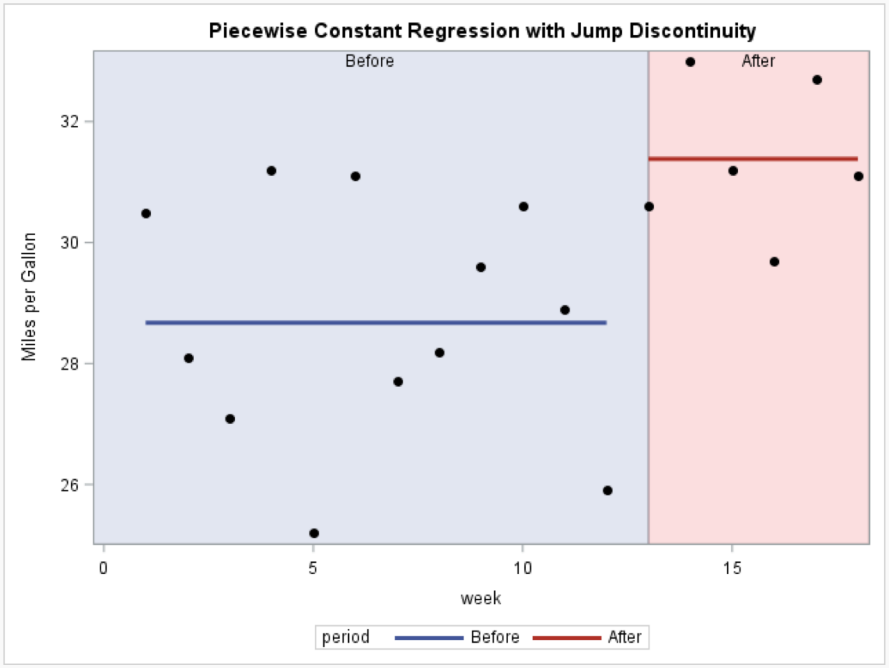
Recall that to avoid overfitting, we sometimes elect to exclude a number of terms in a linear model.

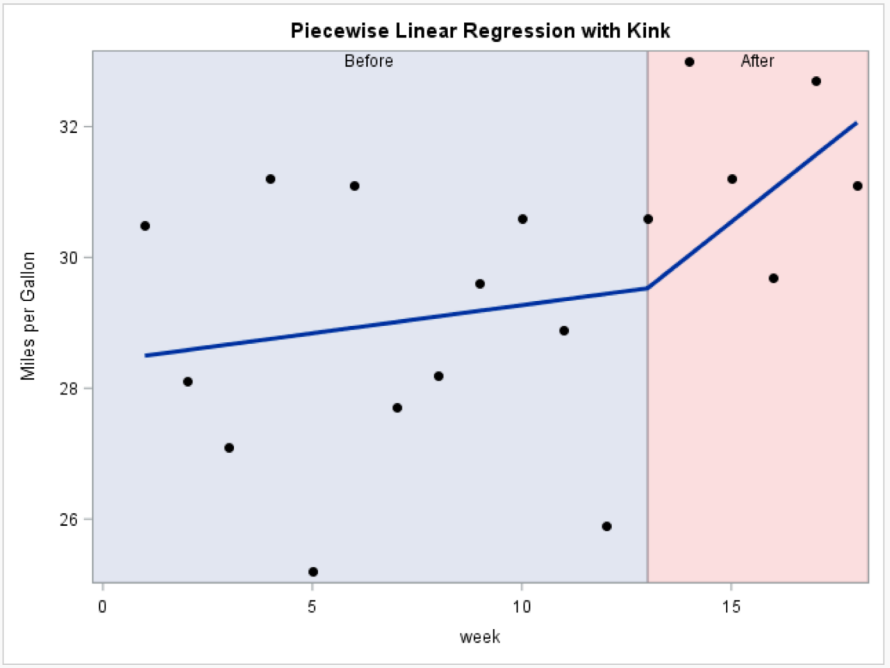
It is standard practice to always include the ***main effects*** in the model. That is, we always include the terms involving only one predictor, $\beta_1 X_1$, $\beta_2 X_2$ etc.

Question: Why are the ***main effects*** important?

Question: In what type of model would it make sense to include the interaction term without one of the main effects?

How would you *parameterize* these model?





$$\hat{Y} = \beta_0 + \beta_1 \cdot I(X \geq 13)$$

$$\hat{Y} = \beta_0 + \beta_1 X + \beta_2 \cdot (X - 13) \cdot I(X \geq 13))$$

How Many Interaction Terms?

This NYC taxi and Uber dataset has 1.9 million Taxi and Uber trips. Each trip is described by p = 23 useable predictors (and 1 response variable). How many interaction terms are there?

- Two-way interactions: $\binom{p}{2} = \frac{p(p-1)}{2} = 253$
- Three-way interactions: $\binom{p}{3} = \frac{p(p-1)(p-2)}{6} = 1771$
- Etc.

The total number of all possible interaction terms (including main effects) is.

$$\sum_{k=0}^{p} \binom{p}{k} = 2^p \approx 8.3\text{million}$$

What are some problems with building a model that includes all possible interaction terms?

How Many Interaction Terms?

In order to wrangle a data set with over 1 billion observations, we could use random samples of 100k observations from the dataset to build our models. If we include all possible interaction terms, our model will have 8.3 mil parameters. **We will not be able to uniquely determine 8.3 mil parameters with only 100k observations**. In this case, we call the model ***unidentifiable***.

In practice, we can:

- increase the number of observation
- consider only scientifically important interaction terms
- perform variable selection
- perform another ***dimensionality reduction*** technique like PCA

Big Data and High Dimensionality

What is 'Big Data'?

In the world of Data Science, the term *Big Data* gets thrown around a lot. What does *Big Data* mean?

A rectangular data set has two dimensions: number of observations (n) and the number of predictors (p). Both can play a part in defining a problem as a *Big Data* problem.

What are some issues when:

- n is big (and p is small to moderate)?
- p is big (and n is small to moderate)?
- n and p are both big?

When *n* is big

When the sample size is large, this is typically not much of an issue from the statistical perspective, just one from the computational perspective.

- Algorithms can take forever to finish. Estimating the coefficients of a regression model, especially one that does not have closed form (like LASSO), can take a while. Wait until we get to Neural Nets!
- If you are tuning a parameter or choosing between models (using CV), this exacerbates the problem.

What can we do to fix this computational issue?

- Perform 'preliminary' steps (model selection, tuning, etc.) on a subset of the training data set. 10% or less can be justified

Keep in mind, big *n* doesn't solve everything

The era of Big Data (aka, large *n*) can help us answer lots of interesting scientific and application-based questions, but it does not fix everything.

Remember the old adage: "**crap in = crap out**". That is to say, if the data are not representative of the population, then modeling results can be terrible. Random sampling ensures representative data.

Xiao-Li Meng does a wonderful job describing the subtleties involved (WARNING: it's a little technical, but digestible):
https://www.youtube.com/watch?v=8YLdIDOMEZs

When *p* is big

When the number of predictors is large (in any form: interactions, polynomial terms, etc.), then lots of issues can occur.

- Matrices may not be invertible (issue in OLS).
- Multicollinearity is likely to be present
- Models are susceptible to overfitting

This situation is called *High Dimensionality*, and needs to be accounted for when performing data analysis and modeling.

What techniques have we learned to deal with this?

When Does High Dimensionality Occur?

The problem of high dimensionality can occur when the number of parameters exceeds or is close to the number of observations. This can occur when we consider lots of interaction terms, like in our previous example. But this can also happen when the number of main effects is high.

For example:

- When we are performing polynomial regression with a high degree and a large number of predictors.
- When the predictors are genomic markers (and possible interactions) in a computational biology problem.
- When the predictors are the counts of all English words appearing in a text.

How Does sklearn handle unidentifiability?

In a parametric approach: if we have an over-specified model ($p > n$), the parameters are unidentifiable: we only need $n - 1$ predictors to perfectly predict every observation ($n - \mathbf{1}$ because of the intercept).

So what happens to the 'extra' parameter estimates (the extra β's)?

- the remaining $p - (n - 1)$ predictors' coefficients can be estimated to be anything. Thus there are an infinite number of sets of estimates that will give us identical predictions. There is not one unique set of $\hat{\beta}$'s.

What would be reasonable ways to handle this situation? How does sklearn handle this? When is another situation in which the parameter estimates are unidentifiable? What is the simplest case?

Perfect Multicollinearity

The $p > n$ situation leads to perfect collinearity of the predictor set. But this can also occur with a redundant predictors (ex: putting X_j twice into a model). Let's see what sklearn in this simplified situation:

```
In [9]: # investigating what happens when two identical predictors are used

        logit1 = LogisticRegression(C=1000000,solver="lbfgs").fit(heart_df[['Age']],y)
        logit2 = LogisticRegression(C=1000000,solver="lbfgs").fit(heart_df[['Age','Age']],y)

        print("The coef estimate for Age (when in the model once):",logit1.coef_)
        print("The coef estimates for Age (when in the model twice):",logit2.coef_)

        The coef estimate for Age (when in the model once): [[0.05198618]]
        The coef estimates for Age (when in the model twice): [[0.02599311 0.02599311]]
```

How does this generalize into the high-dimensional situation?

A Framework For Dimensionality Reduction

One way to reduce the dimensions of the feature space is to create a new, smaller set of predictors by taking linear combinations of the original predictors.

We choose $Z_1, Z_2, \ldots, Z_m$, where $m \leq p$ and where each Z_i is a linear combination of the original p predictors

$$Z_i = \sum_{j=1}^{p} \phi_{ji} X_j$$

for fixed constants ϕ_{ji}. Then we can build a linear regression regression model using the new predictors

$$Y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_m Z_m + \varepsilon$$

Notice that this model has a smaller number (m+1 < p+1) of parameters.

A Framework For Dimensionality Reduction (cont.)

A method of dimensionality reduction includes 2 steps:

- Determine a optimal set of new predictors $Z_1,\ldots, Z_m$, for $m < p$.
- Express each observation in the data in terms of these new predictors. The transformed data will have m columns rather than p.

Thereafter, we can fit a model using the new predictors.

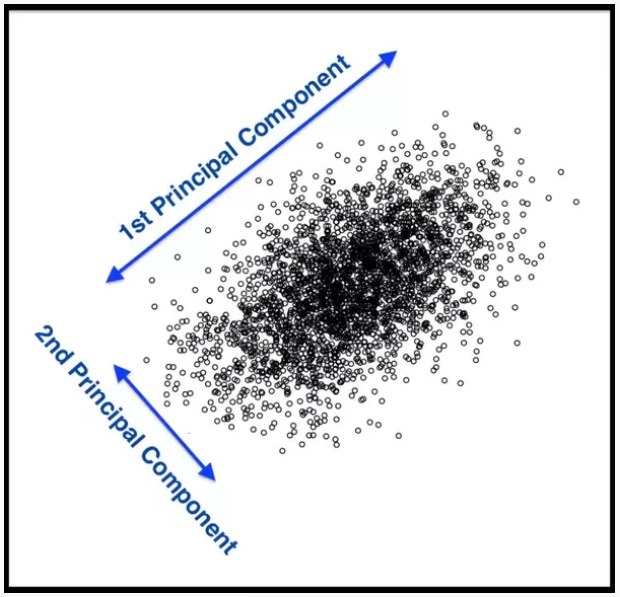
The method for determining the set of new predictors (what do we mean by an optimal predictors set?) can differ according to application. We will explore a way to create new predictors that captures the *essential* variations in the observed predictor set.

You’re Gonna Have a Bad Time…

Principal Components Analysis (PCA)

Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a method to identify a new set of predictors, as linear combinations of the original ones, that captures the `maximum amount' of variance in the observed data.



PCA (cont.)

Principal Components Analysis (PCA) produces a list of *p principle components* $Z_1,\ldots, Z_p$ such that

- Each Z_i is a linear combination of the original predictors, and it's vector norm is 1
- The Z_i's are pairwise orthogonal
- The Z_i's are ordered in decreasing order in the amount of captured observed variance.

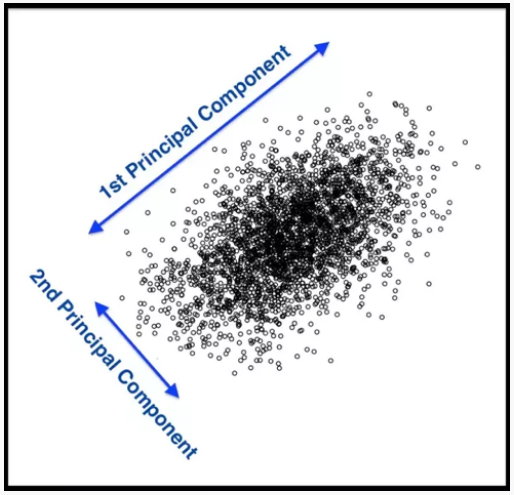
That is, the observed data shows more variance in the direction of Z_1 than in the direction of Z_2.

To perform dimensionality reduction we select the top *m* principle components of PCA as our new predictors and express our observed data in terms of these predictors.

The Intuition Behind PCA

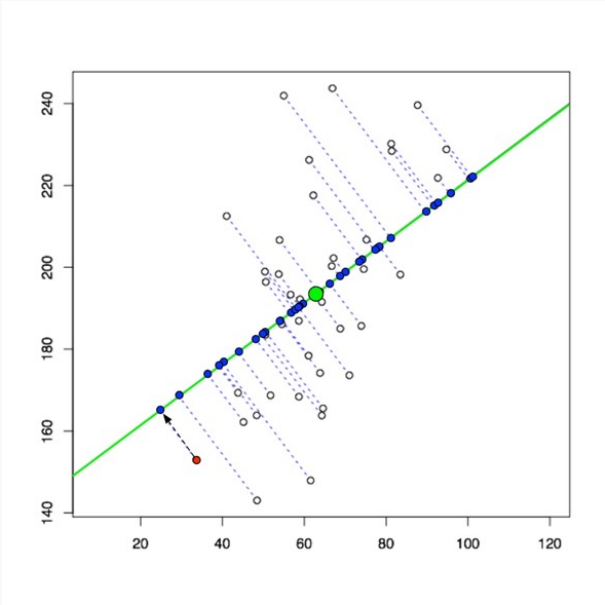
Top PCA components capture the most of amount of variation (interesting features) of the data.

Each component is a linear combination of the original predictors - we visualize them as vectors in the feature space.



The Intuition Behind PCA (cont.)

Transforming our observed data means projecting our dataset onto the space defined by the top *m* PCA components, these components are our new predictors.



The Math behind PCA

PCA is a well-known result from linear algebra. Let **Z** be the *n* x *p* matrix consisting of columns $Z_1,\ldots, Z_p$ (the resulting PCA vectors), **X** be the *n* x *p* matrix of $X_1,\ldots, X_p$ of the original data variables (each standardized to have mean zero and variance one, and without the intercept), and let **W** be the *p* x *p* matrix whose columns are the eigenvectors of the square matrix $\mathbf{X}^T\mathbf{X}$, then:

$$\mathbf{Z}_{n\times p} = \mathbf{X}_{n\times p}\mathbf{W}_{p\times p}$$

Implementation of PCA using linear algebra

To implement PCA yourself using this linear algebra result, you can perform the following steps:

- Standardize each of your predictors (so they each have mean = 0, var = 1).
- Calculate the eigenvectors of the $\mathbf{X}^T\mathbf{X}$ matrix and create the matrix with those columns, **W**, in order from largest to smallest eigenvalue.
- Use matrix multiplication to determine $\mathbf{Z} = \mathbf{XW}$.

Note: this is not efficient from a computational perspective. This can be sped up using Cholesky decomposition.

However, PCA is easy to perform in Python using the `decomposition.PCA` function in the `sklearn` package.

PCA example in sklearn

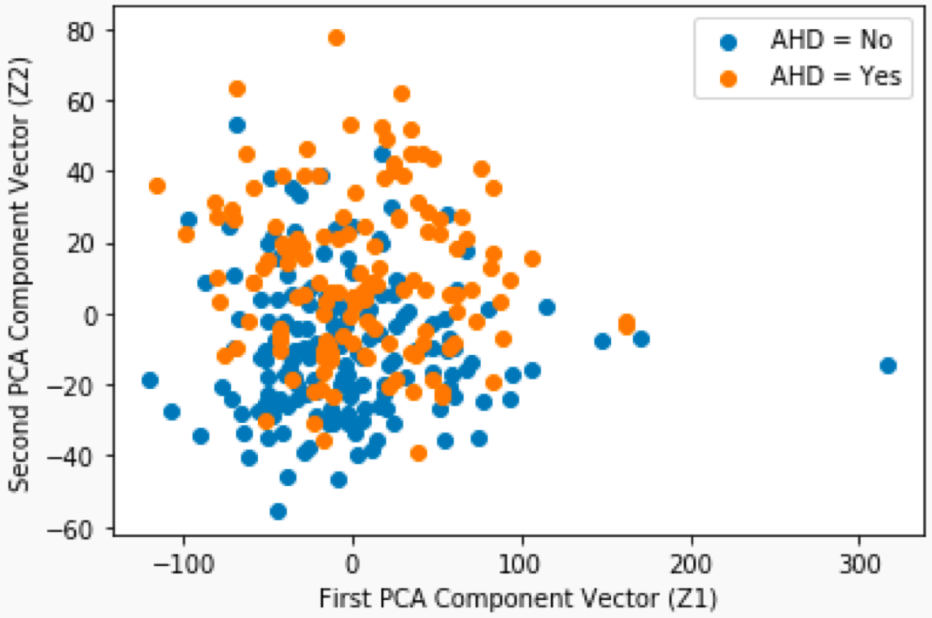
In [11]:
```
X = heart_df[['Age','RestBP','Chol','MaxHR']]

# create/fit the 'full' pca transformation
pca = PCA().fit(X)

# apply the pca transformation to the full predictor set
pcaX = pca.transform(X)

# convert to a data frame
pcaX_df = pd.DataFrame(pcaX, columns=[['PCA1' , 'PCA2', 'PCA3', 'PCA4']])

# here are the weighting (eigen-vectors) of the variables (first 2 at least)
print("First PCA Component (w1):",pca.components_[0,:])
print("Second PCA Component (w2):",pca.components_[1,:])

# here is the variance explained:
print("Variance explained by each component:",pca.explained_variance_ratio_)
```

```
First PCA Component (w1): [ 0.03839966  0.05046168  0.99798051 -0.0037393 ]
Second PCA Component (w2): [ 0.180616    0.10481151 -0.01591307 -0.9778237 ]
Variance explained by each component: [0.74831735 0.15023974 0.0852975  0.01614541]
```

PCA example in sklearn

A common plot is to look at the scatterplot of the first two principal components, shown below for the Heart data:

What do you notice?



What's the difference: Standardize vs. Normalize

What is the difference between Standardizing and Normalizing a variable?

- Normalizing means to bound your variable's observations between zero and one. Good when interpretations of "percentage of max value" makes sense.
- Standardizing means to re-center and re-scale your variable's observations to have mean zero and variance one. Good to put all of your variables on the same scale (have same weight) and to turn interpretations into "changes in terms of standard deviation."

Warning: the term "normalize" gets incorrectly used all the time (online, especially)!

When to Standardize vs. Normalize

When should you do each?

- Normalizing is only for improving interpretation (and dealing with numerically very large or small measures). Does not improve algorithms otherwise.
- Standardizing can be used for improving interpretation and should be used for specific algorithms. Which ones? Regularization and PCA (so far)!

*Note: you can standardize without assuming things to be [approximately] Normally distributed! It just makes the interpretation nice if they are Normally distributed.

PCA for Regression (PCR)

PCA for Regression (PCR)

PCA is easy to use in Python, so how do we then use it for regression modeling in a real-life problem?

If we use all p of the new Z_j, then we have not improved the dimensionality. Instead, we select the first M PCA variables, $Z_1,...,Z_M$, to use as predictors in a regression model.

The choice of M is important and can vary from application to application. It depends on various things, like how collinear the predictors are, how truly related they are to the response, etc...

What would be the best way to check for a specified problem?

Cross Validation!!!

A few notes on using PCA

- PCA is an unsupervised algorithm. Meaning? It is done independent of the outcome variable.
    - Note: the components as predictors might not be ordered from best to worst!
- PCA is not so good because:
    1. Direct Interpretation of coefficients in PCR is completely lost. So do not do if interpretation is important.
    2. Will not improve predictive ability of a model.
- PCA is great for:
    1. Reducing dimensionality in very high dimensional settings.
    2. Visualizing how predictive your features can be of your response, especially in the classification setting.
    3. Reducing multicollinearity, and thus may improve the computational time of fitting models.

Interpreting the Results of PCR

- A PCR can be interpreted in terms of original predictors…very carefully.
- Each estimated β coefficient in the PCR can be *distributed* across the predictors via the associated component vector, *w*. An example is worth a thousand words:

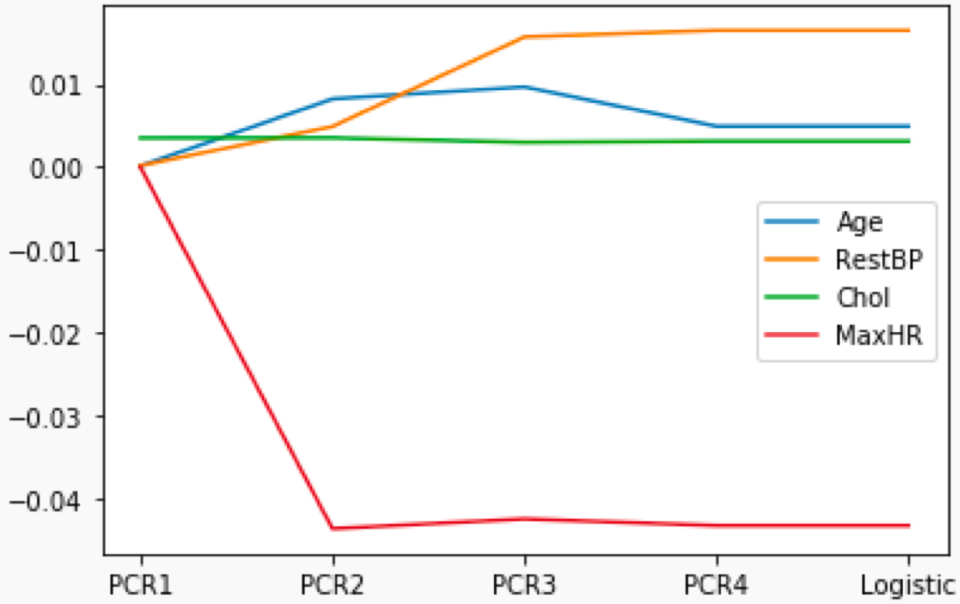
```
In [27]: logit_pcr1 = LogisticRegression(C=1000000,solver="lbfgs").fit(pcaX_df[['PCA1']],y)

         print("Intercept from simple PCR-Logistic:",logit_pca1.intercept_)
         print("'Slope' from simple PCR-Logistic:", logit_pca1.coef_)

         print("First PCA Component (w1):",pca.components_[0,:])


         Intercept from simple PCR-Logistic: [-0.1662098]
         'Slope' from simple PCR-Logistic: [[0.00351092]]
         First PCA Component (w1): [ 0.03839966  0.05046168  0.99798051 -0.0037393 ]
```

- So how can this be transformed back to the original variables?

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 Z_1 = \hat{\beta}_0 + \hat{\beta}_1(\vec{w}_1^T \mathbf{X}) = \hat{\beta}_0 + \hat{\beta}_1 \vec{w}_1^T(\mathbf{X})$$
$$= -0.1162 + 0.00351(0.0384X_1 + 0.0505X_2 + 0.998X_3 - 0.0037X_4)$$
$$= -0.1162 + 0.000135(X_1) + 0.000177(X_2) + 0.0035(X_3) - 0.000013(X_4)$$

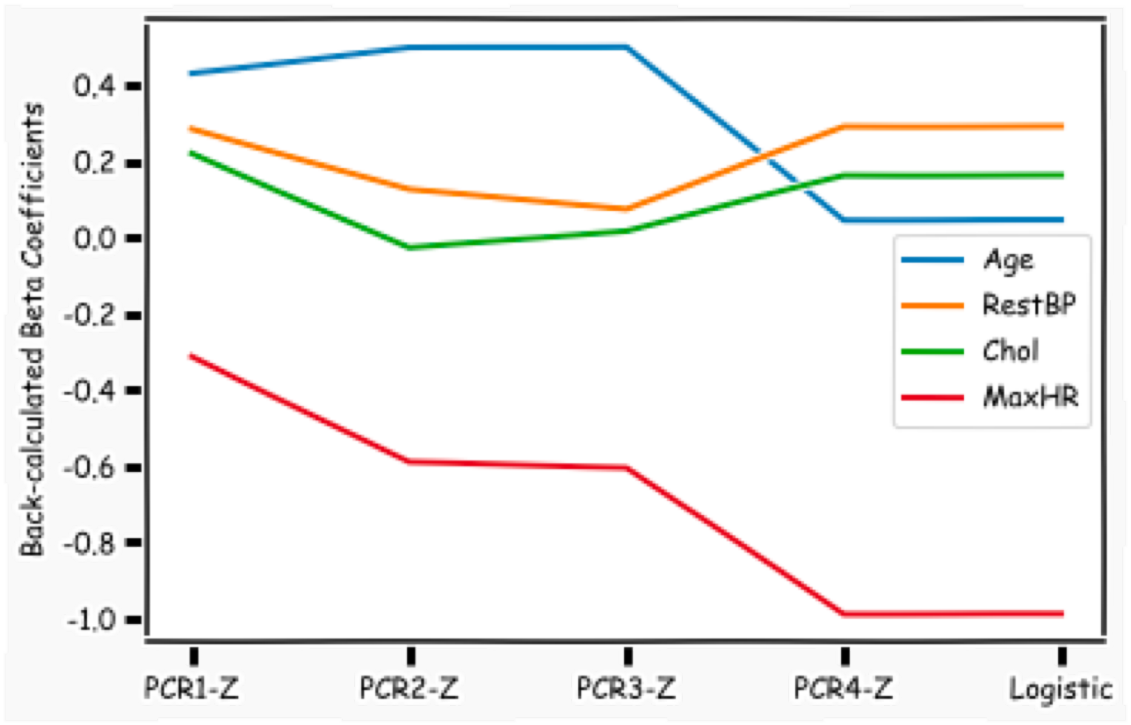
As more components enter the PCR…

- This algorithm can be continued by adding in more and more components into the PCR.



- This plot was done on non-standardized predictors. How would it look different if they were all standardized (both in PCA and plotted here in standardized form of X)?

The same plot with standardized predictors in PCA



Dealing with Missingness

What is missing data? k-Nearest Neighbors

Often times when data is collected, there are some missing values apparent in the dataset. This leads to a few questions to consider:

- How does this show up in pandas?
- How does statsmodels and sklearn handle these NaNs?
- How does this effect our modeling?

Naively handling missingness

What are the simplest ways to handle missing data?

- Drop the observations that have any missing values.
  - Use pd.DataFrame.dropna(axis=0)
- Impute the mean/median (if quantitative) or most common class (if categorical) for all missing values.
  - Use pd.DataFrame.fillna(value=x.mean())

How does statsmodels and sklearn handle these NaNs?

What are some consequences in handling missingness in this fashion?

Missingness Indicator Variable

One simple way to handle missingness in a variable, X_j, is to impute a value (like 0 or $\overline{X_j}$), then create a new variable, $X_{j,miss}$, that indicates this observation had a missing value. If X_j is categorical then just impute 0.

Then include both $X_{j,miss}$ and X_j as predictors in any model.

Illustration is to the right.

X_1	X_2	X_1^*	X_2^*	$X_{1,miss}$	$X_{2,miss}$
10	.	10	0	0	1
5	1	5	1	0	0
21	0	21	0	0	0
15	0	15	0	0	0
16	.	16	0	0	1
.	.	0	0	1	1
21	1	21	1	0	0
12	0	12	0	0	0
.	1	0	1	1	0

Why use a Missingness Indicator Variable?

How does this missingness indicator variable improve the model?

Because the group of individuals with a missing entry may be systematically different than those with that variable measured. Treating them equivalently could lead to bias in quantifying relationships (the β's) and underperform in prediction.

For example: imagine a survey questions asks whether or not someone has ever recreationally used opioids, and some people chose not to respond. Does the fact that they did not respond provide extra information? Should we treat them equivalently as never-users?

This approach essentially creates a third group for this predictor: the "did not respond" group.

Types of Missingness

Sources of Missingness

Missing data can arise from various places in data:

- A survey was conducted and values were just randomly missed when being entered in the computer.
- A respondent chooses not to respond to a question like `Have you ever done cocaine?'.
- You decide to start collecting a new variable (like Mac vs. PC) partway through the data collection of a study.
- You want to measure the speed of meteors, and some observations are just 'too quick' to be measured properly.

The source of missing values in data can lead to the major types of missingness:

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)** - the probability of missingness in a variable is the same for all units. Like randomly poking holes in a data set.
2. **Missing at Random (MAR)** - the probability of missingness in a variable depends only on available information (in other predictors).
3. **Missing Not at Random (MNAR)** - the probability of missingness depends on information that has not been recorded and this information also predicts the missing values.

What are examples of each these 3 types?

Missing completely at random (MCAR)

Missing Completely at Random is the best case scenario, and the easiest to handle:

- Examples: a coin is flipped to determine whether an entry is removed. Or when values were just randomly missed when being entered in the computer.
- Effect if you ignore: there is no effect on inferences (estimates of beta).
- How to handle: lots of options, but best to impute (more on next slide).

Missing at random (MAR)

Missing at random is still a case that can be handled.

- Example(s): men and women respond to the question "have you ever felt harassed at work?" at different rates (and may be harassed at different rates).
- Effect if you ignore: inferences are biased (estimates of β's) and predictions are usually worsened.
- How to handle: use the information in the other predictors to build a model and **impute** a value for the missing entry.

Key: we can fix any biases by modeling and imputing the missing values based on what is observed!

Missing Not at Random (MNAR)

Missing Not at Random is the worst case scenario, and impossible to handle properly:

- Example(s): patients drop out of a study because they experience some really bad side effect that was not measured. Or cheaters are less likely to respond when asked if you've ever cheated.
- Effect if you ignore: there is no effect on inferences (estimates of beta) or predictions.
- How to handle: you can 'improve' things by dealing with it like it is MAR, but you [likely] may never completely fix the bias. And incorporating a **missingness indicator variable** may actually be the best approach (if it is in a predictor).

What type of missingness is present?

Can you ever tell based on your data what type of missingness is actually present?

Since we asked the question, the answer must be no.

It generally cannot be determined whether data really are missing at random, or whether the missingness depends on unobserved predictors or the missing data themselves. The problem is that these potential “lurking variables” are unobserved (by definition) and so can never be completely ruled out.

In practice, a model with as many predictors as possible is used so that the `missing at random' assumption is reasonable.

Imputation Methods

Handling Missing Data

When encountering missing data, the approach to handling it depends on:

1. whether the missing values are in the response or in the predictors. Generally speaking, it is much easier to handle missingness in predictors.
2. whether the variable is quantitative or categorical.
3. how much missingness is present in the variable. If there is too much missingness, you may be doing more damage than good.

Generally speaking, it is a good idea to attempt to **impute** (or 'fill in') entries for missing values in a variable (assuming your method of imputation is a good one).

Imputation Methods

There are several different approaches to imputing missing values:

1. **Impute the mean or median** (quantitative) or most common class (categorical) for all missing values in a variable.
2. Create a new variable that is an **indicator of missingness**, and include it in any model to predict the response (also plug in zero or the mean in the actual variable).
3. **Hot deck imputation**: for each missing entry, randomly select an observed entry in the variable and plug it in.
4. **Model the imputation**: plug in predicted values ($\hat{y}$) from a model based on the other observed predictors.
5. **Model the imputation with uncertainty**: plug in predicted values plus randomness ($\hat{y} + \epsilon$) from a model based on the other observed predictors.

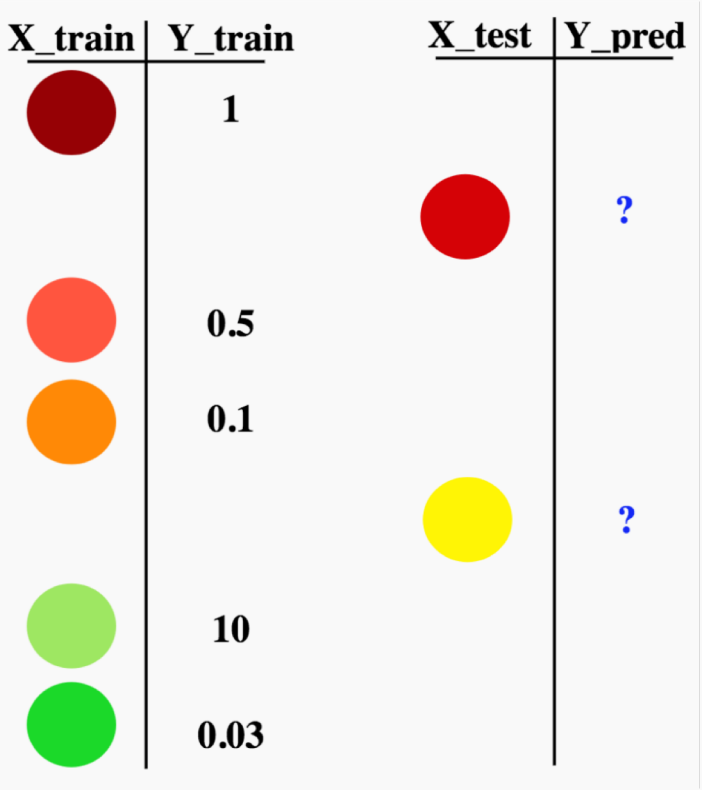
What are the advantages and disadvantages of each approach?

Schematic: imputation through modeling

How do we use models to fill in missing data?

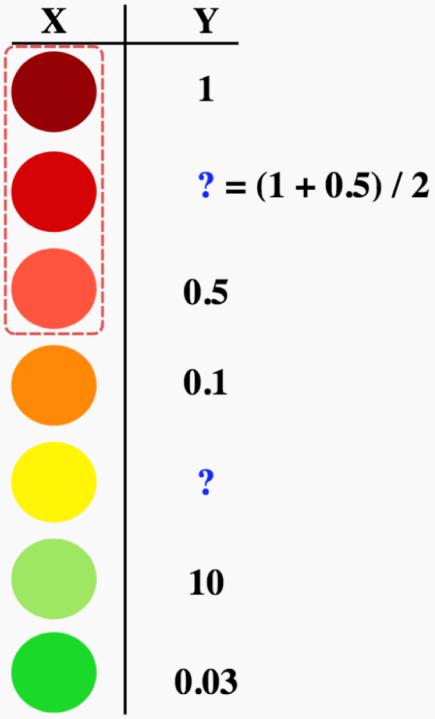
Schematic: imputation through modeling

How do we use models to fill in missing data?



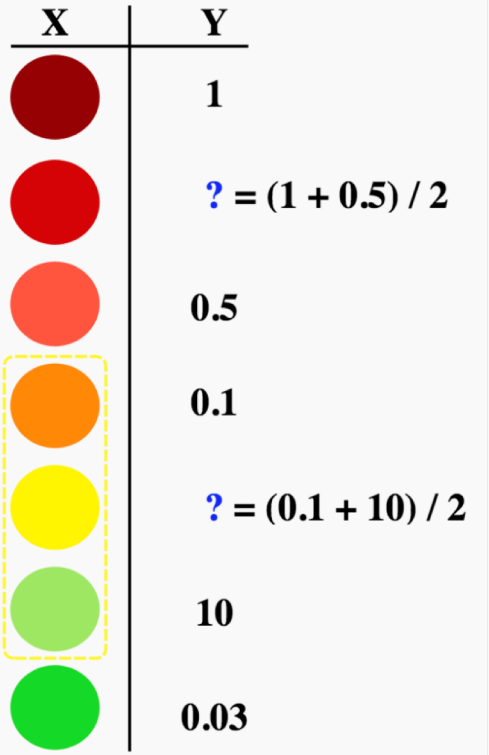
Schematic: imputation through modeling

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



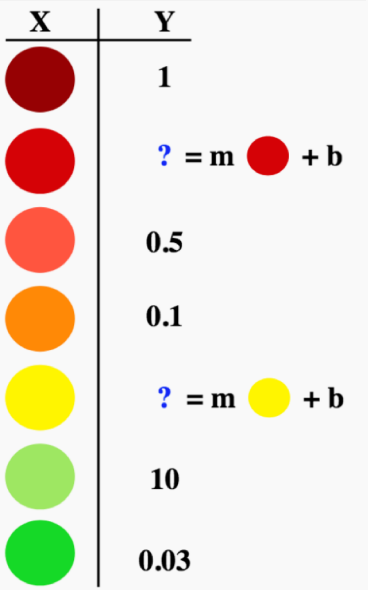
Schematic: imputation through modeling

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



Schematic: imputation through modeling

How do we use models to fill in missing data? Using linear regression?



Where *m* and *b* are computed from the observations (rows) that do not have missingness (we should call them $b = \beta_0$ and $m = \beta_1$).

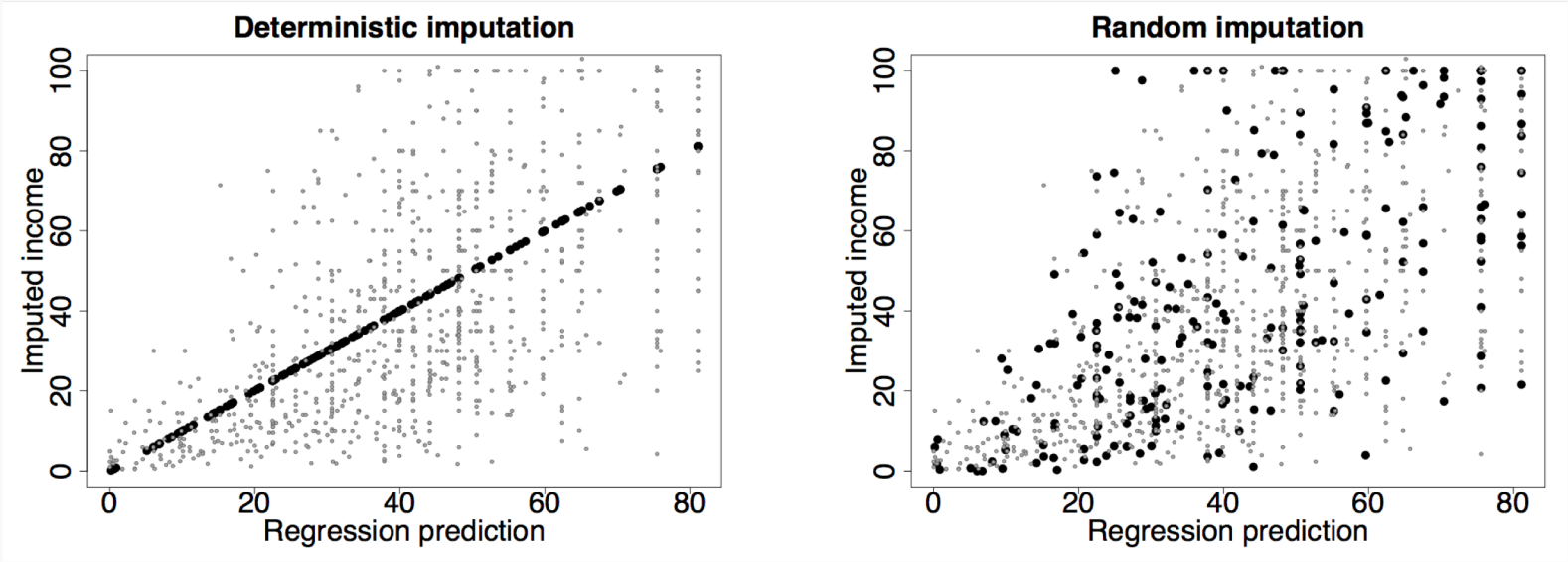
Imputation through modeling with uncertainty

The schematic in the last few slides ignores the fact of imputing with uncertainty. What happens if you ignore this fact and just use the 'best' model to impute values solely on $\hat{y}$?

The distribution of the imputed values will be too narrow and not represent real data (see next slide for illustration). The goal is to impute values that include the uncertainty of the model.

How can this be done in practice in *k*-NN? In linear regression? In logistic regression?

Imputation through modeling with uncertainty: an illustration



Imputation through modeling with uncertainty: an illustration

Recall the probabilistic model in linear regression:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \varepsilon$$

where $\epsilon \sim N(0, \sigma^2)$. How can we take advantage of this model to impute with uncertainty?

It's a 3 step process:

1. Fit a model to predict the predictor variable with missingness from all the other predictors.
2. Predict the missing values from the model in the previous part.
3. Add in a measure of uncertainty to this prediction by randomly sampling from a $N(0, \sigma^2)$ distribution, where σ^2 is the mean square error (MSE) from the model.

Imputation through modeling with uncertainty: *k*-NN regression

How can we use *k*-NN regression to impute values that mimic the error in our observations?

Two ways:

- Use $k = 1$.
- Use any other *k*, but randomly select from the nearest neighbors in $\mathcal{N}_0$. This can be done with equal probability or with some weighting (inverse to the distance measure used).

Imputation through modeling with uncertainty: classifiers

For classifiers, this imputation with uncertainty/randomness is a little easier process. How can it be implemented?

If a classification model (logistic, *k*-NN, etc...) is used to predict the variable with missingness on the observed predictors, then all you need to do is flip a 'biased coin' (or multi-sided die) with the probabilities of coming up for each class equal to the predicted probabilities from the model.

Warning: do not just classify blindly using the predict command in sklearn!

Imputation across multiple variables

If only one variable has missing entries, life is easy. But what if all the predictor variables have a little bit of missingness (with some observations having multiple entries missing)? How can we handle that?

It's an iterative process. Impute X_1 based on X_2, ..., X_p. Then impute X_2 based on X_1 and X_3, ..., X_p. And continue down the line.

Any issues? Yes, not all of the missing values may be imputed with just one 'run' through the data set. So you will have to repeat these 'runs' until you have a completely filled in data set.

Multiple imputation: beyond this class

What is an issue with treating your now 'complete' data set (a mixture of actually observed values and imputed values) as simply all observed values?

Any inferences or predictions carried out will be tuned and potentially overfit to the random entries imputed for the missing entries. How can we prevent this phenomenon?

By performing **multiple imputation**: rerun the imputation algorithm many times, refit the model on the response many times (one time each), and then 'average' the predictions or estimates of β coefficients to perform inferences (also incorporating the uncertainty involved).

Note: this is beyond what we would expect in this class. But it generally a good thing to be aware of.