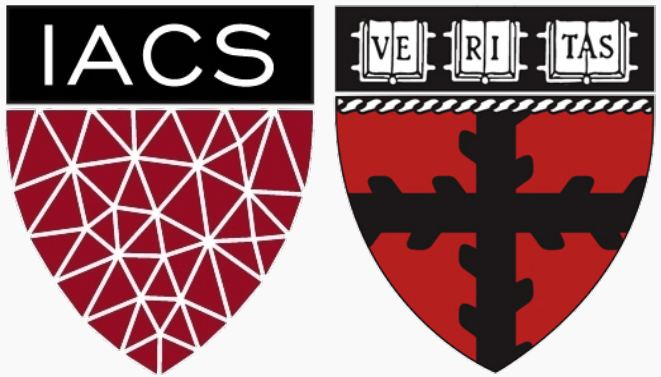
Lecture 14: High Dimensionality & PCA

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner



Lecture Outline

- Interaction Terms and Unique Parameterizations
- Big Data and High Dimensionality
- Principal Component Analysis (PCA)
- Principal Component Regression (PCR)

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?

B2 = relationship of length of trip to fare for uber
B2 + B3 = relationship of length of trip to fare for taxis (B3 compares taxi and uber for each length of trip)

Including Interaction Terms in Models

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

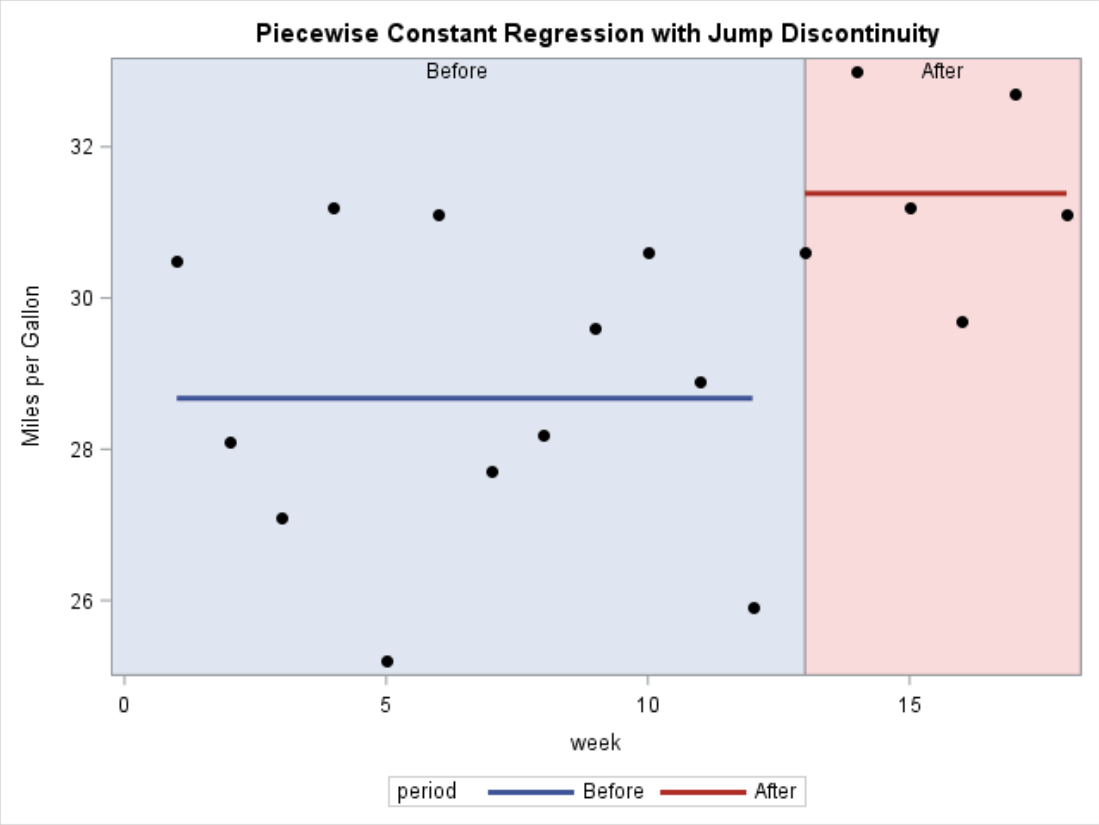
It is standard practice to always include the ***main effects*** in the model. That is, we always include the terms involving only one predictor, $\beta_1 X_1$, $\beta_2 X_2$ etc.

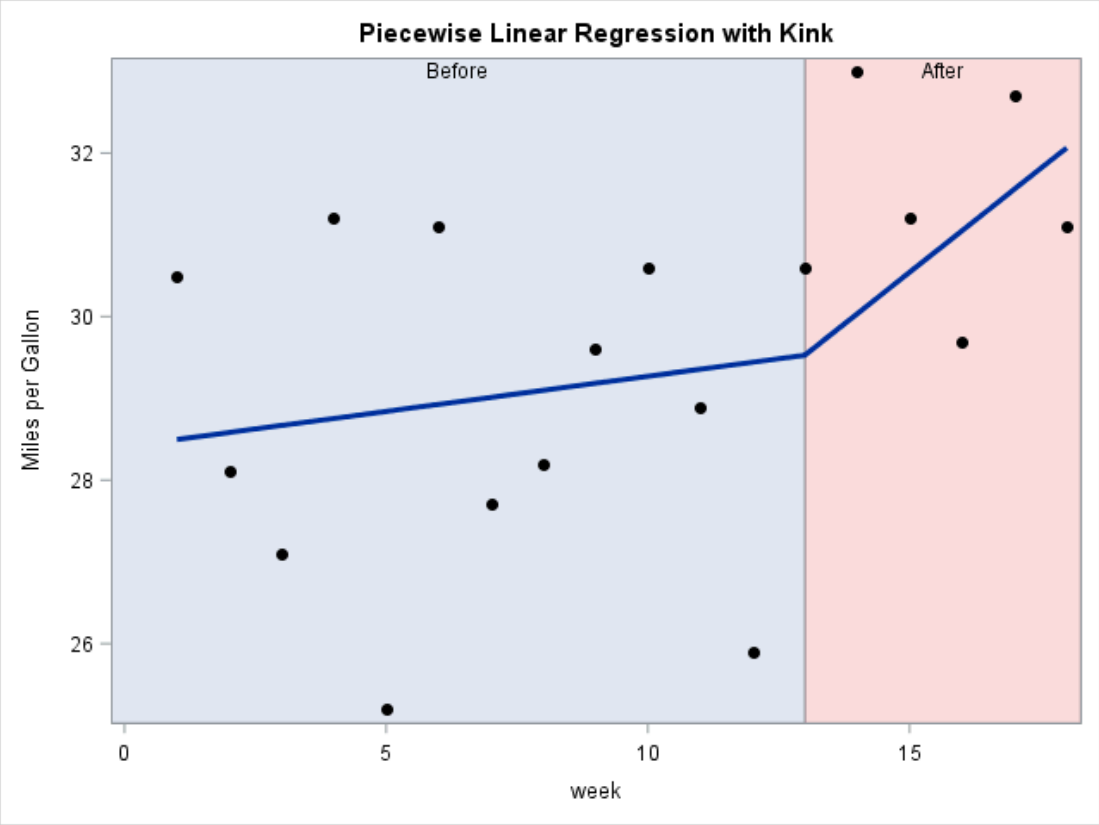
Question: Why are the ***main effects*** important?

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?

No main effect for this interaction term





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

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

when X is less than 13, you have just B1
when X is greater than 13, you add small contributions of B2 to B1 as you move away from X=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***.

no closed form solution here; but we can solve it iteratively and get an answer, but it won't necessarily be the best solution (most minimal loss fxn)

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): Biases get worse with large n
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?

Variable selection thru regularization, cross-validation

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?

Simplest case of unidentifiable model : have the sample predictor twice

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]]
```

distributes the effects equally across the redundant predictors

How does this generalize into the high-dimensional situation?

water down the signal of one predictor if you have lots of collinear ones with that predictor

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

fewer Z than X predictors - Z is a linear combination of the X predictors (keeping the essence of all the 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.

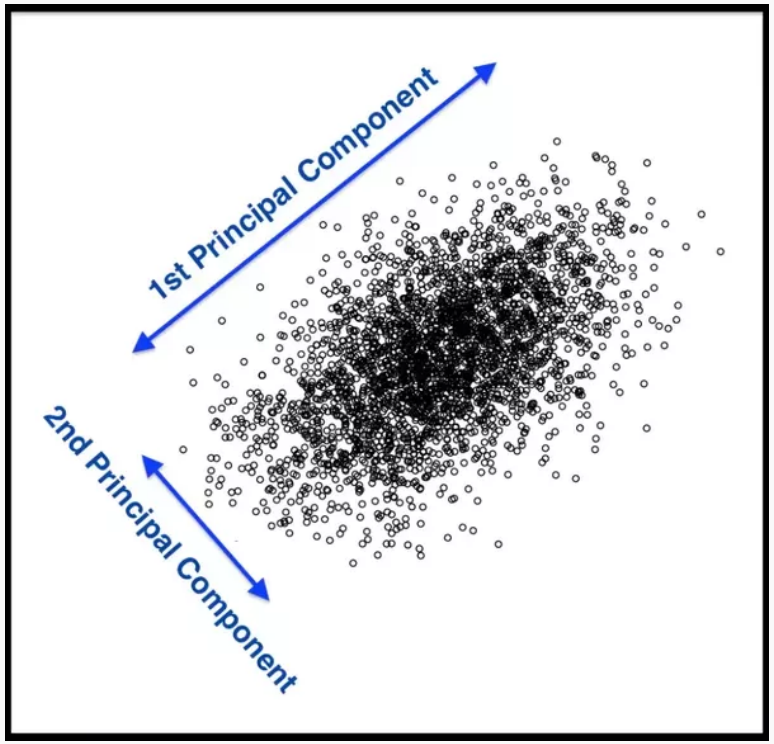
The method for determining the set of new predictors (what do we mean by an optimal predictors set?) can differ according to application. We will explore a way to create new predictors that captures the *essential* variations in the observed predictor set.

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 distance is 1, length is 1 - all on the same scale
- The Z_i's are pairwise orthogonal no collinearity
- The Z_i's are ordered in decreasing order in the amount of captured observed variance. ordered in importance

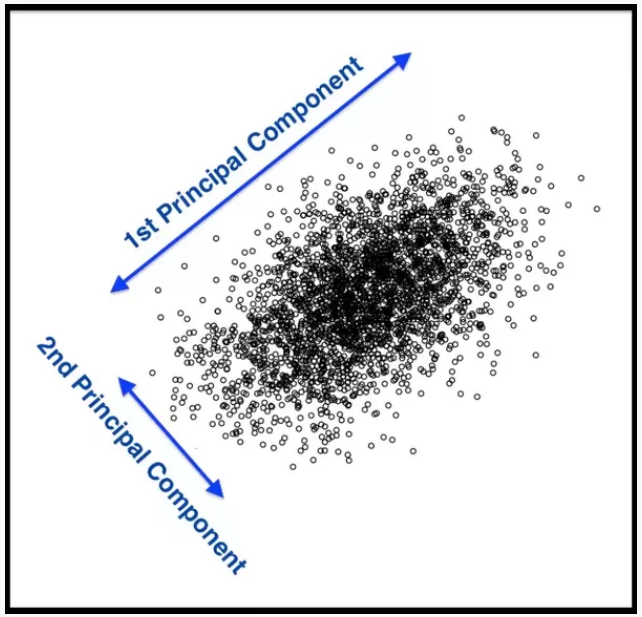
That is, the observed data shows more variance in the direction of Z_1 than in the direction of Z_2.

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

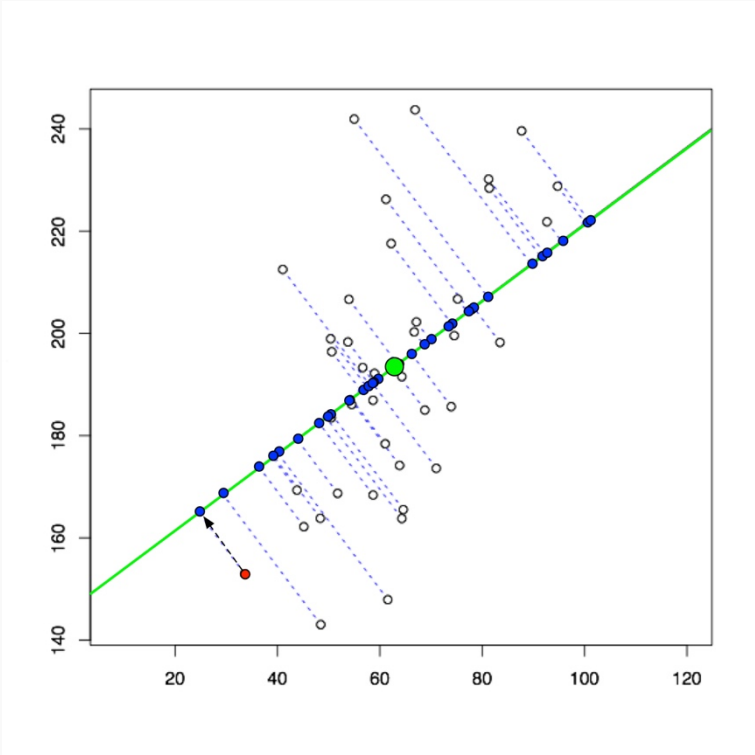
Top PCA components capture the most of amount of variation (interesting features) of the data.

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:

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

w is the transformation from X space to Z space

Implementation of PCA using linear algebra

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

categorical - turn into one-hot encoding, standardize

comparing variances in PCA - you want all your predictors on the same scale

- 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.

predictors that are highly collinear get small weights (W) in Z , those predictors that are uncorrelated with others get bigger weights (W)

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_)
```

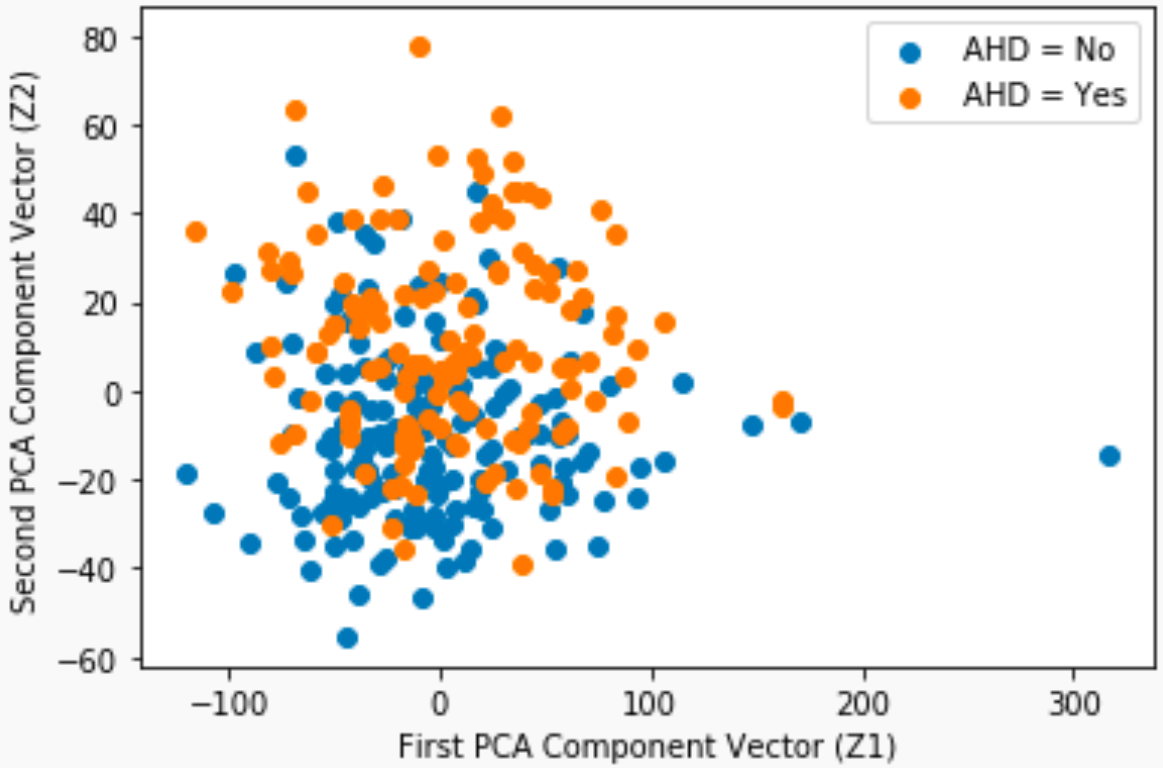
```
First PCA Component (w1): [ 0.03839966  0.05046168  0.99798051 -0.0037393 ]
Second PCA Component (w2): [ 0.180616    0.10481151 -0.01591307 -0.9778237 ]
Variance explained by each component: [0.74831735 0.15023974 0.0852975  0.01614541]
```

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?

way to visually separate two classes when plotting on PC space - it could show the predictive ability of your features to separate the classes, but be careful in interpreting the success of your model based on these plots (could underestimate it)



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, $Z_1,\ldots,Z_M$, 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!!!

choose your hyperparameter M

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! in terms of predictive ability
- 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:

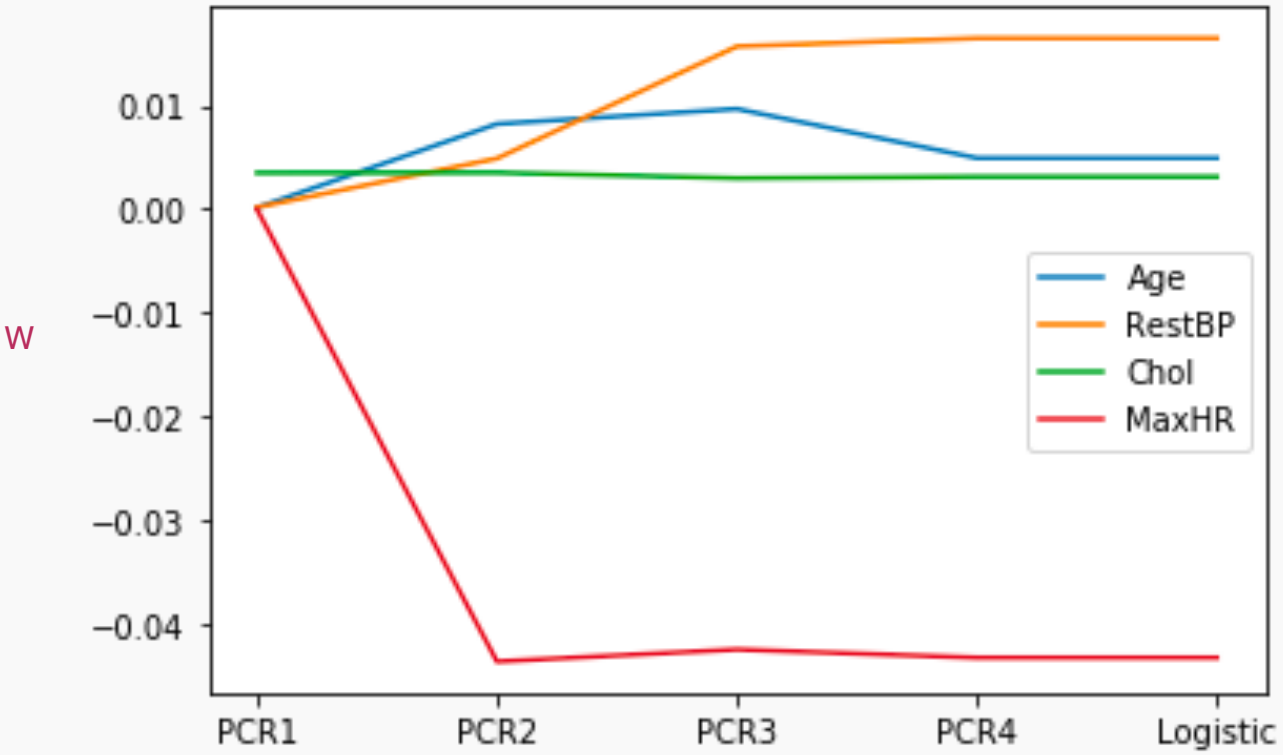
```
In [27]: logit_pcr1 = LogisticRegression(C=1000000,solver="lbfgs").fit(pcaX_df[['PCA1']],y)

         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.

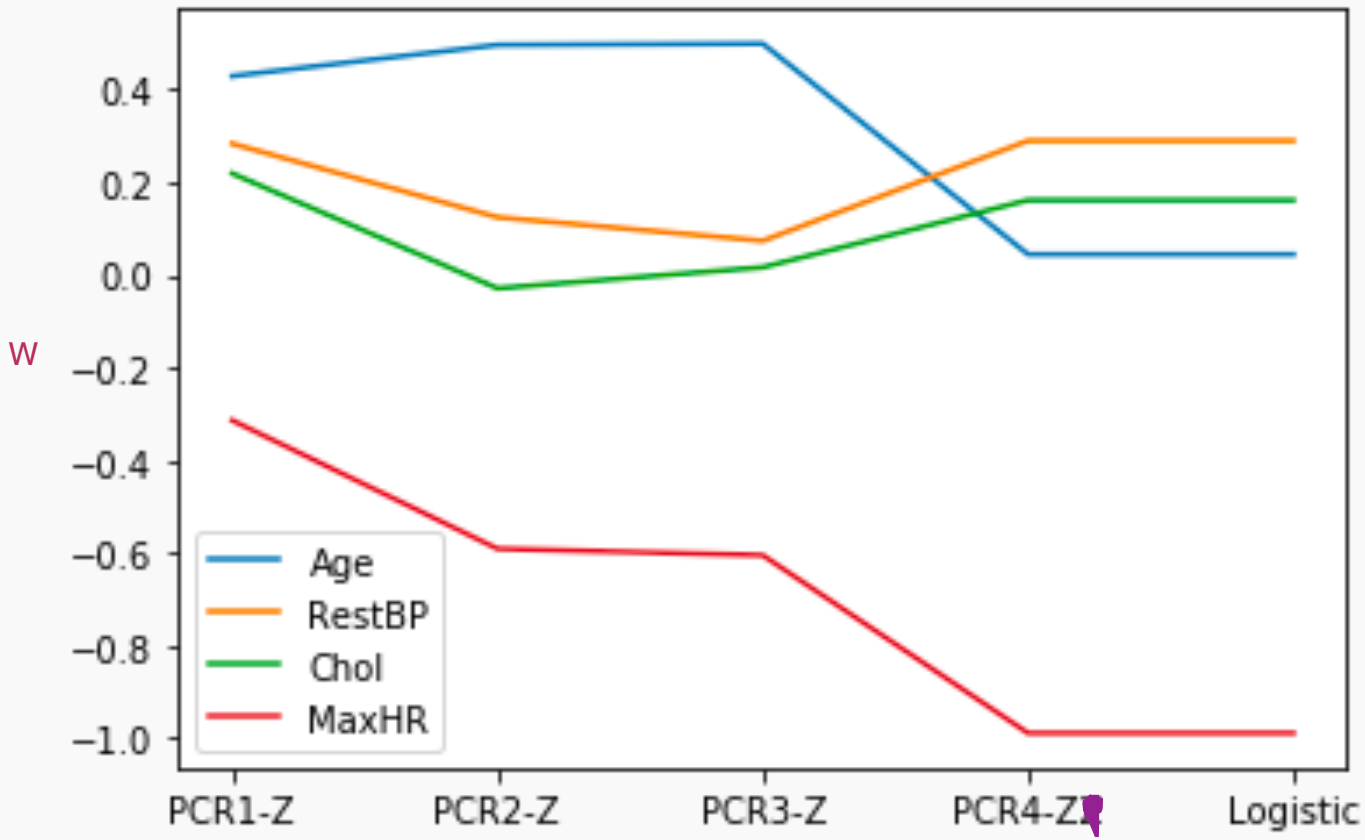


PC1 = Cholesterol is predominant
PC2 = Max HR is predominant

do not distribute weight across all predictors evenly

- 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



More structured pattern in how weights on predictors vary with PC components (PC1 most var captured to PC4 least captured)

can only have N (num obs) or P (num predictors) PCs whatever is smaller