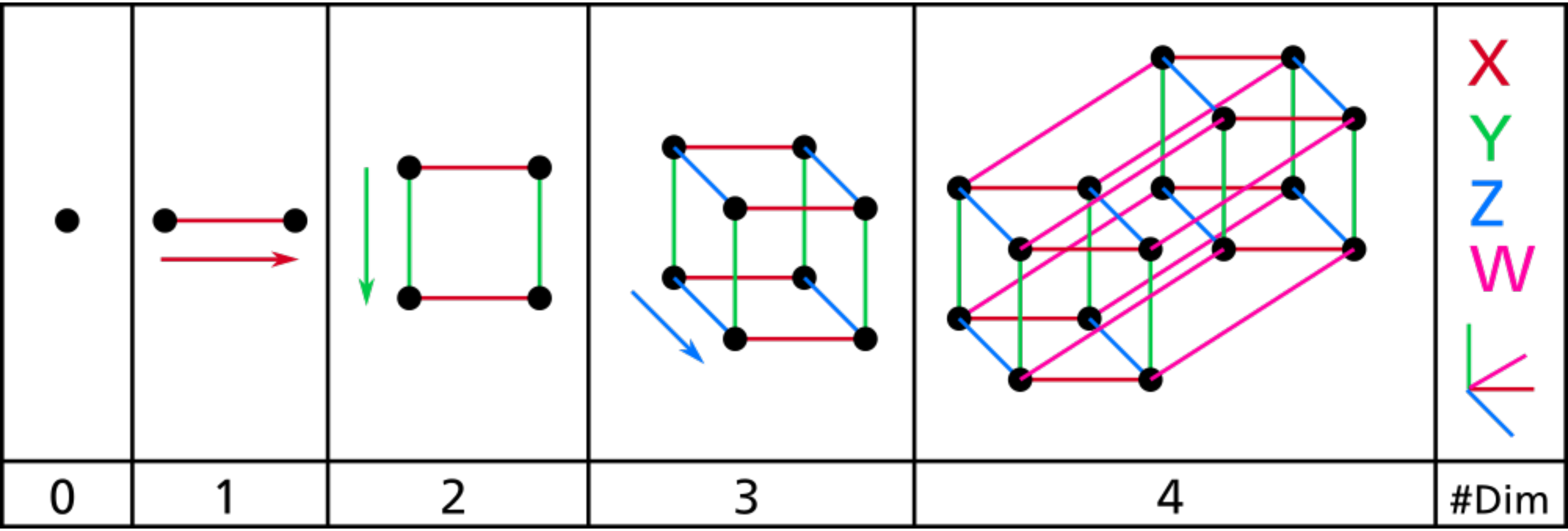
Feature Selection.
Feature Extraction.
Dimensionality Reduction.

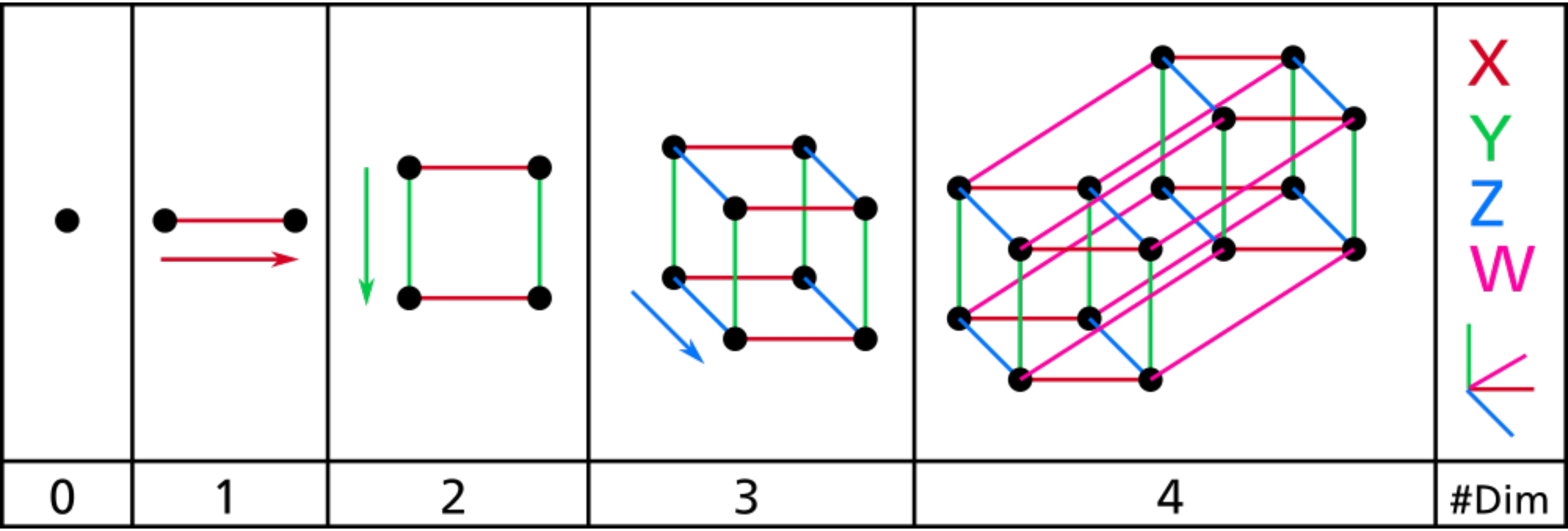
METIS

datascope

Curse of Dimensionality

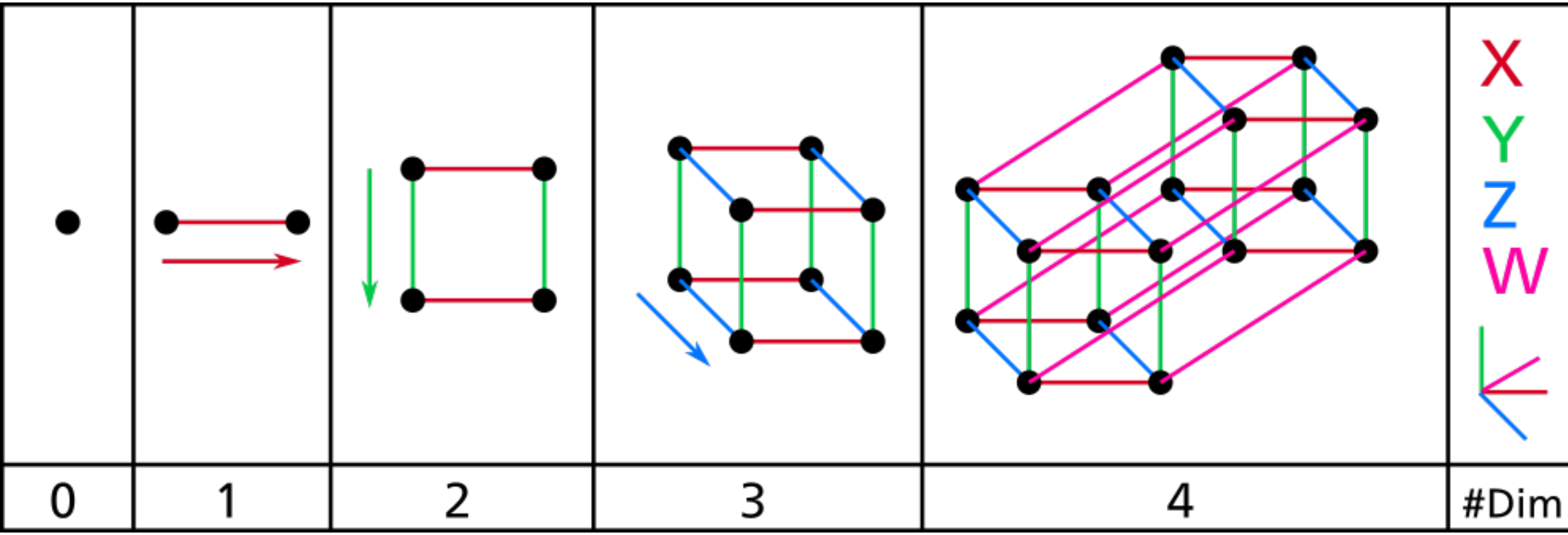


Curse of Dimensionality

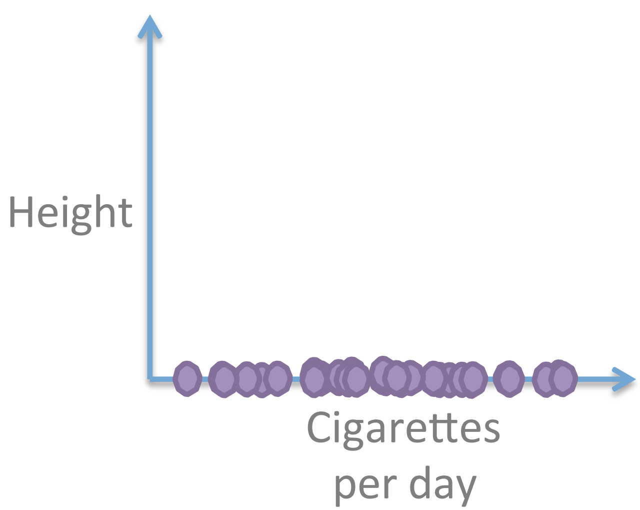


One dimension:
Small space
Being close quite probable

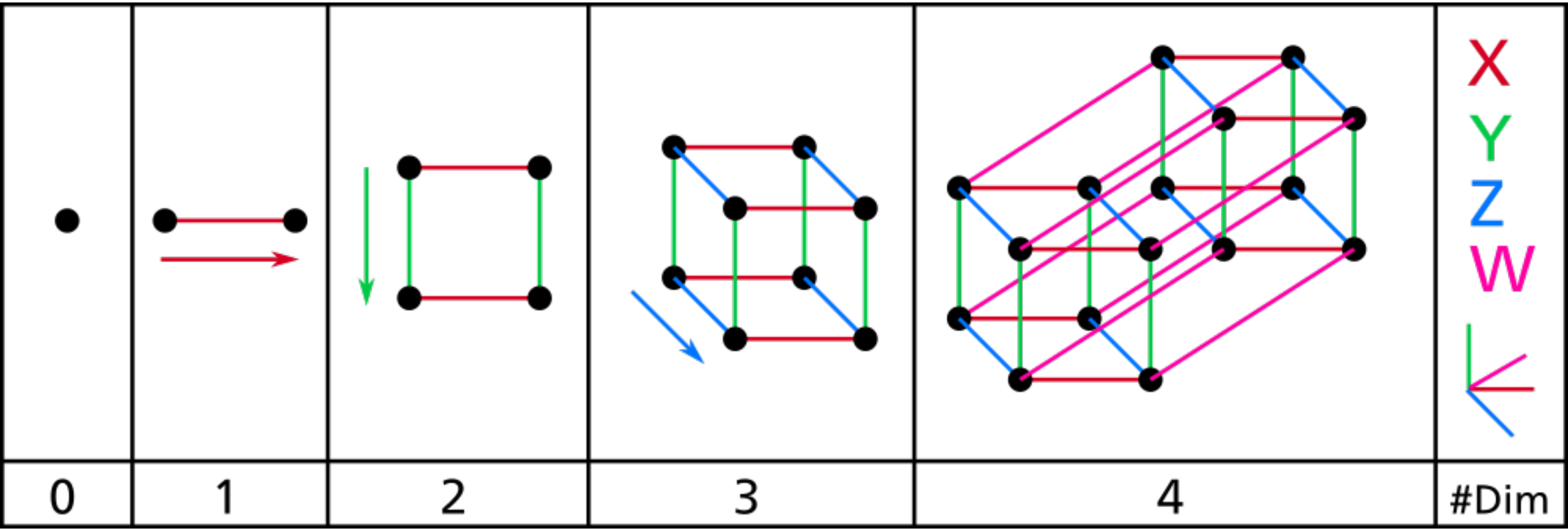
Curse of Dimensionality

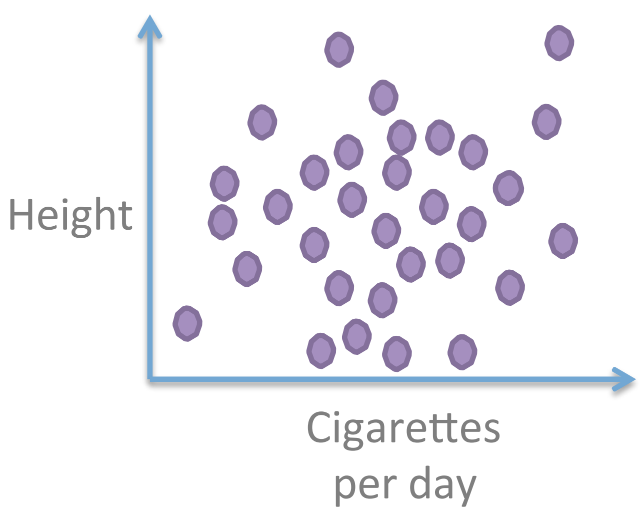


Two dimensions



Curse of Dimensionality



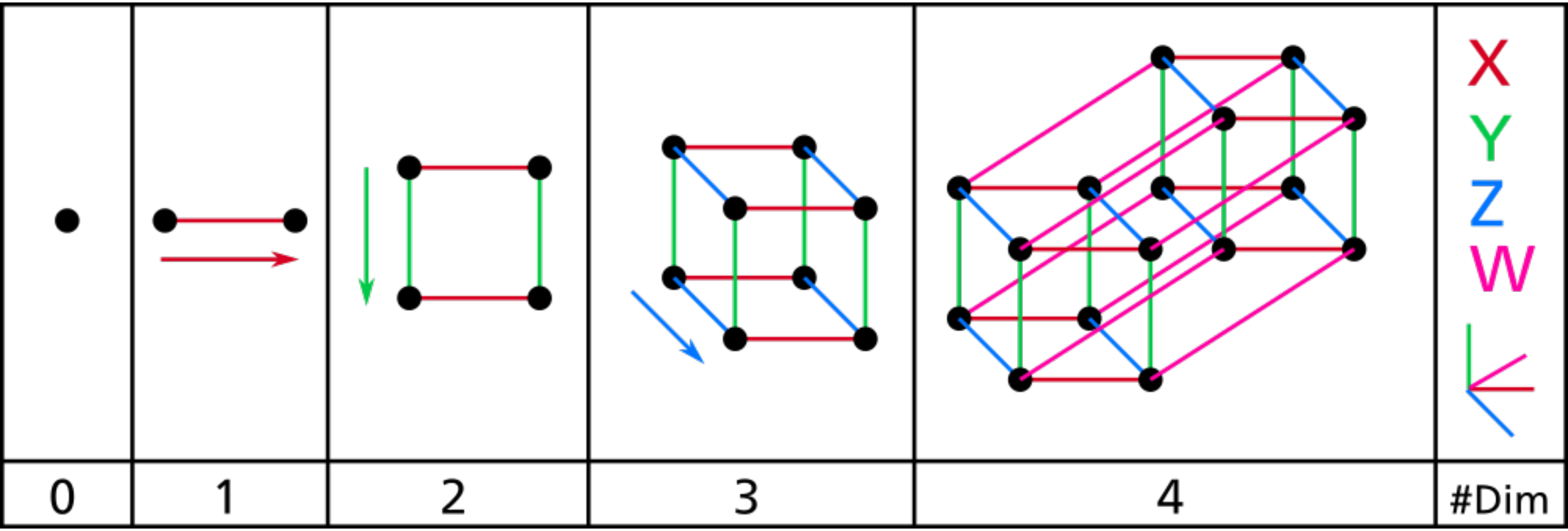


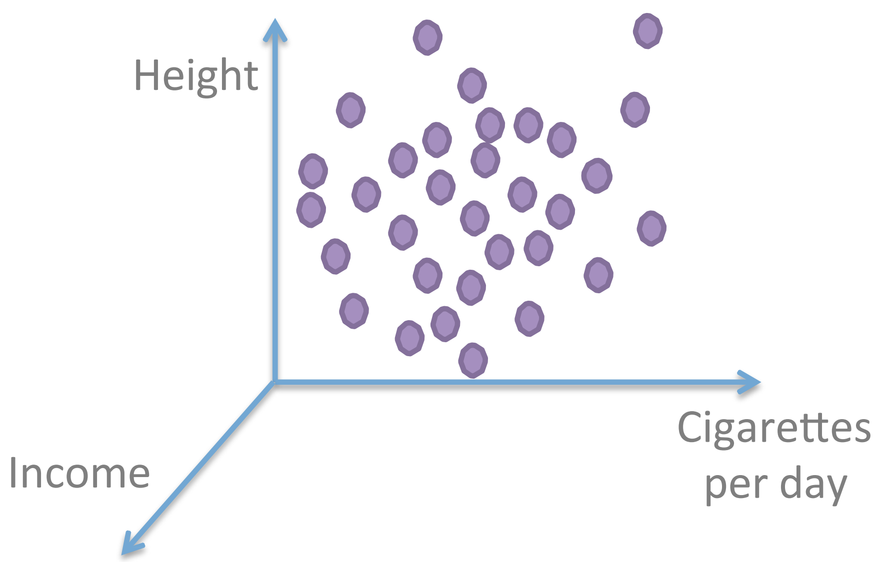
Two dimensions:

More space but still not so much

Being close not improbable

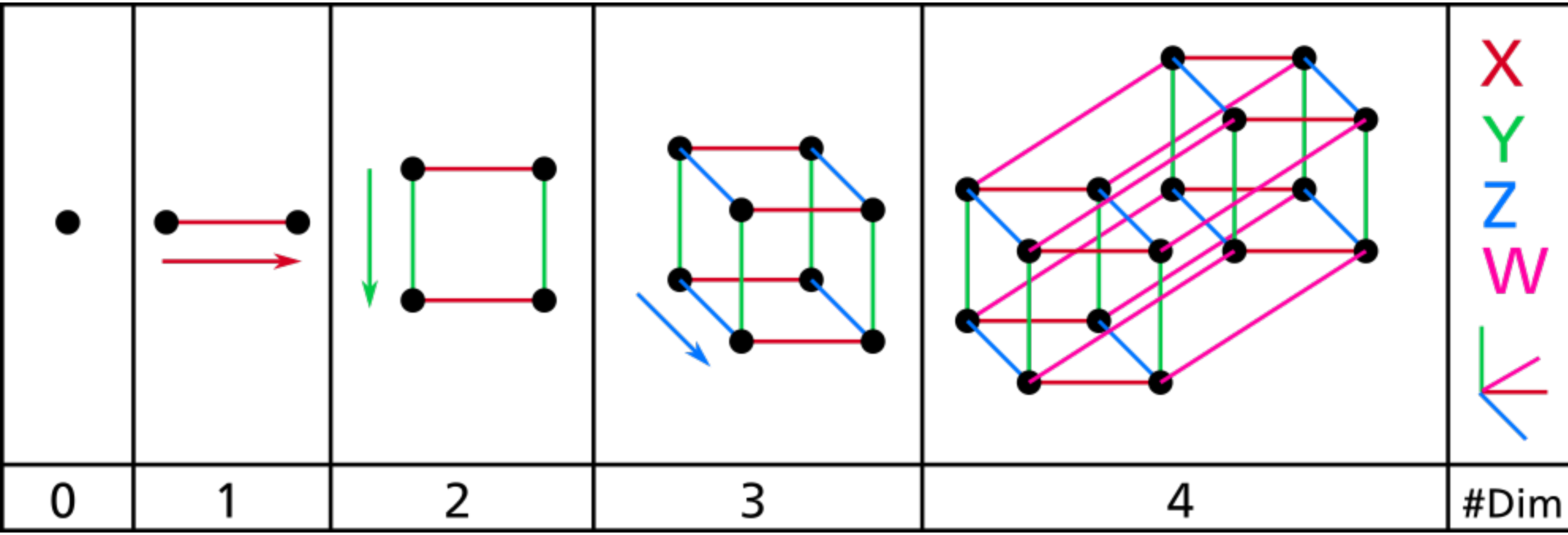
Curse of Dimensionality

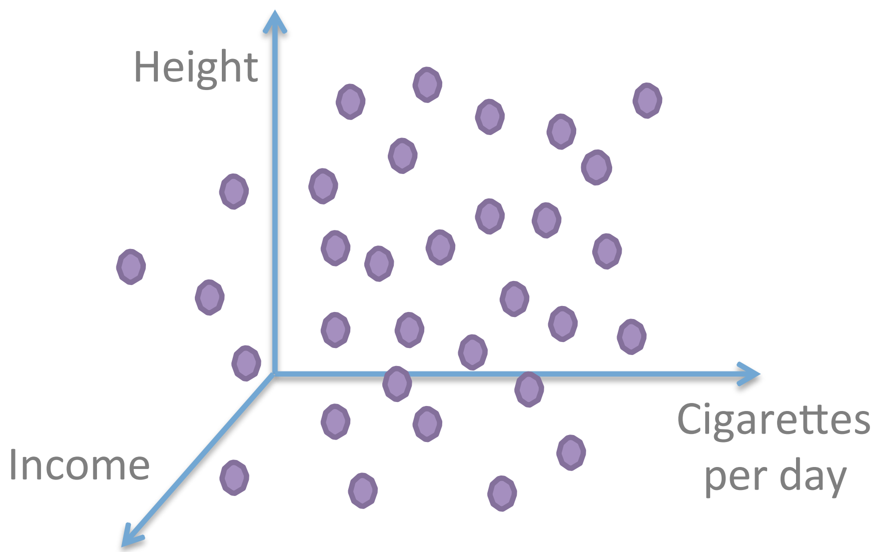




Three dimensions

Curse of Dimensionality



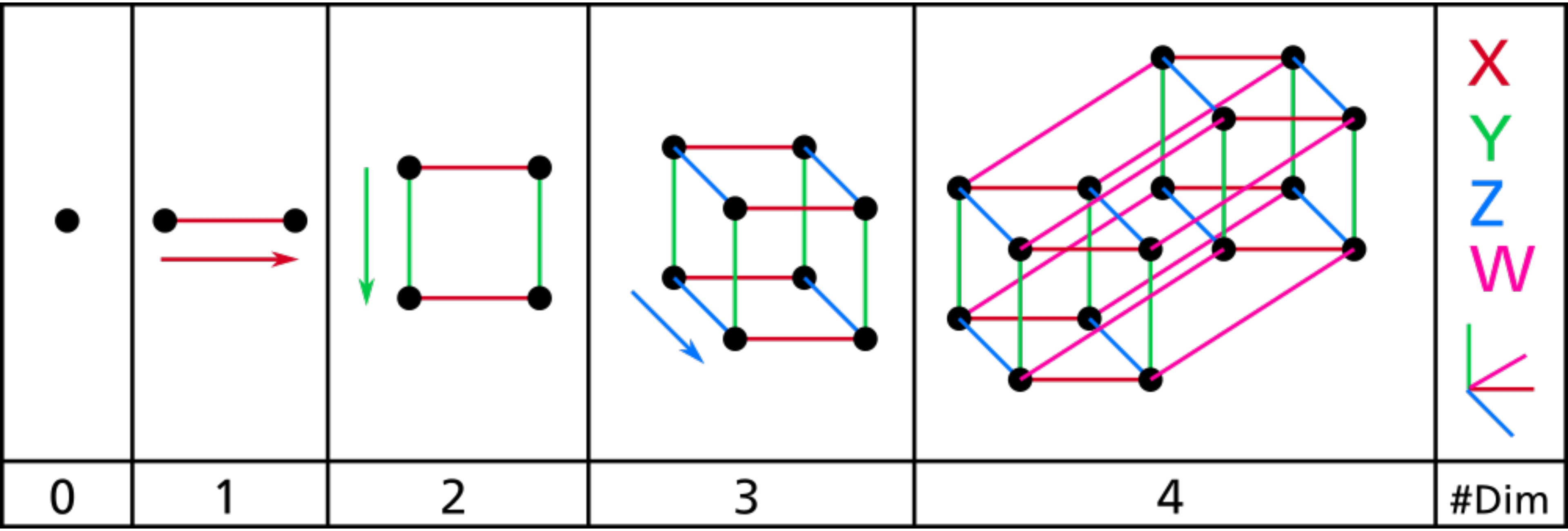


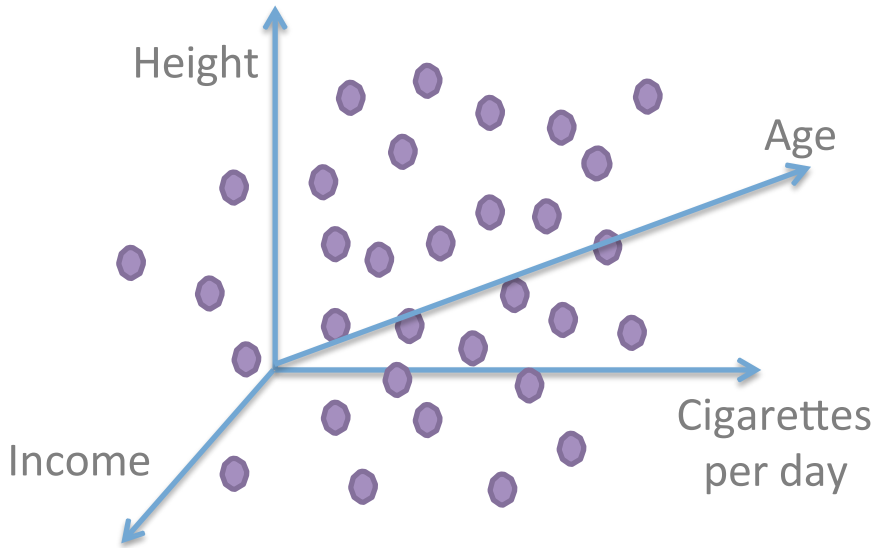
Three dimensions:

Much larger space

Being close less probable

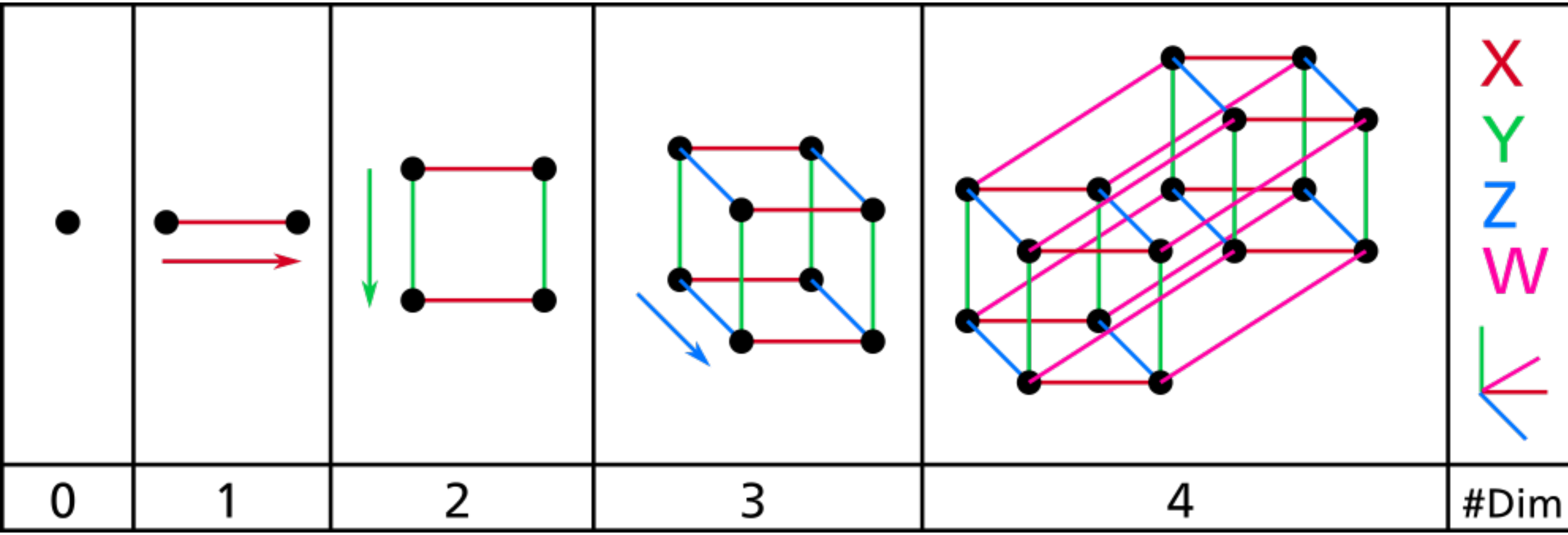
Curse of Dimensionality

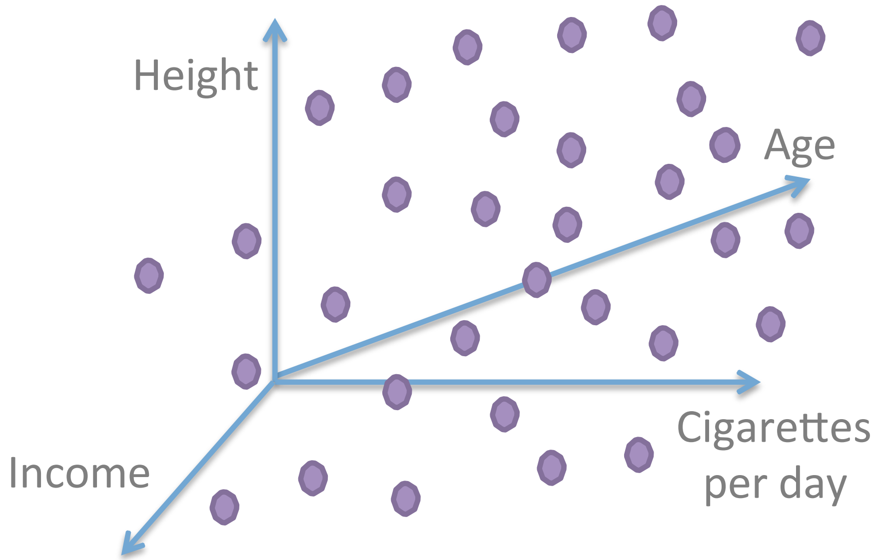




Four dimensions

Curse of Dimensionality



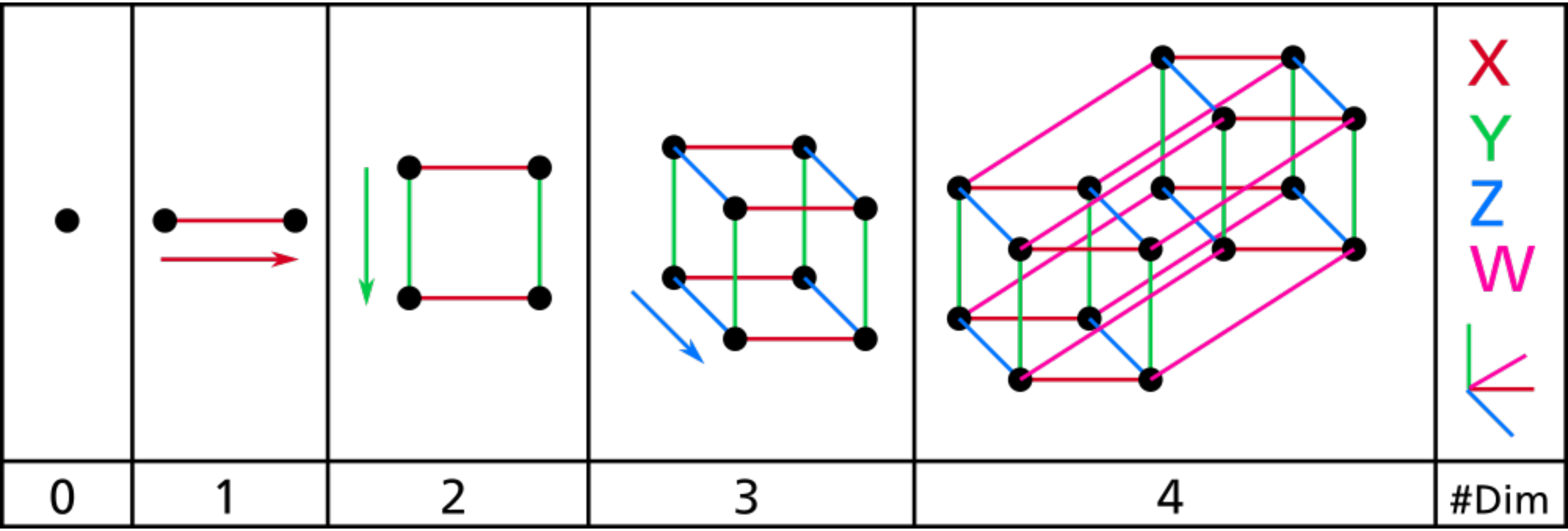


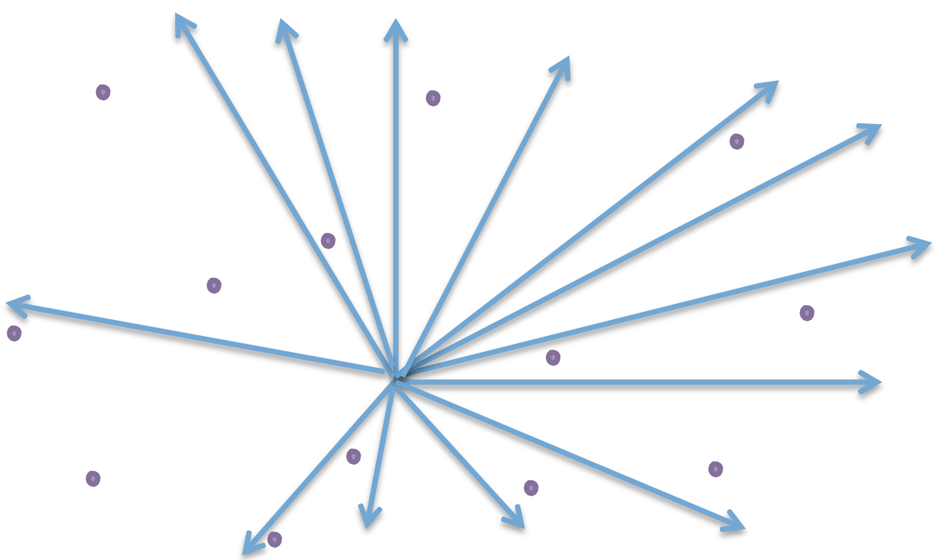
Four dimensions:

Omg so much space

Being close quite improbable

Curse of Dimensionality





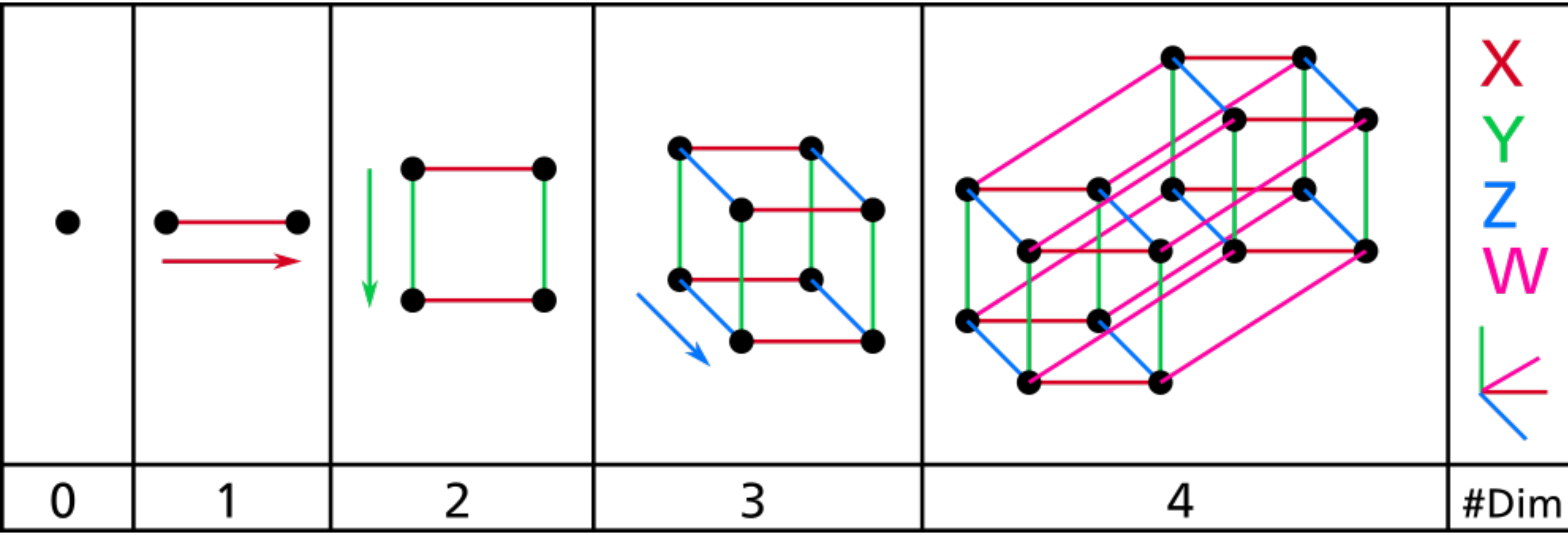
Thousand dimensions:

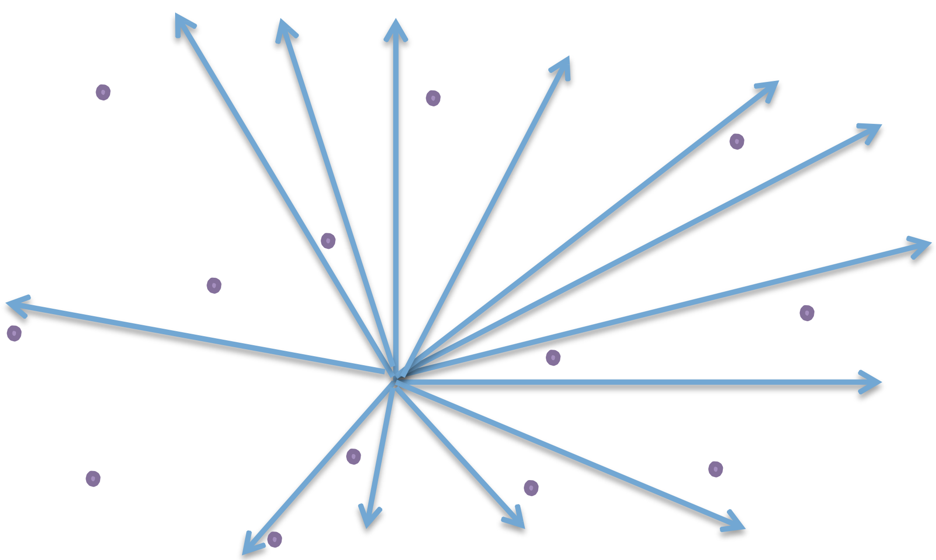
Helloooo... hellooo.. helloo...

Can anybody hear meee.. mee.. mee.. mee..

So alone....

Curse of Dimensionality

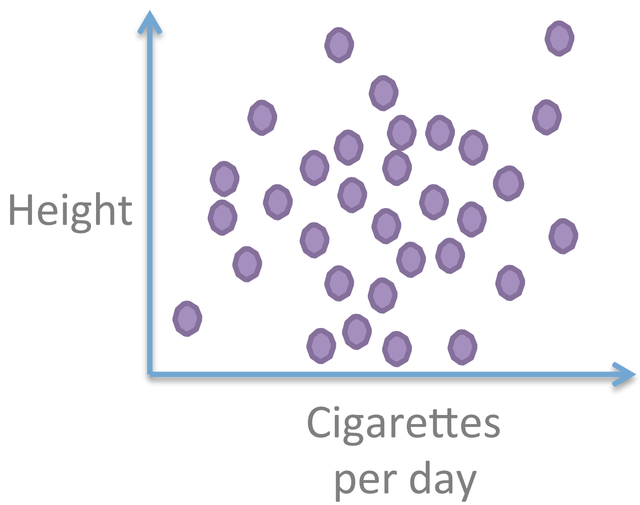


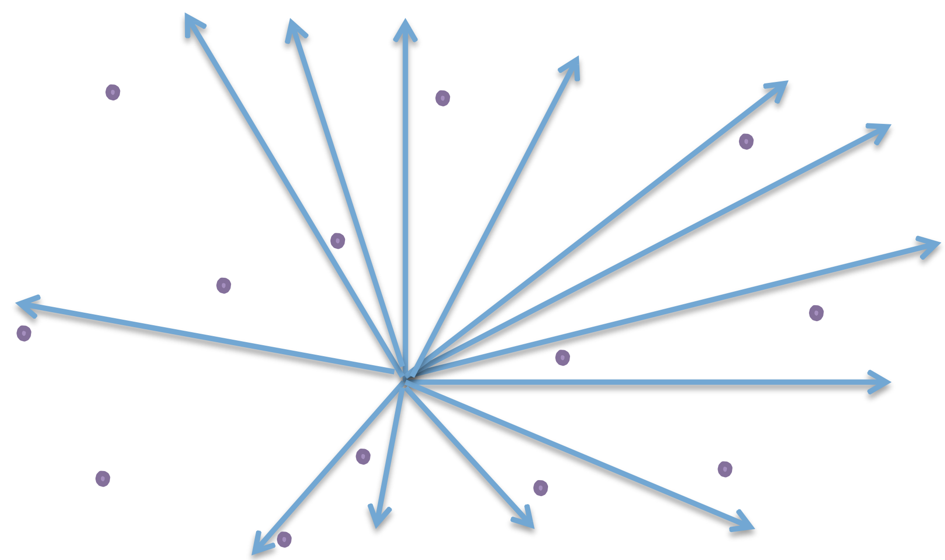


Thousand dimensions:
I specified you with such high resolution, with so much detail, that you don’t look like anybody else anymore. You’re unique.

Curse of Dimensionality

Classification, clustering and other analysis methods become exponentially difficult with increasing dimensions.

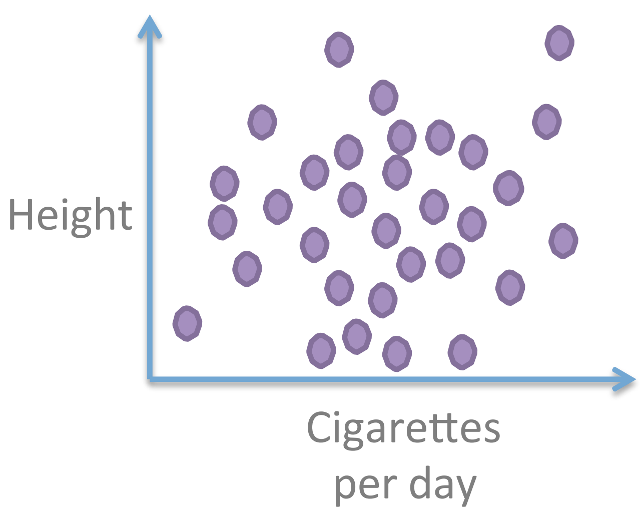


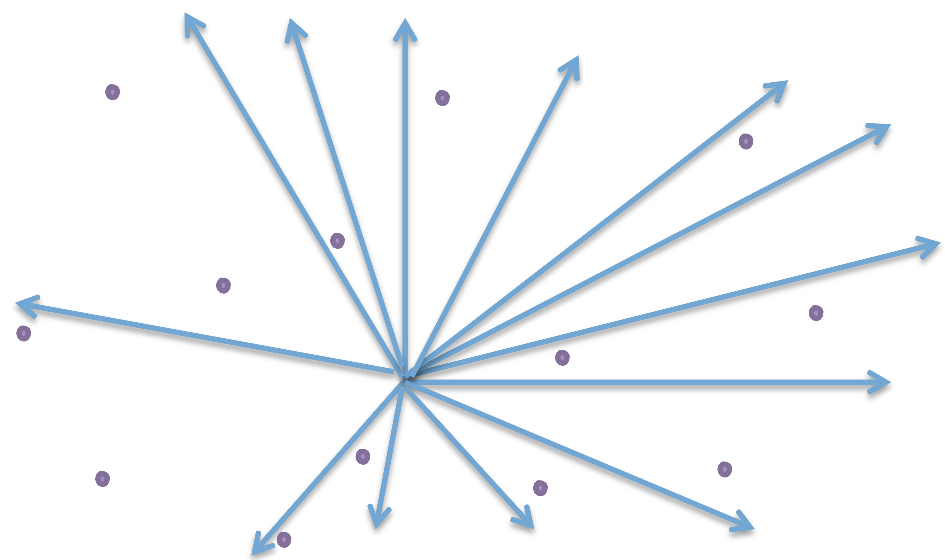


Curse of Dimensionality

Classification, clustering and other analysis methods become exponentially difficult with increasing dimensions.

To understand how to divide that huge space, we need a whole lot more data (usually much more than we do or can have).



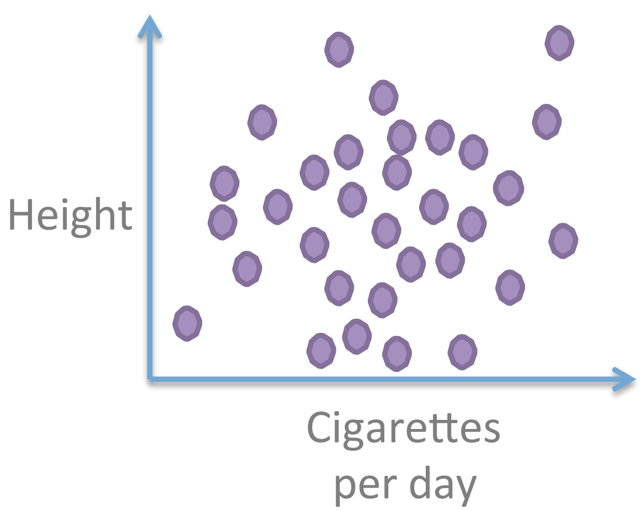


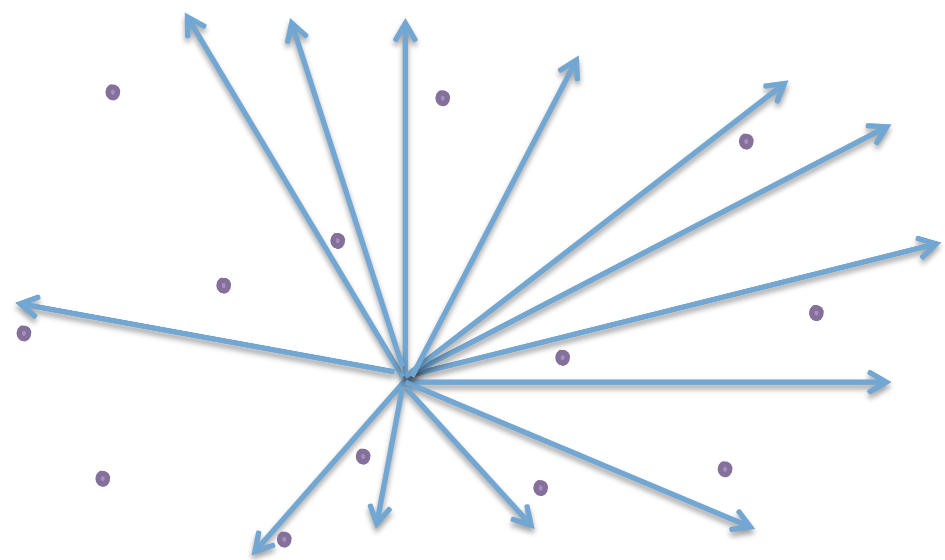
Dimensionality Reduction

Lots of features, lots of data is best. But what if you don't have the luxury of ginormous amounts of data?

Not all features provide the same amount of information.

We can reduce the dimensions (compress the data) without necessarily losing too much information.





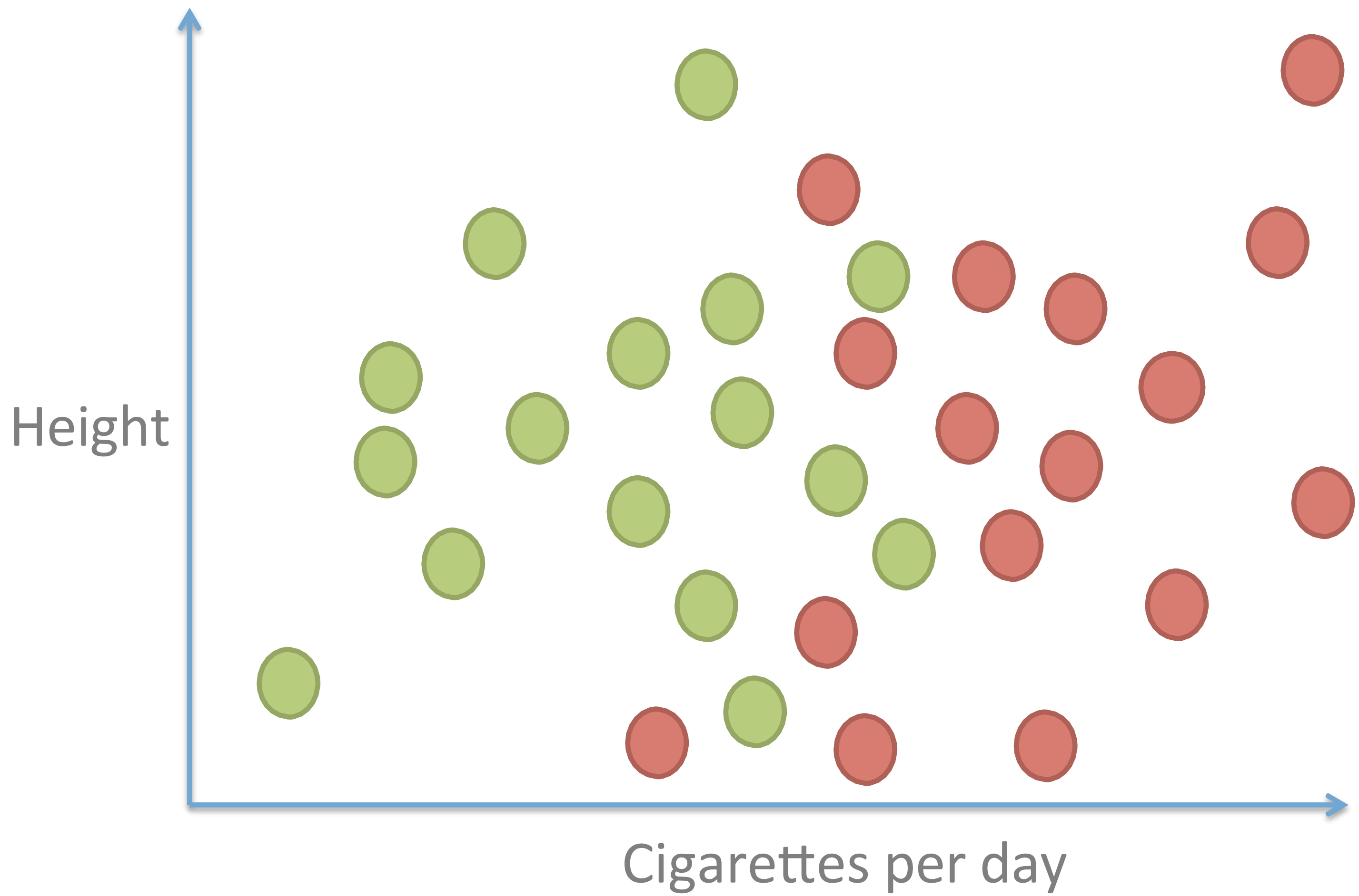
Feature Selection

Feature Selection

Dropping some features loses information,
but gains more on the curse of dim.

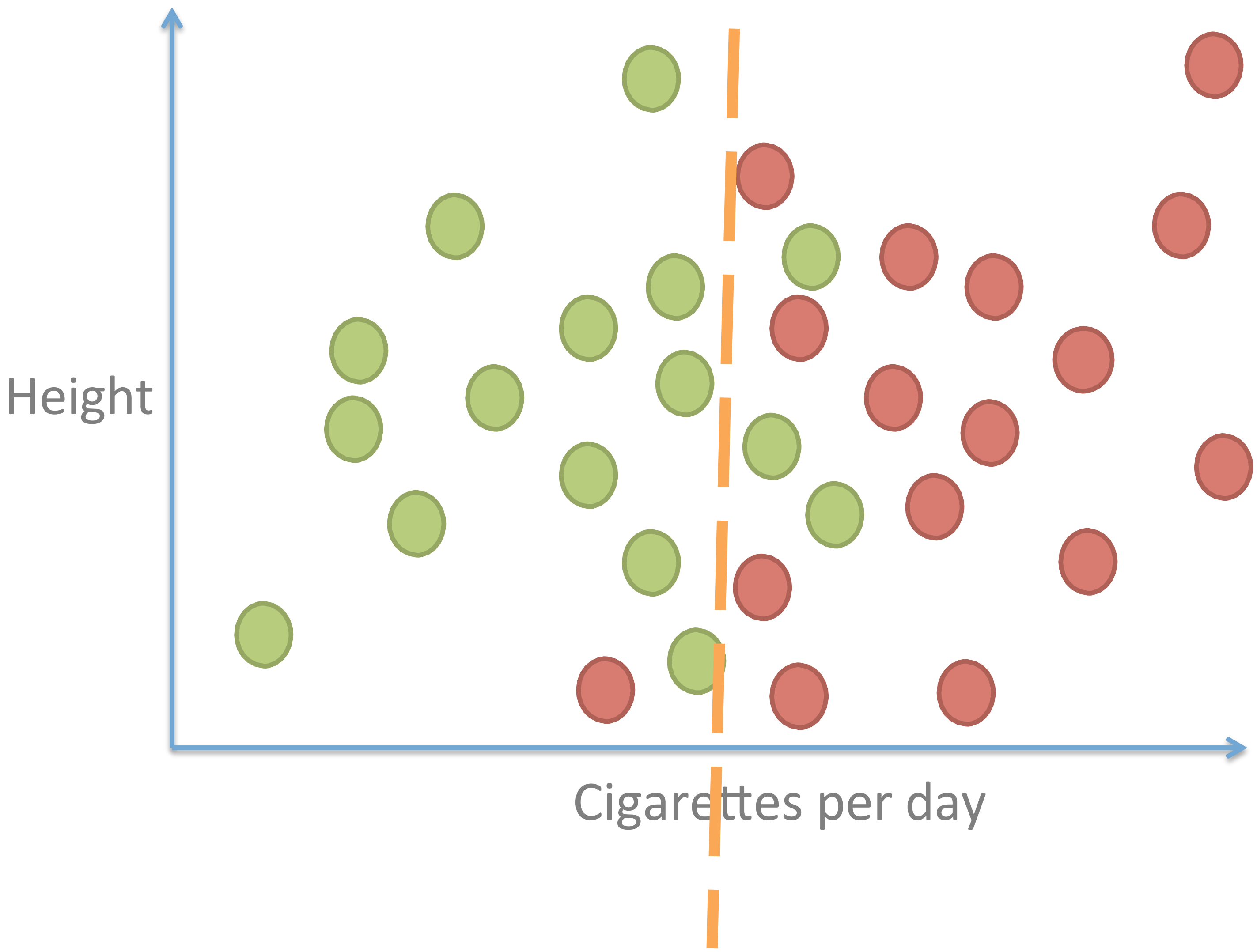
Feature Selection

Healthy / Heart Disease



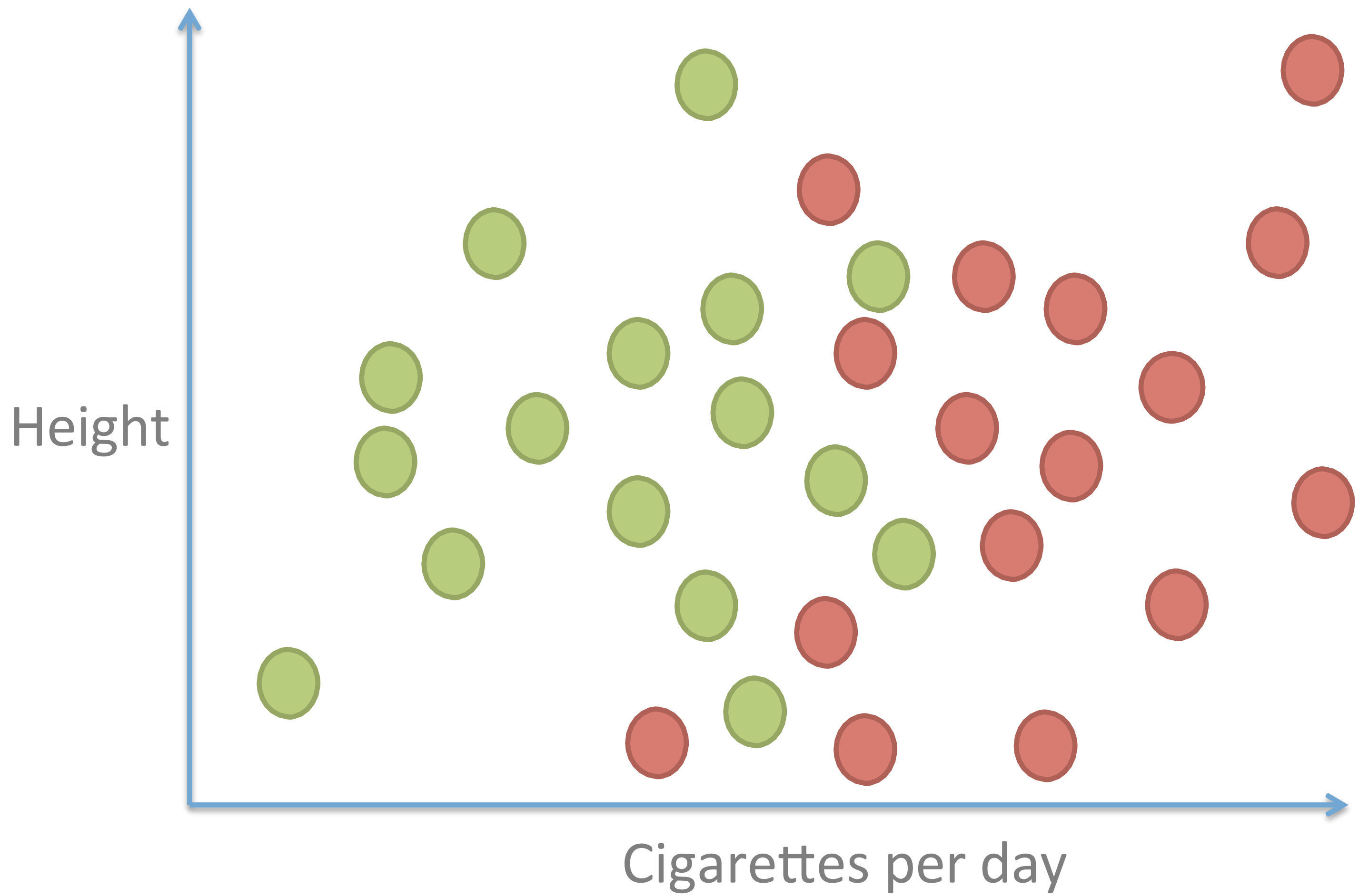
Feature Selection

Healthy / Heart Disease



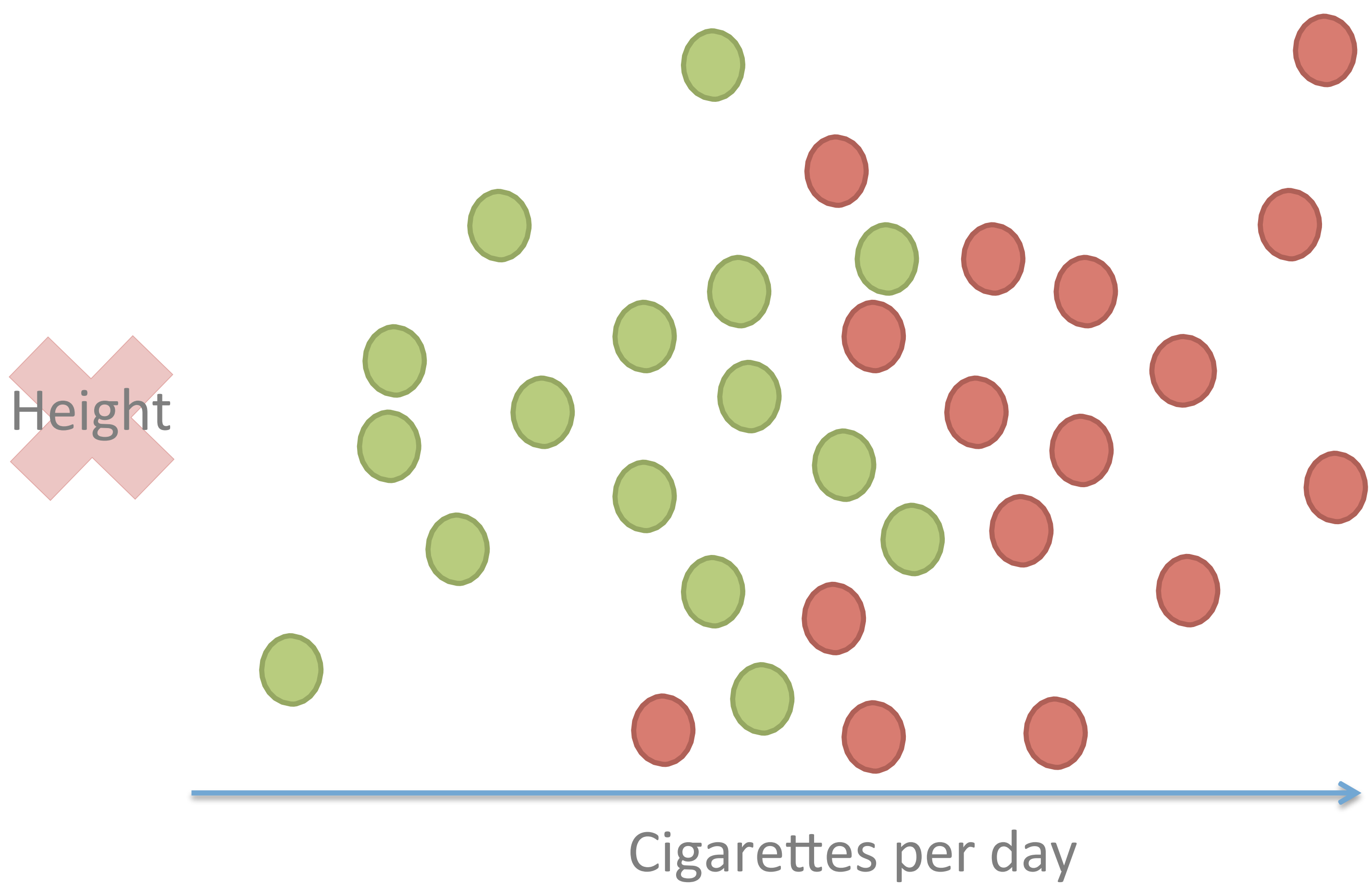
Feature Selection

Healthy / Heart Disease



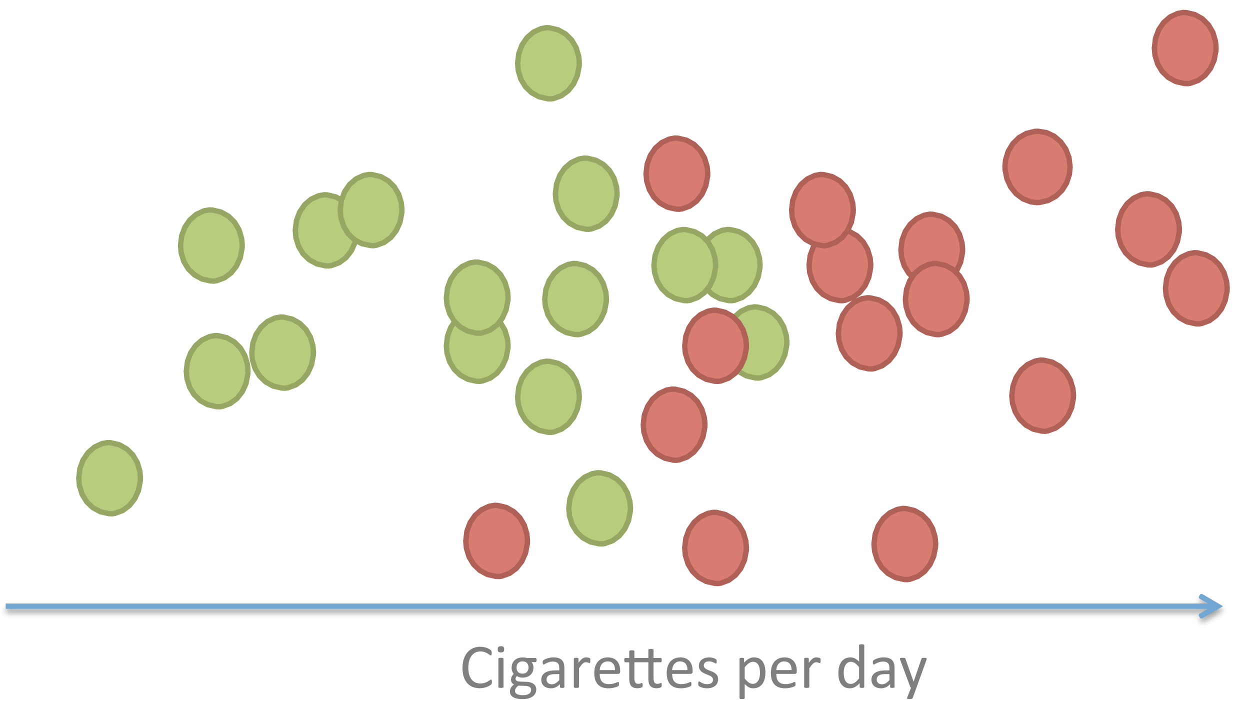
Feature Selection

Healthy / Heart Disease



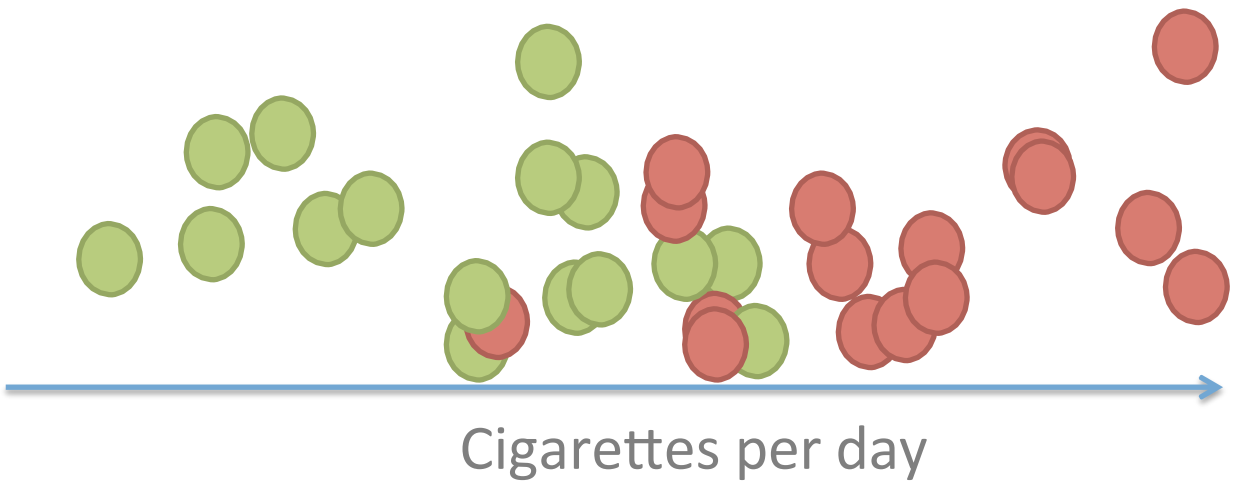
Feature Selection

Healthy / Heart Disease



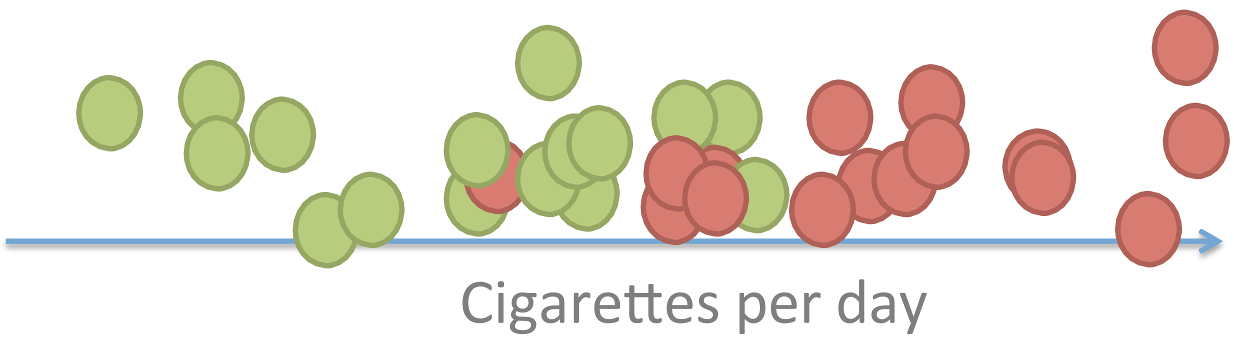
Feature Selection

Healthy / Heart Disease



Feature Selection

Healthy / Heart Disease

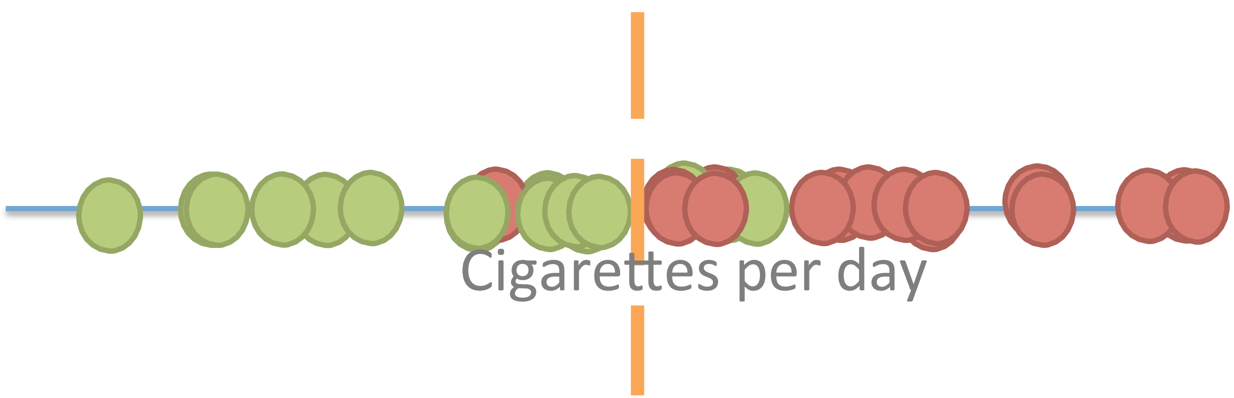


Feature Selection

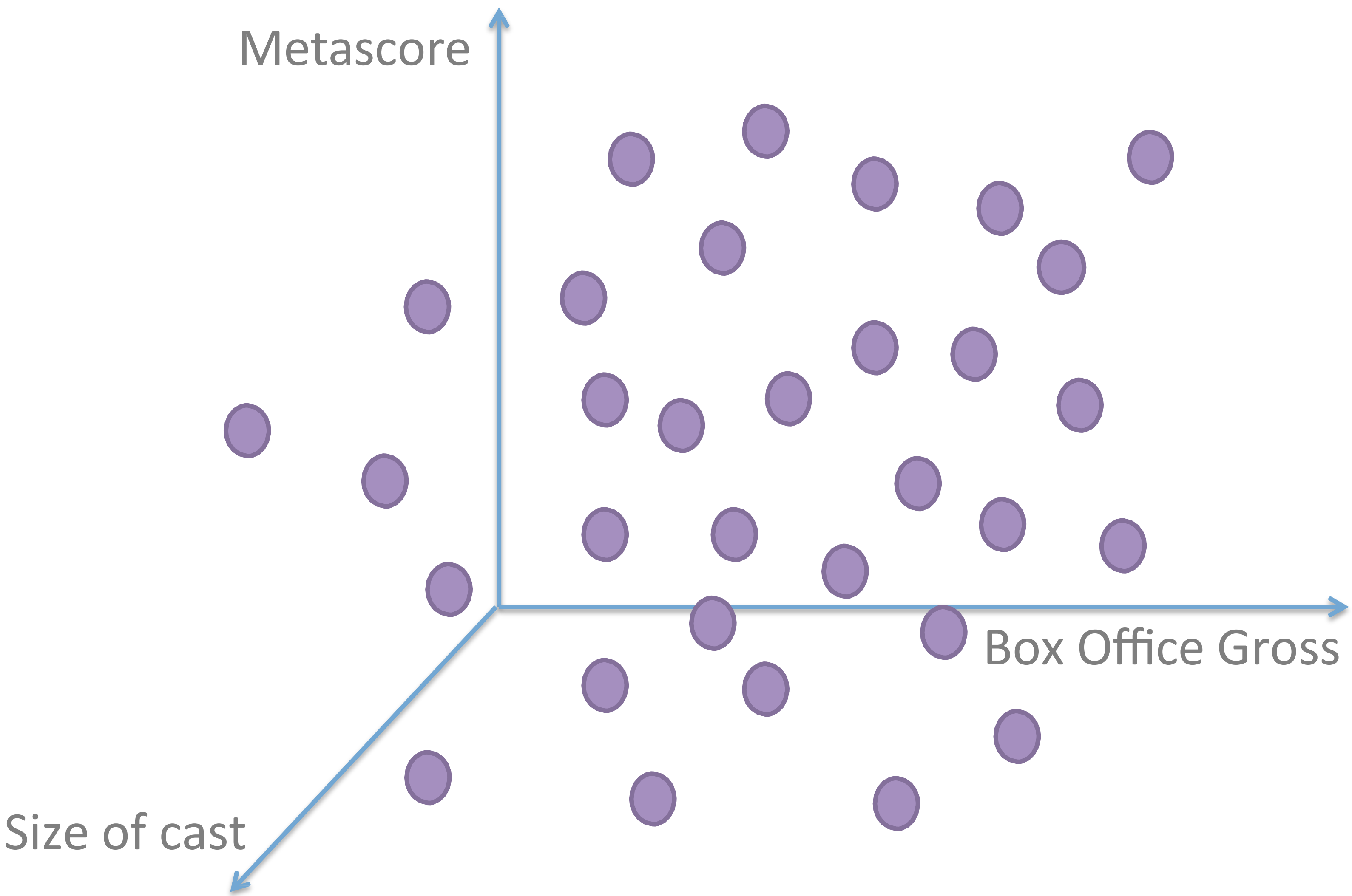
Healthy / Heart Disease

Feature Selection

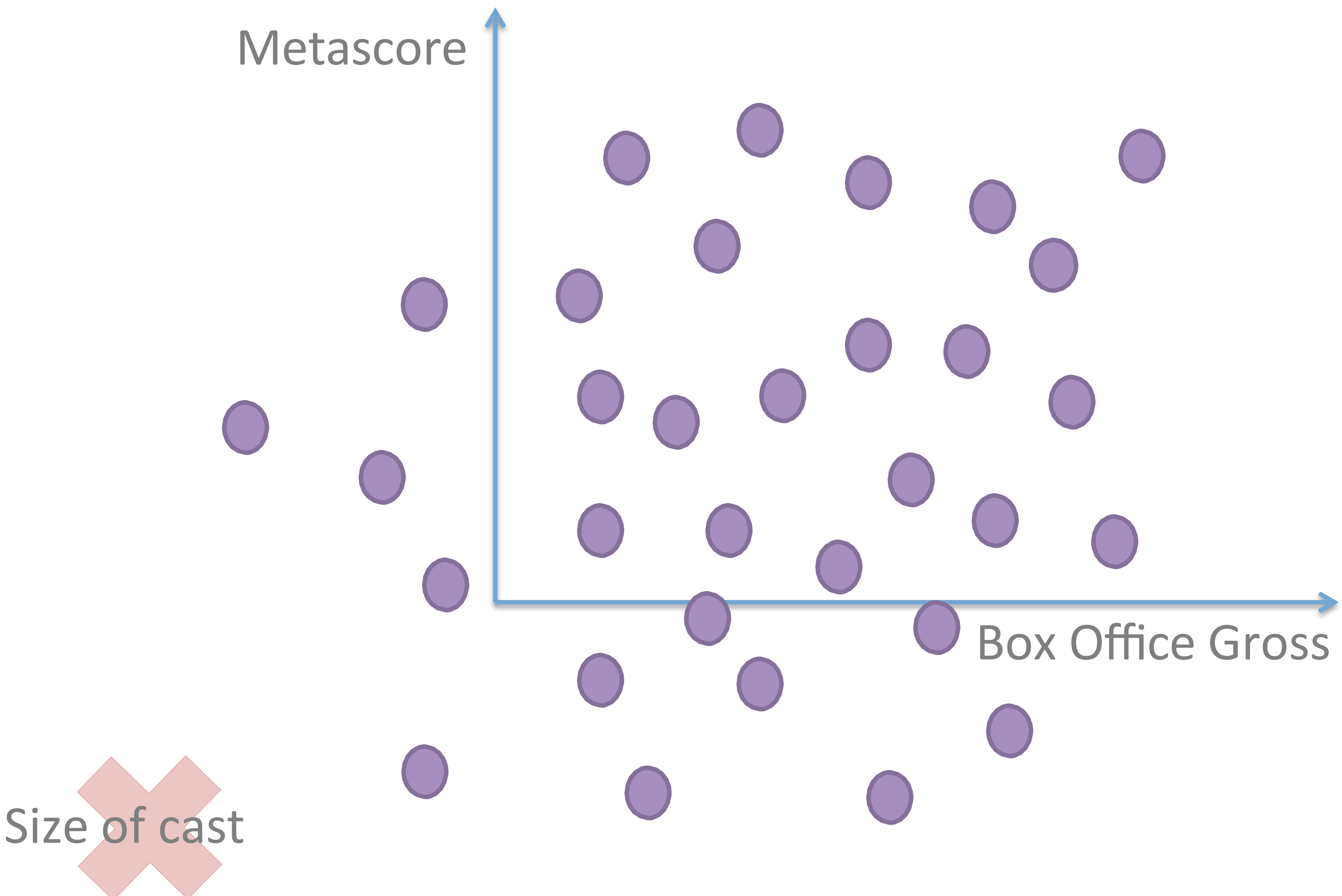
Healthy / Heart Disease



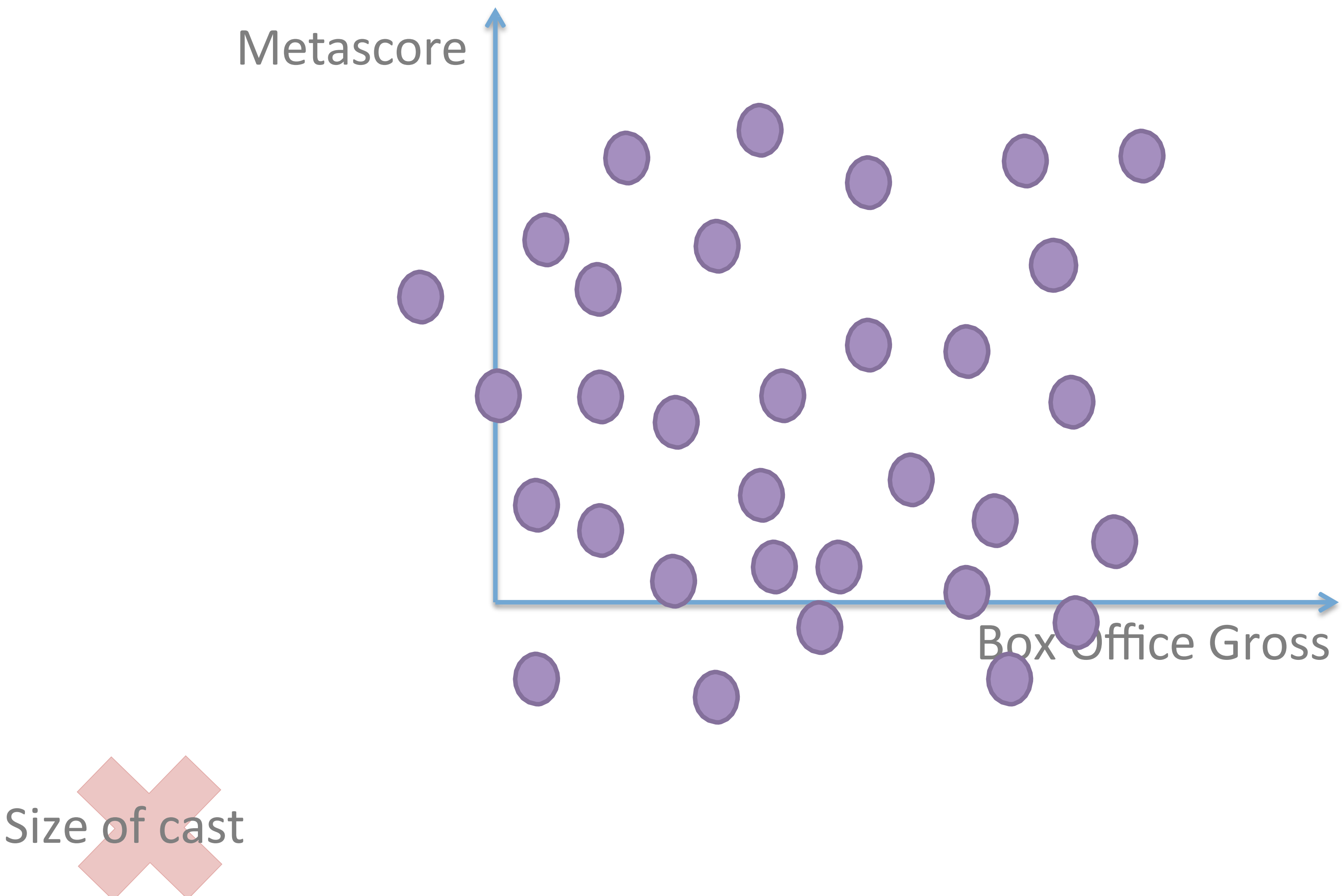
Feature Selection



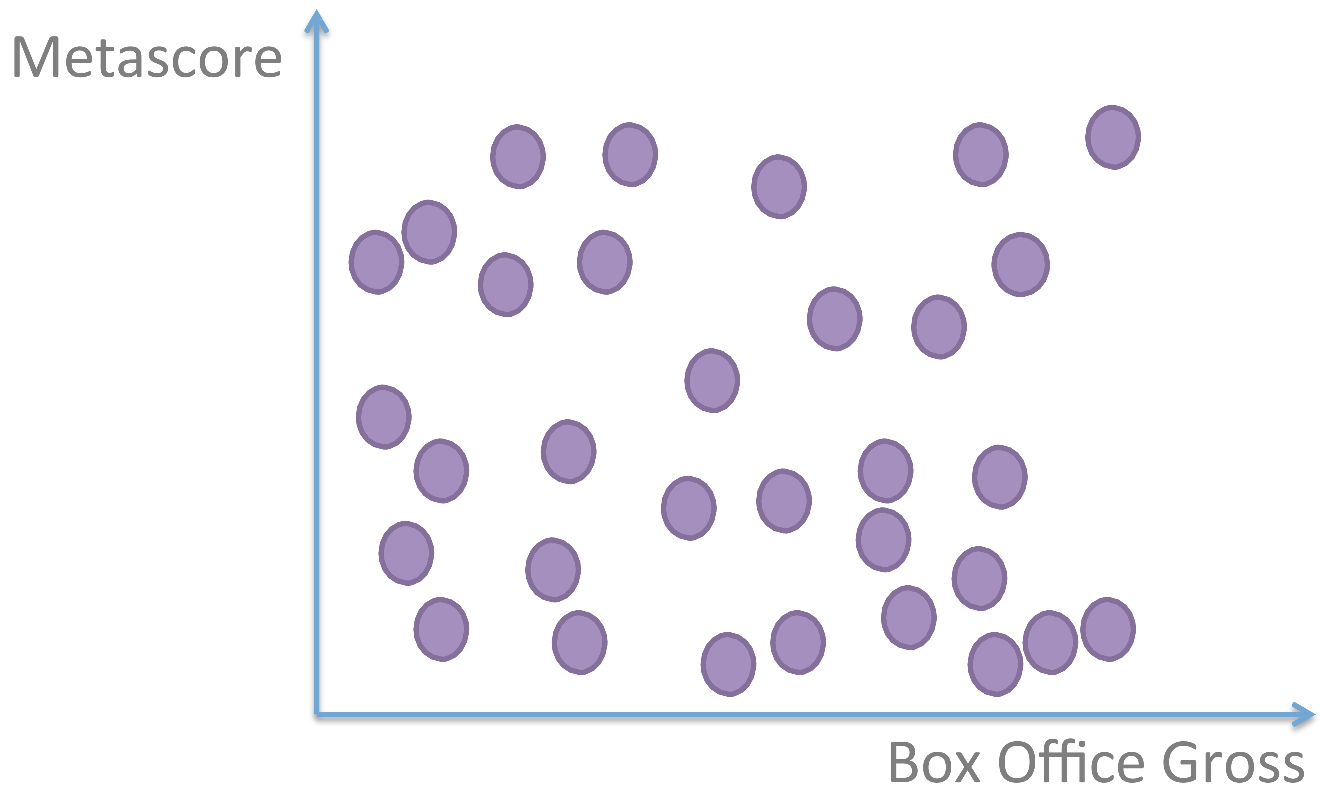
Feature Selection



Feature Selection



Feature Selection



Feature Selection

Common Sense

Experiments (remove a feature, fit again, evaluate results)

Regularization (in regression)

To get a quick idea on the features,
Or to eliminate the useless 80% of 1000 features:
sklearn.feature_selection

Feature Selection

sklearn.feature_selection

VarianceThreshold

If the values of a feature are pretty much the same for all the points, we can't use it to distinguish them.

Features with low variance cannot carry much information. Drop the features that have variation below a threshold.

Feature Selection

sklearn.feature_selection

VarianceThreshold

```
from sklearn.feature_selection import VarianceThreshold
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
```

oscar?	emmy?	bafta?
0	0	1
0	1	0
1	0	0
0	1	1
0	1	0
0	1	1

Feature Selection

sklearn.feature_selection

VarianceThreshold

```
from sklearn.feature_selection import VarianceThreshold
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
```

oscar?	emmy?	bafta?
0	0	1
0	1	0
1	0	0
0	1	1
0	1	0
0	1	1

emmy?	bafta?
0	1
1	0
0	0
1	1
1	0
1	1

Feature Selection

sklearn.feature_selection

SelectKBest

Check each feature's relation to the target one by one.
Give each feature a score.
This score estimates how much information it carries.
Drop the lowest scoring features.
(Only use the best K features)

Feature Selection

sklearn.feature_selection

SelectKBest

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
iris = load_iris()
X, y = iris.data, iris.target
X.shape

X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
```

specific score function we use here: **chi2**

chi2 is a test statistic measuring independence between a feature and the target (labels).

A good estimate of useful information for this classification.

https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test#Test_of_independence

Feature Selection

sklearn.feature_selection

SelectKBest

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
iris = load_iris()
X, y = iris.data, iris.target
X.shape

X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
```

specific score function we use here: **chi2**

Drop all but two features.

We are keeping the top 2 features that are the least independent from the target labels.

https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test#Test_of_independence

Feature Selection
sklearn.feature_selection

SelectKBest

Classification score functions:

chi2 Test of independence
f_classif ANOVA F-test

Regression score function:

f_regression The same F-test as reported by statsmodels, related to fitting only with this given feature

https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test#Test_of_independence

Feature Selection

sklearn.feature_selection

SelectKPercentile

Same as SelectKBest, but instead of saying
“Keep the top k features”,

you’re saying
“Keep the top k% of all the features”.

Feature Selection
sklearn.feature_selection

Recursive Feature Elimination

Fit a model with every feature.
Find out which feature contributes the least.
Remove that one feature, fit with the rest only.
Repeat, dropping features one by one.

Stop when you’ve reached the planned feature number.

Feature Selection

sklearn.feature_selection

Recursive Feature Elimination

```
>>> from sklearn.datasets import make_friedman1
>>> from sklearn.feature_selection import RFE
>>> from sklearn.svm import SVR
>>> X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
>>> estimator = SVR(kernel="linear")
>>> selector = RFE(estimator, 5, step=1)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True,  True,  True,  True,  True,
        False, False, False, False, False], dtype=bool)
>>> selector.ranking_
array([1, 1, 1, 1, 1, 6, 4, 3, 2, 5])
```

Needs an estimator with coefficients

(Logistic Regression works, KNN doesn't)

Uses model.coef_ as feature contributions. Scale!!!

Lasso (L1) for Feature Selection

L1 Regularization while fitting a model with coefficients will set some of those to zero. This removes them.

Pick any model that has regularization
(Logistic Regression, LinearSVC, etc.)

If you call fit(X,Y), it fits normally.
If you call fit_transform(X,Y) instead, it works as a feature selector, fits and removes features with coef_ set to zero.

Lasso (L1) for Feature Selection

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> X_new = LinearSVC(C=0.01, penalty="l1", dual=False).fit_transform(X, y)
>>> X_new.shape
(150, 3)
```

Tree Based Selection

Decision Trees are classifiers.
In a decision tree, you can measure the impact of a feature in a single split decision.

Random Forests are ensemble models of a bunch of trees. After fitting a Random Forest, you can look at the decision impacts of a feature in the entire forest.

That’s a score. You can judge a feature’s importance by it.

Feature Selection
sklearn.feature_selection

Tree Based Selection

```
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> clf = ExtraTreesClassifier()
>>> X_new = clf.fit(X, y).transform(X)
>>> clf.feature_importances_
array([ 0.04...,  0.05...,  0.4...,  0.4...])
>>> X_new.shape
(150, 2)
```

Feature Selection

Common Sense

Experiments (remove a feature, fit again, evaluate results)

Regularization (in regression)

To get a quick idea on the features,
Or to eliminate the useless 80% of 1000 features:
sklearn.feature_selection

Feature Extraction

Do I have to choose the dimensions
among existing features?

Feature Extraction

Do I have to choose the dimensions among existing features?

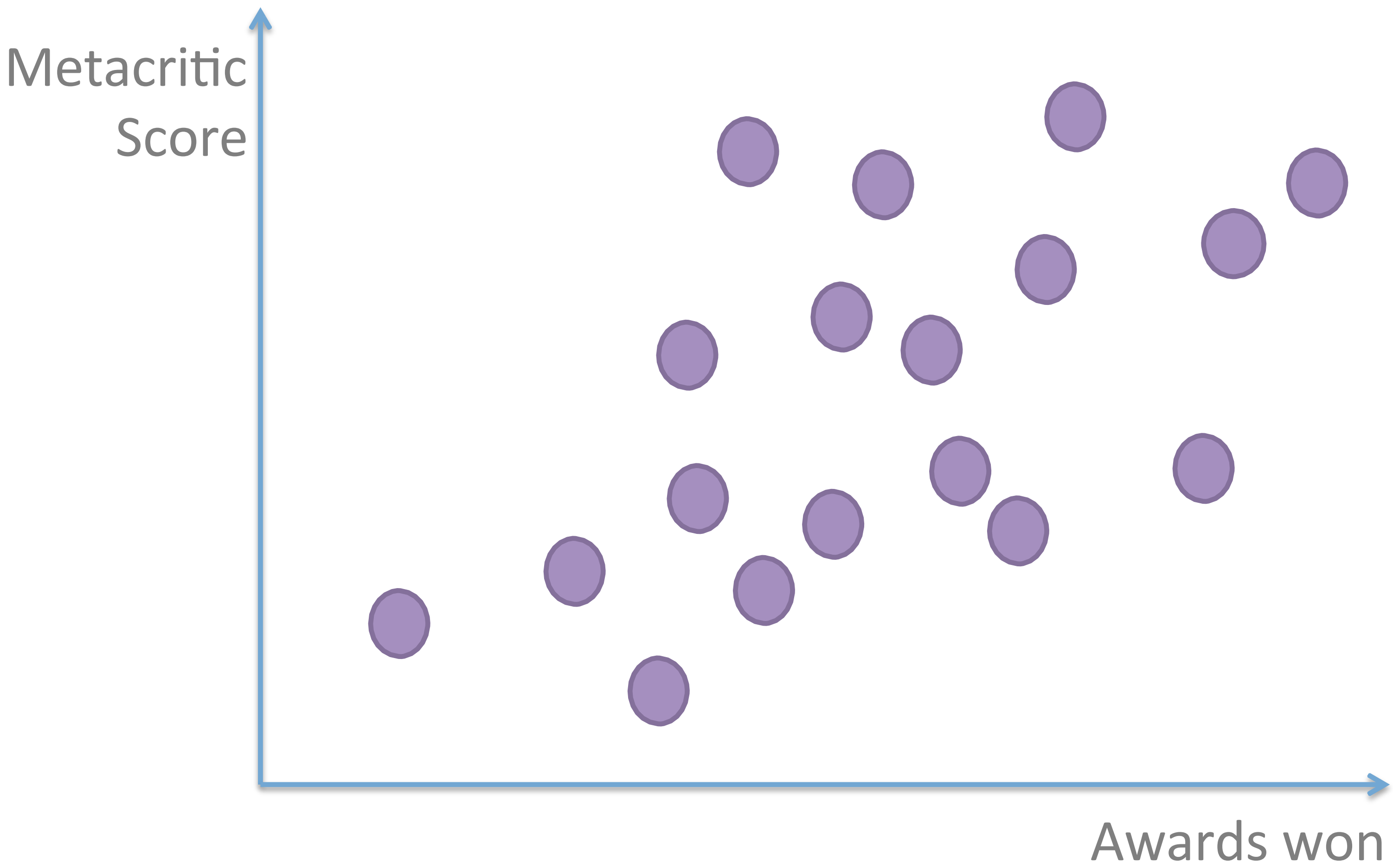
Instead of the columns won_oscar?, won_emmy?, won_golden_globe?, won_actor's_guild? (4 dummy features), try using number_of_awards_won (1 feature).

You're throwing away some information, but gaining on the curse of dimensionality arena.

Or try %_nominations_that_turn_to_awards. Combine features. Use common sense, perform hypothesis driven trials on the feature set and measure performance.

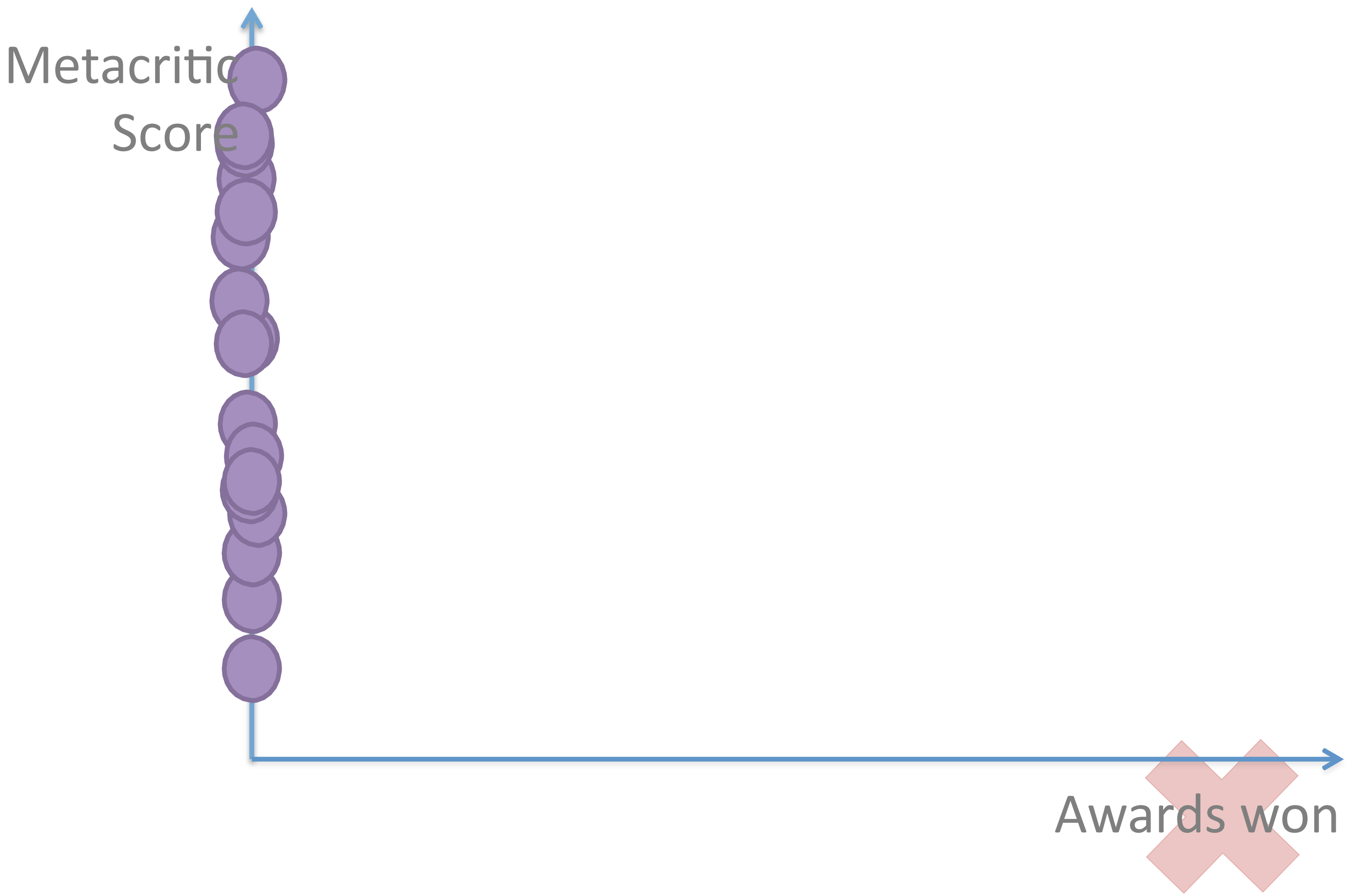
Feature Extraction

Do I have to choose the dimensions among existing features?

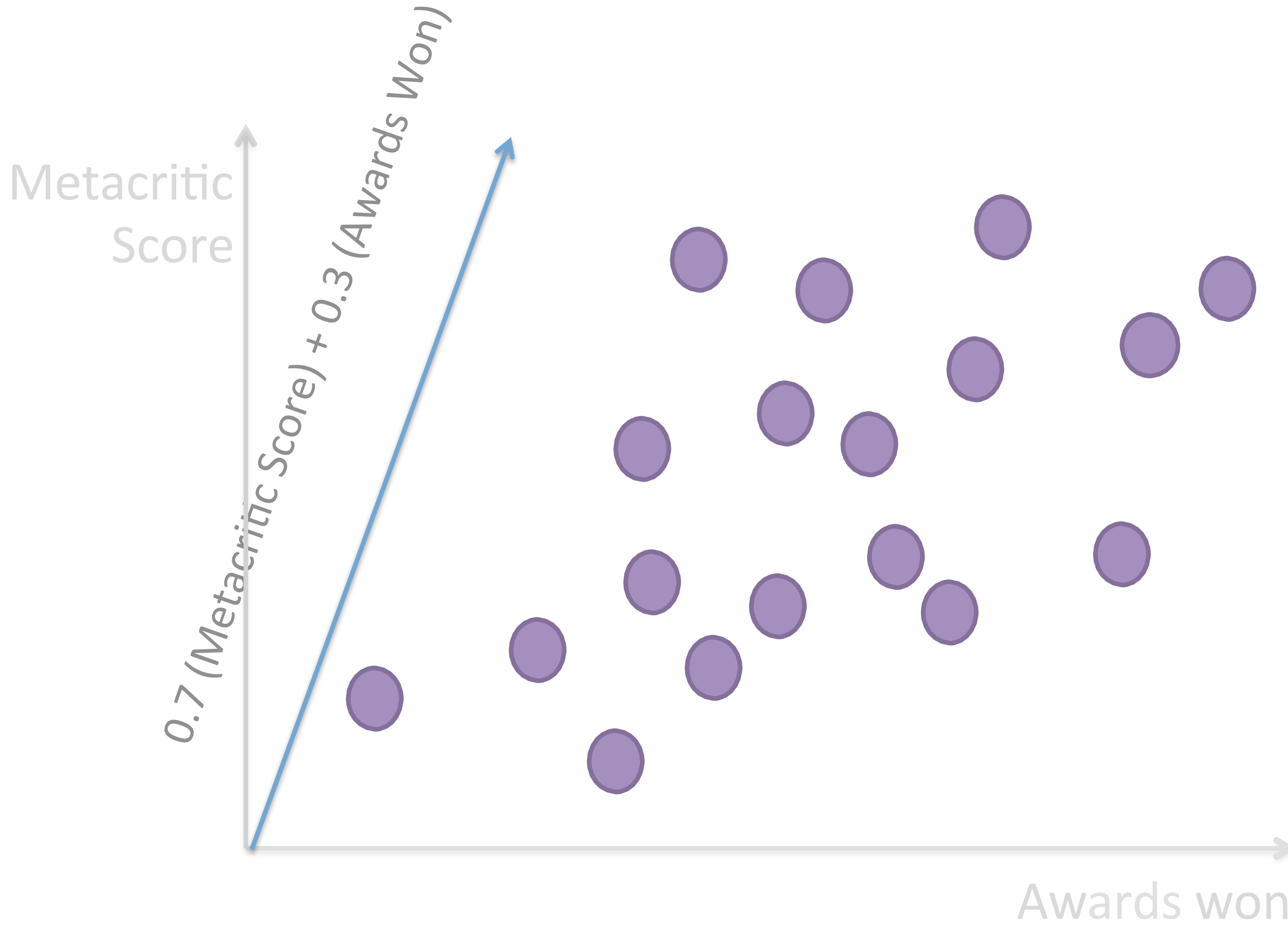


Feature Extraction

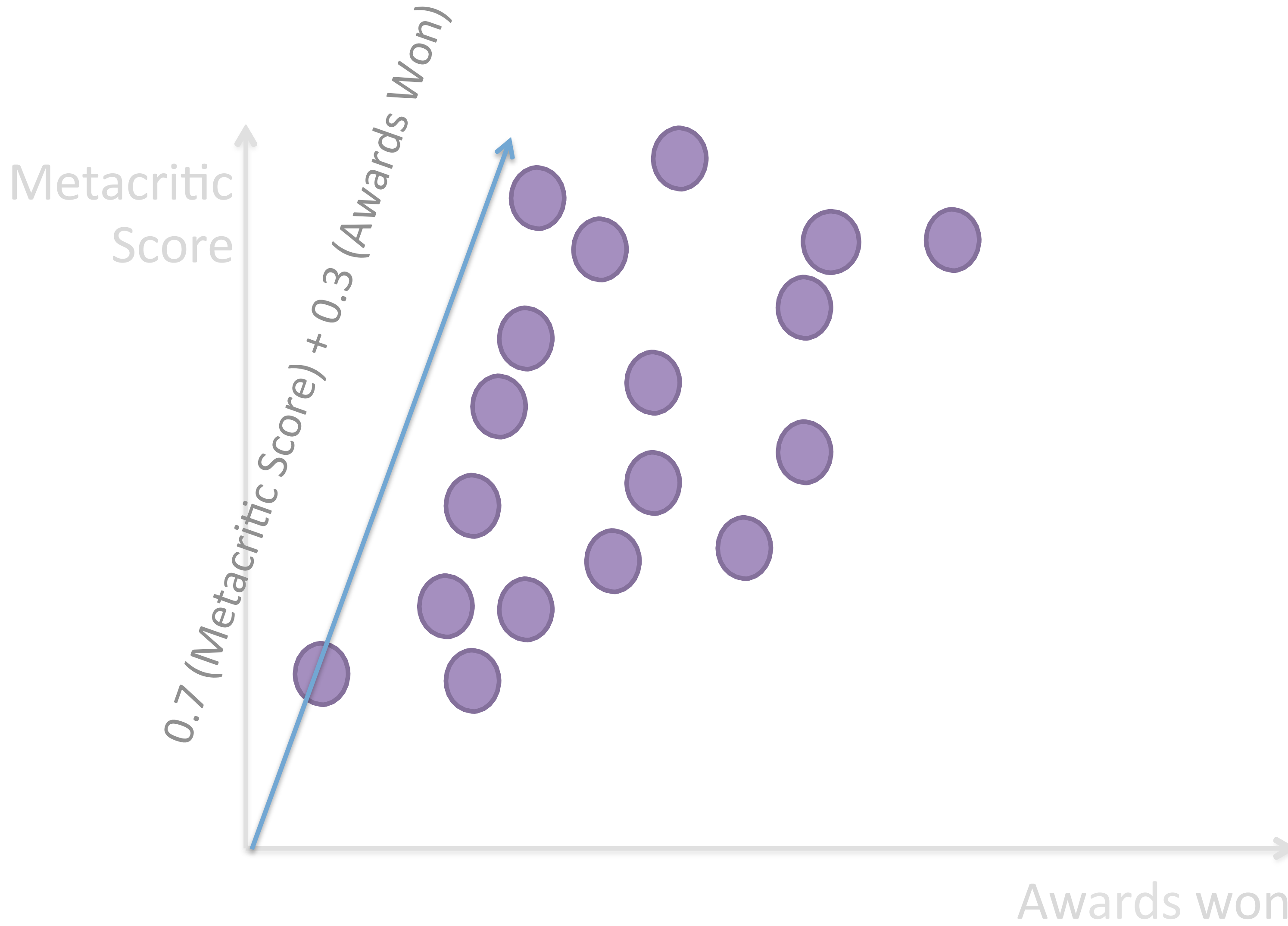
Do I have to choose the dimensions among existing features?



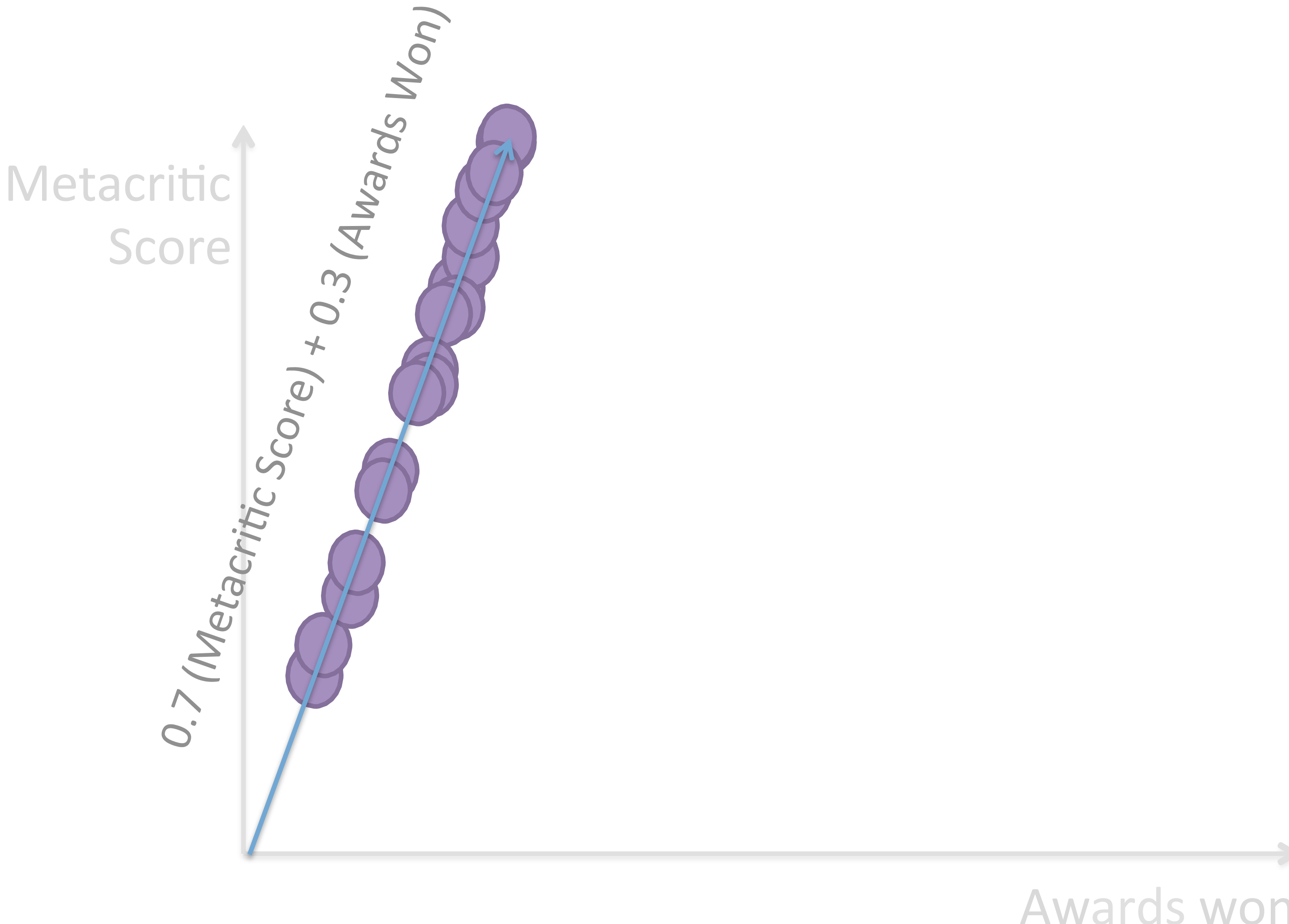
Feature Extraction



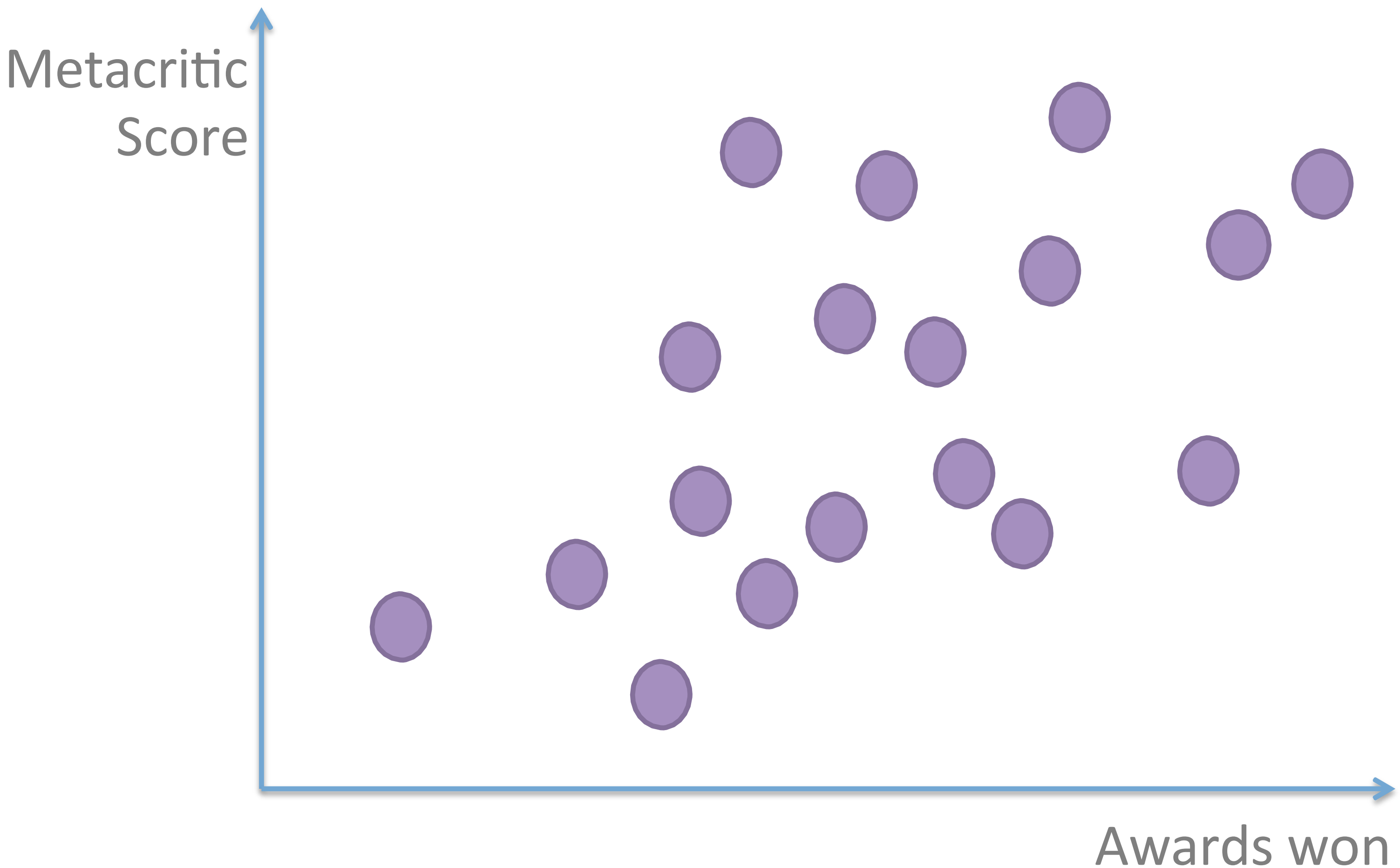
Feature Extraction



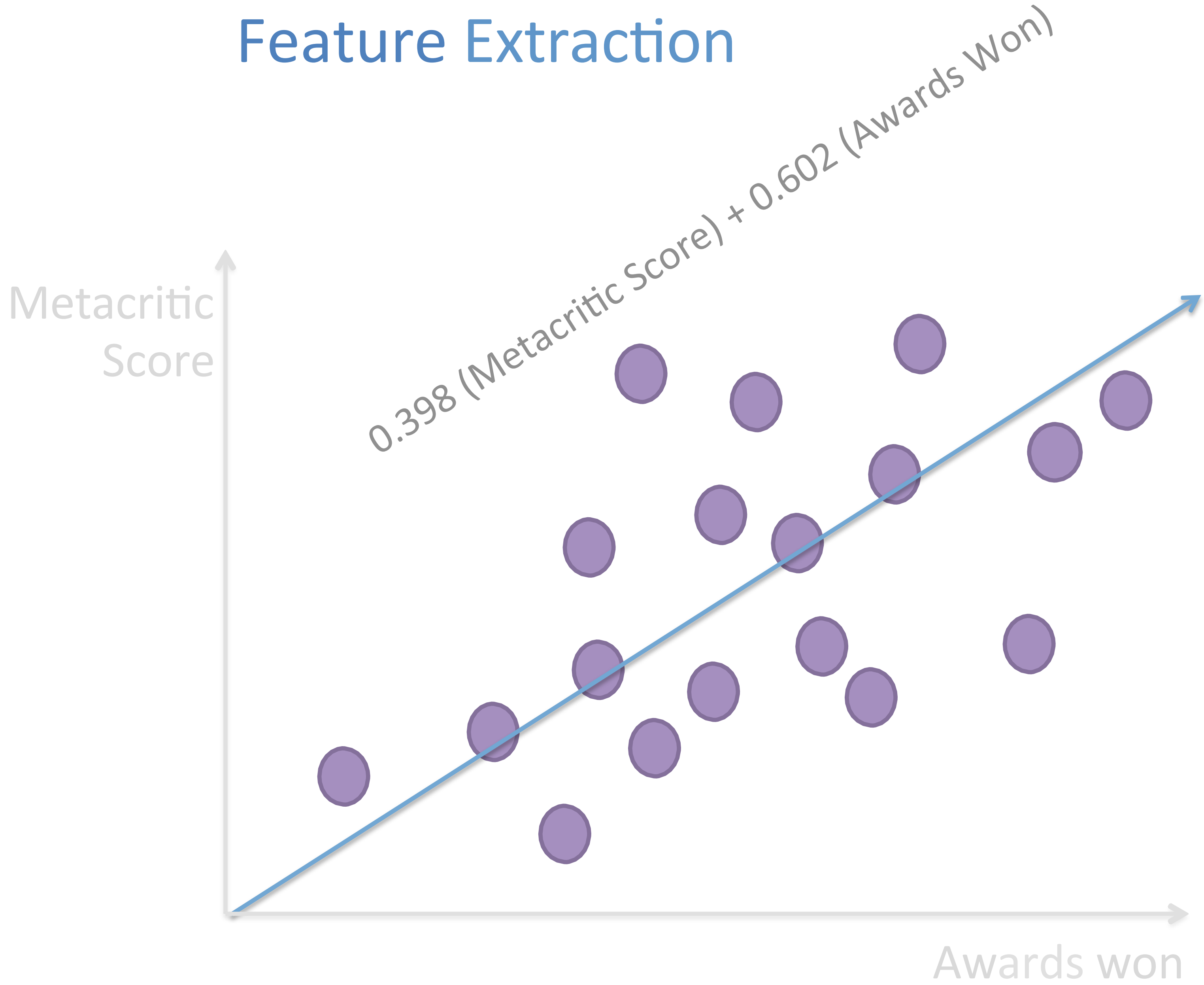
Feature Extraction



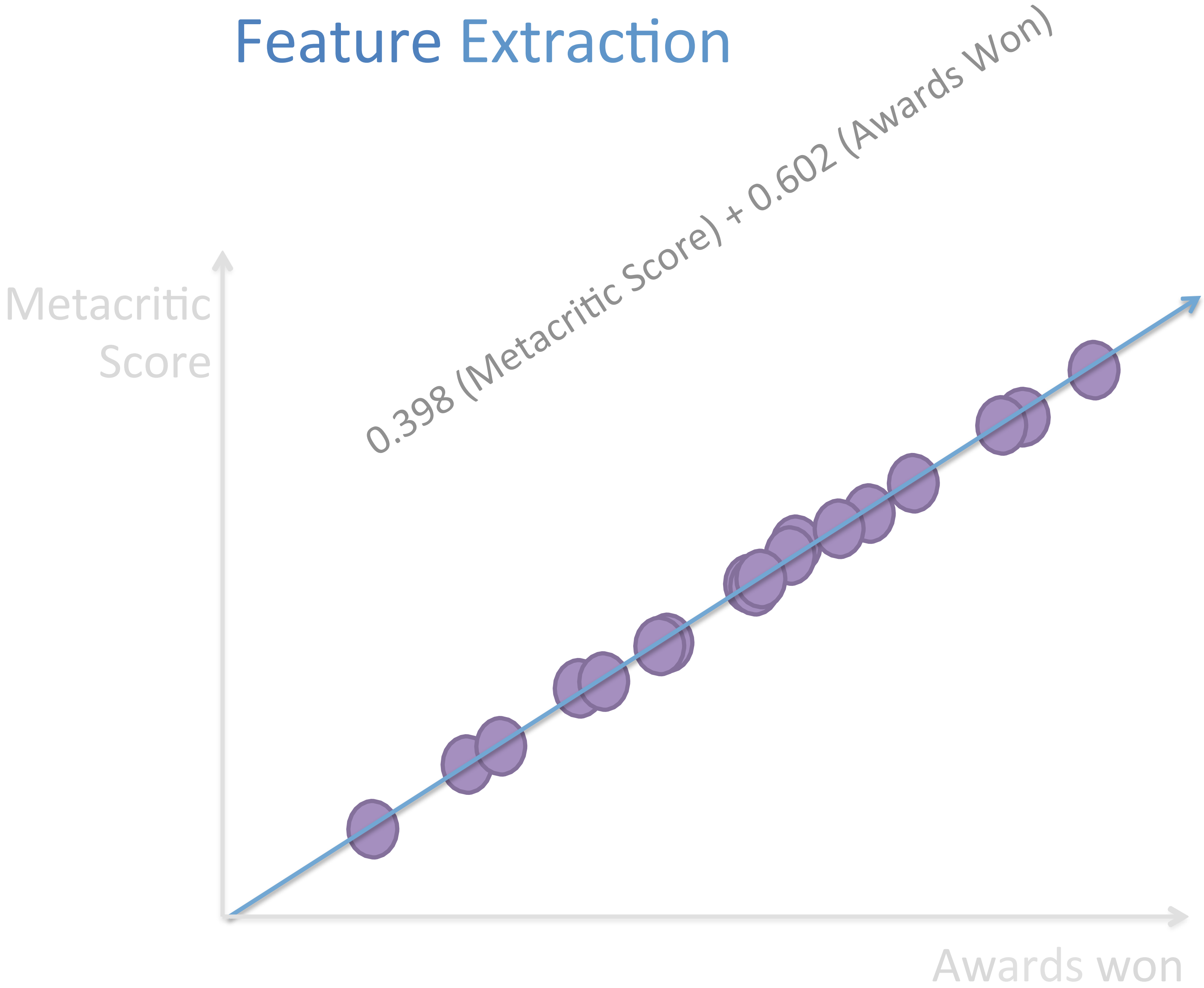
Feature Extraction



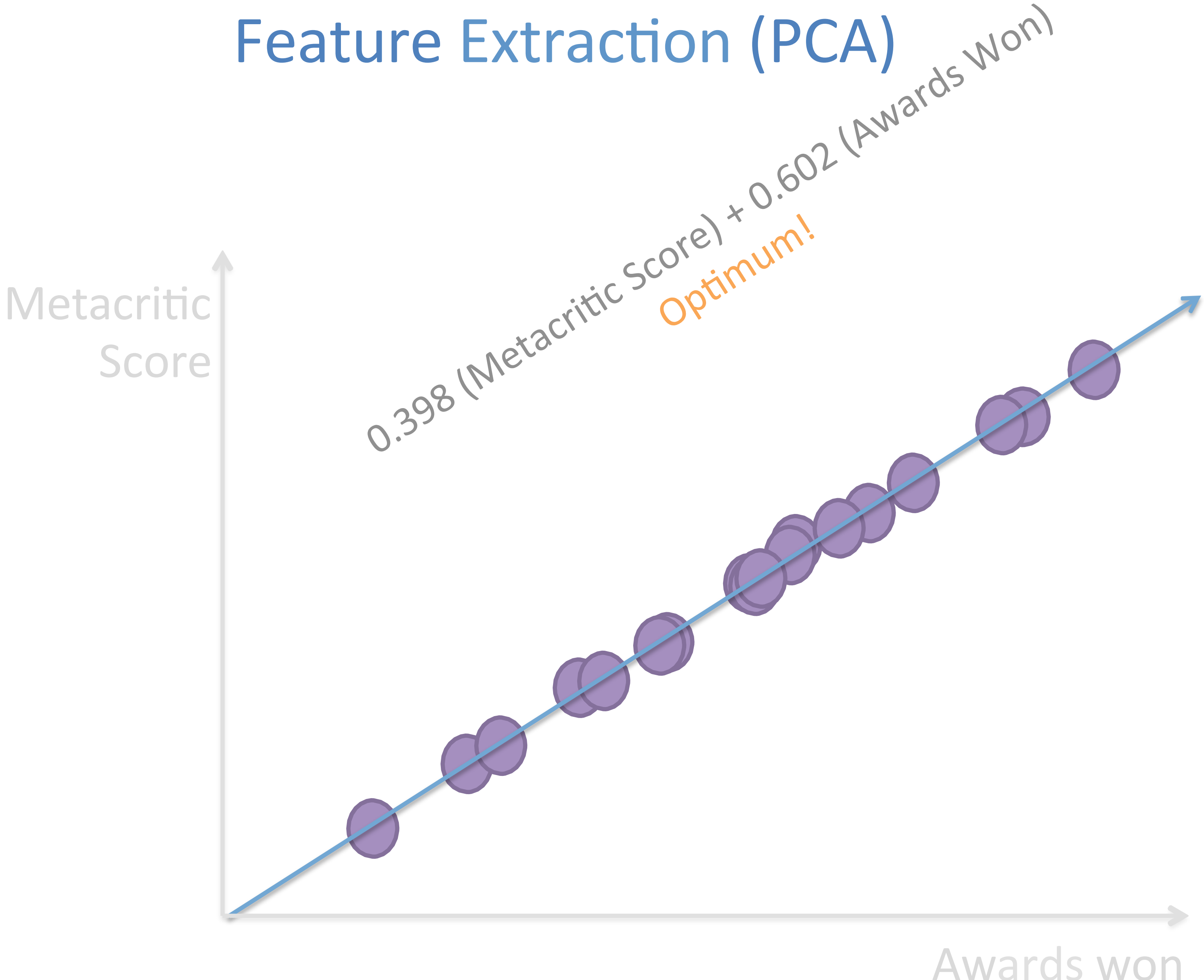
Feature Extraction



Feature Extraction



Feature Extraction (PCA)



Feature Extraction

Advantage: You retain more information
Disadvantage: You lose interpretability

Feature Extraction

Advantage: You retain more information
Disadvantage: You lose interpretability

2D

Oscar_or_not = logit(β_1(Metascore) + β_2(Awards))

Feature Extraction

Advantage: You retain more information
Disadvantage: You lose interpretability

2D

$$\text{Oscar\_or\_not} = \text{logit}(\ \beta_1(\text{Metascore}) + \beta_2(\text{Awards})\ )$$

Feature selection 1D

$$\text{Oscar\_or\_not} = \text{logit}(\ \beta_1(\text{Metascore})\ )$$

Feature Extraction

Advantage: You retain more information
Disadvantage: You lose interpretability

2D

Oscar_or_not = logit(β_1(Metascore) + β_2(Awards))

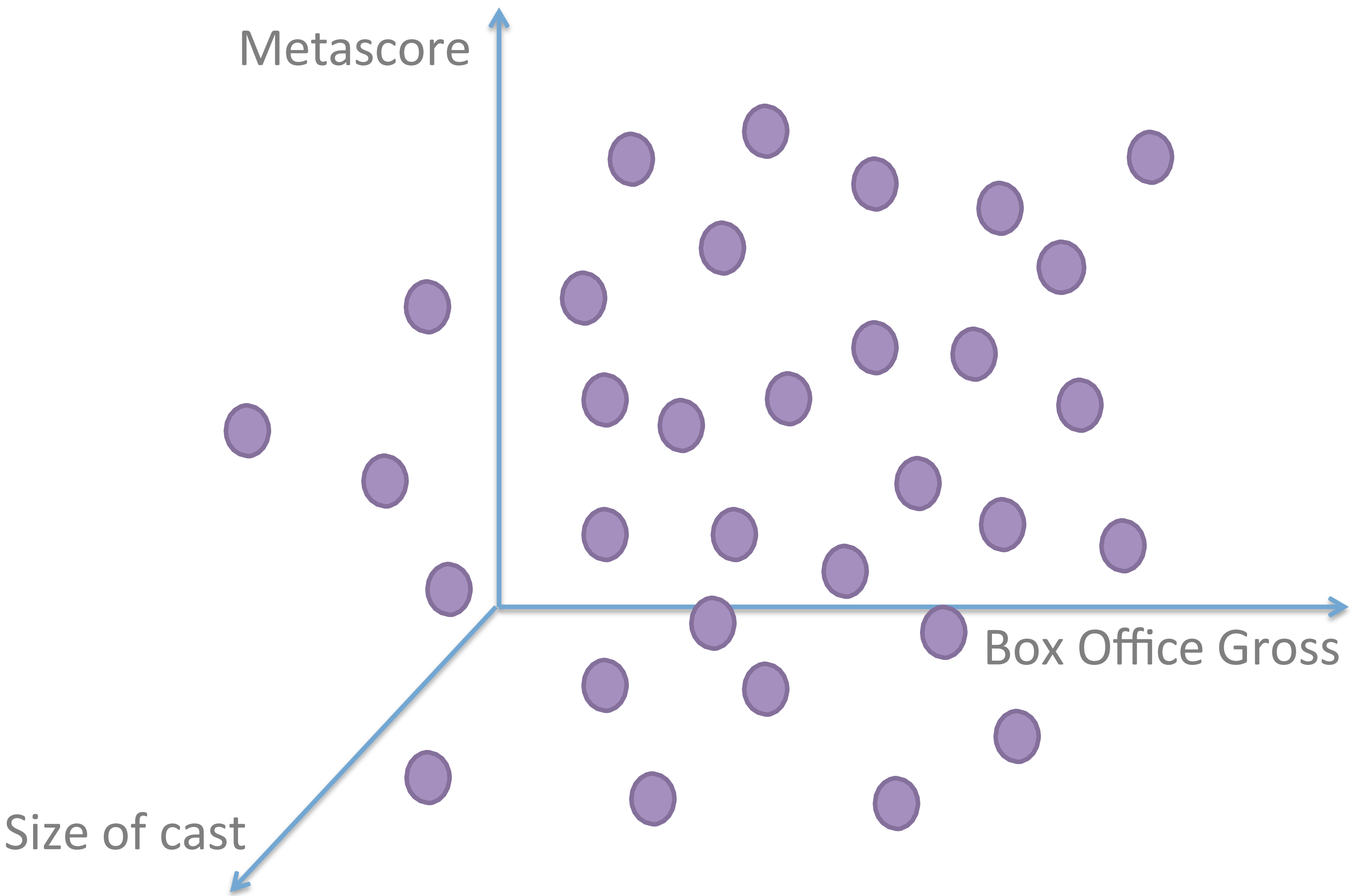
Feature selection 1D

Oscar_or_not = logit(β_1(Metascore))

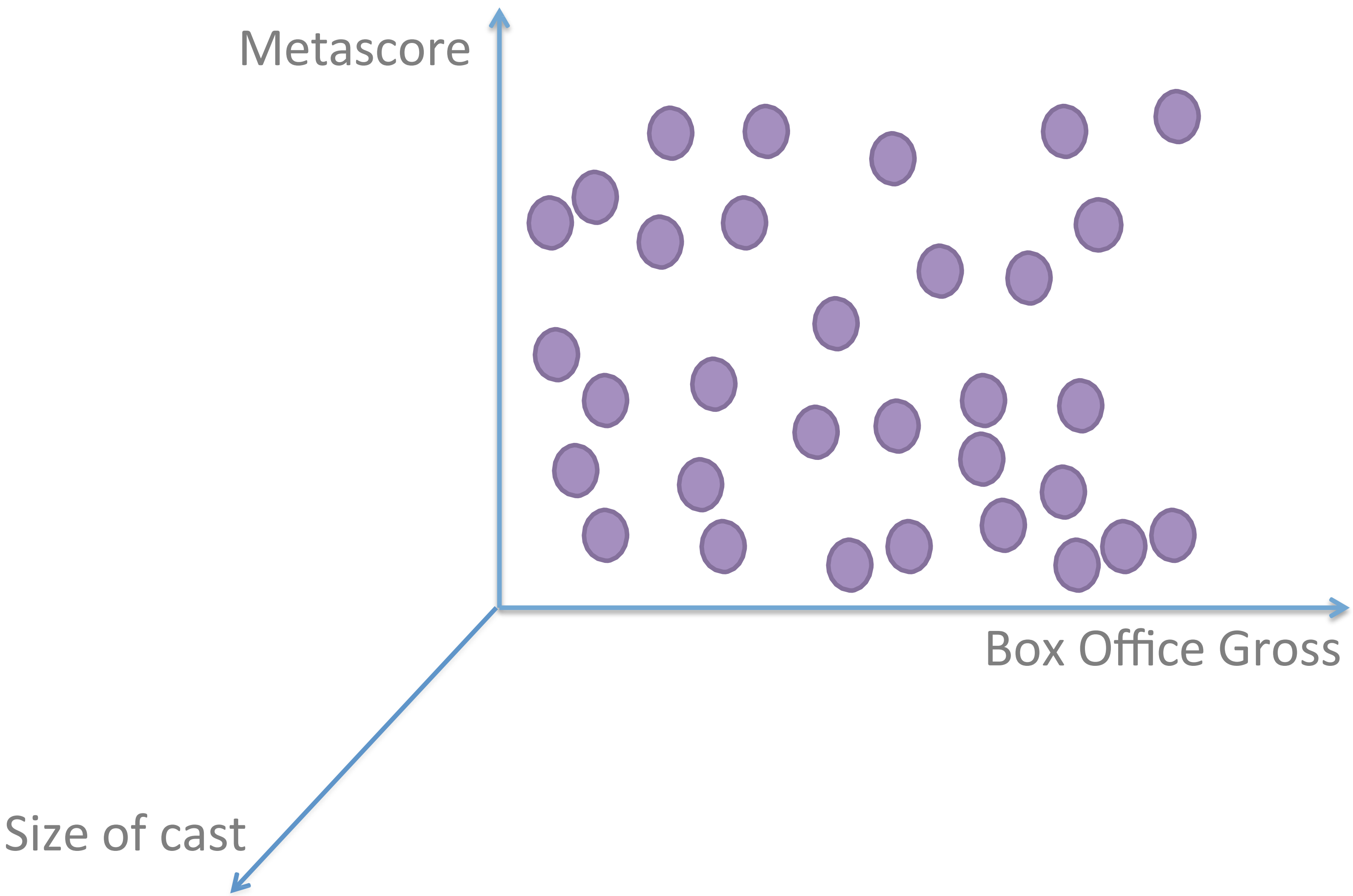
Feature extraction 1D

Oscar_or_not = logit(β_1(0.4*Metascore + 0.6*Awards))

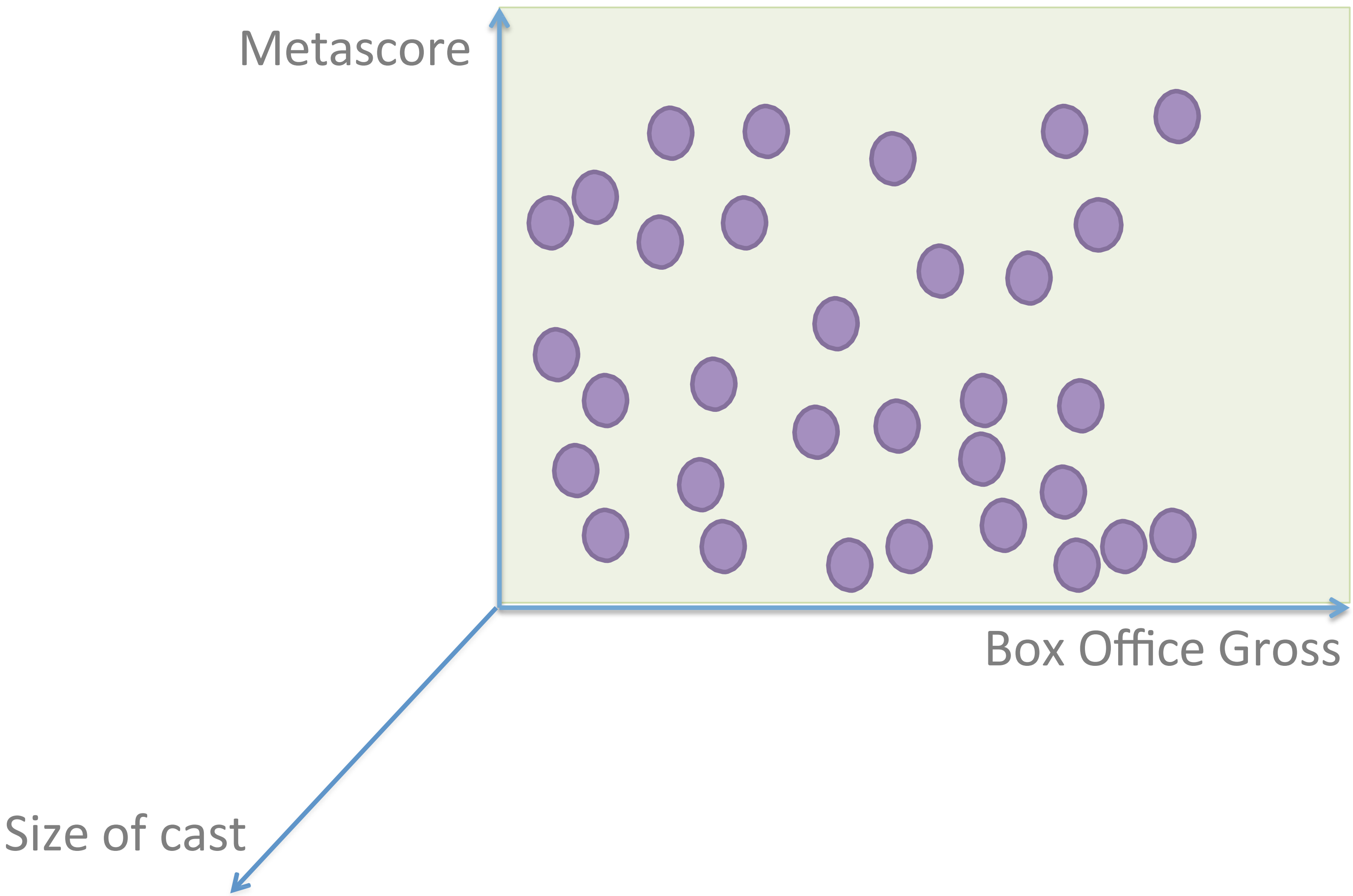
3D → 2D Feature Selection



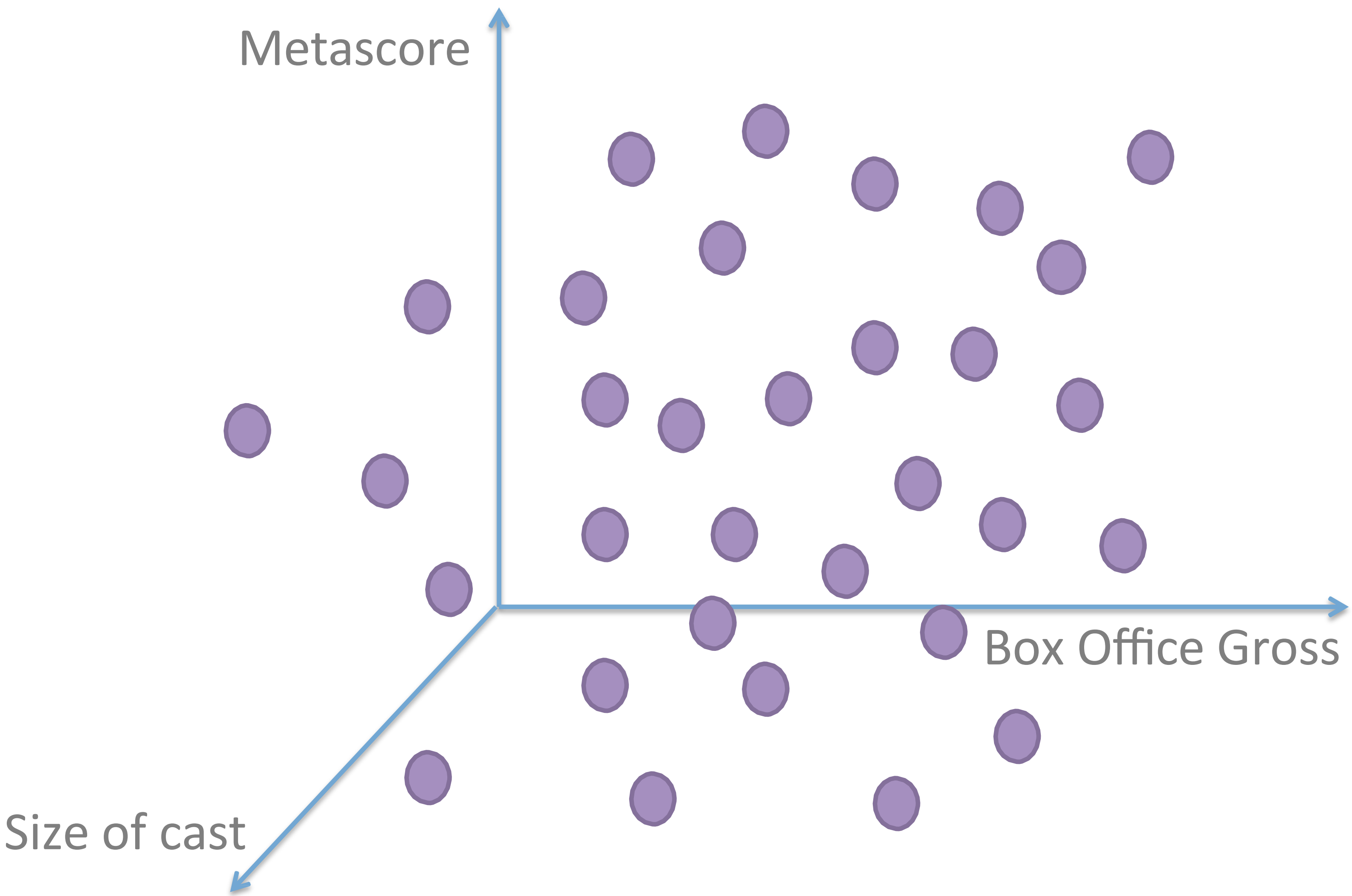
3D → 2D Feature Selection



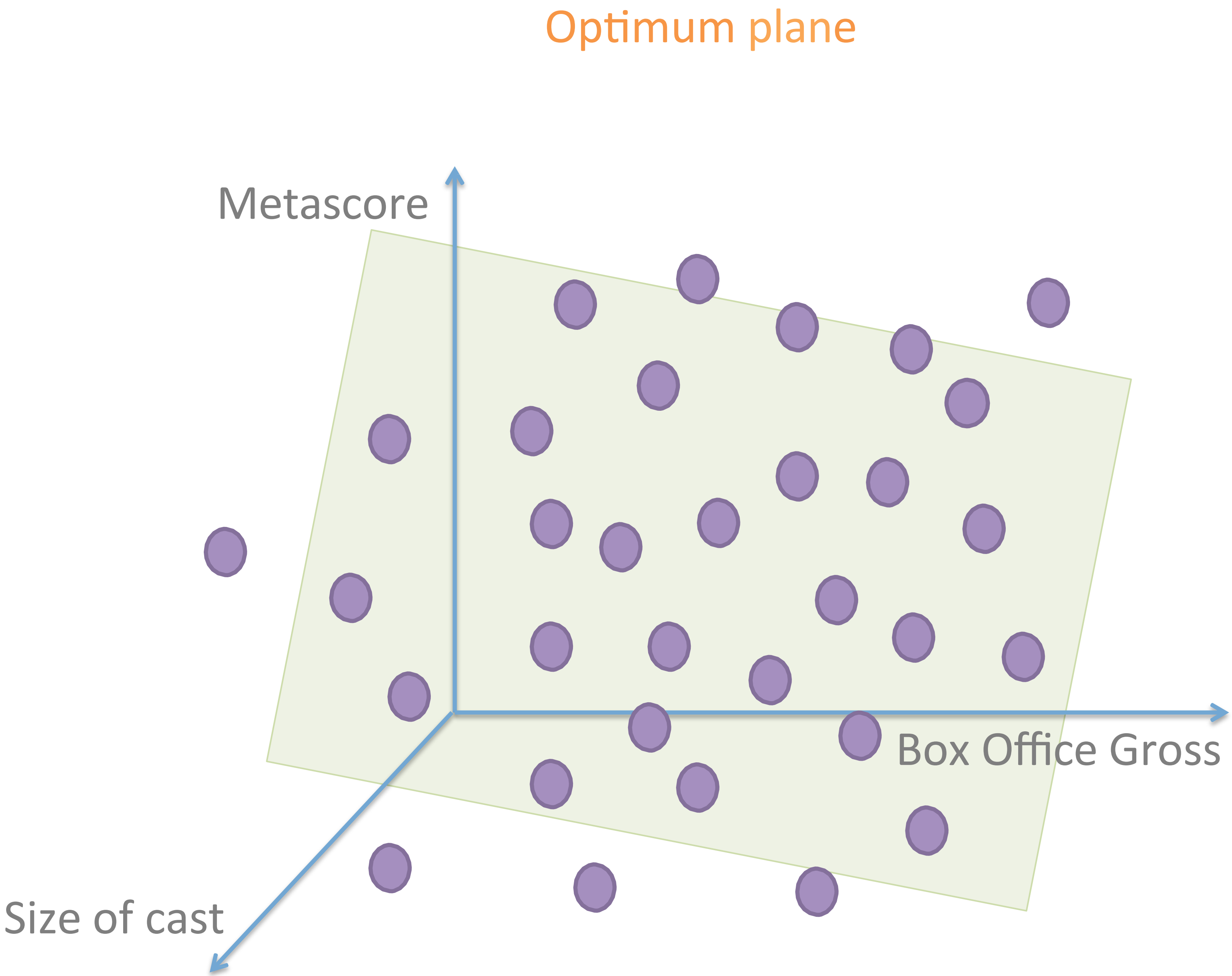
3D → 2D Feature Selection



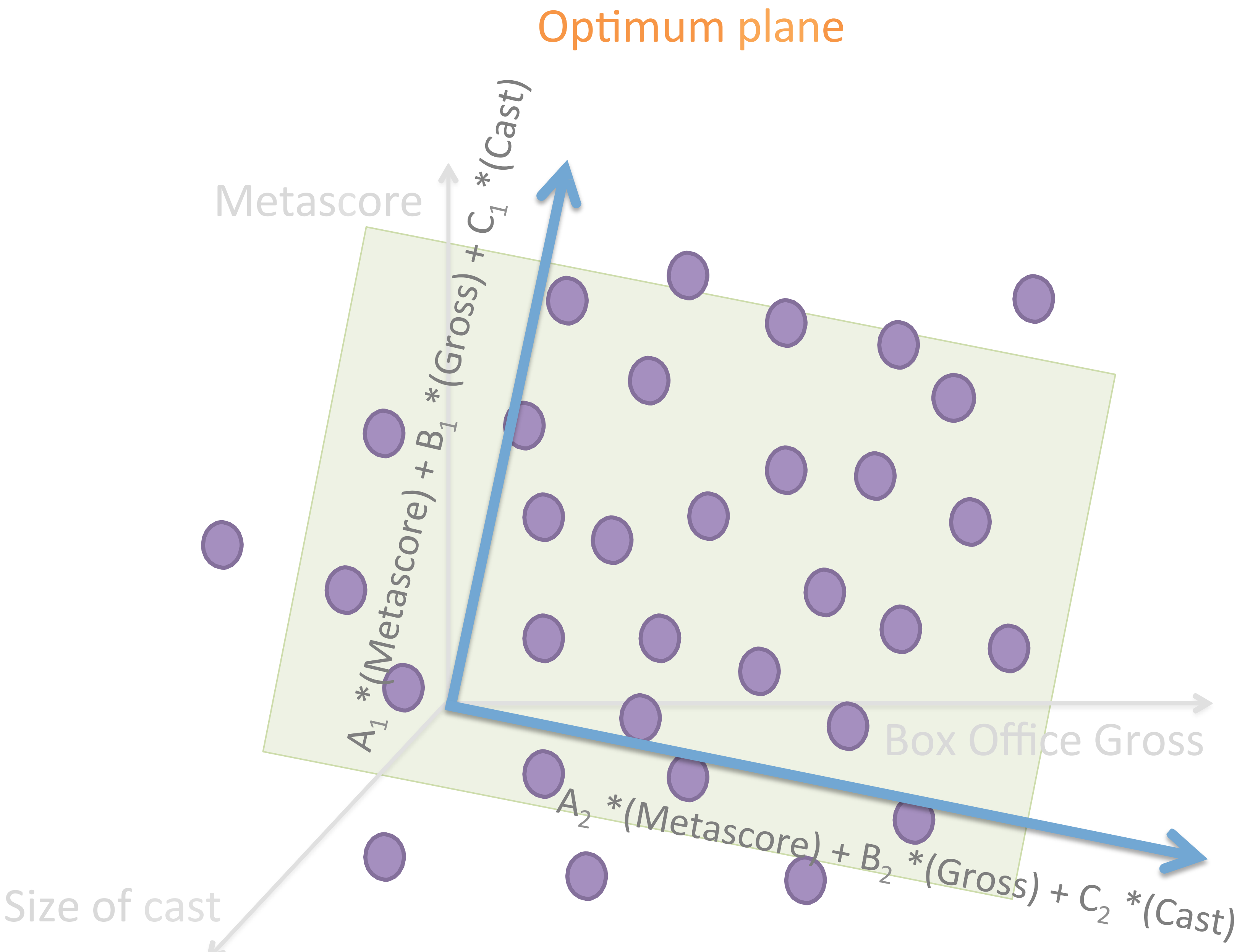
3D → 2D Feature Extraction (PCA)



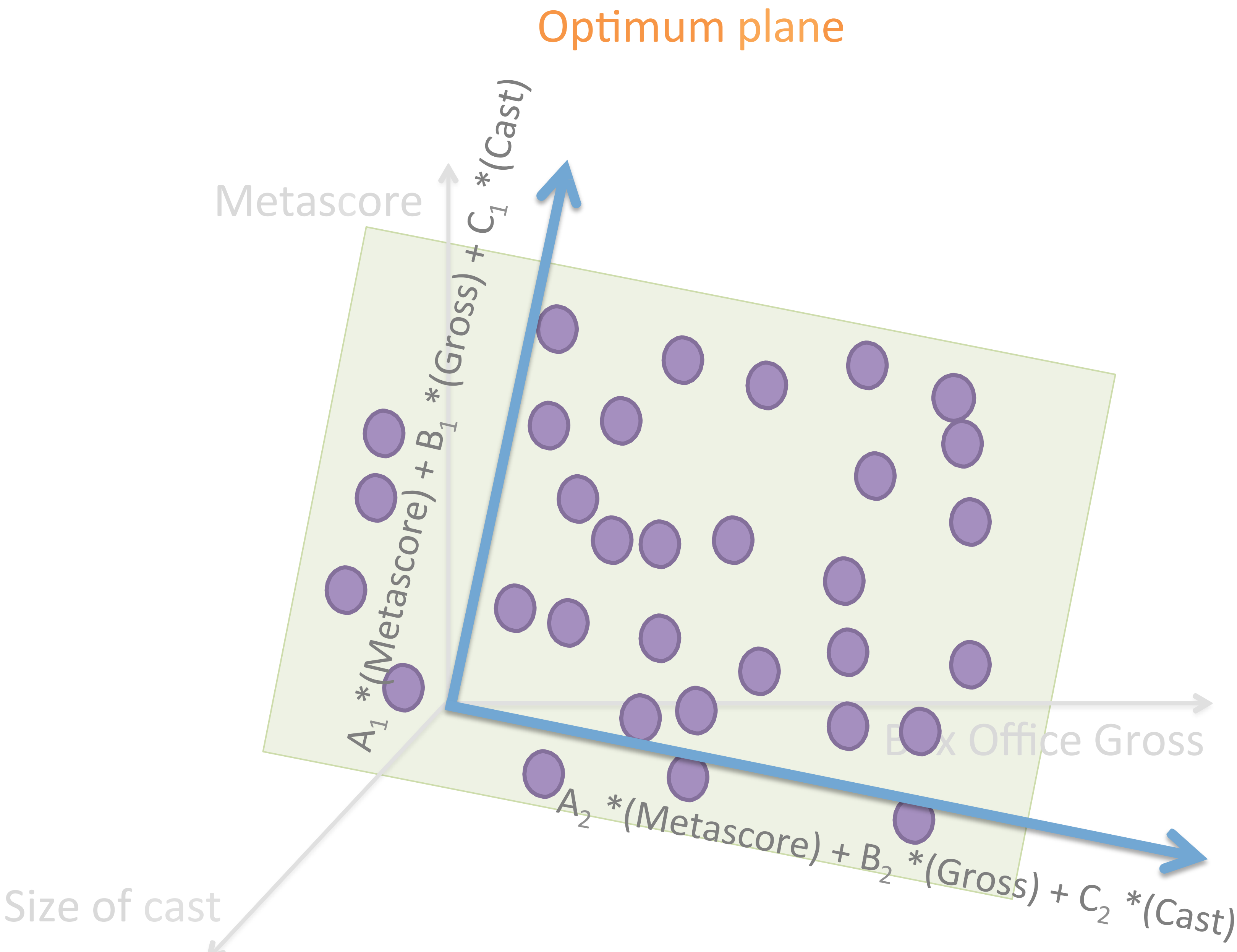
3D → 2D Feature Extraction (PCA)



3D → 2D Feature Extraction (PCA)



3D → 2D Feature Extraction (PCA)



PCA Math

Singular Value Decomposition

Vectors defining the reduced hyperplane are eigenvectors of the covarience matrix of the features.

```python
from sklearn.decomposition import PCA

reducer = PCA( n_components = 20 )
reduced_X = reducer.fit_transform(X)
model.fit(reduced_X, Y)


# When you need to predict:
reduced_X_new = reducer.transform(X_new)
model.predict(reduced_X_new)
```

How and why to use PCA

Improving your clustering

Improving your classification
(alternative to feature selection)

Dealing with sparse features

Visualizing high dimensional data in 2D or 3D

Data compression with little loss

My model is not awesome enough.

What do I do?

What do I do?

<u>Feature selection</u>
<u>Model parameters</u>
(K for KNN, C for regularization, etc.)
Feature extraction
Functional forms of features
Feature interactions
Sensible combinations of features
Try different algorithms
PCA