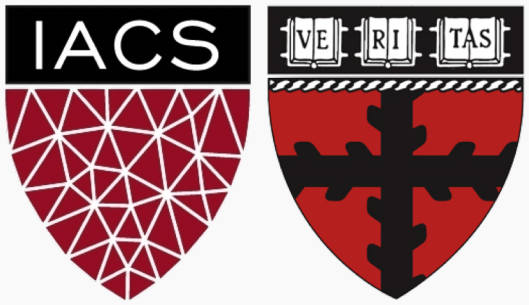
Lecture 8: High Dimensionality & PCA

CS109A Introduction to Data Science

Pavlos Protopapas and Kevin Rader



Announcements

Homeworks:

HW3 is due tonight, late day til tomorrow.

HW4 is an individual HW. Only private piazza posts.

Projects:

- **Milestone 1:** remember to submit your project groups and topic preferences. Be sure to follow directions on what to submit!
- Expect to hear from us quickly as to the topic assignments. Vast majority of groups will get their first choice.

Lecture Outline

Regularization wrap-up

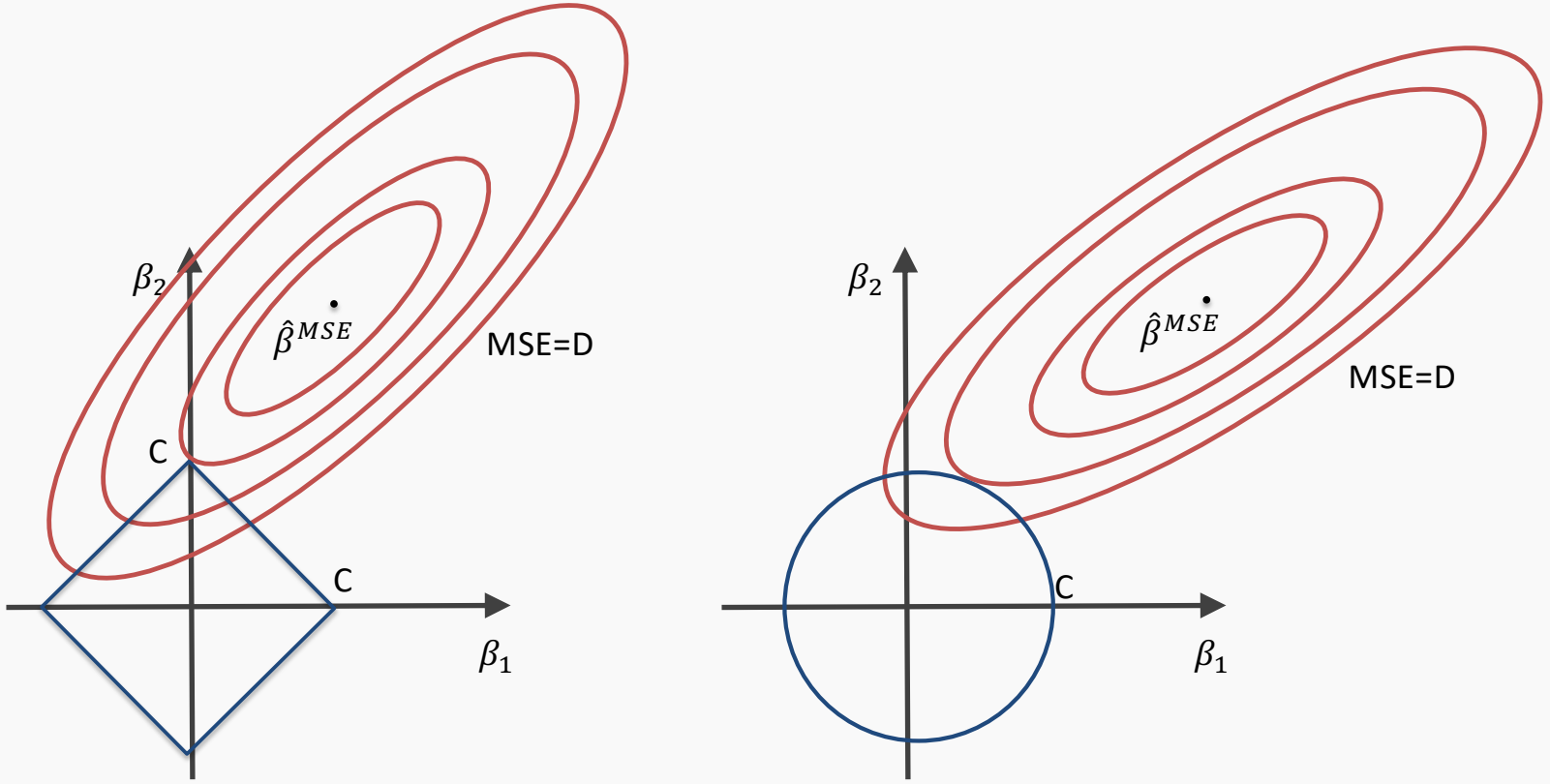
Probabilistic perspective of linear regression

Interaction terms: a brief review

Big Data and High dimensionality

Principle component analysis (PCA)

The Geometry of Regularization



Variable Selection as Regularization

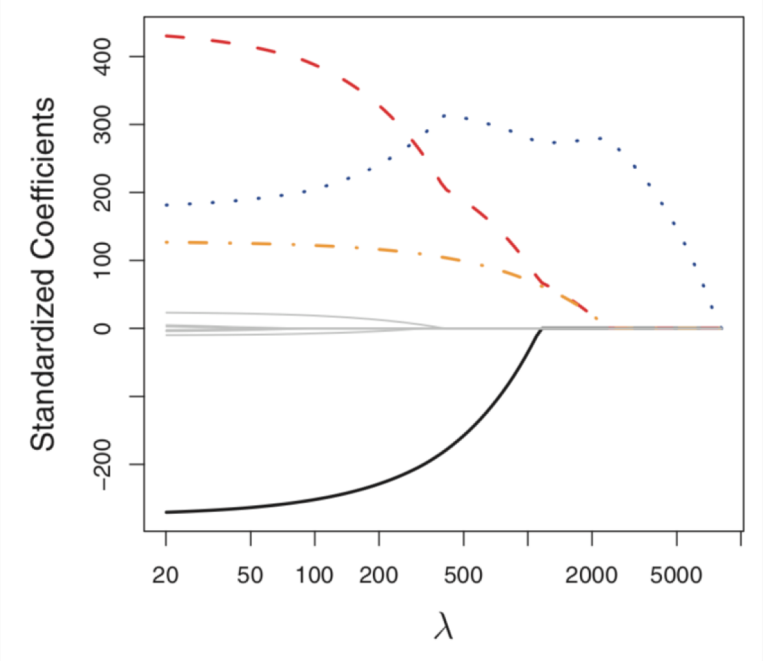
Since LASSO regression tend to produce zero estimates for a number of model parameters - we say that LASSO solutions are **sparse** - we consider LASSO to be a method for variable selection.

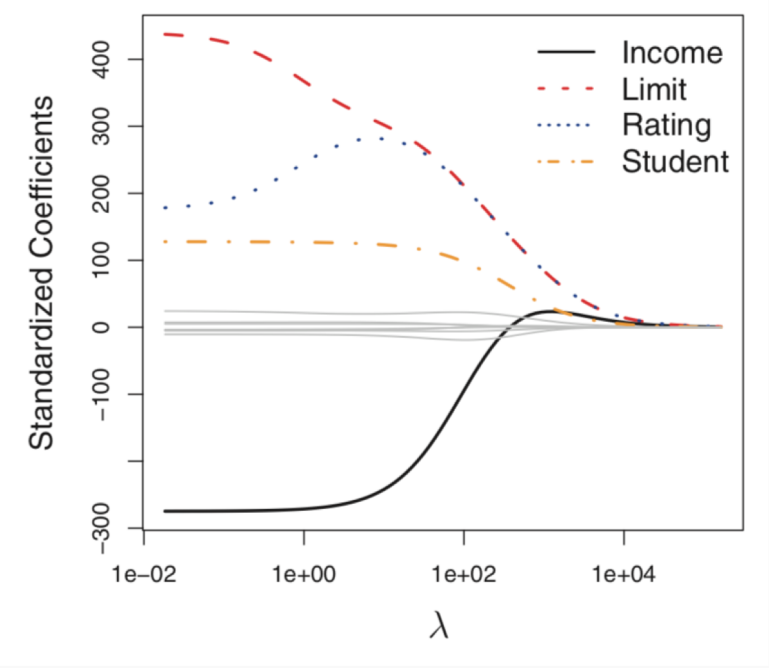
Many prefer using LASSO for variable selection (as well as for suppressing extreme parameter values) rather than stepwise selection, as LASSO avoids the statistic problems that arises in stepwise selection.

Question: What are the pros and cons of the two approaches?

LASSO vs. Ridge: $\hat{\beta}$ estimates as a function of λ

Which is a plot of the LASSO estimates? Which is a plot of the Ridge estimates?





General Guidelines: LASSO vs. Ridge

Regularization methods are great for several reasons. They help:

- Reduce overfitting
- Deal with Multicollinearity
- With Variable Selection

Keep in mind, when sample sizes are large ($n >> 10{,}000$) then regularization might not be needed (unless p is also very large). OLS often does very well when linearity is reasonable and overfitting is not a concern.

When to use each: Ridge generally is used to help deal with multicollinearity, and LASSO is generally used to deal with overfitting. But do them both and CV!

Behind Ordinary Lease Squares, AIC, BIC

Likelihood Functions

Recall that our statistical model for linear regression in matrix notation is:

$$Y = X\beta + \epsilon$$

It is standard to suppose that $\epsilon \sim N(0, \sigma^2)$. In fact, in many analyses we have been making this assumption. Then,

$$y|\beta, x, \epsilon \sim \mathcal{N}(x\beta, \sigma^2)$$

Question: Can you see why?

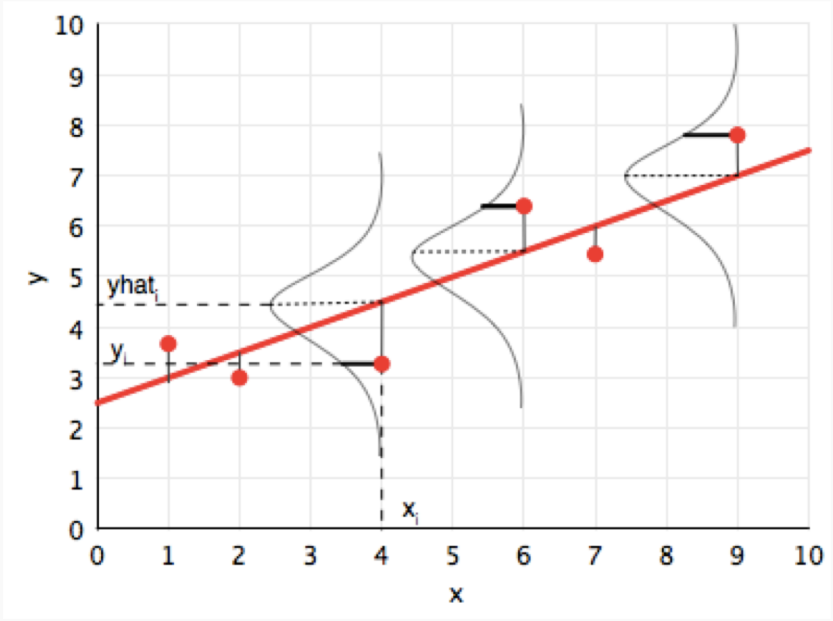
Note that $N(x\beta, \sigma^2)$ is naturally a function of the model parameters β, since the data is fixed.

Likelihood Functions

We call:

$$\mathcal{L}(\beta) = \mathcal{N}(x\beta, \sigma^2)$$

the **likelihood function**, as it gives the likelihood of the observed data for a chosen model $\boldsymbol{\beta}$.



Maximum Likelihood Estimators

Once we have a likelihood function, $\mathcal{L}(\boldsymbol{\beta})$, we have strong incentive to seek values of to maximize $\mathcal{L}$.

Can you see why?

The model parameters that maximizes $\mathcal{L}$ are called **maximum likelihood estimators (MLE)** and are denoted:

$$\boldsymbol{\beta}_{\text{MLE}} = \underset{\boldsymbol{\beta}}{\text{argmax}}\, \mathcal{L}(\boldsymbol{\beta})$$

The model constructed with MLE parameters assigns the highest likelihood to the observed data.

Maximum Likelihood Estimators

But how does one maximize a likelihood function?

Fix a set of *n* observations of *J* predictors, $\boldsymbol{X}$, and a set of corresponding response values, $\boldsymbol{Y}$; consider a linear model $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \epsilon$.

If we assume that $\epsilon \sim N(0, \sigma^2)$ then the likelihood for each observation is

$$\mathcal{L}_i(\boldsymbol{\beta}) = \mathcal{N}(y_i; \boldsymbol{\beta}^\top \boldsymbol{x}_i, \sigma^2)$$

and the likelihood for the entire set of data is

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} \mathcal{N}(y_i; \boldsymbol{\beta}^\top \boldsymbol{x}_i, \sigma^2)$$

Maximum Likelihood Estimators

Through some algebra, we can show that maximizing $\mathcal{L}(\boldsymbol{\beta})$, is equivalent to minimizing MSE:

$$\boldsymbol{\beta}_{MLE} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}}\, \boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, MSE$$

Minimizing MSE or RSS is called **ordinary least squares**.

Using Interaction Terms

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 tax or an Uber, X_2 is the times of day of the pickup and Y is the length of the ride.

What is the interpretation of β_3?

Including Interaction Terms in Models

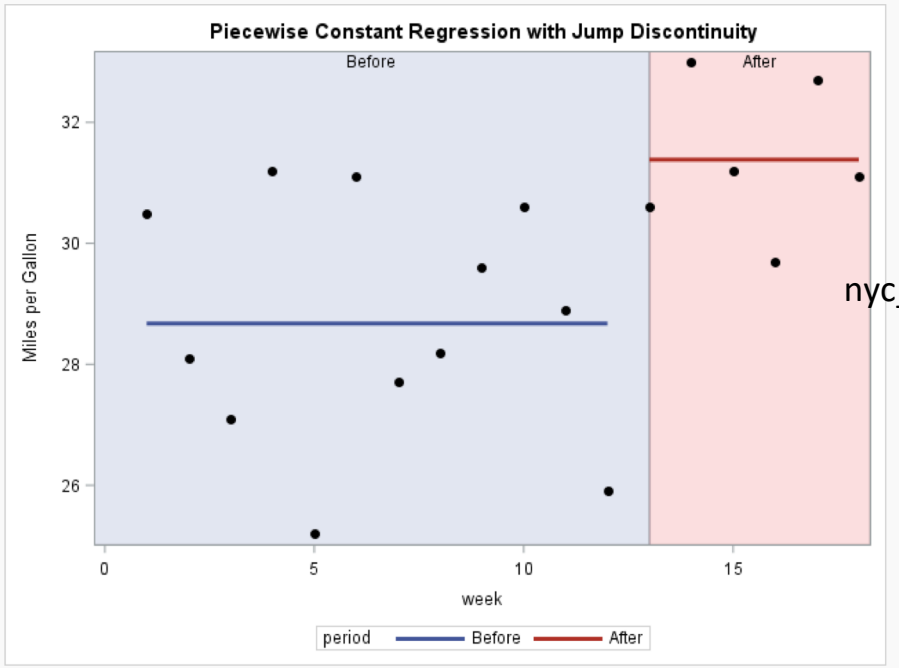
Recall that to avoid overfitting, we sometimes elect to exclude a number of terms in a linear model.

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.

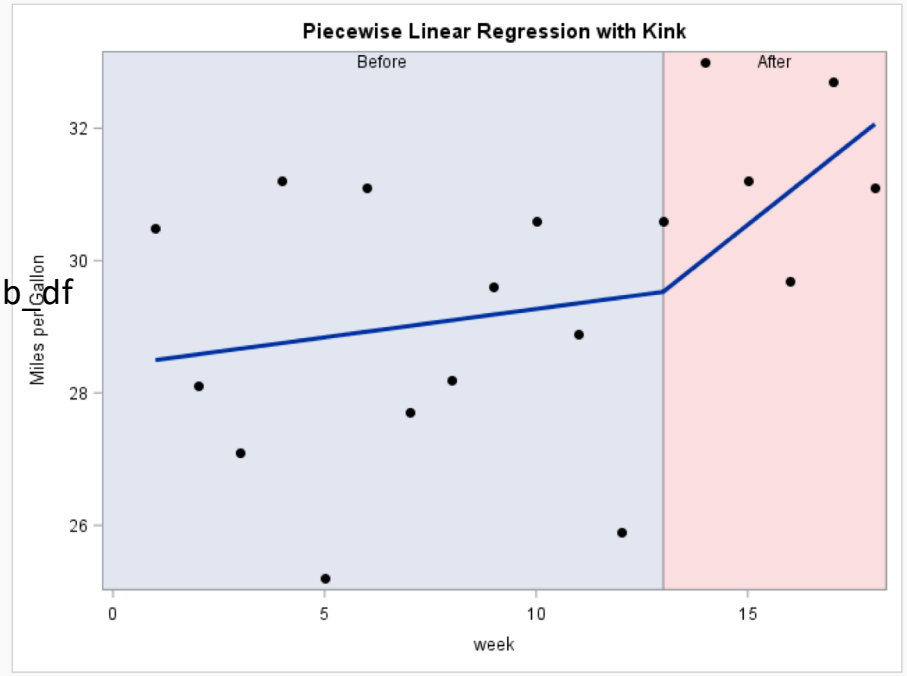
Question: Why are the ***main effects*** important?

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?



nyc_cab_df



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

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.

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 $m < p$ 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

$$Y = \beta_0 + \beta_1 Z_1 + \ldots + \beta_m Z_m + \epsilon.$$

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.

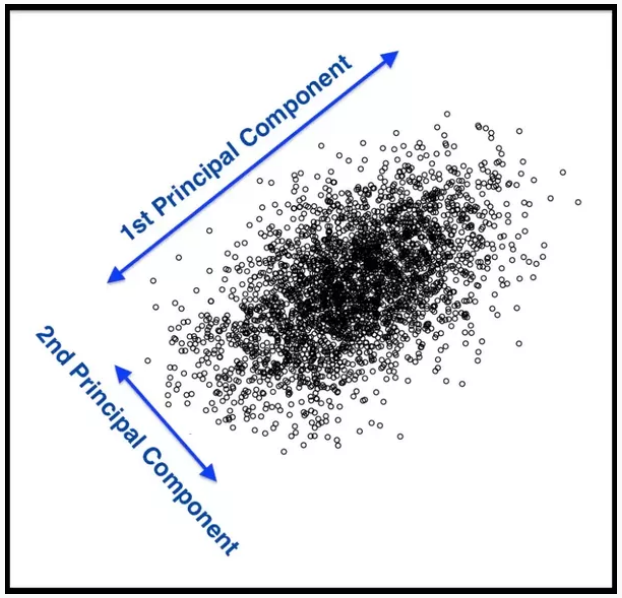
Thereafter, we can fit a model using the new predictors.

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 variations in the observed data.

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.

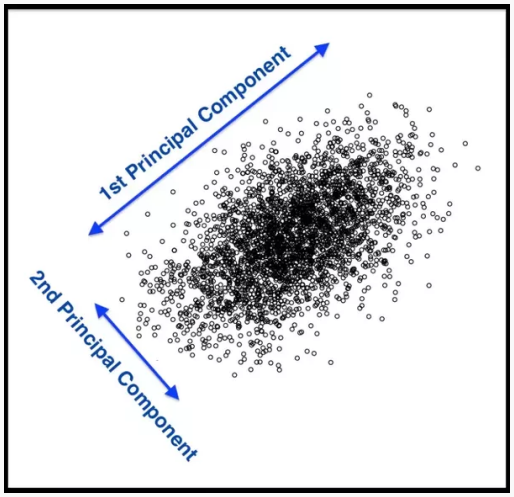
That is, the observed data shows more variance in the direction of Z_1 than in the direction of Z_2.

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

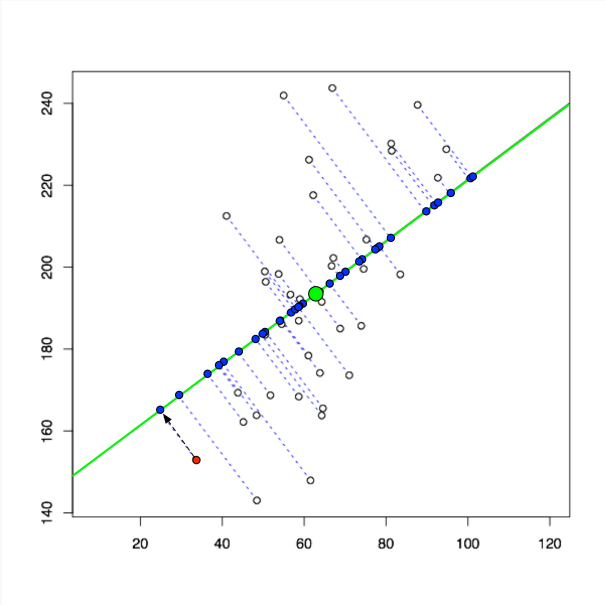
Top PCA components capture the most of amount of variation (interesting features) of the data.

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

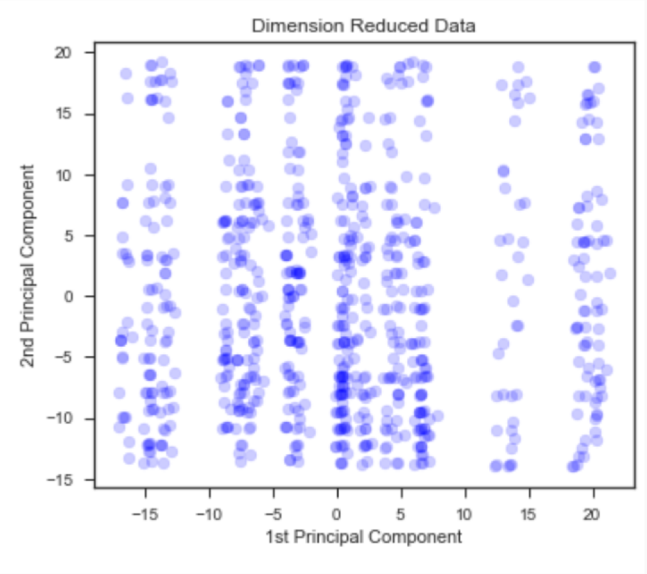
In [24]:
```
pca_all = PCA()
pca_all.fit(X_non_test)

print('First 4 principal components:\n', pca_all.components_[0:4])
print('Explained variance ratio:\n', pca_all.explained_variance_ratio_)
```

```
First 4 principal components:
 [[  2.39129056e-02  -1.52589992e-03   2.87820181e-03   8.15273795e-01
    5.30856898e-01  -1.66247307e-01  -1.57138583e-01   1.63510006e-02
   -9.18124937e-03  -0.00000000e+00   0.00000000e+00  -1.59625033e-02]
 [ -4.81556857e-02  -1.64875776e-03   8.71237863e-03  -1.95383366e-01
   -1.16363766e-01  -6.82620696e-01  -6.92730264e-01  -2.87939549e-03
    2.04933364e-03  -0.00000000e+00  -0.00000000e+00   5.33823735e-03]
 [  9.62900731e-01  -5.74858553e-03   2.52808711e-01  -2.66439844e-02
   -2.31060283e-02  -4.15207358e-02  -1.18366711e-02   6.63443885e-02
   -3.68951248e-02  -0.00000000e+00  -0.00000000e+00  -2.48879974e-03]
 [ -4.24750484e-03  -1.28064076e-03  -8.79072218e-03   5.43141125e-01
   -8.38916835e-01   2.59209473e-04  -1.26252332e-02   1.35518981e-02
   -7.80620455e-03  -0.00000000e+00  -0.00000000e+00  -2.67441606e-02]]
Explained variance ratio: [  3.52498964e-01   3.12127677e-01   2.40830333e-01   4.55437699e-02
   3.29216180e-02   7.43824785e-03   4.94773071e-03   2.95866750e-03
   7.23712321e-04   9.27988146e-06   0.00000000e+00   0.00000000e+00]
```

PCA example in sklearn

A common plot is to look at the scatterplot of the first two principal components, shown below for the NYC Taxi 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!

*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?

Train, Test, and Cross Validation!!!

A few notes on using PCA

- PCA is an unsupervised algorithm. Meaning? It is done independent of the outcome variable.
- PCA is not so good because:
    1. 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 (more to come in Module 2).
    3. Reducing multicollinearity, and thus may improve the computational time of fitting models.

A few notes on using PCA

- PCA is an unsupervised algorithm. Meaning? It is done independent of the outcome variable.
- PCA is not so good because:
    1. 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 (more to come in Module 2).
    3. Reducing multicollinearity, and thus may improve the computational time of fitting models.