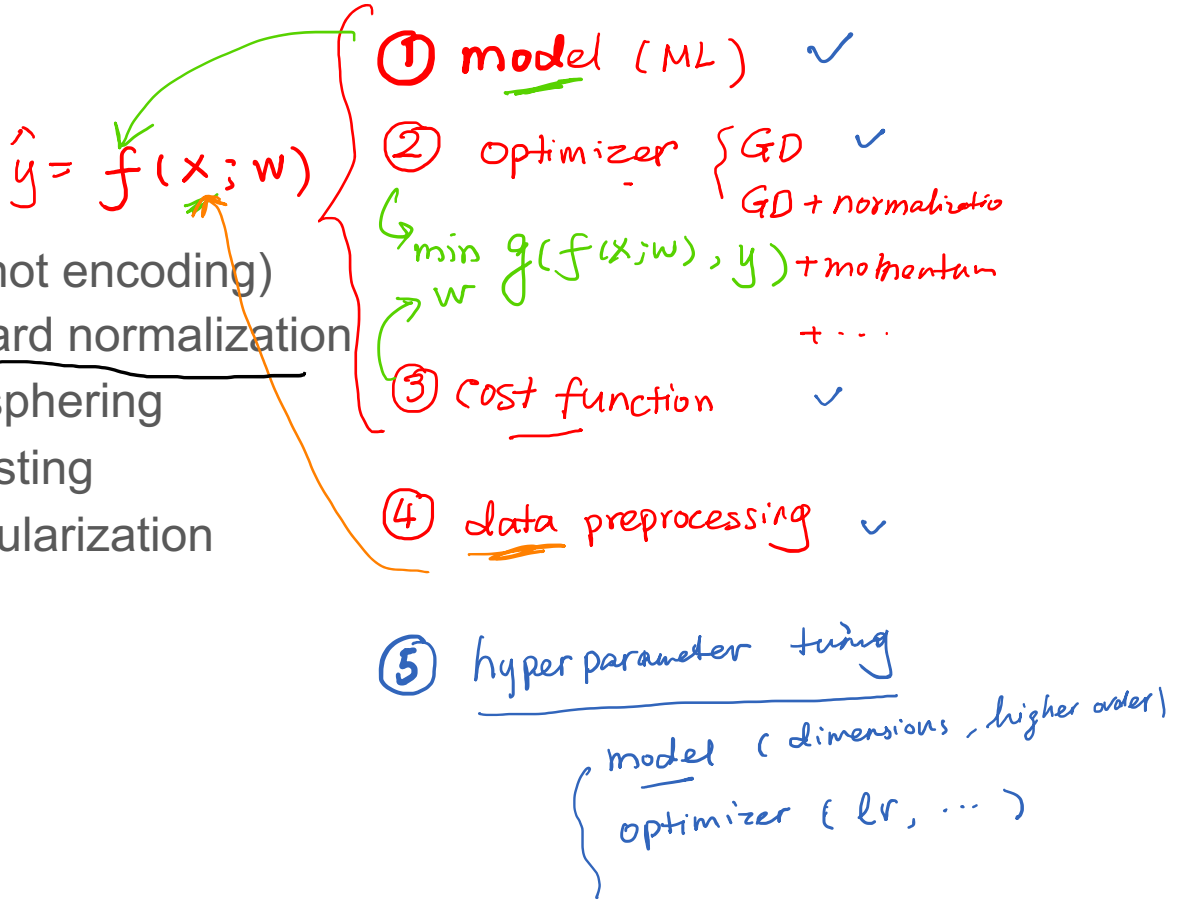
Feature Engineering and Selection

Instructor: Hui Guan

Slides adapted from: https://github.com/jermwatt/machine_learning_refined

Outline

- Histogram features (one-hot encoding)
- Feature scaling via standard normalization
- Feature scaling via PCA sphering
- Feature selection via Boosting
- Feature selection via Regularization



$f(x; w)$

input　parameter

"numerical values"

1. medical record: ..., "A/AB/B/O", ...

① data = attribute: blood type $\in \{A, AB, B, O\}$

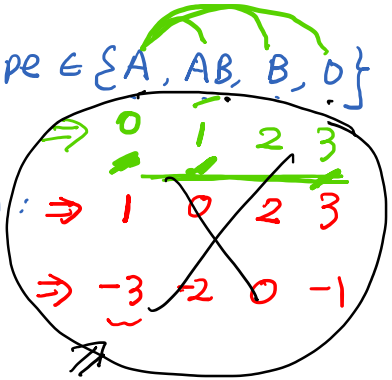
↓

x: Features (numerical):

A: 0 0 0 1
AB: 0 0 1 0
B: 0 1 0 0
O: 1 0 0 0

"one-hot encoding"

no assumptions on similarities



bad "bias"

9.2 Histogram features

2. review: text

x_1: I like this class → $\in \mathbb{R}^{d=7}$

x_2: I hate the other class → $\in \mathbb{R}^{d}$

x_3: I like this class; ...; ...?

:

$\in \mathbb{R}^{p}$

unique words = (I, like, hate, this, the, class, other)

	I	like	hate	this	the	class	other
x_1:	1/√4	1	0	1	0	1	0
x_3:	4/√(4²+5²...3²)	5	0	3	9	4	0

Unit vector

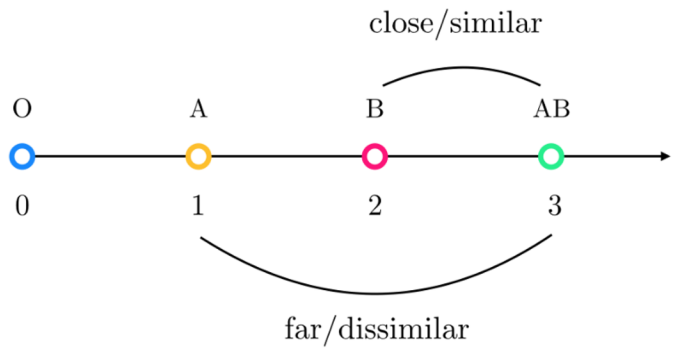
"BOW"

Histogram features for categorical data

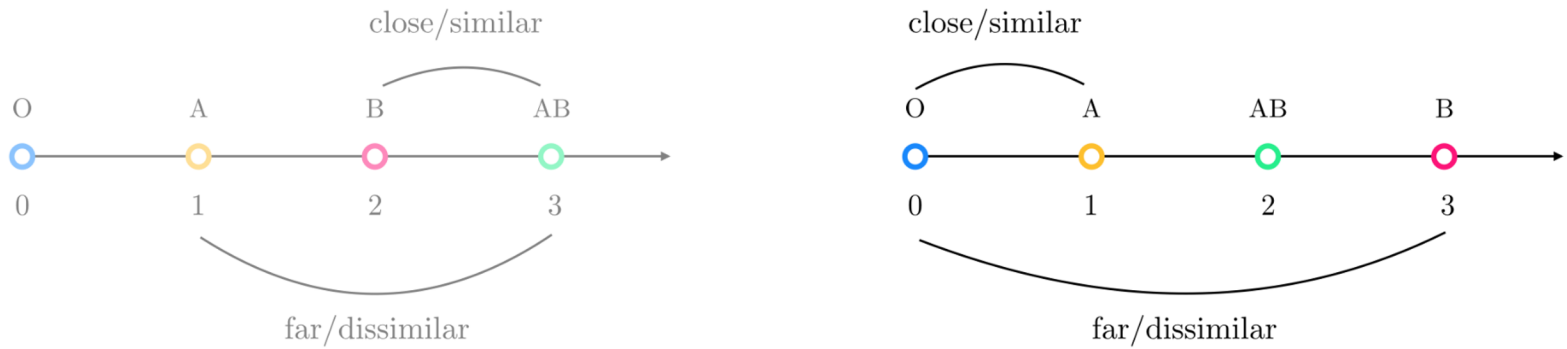
- Every machine learning paradigm requires that the data we deal with consists strictly of **numerical values**, however raw data does not always come pre-packaged in this manner.
- Consider, for instance, a hypothetical medical dataset consisting of several patients' vital measurements such as blood pressure, blood glucose level, and blood type, as input.

- The first two features (i.e., blood pressure and blood glucose level) are naturally numerical, and hence ready for supervised or unsupervised learning.
- Blood type on the other hand is a **categorical feature**, taking on as value one of four categories: O, A, B, and AB. How can one translate these categories into numerical values decipherable by a machine learning algorithm?

- A first, intuitive approach would be to **represent each category with a distinct real number**, e.g., by assigning 0 to the blood type O, 1 to A, 2 to B, and 3 to AB, as shown in the left panel of the figure below.
- Here, the way we assign numbers to each category is important. In this particular instance, by assigning 1 to A, 2 to B, and 3 to AB, we have tacitly made the assumption that, everything else the same, an individual with blood type AB is more 'similar' to one with blood type B than one with blood type A.



- This is because the number 3 is closer to number 2 than it is to number 1 on the real axis.
- One could argue that it is more appropriate to switch the numbers used to represent categories B and AB so that AB now sits between (and hence equidistant from) A and B, as shown in the right panel of the figure below.
- With this reassignment, blood type O still neighbors A but is now maximally away from B - an assumption that may or may not be true in reality.



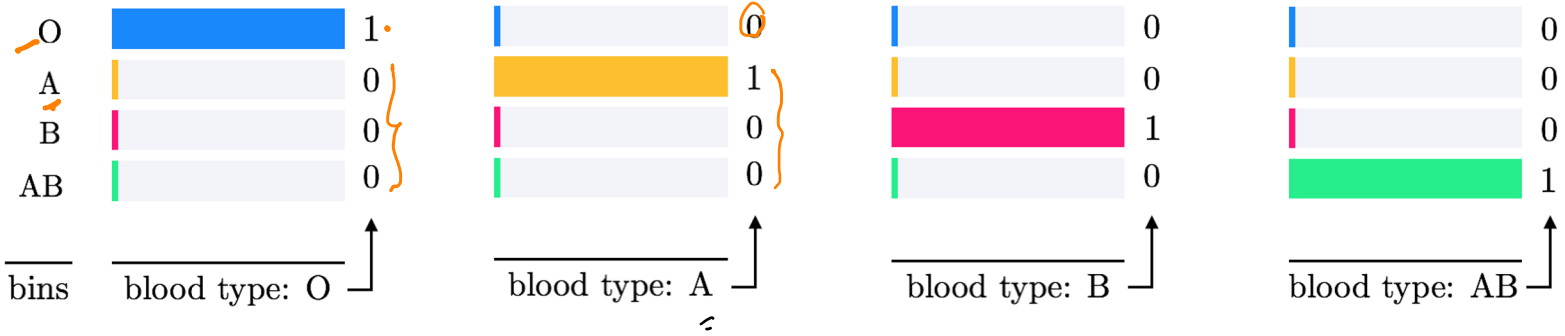
- The crux of the matter here is that **there is always a natural order to any set of numbers chosen on the real axis**, and by using such numbers we inevitably make assumptions about similarity or dissimilarity of the existing categories in the data.
- In most cases we want to avoid making such assumptions that fundamentally change the shape/geometry of the problem, especially when we lack the intuition/knowledge necessary for determining similarity between different categories.

- A better way of encoding categorical features is to use a **simple histogram whose bins are all categories** present in the categorical feature of interest (here, blood type).
- Then, an individual with blood type O is no longer represented by a single number but by a four-binned histogram wherein the count associated with bin O is 1 while all the rest are set to zero.

one-hot encoding

O = 1000

A: 0100



This way, all blood type representations (each a four dimensional vector with a single one and three zeros) are equidistant from one another.

This method of encoding categorical features, sometimes referred to as one-hot encoding, can also be thought of as replacing the original blood type feature with four new 'dummy' features that take on binary values:

Is blood type **O**? 1 if yes; 0 if no

Is blood type **A**? 1 if yes; 0 if no

Is blood type **B**? 1 if yes; 0 if no

Is blood type **AB**? 1 if yes; 0 if no

Histogram features for text data

- Many popular uses of machine learning, including sentiment analysis, spam detection, document categorization or clustering, and more, are based on text data (e.g., online news articles, emails, social-media updates, etc.).
- However with text data, the initial input (i.e., the document itself) requires a significant amount of preprocessing and transformation prior to further feature design and supervised/unsupervised learning.

- The most basic yet widely used feature of a document for machine learning tasks is called a Bag of Words (BoW) histogram or feature vector.
- Here we introduce the BoW histogram and discuss its strengths, weaknesses, and common extensions.

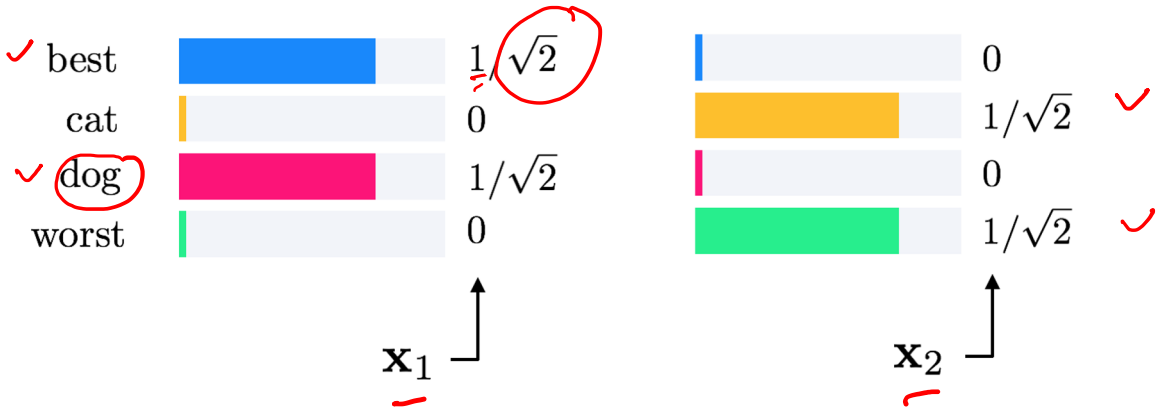
- **A BoW feature vector** of a document is a simple histogram count of the different words it contains with respect to a single corpus or collection of documents minus those non-distinctive words that do not characterize the document.

1) dogs are the best
2) cats are the worst

"stop words"

dog
"dogs" → "dog"

"Stemming"

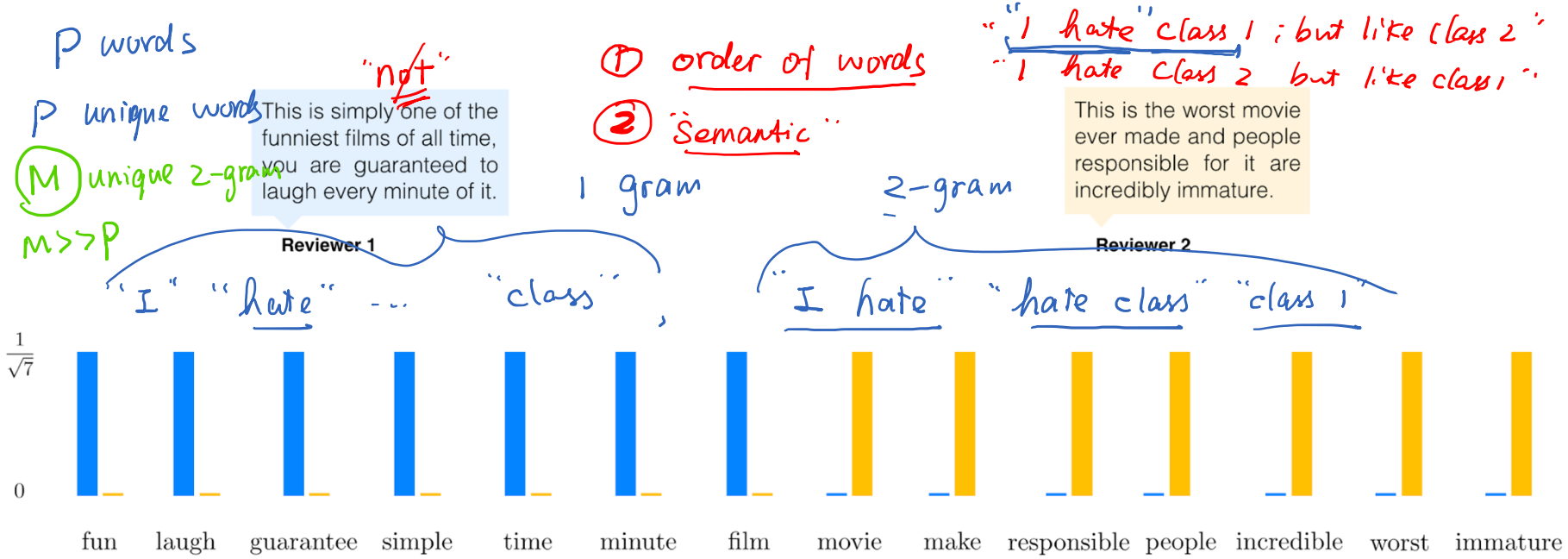


- Notice that in creating the BoW histograms uninformative words such as 'are' and 'the', typically referred to as "stop words", are not included in the representation.
- Further notice that we count the singular 'dog' and 'cat' in place of their plural which appeared in the original documents.

- This preprocessing step is commonly called "stemming," where related words with a common stem or root are reduced to and then represented by their "common root."
- For instance, the words 'learn', 'learning', 'learned', and 'learner', in the final BoW feature vector are all represented by and counted as 'learn'. Additionally, each BoW vector is "**normalized to have unit length**."

Example: Sentiment analysis

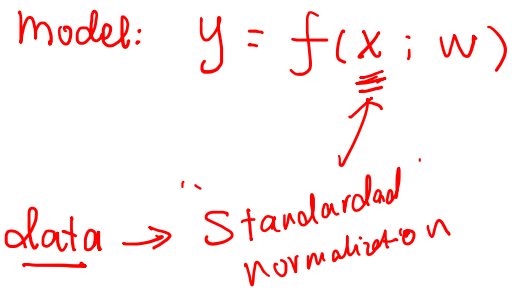
- Determining the aggregated feelings of a large base of customers, using text-based content like product reviews, tweets, and comments, is commonly referred to as sentiment analysis
- Classification models are often used to perform sentiment analysis, learning to identify consumer data of either positive or negative feelings.



BoW representation of two movie review excerpts, with words (after the removal of stop words and stemming) shared between the two reviews listed along the horizontal axis. The vastly different opinion of each review is reflected very well by the BoW histograms, which have zero correlation

Limitations of BoW

- Because the BoW vector is such a simple representation of a document, completely ignoring word order, punctuation, etc., it can only provide a gross summary of a document's contents and **is thus not always distinguishing**.
- For example, the two documents ``*dogs are better than cats*`` and ``*cats are better than dogs*`` would be considered the same document using BoW representation, even though they imply completely opposite relations.



9.3 Feature Scaling via Standard Normalization

Q: Apply STN(X), does it change minimum cost value? { Y / N

$\min_w \|X^T w - y\|_2^2$

Standard normalization of classification data

raw data

$$\frac{x_{p,n} - \mu_n}{\sigma_n}$$

$$\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$$

$$\sigma_n = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\left(x_{p,n} - \mu_n\right)^2}.$$

X : n2000

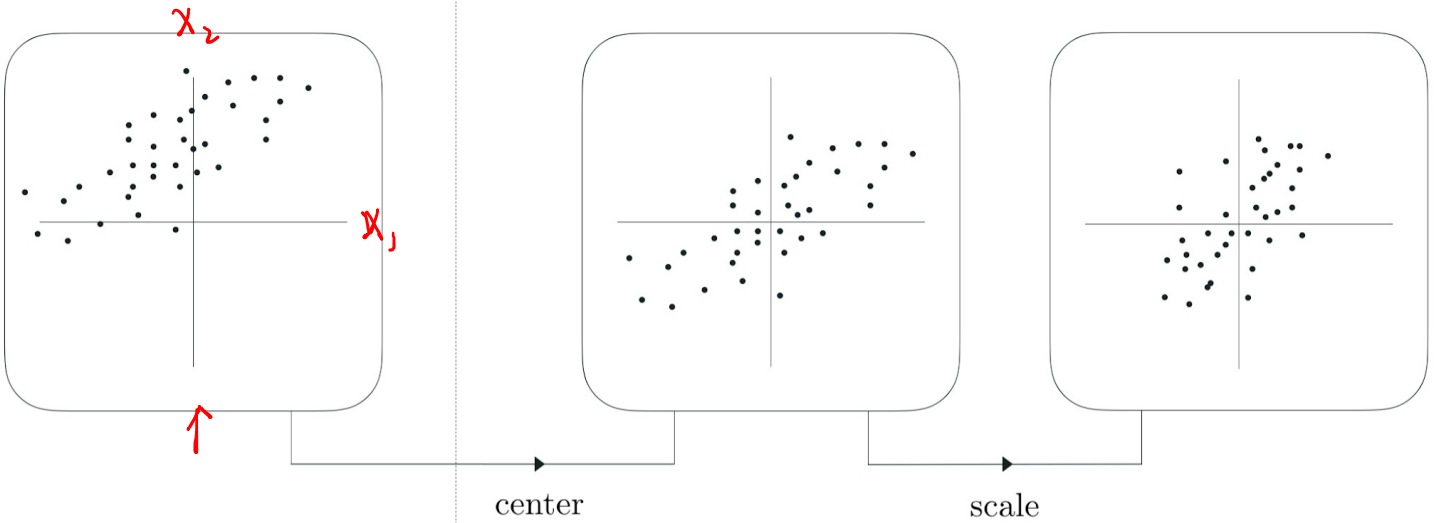
$y = w_1 x + w_0$

$\frac{\partial g}{\partial w_1} = \cdots \cdot X^{2000} = \cdots$; $\frac{\partial g}{\partial w_0} = (\quad) \cdot 1 \approx$!

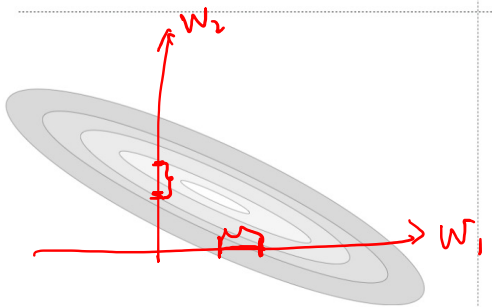
Python implementation:
https://colab.research.google.com/github/jermwatt/machine_learning_refined/blob/main/notes/9_Feature_engineer_select/9_3_Scaling.ipynb

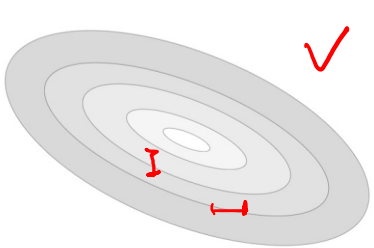
- First and foremost, they is indicative of a more general truth regarding machine learning cost functions: ***normalizing the input features of a dataset by mean centering and scaling by the standard deviation of each input will results in a cost function with less elliptical and more 'circular' contours*.**
- This includes regression, (two-class / multi-class) classification, as well as unsupervised learning cost functions.

data space



cost function





✓ contour
circular

- Why is it the case that making the contours of a cost function less 'elliptical' and more 'circular' helps make optimization easier when using a first order optimization method like gradient descent?
- Because the gradient descent direction always points perpendicular to the contours of a cost function.

- With more circular contours the gradient descent direction starts pointing more in the direction of the cost function minima, making each gradient descent step much more effective.
- This also helps explain why **we can typically use a much larger steplength parameter** α when minimizing a cost function of standard normalized data - since the gradient descent direction points in a better direction we can freely travel in it much further at each step.

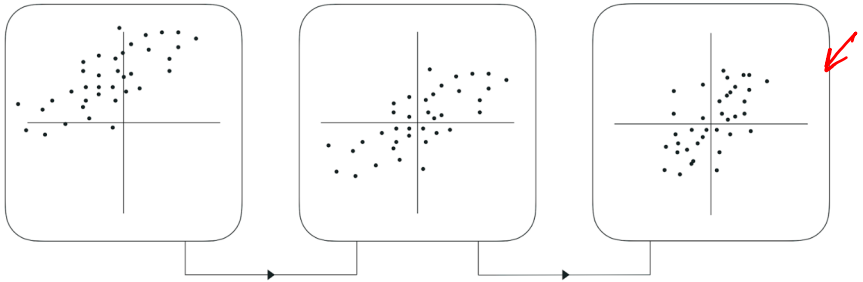
9.5 Feature scaling via PCA sphering

- Before discussing PCA-sphereing, let us recall a few key notations and ideas from our discussion of PCA itself.
- Stacking our input data together column-wise we create our $N \times P$ data matrix $\mathbf{x}$
- We then denote $\frac{1}{P}\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{N \times N}$ the regularized covariance matrix of this data
- And $\frac{1}{P}\mathbf{X}\,\mathbf{X}^T + \lambda \mathbf{I}_{N \times N} = \mathbf{V}\,\mathbf{D}\,\mathbf{V}^T$ its eigenvalue/vector decomposition.

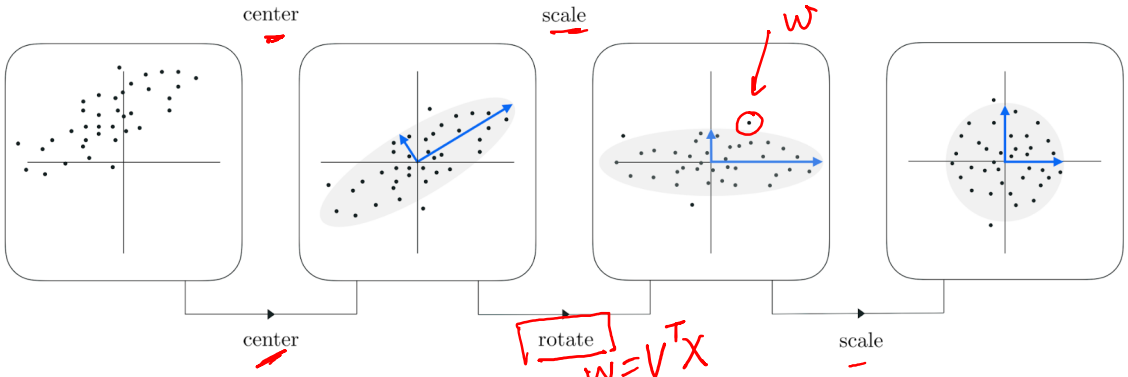
- Also recall that when performing PCA we first **mean-center** our dataset.
- We then aim to represent each of our mean-centered datapoints
- In the space spanned by the principal components we can represent the entire set of transformed mean-centered data as

(PCA transformed data) $$\mathbf{W} = \mathbf{V}^T\mathbf{X}.$$

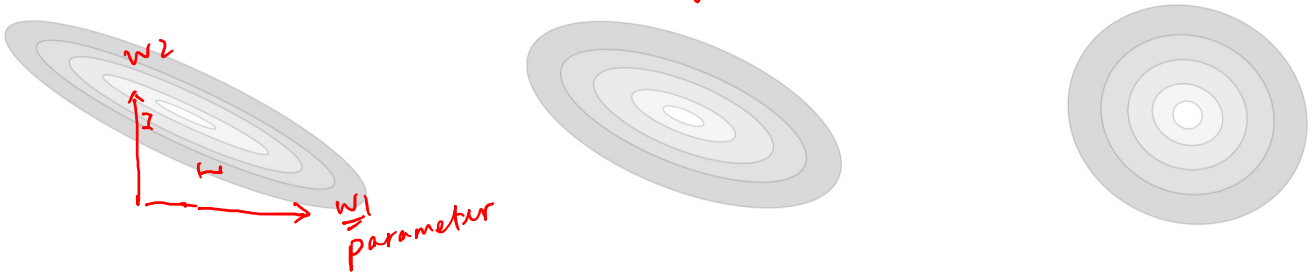
encoding



⇐ Standard normalization



$w = V^T X$



original cost

standard-normalized cost

PCA-sphered cost

- Because PCA sphering first rotates the data prior to scaling, **it typically results in more compact transformed data, and a transformed cost function with more 'circular' contours.**

Standard normalization scheme:

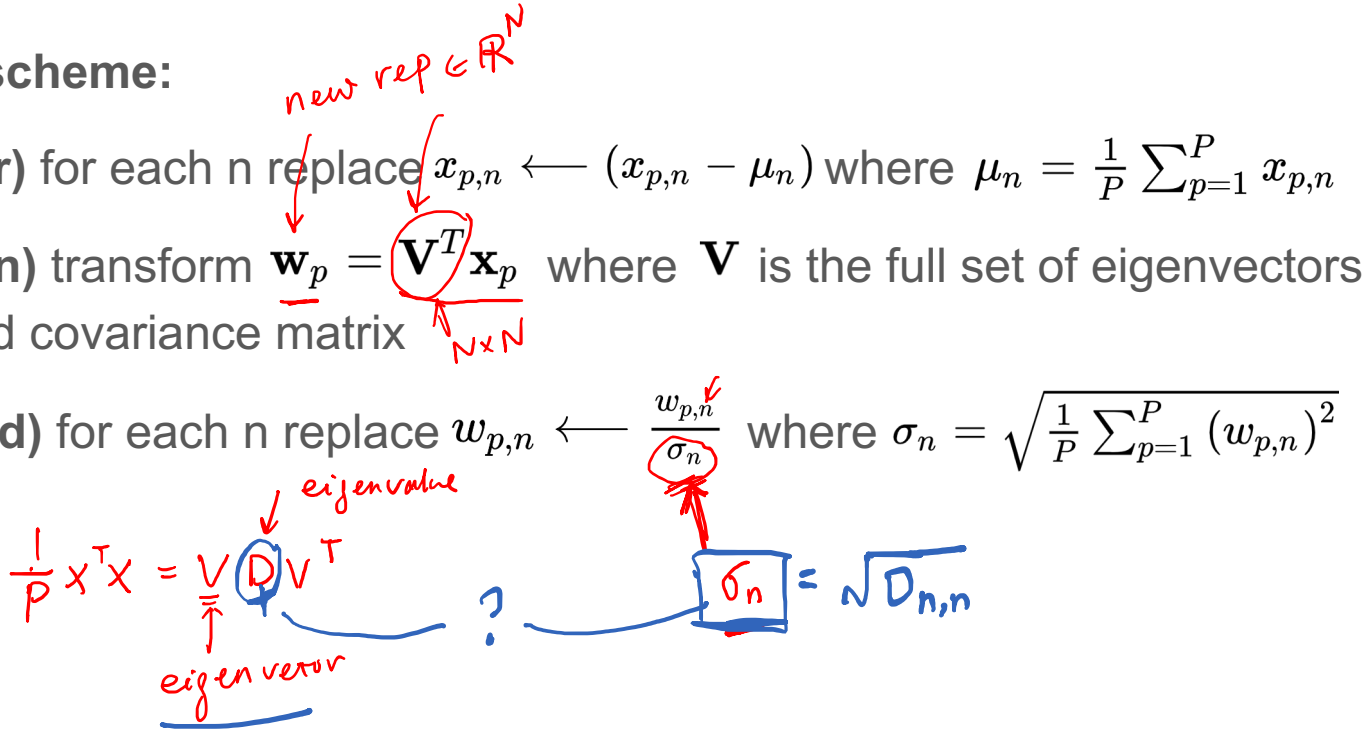
1. **(mean center)** for each n replace $x_{p,n} \longleftarrow (x_{p,n} - \mu_n)$ where $\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$
2. **(divide off std)** for each n replace $x_{p,n} \longleftarrow \frac{x_{p,n}}{\sigma_n}$ where $\sigma_n = \sqrt{\frac{1}{P}\sum_{p=1}^{P} (x_{p,n})^2}$

PCA-sphering scheme:

1. **(mean center)** for each n replace $x_{p,n} \longleftarrow (x_{p,n} - \mu_n)$ where $\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$

2. **(PCA rotation)** transform $\mathbf{w}_p = \mathbf{V}^T\mathbf{x}_p$ where $\mathbf{V}$ is the full set of eigenvectors of the regularized covariance matrix

3. **(divide off std)** for each n replace $w_{p,n} \longleftarrow \frac{w_{p,n}}{\sigma_n}$ where $\sigma_n = \sqrt{\frac{1}{P}\sum_{p=1}^{P}(w_{p,n})^2}$



PCA-sphering scheme:

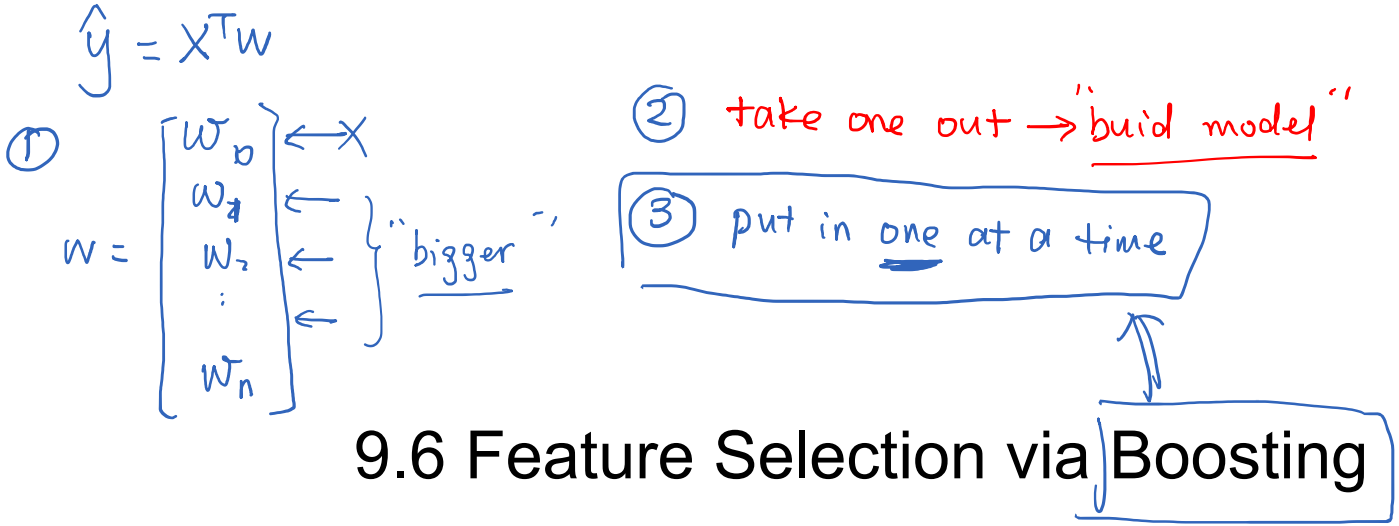
1. **(mean center)** for each n replace $x_{p,n} \longleftarrow (x_{p,n} - \mu_n)$ where $\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n}$

2. **(PCA rotation)** transform $\mathbf{w}_p = \mathbf{V}^T\mathbf{x}_p$ where $\mathbf{V}$ is the full set of eigenvectors of the regularized covariance matrix

3. **(divide off std)** for each n replace $w_{p,n} \longleftarrow \frac{w_{p,n}}{d_n^{1/2}}$ where $d_n^{1/2}$ is the square root of the n^{th} eigenvalue of the regularized covariance matrix

$$\mathbf{X} \longleftarrow \mathbf{D}^{-\frac{1}{2}}\mathbf{V}^T\mathbf{X}.$$

$$D^{-\frac{1}{2}} = \frac{1}{D^{1/2}} = \frac{1}{\sqrt{D}}$$



9.6 Feature Selection via Boosting

- Now at the start of the weight learning process ***we begin with just the bias***.
- That is we begin with a model - which we will denote as model_0 - that consists of the bias w_0 alone

$$\text{model}_0\left(\mathbf{x}, \mathbf{w}\right) = w_0.$$

- To begin we first tune the bias parameter w_0 by minimizing an appropriate cost over this variable alone.
- For example, if we are performing regression employing the Least Squares cost and we minimize

$$\frac{1}{P}\sum_{p=1}^{P}\left(\text{model}_0\left(\mathbf{x},\mathbf{w}\right)-y_p\right)^2=\frac{1}{P}\sum_{p=1}^{P}\left(w_0-y_p\right)^2$$

- This gives the optimal value for our bias $w_0 \longleftarrow w_0^\star$. Thus our starting model is now

$$\text{model}_0\left(\mathbf{x}, \mathbf{w}\right) = w_0^\star.$$

- Next, at the first round of boosting, in order to determine the most important feature-touching weight $w_1, w_2, \ldots, w_N$ we *try out each one*...
- ...by minimizing a cost over each one individually, having already set the bias optimally.
- That is we minimize N cost functions over a single feature-touching weight alone.

- So e.g., in the case of Least Squares regression the n^{th} of these subproblems takes the form

$$w_n^* = \underset{w_n}{\text{minimize}} \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}_0\left(\mathbf{x}_p, \mathbf{w}\right) + w_n x_{n,p} - y_p\right)^2$$
$$= \frac{1}{P}\sum_{p=1}^{P}\left(w_0^\star + x_{n,p}w_n - y_p\right)^2.$$

- Again, the bias weight has already been set optimally and is not tuned in each subproblem above - we only tune the weight w_n.

- The weight that produces the smallest value is the one that helps best explain the relationship between the input and output of our dataset.
- It is therefore most important feature-touching weight we learn.
- Denoting this weight as w_{s_1} , we then fix it at its optimally determined value $w_{s_1} \longleftarrow w_{s_1}^{\star}$ (discarding all other weights tuned in each of these subproblems).

- We update our `model` accordingly.
- We call our updated `model` model_1 which is a sum of our optimal bias and this newly determined optimal feature-touching weight

$$\text{model}_1\left(\mathbf{x}, \mathbf{w}\right) = \text{model}_0\left(\mathbf{x}, \mathbf{w}\right) + x_{s_1} w^\star_{s_1} = w^\star_0 + x_{s_1} w^\star_{s_1}.$$

w_0^*

$+ x_n \cdot w_n^*$

$n = 1, 2 \ldots N,\ n \neq S_1$

$N-1$

- At the second round of boosting we determine the *second* most important feature-touching weight.
- To do this we then sweep through the remaining N-1 weights (i.e., excluding w_0 and w_{s_1} which we have already tuned).
- We try out each one individually by minimizing a cost over each weight independently.

- In the case of Least Squares regression n^{th} such cost looks like

$$w_n^* = \arg\min \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}_1\left(\mathbf{x}_p, \mathbf{w}\right) + w_n x_{n,p} - y_p\right)^2$$

$$= \frac{1}{P}\sum_{p=1}^{P}\left(w_0^\star + x_{s_1} w_{s_1}^\star + x_{n,p} w_n - y_p\right)^2.$$

$\{w_1^*, \dots w_N^*\} \Rightarrow$ min cost

- Again with each of these subproblems $w_0^\star$ and $w_{s_1}^\star$ have already been set optimally, we only tune the weight w_n in each instance.

- The weight producing the *smallest* value from these $(N-1)$ subproblems tells us the second most important feature.
- Denoting the cost minimizing weight w_{s_2}, we then *fix* the value of this weight at its optimally determined value $w_{s_2} \longleftarrow w_{s_2}^{\star}$
- Updating our model accordingly, calling it model_2, we have

$$\text{model}_2\left(\mathbf{x}, \mathbf{w}\right) = \text{model}_1\left(\mathbf{x}, \mathbf{w}\right) + x_{s_2} w_{s_2}^{\star} = w_0^{\star} + x_{s_1} w_{s_1}^{\star} + x_{s_2} w_{s_2}^{\star}.$$

- More generally, at the M^{th} round of boosting, to determine the M^{th} most important feature-touching weight we follow the same pattern.
- When we do this we have already determined the optimal setting of our bias and the top $M-1$ most important feature-touching weights, our model taking the form

$$\begin{aligned}\text{model}_{M-1}(\mathbf{x}, \mathbf{w}) &= \text{model}_{M-2}(\mathbf{x}, \mathbf{w}) + x_{s_{m-1}} w^\star_{s_{M-1}} \\ &= w^\star_0 + x_{s_1} w^\star_{s_1} + \cdots + x_{s_{M-1}} w^\star_{s_{M-1}}.\end{aligned}$$

- Here $w^\star_{M-1}$ denotes our optimally tuned $(M-1)^{th}$ most important feature-touching weight.

- We then must setup and solve $(N - M + 1)$ subproblems, one for each feature-touching weight we have not yet resolved.

N Features
M-1 Features most important

- For example, in the case of Least Squares regression the n^{th} of these takes the form

$$\frac{1}{P}\sum_{p=1}^{P}\left(\text{model}_{M-1}\left(\mathbf{x}_p, \mathbf{w}\right) + w_n x_{n,p} - y_p\right)^2$$

$n \neq S_1, S_2 \cdots S_{M-1}$

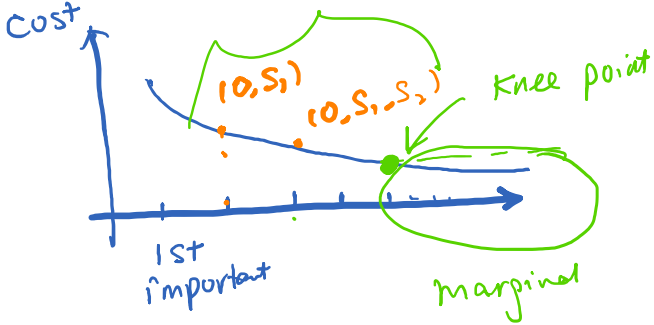
- Here again in each case we only tune the individual weight w_n .

- Denoting this weight w_{s_M} we then fix its optimal value $w_{s_M} \longleftarrow w^{\star}_{s_M}$ and add its contribution to the running model as

$$\text{model}_M\left(\mathbf{x}, \mathbf{w}\right) = \text{model}_{M-1}\left(\mathbf{x}, \mathbf{w}\right) + x_{s_M} w^{\star}_{s_M}.$$

- Given that we have N input features we can continue until $M \leq N + 1$.
- Note too how in building model_M we have constructed a sequence of models .

$$\left\{\text{models}_m\right\}_{m=1}^{M}$$



- This method of model building (via the construction of a simpler series of models) adding one feature at-a-time...
- ...and tuning only the parameters of the feature added keeping all others fixed at their previously tuned values...
- ...is referred to as *boosting*, or similarly as *forward stage-wise selection*.

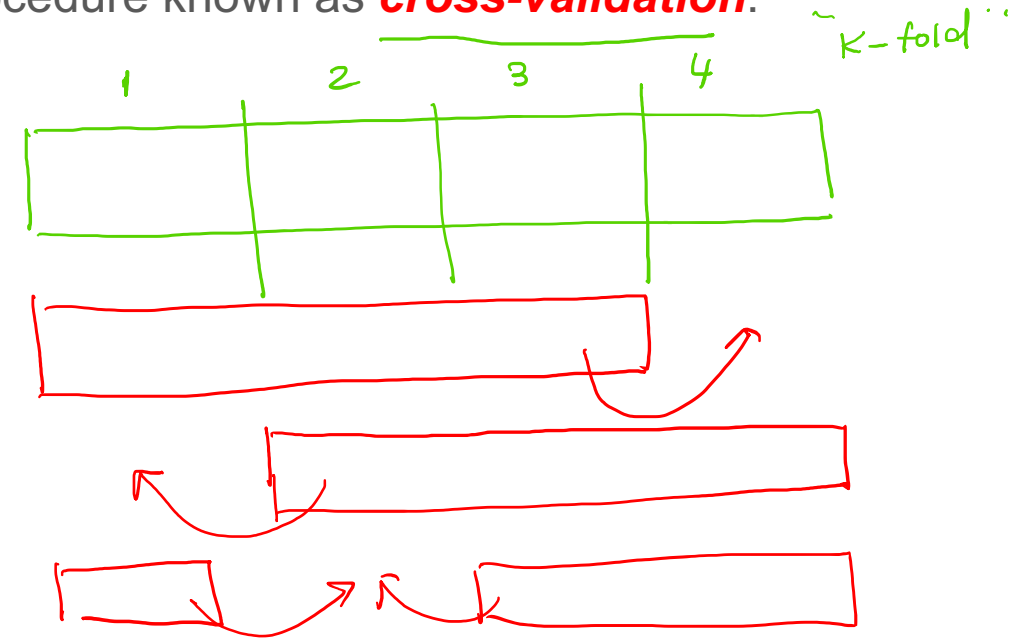
- Because we are trying to determine the *importance* of each input feature here that before we begin the process of boosting, we ***always need to standard normalize the input to our dataset***
- By normalizing each input feature distribution we can fairly compare each input feature's contribution and determine the importance of each.

How many features should be selected?

- Because feature selection is done for the purposes of *human interpretation* the number of features M to select can be based on several factors.
- Option 1: A benchmark value for M can be hand chosen based on the desire to explore a dataset, and the procedure halted once this number of rounds have completed.
- Option 2: One can also halt exploration when adding additional features to the model results in very little decrease in the cost, as most of the correlation between inputs and outputs has already been explained.

Knee point

Option 3: Finally, M can be chosen entirely based on the sample statistics of the dataset via a procedure known as ***cross-validation***.



Limitation of boosting

- While each round of boosting demands we solve a number of subproblems, each one is a minimization with respect to *only a single weight* and is therefore very easy to solve virtually regardless of the local optimization scheme used.
- This makes *boosting* a computationally effective approach to feature selection, and allows it to scale to datasets with large N.
- A weakness inherent in doing this - that is in determining *one feature-touching weight at a time* - is that **interactions between features / feature-touching weights can be potentially missed.**

- To capture potentially missed interactions between groups of features / weights one might naturally extend the boosting idea and try to add *a group of R feature-touching weights* at each round instead of just one weight.
- However a quick calculation shows that this idea would quickly fail to scale. In order to determine the first best group of R feature-touching weights at the the first stage of this approach we would need to try out *every combination of R weights* by solving a subproblem for each.

- The problem is that there are combinatorially many subgroups of size R.
- More precisely there are $\binom{N}{R} = \frac{R!}{N!(N-R)!}$ subproblems, and thus equally many subproblems to solve (this is far too many problems to solve in practice e.g., $\binom{100}{5} = 75,287,520$).

9.7 Feature Selection via Regularization

- The simple linear combination of two functions f_1 and f_2

$$f(\mathbf{w}) = f_1(\mathbf{w}) + \lambda f_2(\mathbf{w})$$

cost fun.

regularization term

- is often referred to as **regularization** in the parlance of machine learning.
- More specifically the function f_2 is called a regularizer in the jargon of machine learning, since by adding it to function f_1 we adjust its shape
- and $\lambda \geq 0$ is referred to as a penalty or regularization parameter.

Suppose we added a generic function $h(\cdot)$ to one of our supervised learning functions

e.g., the Least Squares $g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$, giving a regularized linear combination of our cost function

$$f(\mathbf{w}) = g(\mathbf{w}) + \lambda h(\mathbf{w}).$$

$\|w\|_2^2$

Regularization using the ℓ_0 norm

- The ℓ_0 vector norm, written as $\|\mathbf{w}\|_0$, measures 'magnitude' as

$$\|\mathbf{w}\|_0 = \text{number of non-zero entries of } \mathbf{w}.$$

- If we regularize a cost g using it

$$f(\mathbf{w}) = g(\mathbf{w}) + \lambda \|\mathbf{w}\|_0$$

Regularization using the ℓ_1 norm

- The ℓ_1 vector norm, written as $\|\mathbf{w}\|_1$, measures 'magnitude' as

$$\|\mathbf{w}\|_1 = \sum_{n=0}^{N} |w_n| .$$

- If we regularize a regression cost g (or - as we will see - any machine learning cost) using it

$$f(\mathbf{w}) = g(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

- notice what we are doing: we are penalizing the regularized cost *based on the sum of the absolute value of the entries of $\mathbf{w}$.

https://scikit-learn.org/stable/modules/linear_model.html#lasso

1.1.3. Lasso

The `Lasso` is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer non-zero coefficients, effectively reducing the number of features upon which the given solution is dependent. For this reason, Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero coefficients (see Compressive sensing: tomography reconstruction with L1 prior (Lasso)).

Mathematically, it consists of a linear model with an added regularization term. The objective function to minimize is:

$$\min_w \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha ||w||_1$$

The lasso estimate thus solves the minimization of the least-squares penalty with $\alpha||w||_1$ added, where α is a constant and $||w||_1$ is the ℓ_1-norm of the coefficient vector.

The implementation in the class `Lasso` uses coordinate descent as the algorithm to fit the coefficients. See Least Angle Regression for another implementation:

```
>>> from sklearn import linear_model
>>> reg = linear_model.Lasso(alpha=0.1)
>>> reg.fit([[0, 0], [1, 1]], [0, 1])
Lasso(alpha=0.1)
>>> reg.predict([[1, 1]])
array([0.8])
```

The function `lasso_path` is useful for lower-level tasks, as it computes the coefficients along the full path of possible values.

Example: Regularization using the ℓ_2 norm

$$g(w) = \text{loss}(\quad) + \lambda_1 \cdot \|w\|_1 + \lambda_2 \|w\|_2$$

"sparsity"

"magnitude small"

- The ℓ_1 vector norm, written as $\|\mathbf{w}\|_2$, measures 'magnitude' as

$$\|\mathbf{w}\|_2 = \sqrt{\left(\sum_{n=0}^{N} w_n^2\right)}.$$

- If we regularize a regression cost g (or - as we will see - any machine learning cost) using it

$$f(\mathbf{w}) = g(\mathbf{w}) + \lambda \|\mathbf{w}\|_2$$

- notice what we are doing: we are *penalizing* the regularized cost based on the sum of squares of the entries of $\mathbf{w}$.

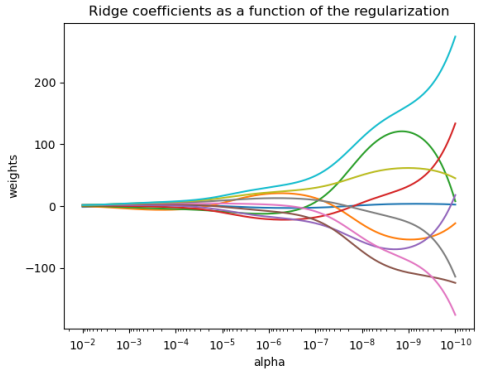
1.1.2. Ridge regression and classification

1.1.2.1. Regression

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.



As with other linear models, **Ridge** will take in its `fit` method arrays `X`, `y` and will store the coefficients w of the linear model in its `coef_` member:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

sklearn.linear_model.Ridge

class sklearn.linear_model.**Ridge**(*alpha=1.0*, ***, *fit_intercept=True*, *copy_X=True*, *max_iter=None*, *tol=0.0001*, *solver='auto'*, *positive=False*, *random_state=None*) [source]

Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the User Guide.

Feature selection via ℓ_1 regularization

- Of the two sparsity-inducing norms described here the ℓ_0 norm - while promoting sparsity to the greatest degree - is also the most challenging to employ since it is *discontinuous*.
- This makes the minimization of an ℓ_0 regularized cost function quite difficult.
- While the ℓ_1 norm induces sparsity to less of a degree, it is ***convex and continuous*** and so we have no practical problem minimizing a cost function where it is used as regularizer.
- **Because of this practical advantage the ℓ_1 norm is by far the more commonly used regularizer for feature selection.**

- Using our individual notation for the bias and feature-touching weights

$$\text{(bias): } b = w_0 \qquad \text{(feature-touching weights): } \quad \boldsymbol{\omega} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}.$$

- we can write this general ℓ_1 regularized cost function equivalently as

$$f\left(b, \boldsymbol{\omega}\right) = g\left(b, \boldsymbol{\omega}\right) + \lambda \left\|\boldsymbol{\omega}\right\|_1.$$

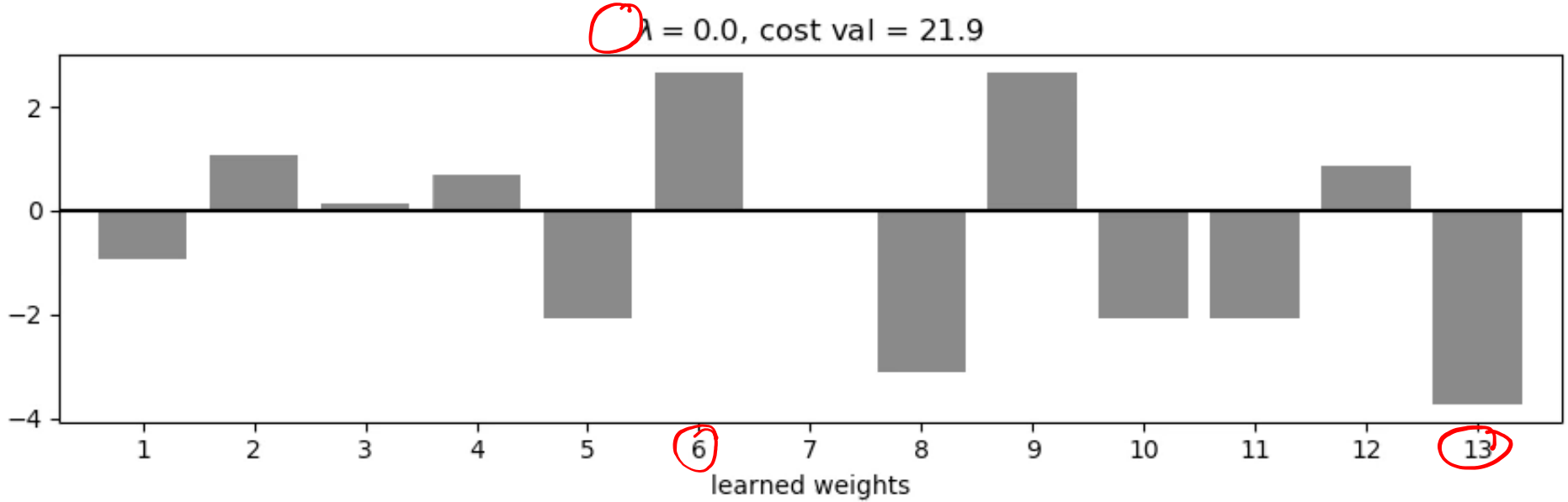
What value should λ be set too?

- Because feature selection is done for the purposes of *human interpretation* the value of λ can be based on several factors.
- Option 1: A benchmark value for λ can be hand chosen based on the desire to explore a dataset, finding a value that provides sufficient sparsity while retaining a low cost value.
- Option 2: λ can be chosen entirely based on the sample statistics of the dataset via a procedure known as *cross-validation*.

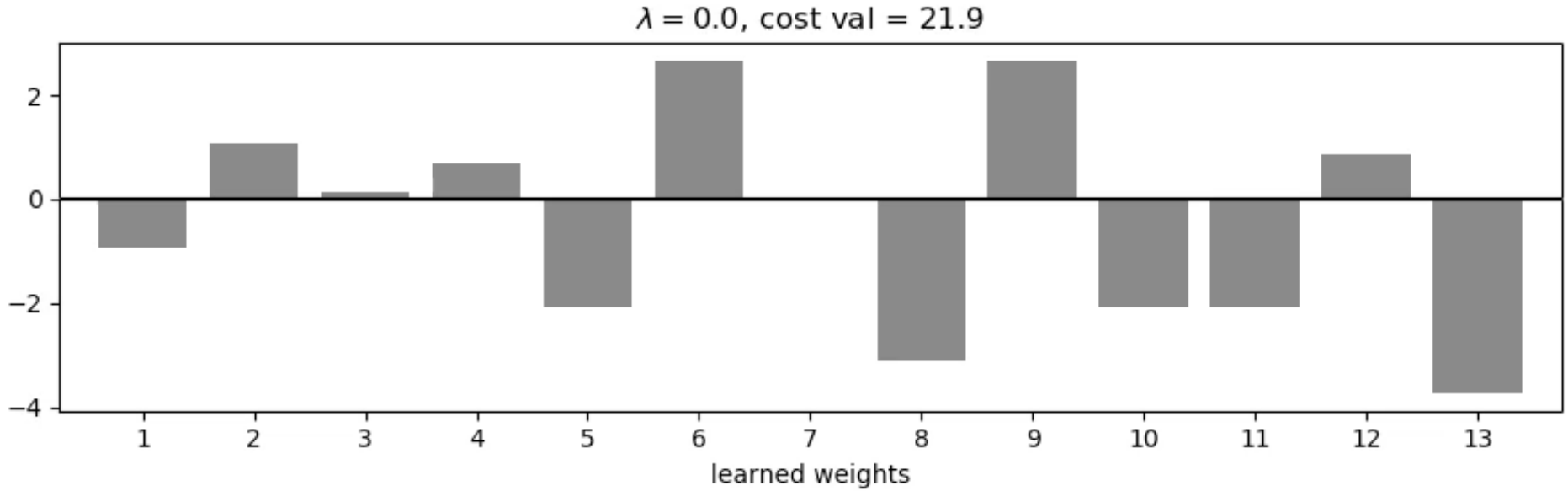
- Regardless of how we select λ , note that because we are trying to determine the *importance* of each input feature here that before we begin the process of regularization we ***always need to standard normalize the input to our dataset***.
- By normalizing each input feature distribution, we can fairly compare each input feature's contribution and determine the importance of each.

Exploring features for predicting housing prices via ℓ_1 regularized regression

- In this example we use the Boston Housing dataset.
- This dataset consists of P=506 datapoints with **N=13 input features** relating various statistics about homes in the Boston area to their median values (in U.S. dollars).
- We use ℓ_1 regularization with 50 evenly spaced values for λ in the range $[0, 130]$


λ = 0.0, cost val = 21.9
2
0
−2
−4
1
2
3
4
5
6
7
8
9
10
11
12
13
learned weights

- Below we show the an analogous run - using the same range of values for λ - employing the ℓ_2 norm.
- This does not sparsify the resulting weights like the ℓ_1 norm does, and so results in optimal weights that are less human interpretable.


λ = 0.0, cost val = 21.9
2
0
−2
−4
1
2
3
4
5
6
7
8
9
10
11
12
13
learned weights

Comparing regularization and boosting

- While boosting is an efficient greedy scheme, the *regularization* idea detailed above can be computationally intensive to perform since for each value of λ tried a full run of local optimization must be completed.
- On the other hand, while boosting is a 'bottom-up' approach that identifies individual features one-at-a-time, regularization takes a more 'top-down' approach and identifies important features *all at once*.
- In principle this allows regularization to uncover groups of important features that may be correlated in such an interconnected way with the output that they will be missed by boosting.

Summary

- Histogram features (one-hot encoding)
- Feature scaling via standard normalization
- Feature scaling via PCA sphering
- Feature selection via Boosting
- Feature selection via Regularization

model $(\dot{X}, w)$

$\dot{X} \in \mathbb{R}^{(1+N)\times P}$, N: # features

$w \in \mathbb{R}^{(N+1)\times C}$, C: # classes

① features not numerical

$x_{raw} \longrightarrow x_{numerical}$

② some features have large mean/variance

$\dot{X} = \begin{bmatrix} 1 \\ X \end{bmatrix} \leftarrow \begin{cases} X = \text{Standerd-norm}(X_{raw}) \\ X = \text{PCA-sphering}(X_{raw}) \end{cases}$

③ most important features

{ Boosting
Regularization : "l_1-norm"

Midterm (sneak peak) - More details announced later

- Date: Thursday, Nov 16th, 5:30 PM-6:45 PM
- In person paper-based exam (closed book, no electronic devices allowed)
  - In person: MOR1N375, MOR20131, *double spacing*
  - Remote section: on zoom with camera ON; same time
    - Allow extra mins to take photos of the solutions and upload to gradescope
- Format:
  - **50 T/F questions** + Optional statement on why True/False. [Tentative]
  - Similar to the T/F questions we have in the class
    - Will give you a sample of T/F questions before midterm
    - Will give you a review of the core concepts to know before midterm
- Solid excuses (doctor-proof on sickness, religion, disability services, time zone problem for online students only etc.) needed for personalizing the exam time.