ECBM E4040
Neural Networks and Deep Learning

Machine Learning
Data Representations and Algorithms

Zoran Kostić
Columbia University
Electrical Engineering Department &
Data Sciences Institute

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

References and Acknowledgments

- Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, http://www.deeplearningbook.org/
- John Paisley, COMS W4721 Machine Learning for Data Science, http://www.columbia.edu/~jwp2128/Teaching/W4721/Spring2017/W4721Spring2017.html
- A Course in Machine Learning, by Hal Daumé III, http://ciml.info/
- Machine Learning Exercises by John Wittenauer https://github.com/jdwittenauer/ipython-notebooks
- Nvidia DLI Teaching Kit

Outline - Machine Learning Basics

- Data Representation
  - PCA, t-SNE Summary
- Curse of dimensionality
- Algorithms
  - Building algorithms, practical considerations
  - Logistic Regression
  - Support Vector Machines
  - Softmax

Traditional Pattern Recognition vs. Deep Learning

(1) The traditional model of pattern recognition (since the late 50's)

Fixed/engineered features (or fixed kernel) + trainable classifier

Hand-crafted Feature Extractor → "Simple" Trainable Classifier →

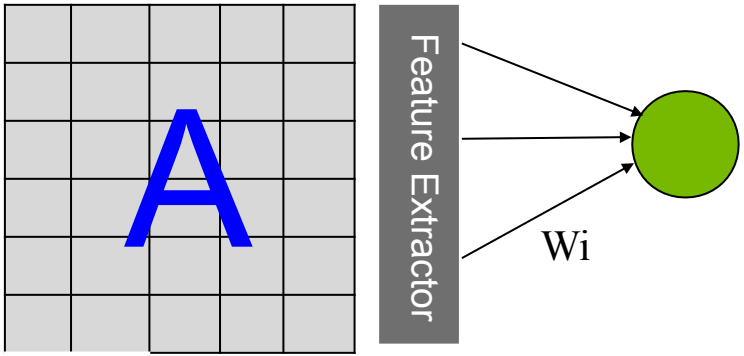
(2) End-to-end learning / Feature learning / Deep learning

Trainable features (or kernel) + trainable classifier

Trainable Feature Extractor → Trainable Classifier →

Traditional Pattern Recognition vs. Deep Learning

- The first learning machine: the Perceptron.
- Built at Cornell in 1960.
- The Perceptron was a linear classifier on top of a simple feature extractor.
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.



$$y = sign\left(\sum_{i=1}^{N} W_i F_i(x) + b\right)$$

Designing a feature extractor requires considerable efforts by experts.

Data Representation

Principal Component Analysis

Unsupervised Learning - Data Representation: A Central Theme in Deep Learning

Unsupervised learning algorithms are those that experience only “feature" but not a supervision signal.

Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples.

Unsupervised Learning - Data Representation: A Central Theme in Deep Learning

Unsupervised learning is usually associated with density estimation,

- learning to draw samples from a distribution, learning to denoise data from some distribution,
- finding a manifold that the data lies near,
- or clustering the data into groups of related examples.

Unsupervised Learning
Learning a Representation of Data

A classic unsupervised learning task is to find the "best" representation of the data.

We are looking for a representation that preserves as much information about x as possible,

while obeying some penalty or constraint aimed at keeping the representation simpler or more accessible than x itself.

Unsupervised Learning
Learning a Representation of Data

What is a simpler representation?

- **Low-dimensional representations attempt to compress as much information about x as possible in a smaller representation.**
- **Sparse representations embed the dataset into a representation whose entries are mostly zeroes for most inputs.**
- **Independent representations attempt to disentangle the sources of variation underlying the data distribution such that the dimensions of the representation are statistically independent.**

Principal Component Analysis (PCA)

PCA is “an orthogonal linear transformation that transfers the data to a new coordinate system

- such that the greatest variance by any projection of the data comes to lie on the first coordinate (principal component),
- the second greatest variance lies on the second coordinate (principal component), and so on.”

Principal Component Analysis (PCA)

PCA is used to reduce dimensions of data without much loss of information.

Applications: machine learning, signal processing, image compression.

Singular Value Decomposition
SVD (requires understanding of)

Linear Algebra:

- **Scalars, vectors, matrices and tensors**
- **Multiplication**
- **Matrix inverse**
- **Linear dependence and span**
- **Norms**
- **Diagonal, symmetric, unit, orthogonal**
- **Eigen-decomposition, SVD, Moore-Penrose inverse**
- **Trace**

Book: Chapter 2.

https://www.cc.gatech.edu/~dellaert/ftp/svd-note.pdf

Principal Component Analysis (PCA)

Here we will demonstrate that the PCA decorrelates the data representation of an (n, m) design matrix $\mathbf{X}$. We will assume that the data has a mean of zero, $\mathbb{E}\mathbf{x} = 0$. The unbiased sample covariance matrix associated with $\mathbf{X}$ is given by:

$$\text{Var}(\mathbf{x}) = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}.$$

Book: sections 2.12, 5.8.1

Principal Component Analysis (PCA)

We consider the SVD of the matrix $\mathbf{X}$,

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\mathbf{\Sigma}$ is an (n, m)-dimensional rectangular diagonal matrix with the singular values of $\mathbf{X}$ on the main diagonal, $\mathbf{U}$ is an (n, n) matrix whose columns are orthonormal (i.e. unit length and orthogonal) and $\mathbf{V}$ is an (m, m) matrix also composed of orthonormal column vectors.

Principal Component Analysis (PCA)

$$(n-1)\text{Var}(\mathbf{x}) = \mathbf{X}^T\mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

and, due to orthonormality $\mathbf{U}^T\mathbf{U} = \mathbf{I}$,

$$\text{Var}(\mathbf{x}) = \frac{1}{n-1}\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T.$$

Similarly, for $\mathbf{z} = \mathbf{V}\mathbf{x}$, we have

$$\text{Var}(\mathbf{z}) = \frac{1}{n-1}\mathbf{Z}^T\mathbf{Z} = \frac{1}{n-1}\mathbf{V}^T\mathbf{X}^T\mathbf{X}\mathbf{V} = \frac{1}{n-1}\mathbf{\Sigma}^2,$$

since $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. The above analysis shows that when we project the data $\mathbf{x}$ to $\mathbf{x}$, via the linear transformation $\mathbf{V}$, the resulting representation has a diagonal covariance matrix (as given by $\mathbf{\Sigma}^2$) which immediately implies that the individual elements of $\mathbf{z}$ are mutually uncorrelated.

PCA

PCA can be thought of as:

- A method for compressing data
- An unsupervised learning algorithm that learns the representation of data

Principal Component Analysis (PCA)
An Example

Assume attributes A1 and A2, and n training examples.

x's denote values of A1 and y's denote values of A2 over the training examples.

Variance of an attribute:

$$\text{var}(A_1) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{(n-1)}$$

Principal Component Analysis (PCA)
An Example

Covariance of two attributes:

$$\text{cov}(A_1, A_2) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

If covariance is positive, the attributes (data dimensions) increase together. If negative, as one increases, the other decreases. If zero, the dimensions are independent of each other.

Principal Component Analysis (PCA)
An Example

Suppose the data has n dimensions (N attributes) $A_1, A_2 \ldots, A_N$.

Covariance matrix is

$$C^{n \times n} = (c_{i,j}), \text{ where } c_{i,j} = \text{cov}(A_i, A_j)$$

Principal Component Analysis (PCA)
An Example - Data

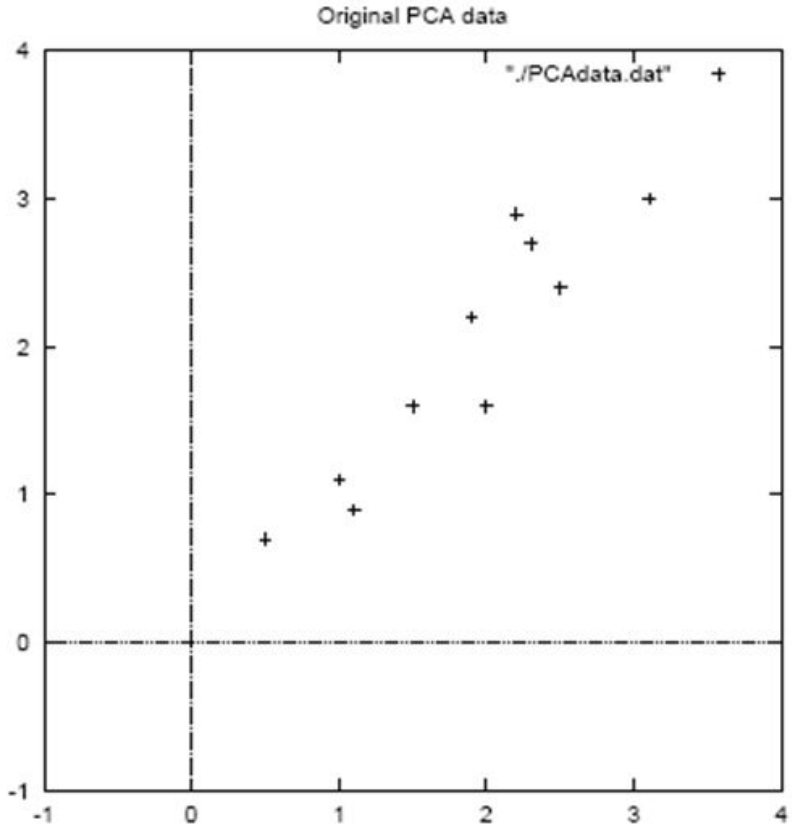
	Hours(H)	*Mark(M)*
Data	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
	19	70
	16	66
	20	80
Totals	167	749
Averages	13.92	62.42

Covariance:

H	M	$(H_i - \bar{H})$	$(M_i - \bar{M})$	$(H_i - \bar{H})(M_i - \bar{M})$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
19	70	5.08	7.58	38.51
16	66	2.08	3.58	7.45
20	80	6.08	17.58	106.89
Total				1149.89
Average				104.54

Table 2.2: 2-dimensional data set and covariance calculation

Principal Component Analysis (PCA)
An Example - Data Plot



Principal Component Analysis (PCA)
An Example

Covariance matrix

$$\begin{pmatrix} \text{cov}(H,H) & \text{cov}(H,M) \\ \text{cov}(M,H) & \text{cov}(M,M) \end{pmatrix} = \begin{pmatrix} \text{var}(H) & 104.5 \\ 104.5 & \text{var}(M) \end{pmatrix}$$

$$= \begin{pmatrix} 47.7 & 104.5 \\ 104.5 & 370 \end{pmatrix}$$

Principal Component Analysis (PCA)
Example - Eigenvectors

Let **M** be an n x n matrix.

- **v** is an eigenvector of **M** if **M** x **v** = λ**v**
- λ is called the eigenvalue associated with **v**

For any eigenvector **v** of **M** and scalar a, **M** x a**v** = λa**v**. This unit vector can be used, where the components satisfy $\sqrt{v_1^2 + \ldots + v_n^2} = 1$.

If M has any eigenvectors, it has n of them, and they are orthogonal to one another.

Thus eigenvectors can be used as a new basis for a n-dimensional vector space.

Principal Component Analysis (PCA)
Example - Adjust Data to Zero Mean

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Mean: 1.81 1.91

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Mean: 0.0 0.0

Principal Component Analysis (PCA)
Example - Covariance Matrix

Covariance Matrix

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

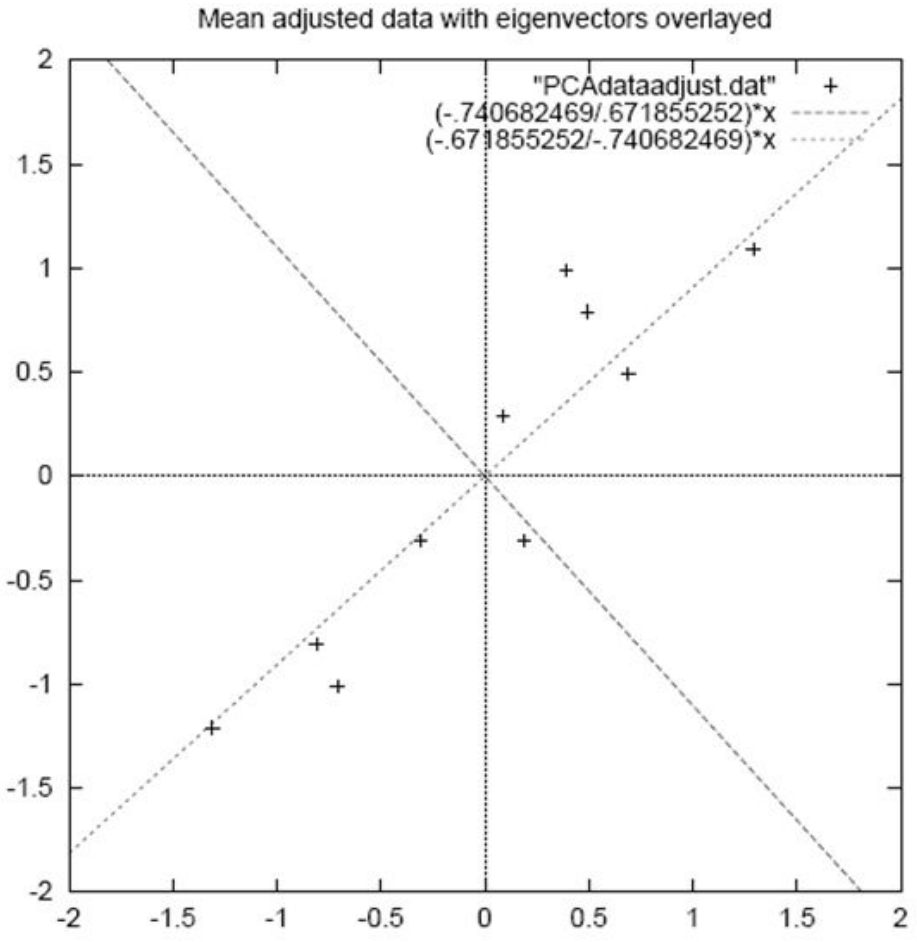
Eigenvalues and eigenvectors

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

Principal Component Analysis (PCA) Example

The eigenvector with the largest eigenvalue traces the linear pattern in the plot of the data.



Principal Component Analysis (PCA)
Example

Order eigenvectors by the order of eigenvalues, from highest to lowest.

$$\mathbf{v}_1 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix} \quad \lambda = 1.28402771$$

$$\mathbf{v}_2 = \begin{pmatrix} -.735178956 \\ .677873399 \end{pmatrix} \quad \lambda = .0490833989$$

In general, there are n components. To reduce the dimensionality to p, ignore n-p components at the bottom of the list.

Principal Component Analysis (PCA) Example - New Feature Vectors

The new feature vector is based on the eigenvectors

$$FeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

or reduced dimension feature vector:

$$FeatureVector2 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix}$$

Principal Component Analysis (PCA)
Example - New Representation of Data

TransformedData = RowFeatureVector x RowDataAdjust

$$RowFeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

$$RowFeatureVector2 = \begin{pmatrix} -.677873399 & -.735178956 \end{pmatrix}$$

$$RowDataAdjust = \begin{pmatrix} .69 & -1.31 & .39 & .09 & 1.29 & .49 & .19 & -.81 & -.31 & -.71 \\ .49 & -1.21 & .99 & .29 & 1.09 & .79 & -.31 & -.81 & -.31 & -1.01 \end{pmatrix}$$

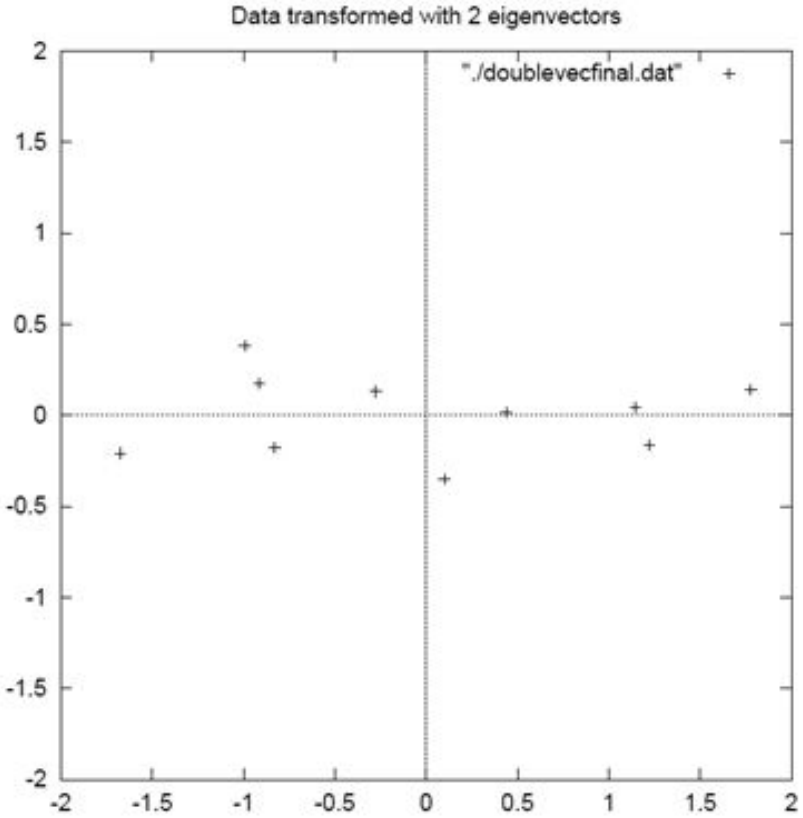
This represents the original data in terms of the chosen principal components (eigenvectors)—that is, along these axes.

Principal Component Analysis (PCA)
Example - New Representation of Data

Transformed Data=

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

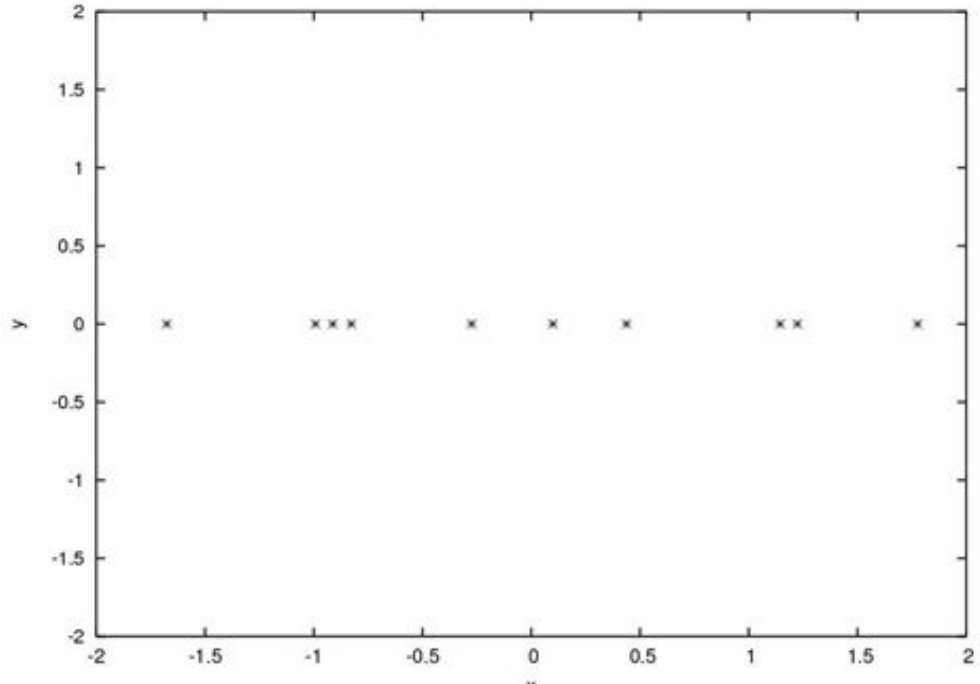
Data transformed with 2 eigenvectors



Principal Component Analysis (PCA)
Example - Reduced Representation of Data

Transformed Data (Single eigenvector)

x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056



Principal Component Analysis (PCA)
Example - Reconstruction of Original Data

Original operation:

- **TransformedData = RowFeatureVector x RowDataAdjust**

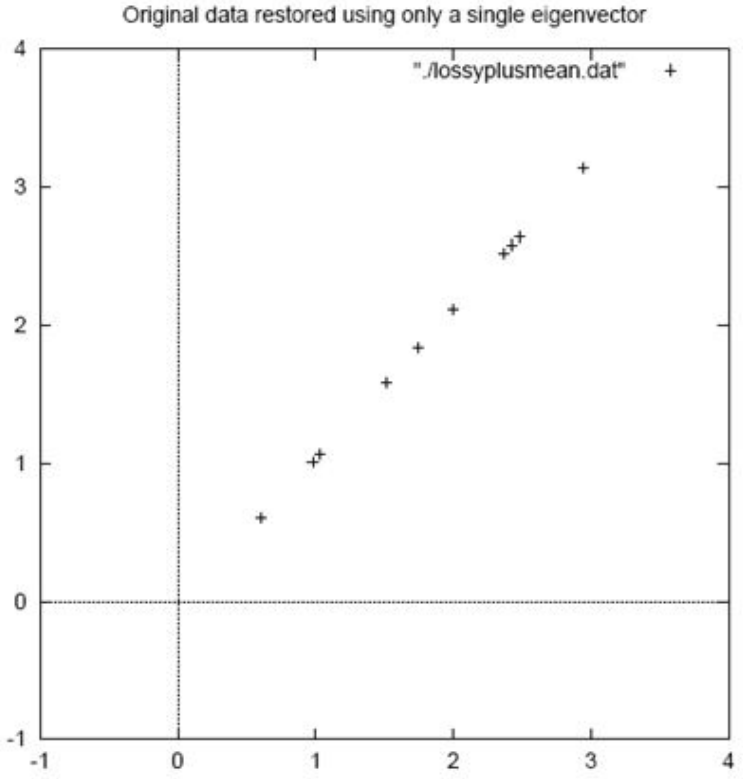
To reconstruct original data

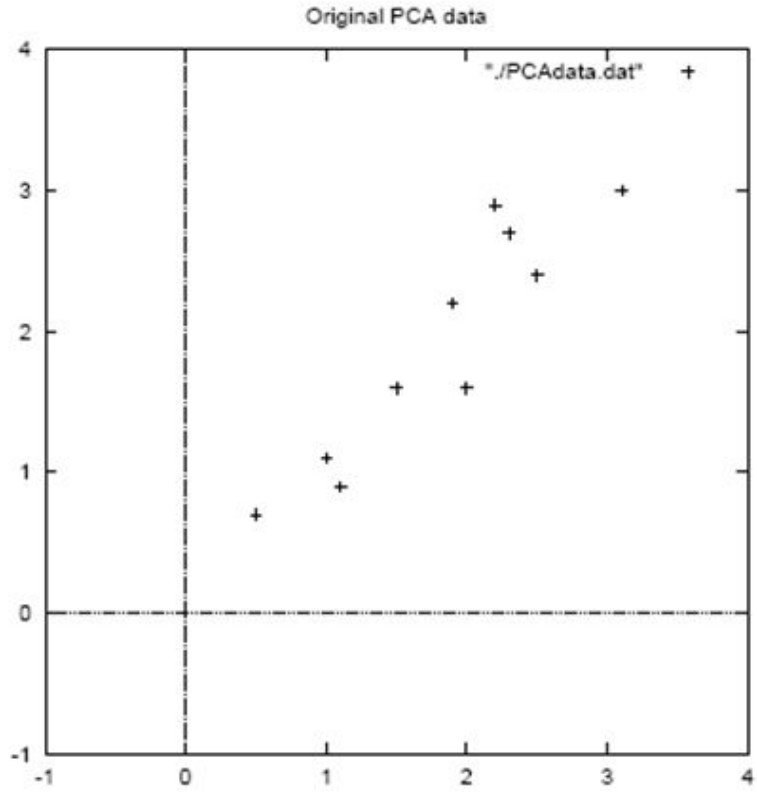
Inverse operation:

- **RowDataAdjust = RowFeatureVector $^{-1}$ x TransformedData**

  = RowFeatureVector^{T} x TransformedData
- **RowDataOriginal = RowDataAdjust + OriginalMean**

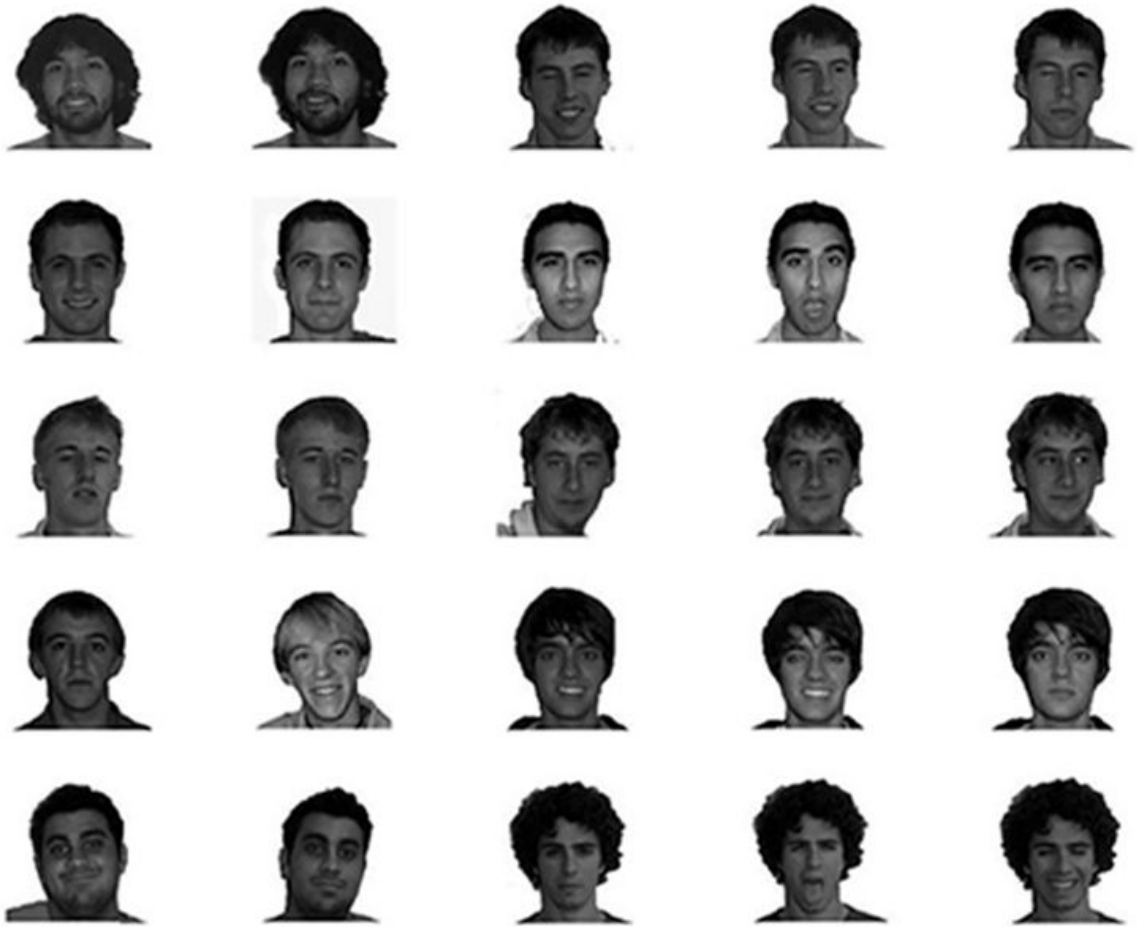
Principal Component Analysis (PCA)
Example - Reconstruction of Original Data





Principal Component Analysis (PCA)

Example - Face



Principal Component Analysis (PCA) Example - Face

- **Raw features are pixel intensity values (2061 features)**
- **Each image is encoded as a vector Γ_i of these features**
- **Compute "mean" face in the training set as**

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

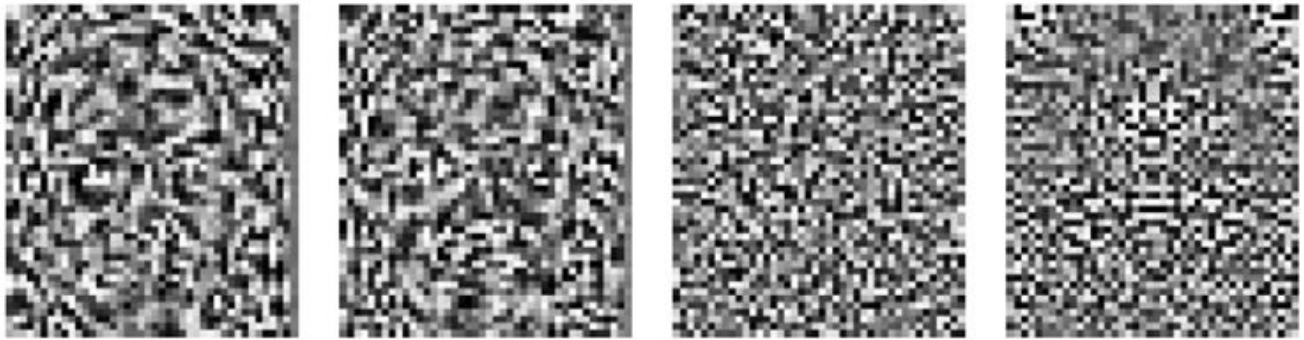
The average face and first four eigenfaces

- **Subtract the mean face from each face vector**

$$\Phi_i = \Gamma_i - \Psi$$

- **Compute the covariance matrix C**
- **Compute the (unit) eigenvectors v_i of C**
- **Keep only the first K principal components (eigenvectors)**

Eigenfaces 15, 100, 200, 250, 300



Eigenfaces 400, 450, 1000, 2000

Principal Component Analysis (PCA)
Example - Face Applications

The eigenfaces encode the principal sources of variation in the dataset, such as absence/presence of facial hair, skin tone, glasses, etc.

We can represent any face as a linear combination of the "basis" faces.

Use this representation for:

- **Face recognition**
  - **e.g., Euclidean distance from known faces**
- **Linear discrimination**
  - **e.g., "glasses" versus "no glasses", or "male" versus "female"**

Principal Component Analysis (PCA)
Python Implementation Example

https://github.com/jdwittenauer/ipython-notebooks/blob/master/notebooks/ml/ML-Exercise7.ipynb

Data Representation

Visualizing Representations: t-SNE

t-SNE(t-distributed stochastic neighbor embedding) Method for Data Representation in Lower Dims

A method for exploring high-dimensional data, can find structure where other dimensionality-reduction algorithms cannot

Maaten, Hinton, "Visualizing Data using t-SNE"

Machine learning algorithm for dimensionality reduction

Nonlinear dimensionality reduction technique

- **Embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot**
- **Similar objects are modeled by nearby points and dissimilar objects are modeled by distant points.**

t-SNE: t-distributed stochastic neighbor embedding Method for Data Representation in Lower Dims

Properties:

- **Creates compelling two-dimensional “maps” from data with hundreds or even thousands of dimensions**
- **Uses different transformations on different regions**
- **Uses tuneable parameter, “perplexity,” which says (loosely) how to balance attention between local and global aspects of your data**
- **Different runs can produce different outputs**
- **There is a possibility for misinterpretation**

t-SNE: t-distributed stochastic neighbor embedding Method for Data Representation in Lower Dims

Two steps:

1. **Constructs a probability distribution over pairs of high-dimensional objects**
   a. **in such a way that similar objects have a high probability of being picked,**
   b. **whilst dissimilar points have an extremely small probability of being picked**
2. **Defines a similar probability distribution over the points in the low-dimensional map,**
   a. **and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map.**

t-SNE Visualization
Perplexity

The perplexity of a discrete probability distribution p is defined as

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

where H(p) is the entropy (in bits) of the distribution and x ranges over events.

In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample.

It can be used to compare probability models.

Low perplexity indicates the probability distribution is good at predicting the sample.

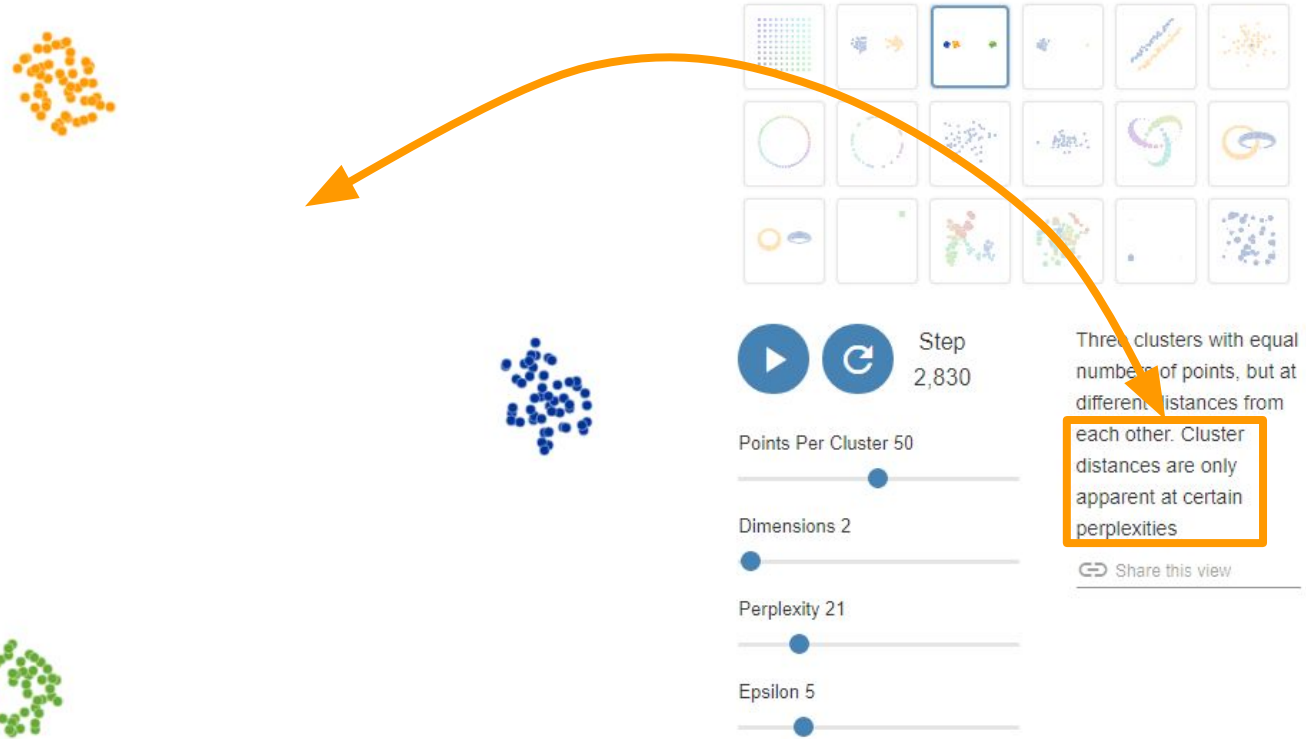
t-SNE Visualization
Method for Data Representation in Lower Dims

Peculiar behavior:

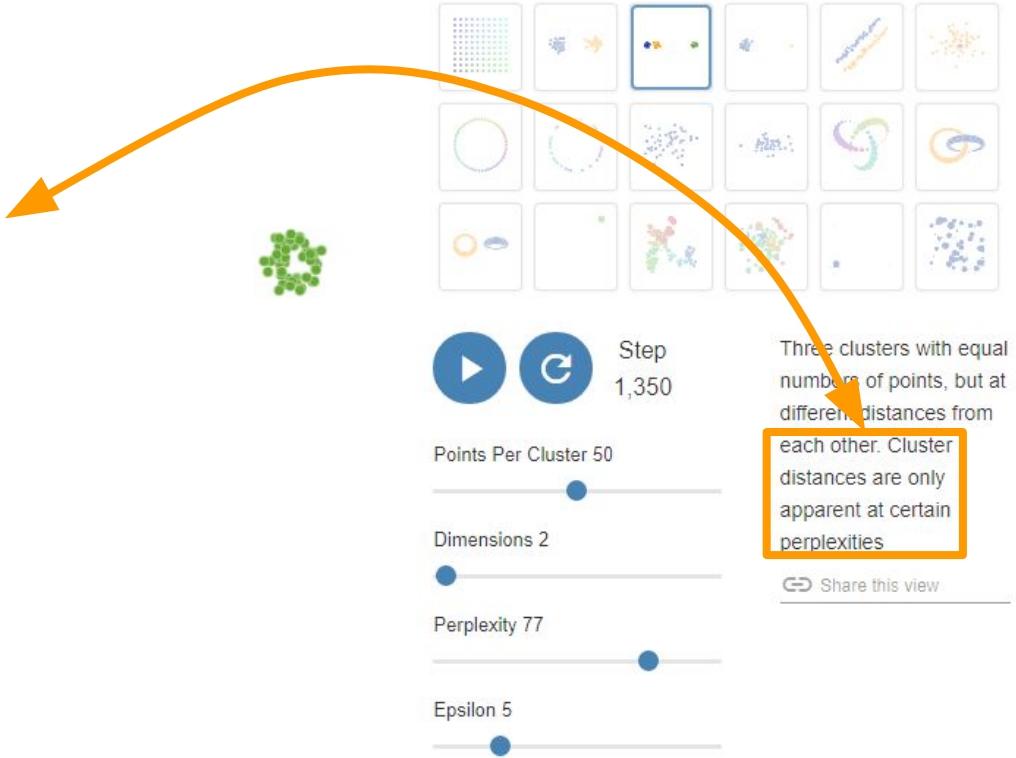
- **Hyperparameters really matter: perplexity 5-50, perplexity really should be smaller than the number of points.**
- **Cluster sizes in a t-SNE plot mean nothing: expands dense clusters, and contracts sparse ones**
- **Distances between clusters might not mean anything**
- **Random noise doesn't always look random**
- **You can see some shapes, sometimes**
- **For topology, you may need more than one plot**

https://distill.pub/2016/misread-tsne/

t-SNE Visualization
Example: Perplexity 21



t-SNE Visualization
Example: Perplexity 77



The Curse of Dimensionality

The Problem

Regularization

The Curse of Dimensionality

In applications, the number of possible distinct configurations of the variables of interest often increases exponentially with the dimensionality. This is known as the curse of dimensionality.

The main challenge: the number of possible configurations of the variables of interest is much larger than the number of training examples.

The Curse of Dimensionality
Illustration

If one wants to build a (good) minimum-error rate classifier, then one needs to get a very good estimate of P (ω_i | **x**).

Is that always possible?

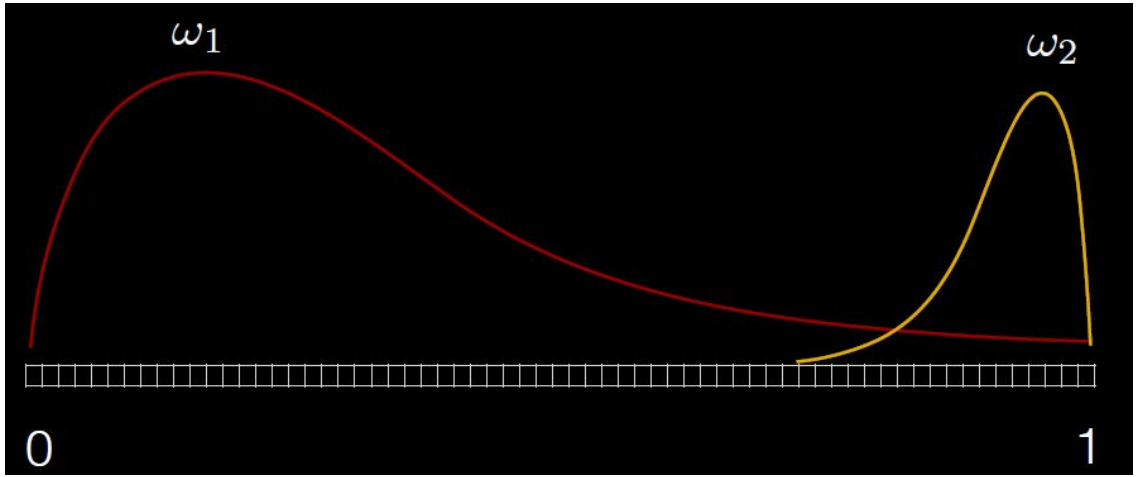
The Curse of Dimensionality
Illustration

Assume feature space with just one dimension, and feature values $x \in [0, 1]$.

Assume 10,000 training samples from which to estimate a posteriori probabilities.

These probabilities can be estimated using a histogram with 100 evenly-spaced bins.

The Curse of Dimensionality
Illustration - One dimension is Fine



On average, each bin would have 100 samples.

One could estimate $\rho(x|\omega_i)$ as the number of samples from class *i* that fall in the same bin that *x* falls into, divided by the total number of samples in that bin .

Belheumeur

The Curse of Dimensionality
Illustration - Three Dimensions?

Assume feature space is just 3-dimensional and the feature $x \in [\,0\,,\,1\,]^3$.

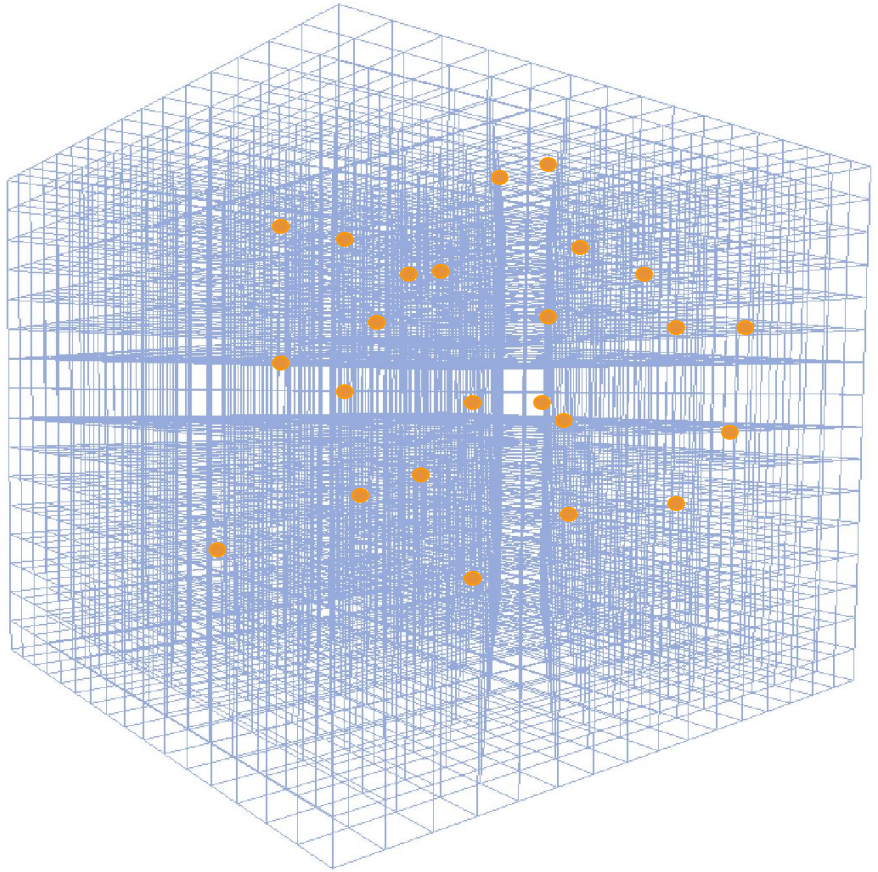
Assume 10,000 training samples from which to estimate a posteriori probabilities.

If one estimates the probabilities using a histogram where the 3-D volume is divided into the bins which have the same width as the bins in the 1-D case, then ...

The Curse of Dimensionality Illustration - 3D

Each bin would have

0.01 samples on average.



NOT nearly enough.

The Curse of Dimensionality
Illustration - Multi-D space

There are never enough training samples to sample high-dimensional feature spaces densely enough.

The Curse of Dimensionality
Illustration - Multi-D Space

High-dimensional spaces defy intuition:

- What portion of the volume of the unit N-dim hypercube is taken by an inscribed sphere?

The Curse of Dimensionality
Illustration - Multi-D Space

Volume of sphere inside volume of cube

$$V_n R_n = \frac{\pi^{n/2}}{\Gamma(n/2+1)}\left(\frac{1}{2}\right)^{20} = \frac{\pi^{n/2}}{10!}\left(\frac{1}{2}\right)^{20} = 2.46x10^{-8}$$

The Curse of Dimensionality
Illustration - Multi-D Space

Volume of sphere inside volume of cube

If one uniformly sampled 100 million points in the unit hypercube,

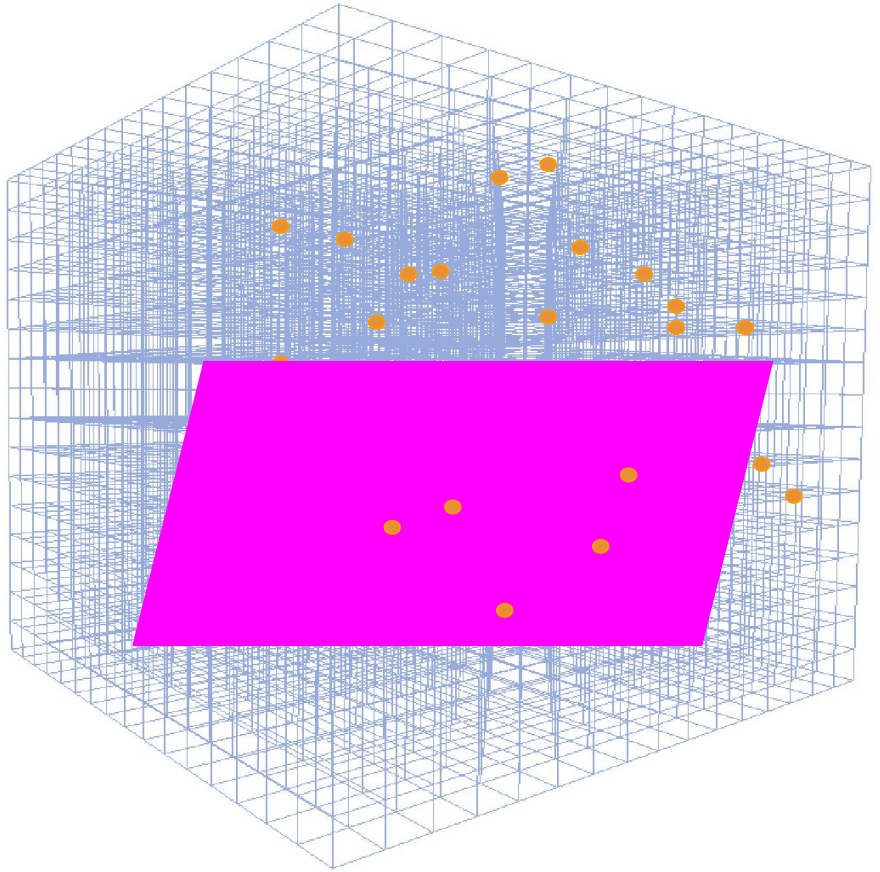
… then one should expect less than 3 of the points would lie within its contained hypersphere.

The Curse of Dimensionality
Illustration - Multi-D Space

So to create classifiers:

- instead of dealing with probabilities, one should look for good discriminant functions
  - which create boundaries between classes in multi-dimensional spaces.

The Curse of Dimensionality Illustration - 3D



The Curse of Dimensionality
Local Constancy and Smoothness Regularization

Because in high-dimensional spaces the number of configurations is huge, much larger than the number of examples, most configurations will have no training example associated with it.

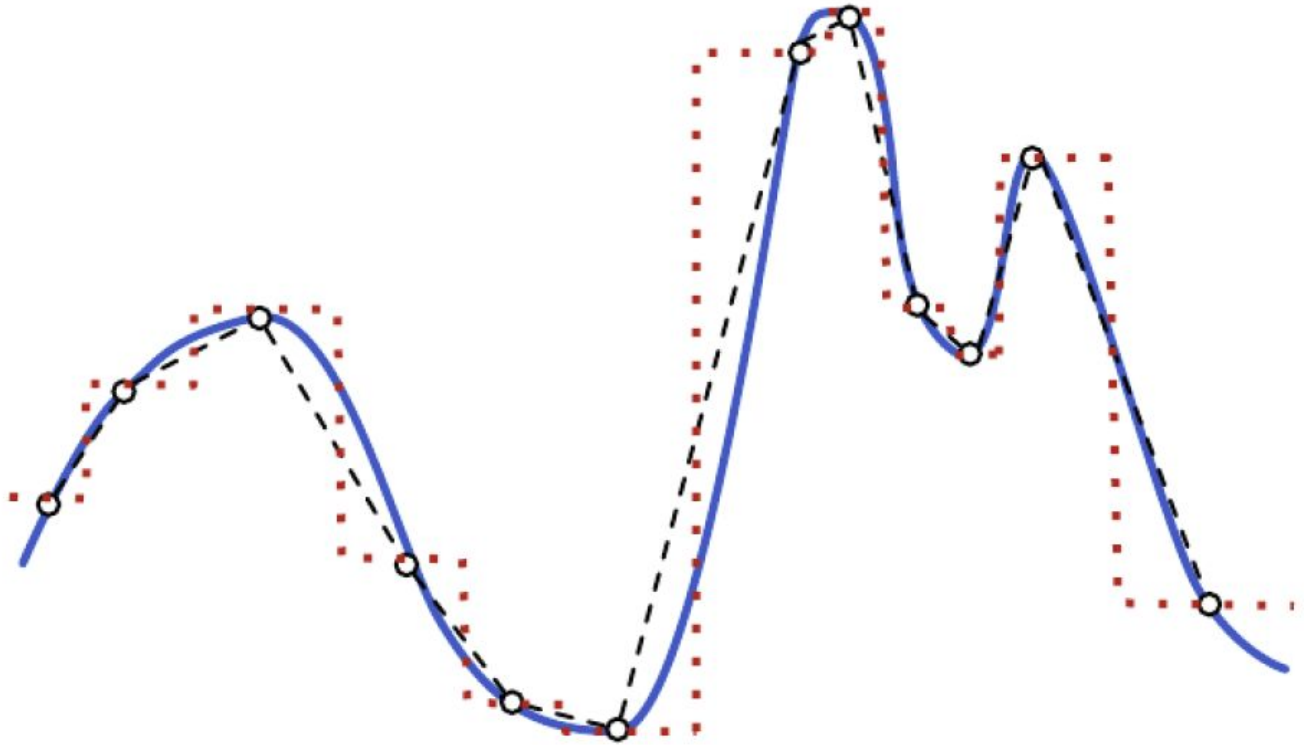
To say something meaningful about these configurations, machine learning algorithms need to be guided by prior beliefs about the kind of function they should learn. These priors are incorporated as explicit beliefs in the form of probability distributions over parameters of the model.

Often, implicit "priors" are the smoothness prior or local constancy prior. The prior belief is that the learned function should be smooth or locally constant within a small region:

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$$

for most $\mathbf{x}$ and small change ϵ.

The Curse of Dimensionality
Local Constancy and Smoothness Regularization



Interpolation between neighboring training examples.

The Curse of Dimensionality
Local Constancy and Smoothness Regularization

Non-parametric kernel density estimation methods and kernel regression methods construct a learned function f of the form

$$f(\mathbf{x}) = b + \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}^i)$$

for classification or regression. If the kernel function k always returns a discrete output such as 0 or 1, this includes the cases where f is piecewise constant. However, better results can be obtained if k is continuous. A common choice is the Gaussian kernel of the form

$$k(\mathbf{u} - \mathbf{v}) = \mathcal{N}(\mathbf{u} - \mathbf{v}; 0, \sigma^2 \mathbf{I}).$$

An important class of kernels is the family of local kernels where $k(\mathbf{u} - \mathbf{v})$ is large when $\mathbf{u} = \mathbf{v}$ and decreases as $\mathbf{u}$ and $\mathbf{v}$ grow farther apart from each other. A local kernel can be thought of as a similarity function that performs template matching, by measuring how closely a test example $\mathbf{x}$ resembles each train example $\mathbf{x}^i$.

The Curse of Dimensionality
Local Constancy and Smoothness Regularization

The use of a local kernel means that these methods are still essentially only interpolating between nearby training examples. One can think of the training examples as control knots of an interpolating spline.

Much of the modern motivation for deep learning is derived from studying the limitations of local template matching and how deep models are able to succeed in cases where local template matching fails.

Algorithm Survey

Building ML Algorithms

Practical considerations, python code

Logistics Regression

Support Vector Machine

Softmax Classifier

Supervised Learning Algorithms

Linear Regression and Regularized Linear Regression
Classification Trees
Regression Trees
Discriminant Analysis (classification)
k-Nearest Neighbors (classification)
Naive Bayes (classification)
Support Vector Machines (SVM) for classification
SVM for regression
Multiclass models for SVM or other classifiers
Classification or Regression Ensembles

Supervised Learning Steps

- Prepare Data
- Choose an Algorithm
- Fit a Model
- Choose a Validation Method
- Examine Fit and Update Until Satisfied
- Use Fitted Model for Predictions

Building Machine Learning Algorithms

All deep learning algorithms are built using a simple recipe:

- combine a specification of a dataset,
- a cost function,
- an optimization procedure and,
- a model.

Machine Learning Algorithms
Python Implementations

Machine Learning Exercises by John Wittenauer

- https://github.com/jdwittenauer/ipython-notebooks
- Code: https://github.com/jdwittenauer/ipython-notebooks/tree/master/notebooks/ml
- Instructions: https://github.com/jdwittenauer/ipython-notebooks/tree/master/exercises/ML
- Data: https://github.com/jdwittenauer/ipython-notebooks/tree/master/data

Probabilistic Supervised Learning Algorithms

Supervised learning algorithms learn to associate some input with some output, given a training set of examples of inputs **x** and outputs **y**.

Probabilistic Supervised Learning Algorithms

Most supervised learning algorithms are based on estimating a probability distribution $p(\mathbf{y}|\mathbf{x})$. We can do this simply by using maximum conditional likelihood estimation to find the best parameter vector $\boldsymbol{\theta}$ for a parametric family of distributions $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. The normal distribution over real-valued numbers that we used for linear regression was parameterized in terms of a mean. Linear regression corresponds to the family

$$p(y|\mathbf{x};\boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{\theta}^T\mathbf{x}, \mathbf{I}).$$

We can generalize linear regression to the classification scenario by defining a different family of probability distributions. If we have two classes, class 0 and class 1, then we need to only specify the probability of one of these classes.

Algorithm Survey

Linear Regression

Building Machine Learning Algorithms
Linear Regression

For example, the linear regression algorithm combines a dataset $(\mathbf{X}, \mathbf{y})$, the cost function

$$J(\mathbf{w}, b) = -\mathbb{E}p_{model}(y|\mathbf{x}),$$

and the model specification

$$p_{model}(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{x}^T\mathbf{w} + b, 1).$$

Typically we then observe that J simplifies to the mean squared error and we can choose to optimize this in closed form by solving the normal equations with the Moore-Penrose pseudo-inverse. By realizing that we can modify any of these components, we can obtain a very wide variety of algorithms.

Building Machine Learning Algorithms
Linear Regression

The cost function typically includes at least one term that causes the learning process to perform statistical estimation. The most common cost function is the negative log-likelihood, so that minimizing the cost function causes maximum likelihood estimation.

This main term of the cost function often decomposes as a sum over training examples of some per-example loss function. For example, the negative conditional log-likelihood of the training data can be written as

$$J(\boldsymbol{\theta}) = \mathbb{E}\ L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{x}^i, y^i, \boldsymbol{\theta})$$

where y is the per-example loss $L(\mathbf{x}, y, \boldsymbol{\theta}) = \log p(y|\mathbf{x}; \boldsymbol{\theta}$. We can design many different cost functions just by taking the expectation across the training set of different per-example loss functions.

Building Machine Learning Algorithms
Linear Regression

The cost function may also include additional terms, such as regularization terms. For example, we can add weight decay to the linear regression cost function to obtain

$$J(\mathbf{w}, b) = \lambda ||\mathbf{w}||_2^2 - \mathbb{E}p_{model}(y|\mathbf{x}),$$

This still allows closed-form optimization.
If the model is non-linear, then most cost functions can no longer be optimized in close form. An iterative numerical optimization procedure, such as gradient descent, provides a solution.

Linear Regression - Normal Equation
Closed Form Way of Doing Linear Regression

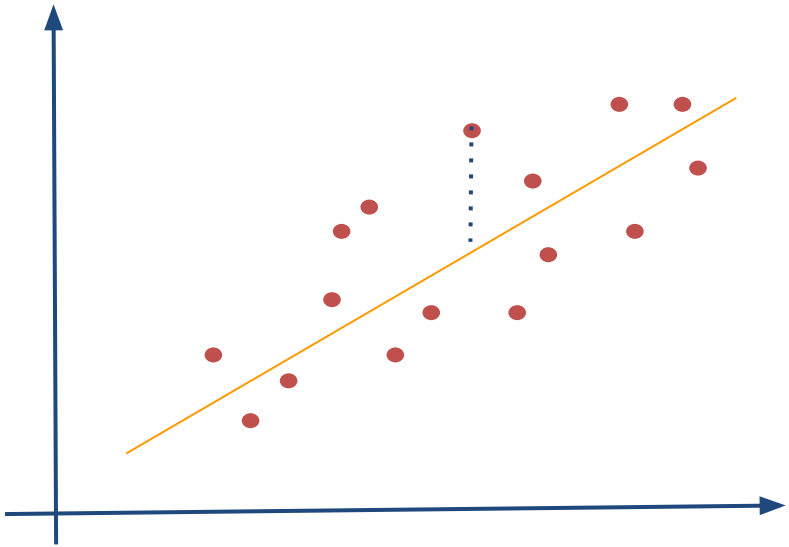
$$2\mathbf{X}^{(train)T}\mathbf{X}^{train}\mathbf{w} - 2\mathbf{X}^{(train)T}\mathbf{y}^{train} = 0$$

or finally

$$\mathbf{w} = (\mathbf{X}^{(train)T}\mathbf{X}^{train})^{-1}\mathbf{X}^{(train)T}\mathbf{y}^{train}.$$

This system of equations is known as the normal equations.

Linear Regression - Iterative Solution
Another Look at Linear Regression



Linear Regression - Iterative Solution
Another Look at Linear Regression

Find the line that minimizes the square distances from the training points.

Write the line equation in the parametric form

$$y=\theta_0+\theta_1 x=\theta^T x=h_\theta(x)$$

$$\theta=\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix},\quad \boldsymbol{x}=\begin{bmatrix}1\\ x\end{bmatrix}$$

Linear Regression - Iterative Solution
Gradient Descent

Given sample points $(x^{(i)}, y^{(i)})$

The loss function is $$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

The gradient is $$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

This can be solved $$\theta(t+1) = \theta(t) - \alpha \nabla_\theta J(\theta(t))$$

Linear Regression

Python Code:

- **https://github.com/jdwittenauer/ipython-notebooks/blob/master/notebooks/ml/ML-Exercise1.ipynb**

Reading:

- **Linear classification Stanford tutorials: http://cs231n.github.io/linear-classify/#intro**
- **Loss function and optimization tutorials: http://cs231n.github.io/optimization-1/**

Algorithm Survey

Logistic Regression

Probabilistic Supervised Learning
Logistic Regression

A distribution over a binary variable is slightly more complicated, because its mean must always be between 0 and 1. One way to solve this problem is to use the logistic sigmoid function to squash the output of the linear function into the interval $(0, 1)$ and interpret that value as a probability:

$$p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T \mathbf{x}).$$

This approach is known as logistic regression. Solution is possible by minimizing the negative log-likelihood ratio via gradient descent.

Logistic Regression
Estimation of Probabilities

It is often required to construct a model which estimates probabilities.

Examples:

- **medical diagnosis: probability of recovery or relapse, probability of miscarriage, etc.**
- **credit scoring: probability of a mortgage being repaid**
- **sports: probability of a team beating a competitor**

Can linear regression be used?

- **No: numerical values obtained by linear regression fall out of boundaries [0.0, 1.0]**

Logistic Regression
Estimation of Probabilities

Models for estimating the probability of class membership

- **logistic regression**
- **linear and quadratic discriminant analysis**
- **neural networks**
- **tree induction**

Logistic Regression
Estimation of Probabilities

Logistic regression: member of the family of methods called generalized linear models ("GLM")

Such models include a linear part followed by some "link function"

- **In neural networks:**
  - **activation functions**
  - **transfer functions**
  - **squashing functions**

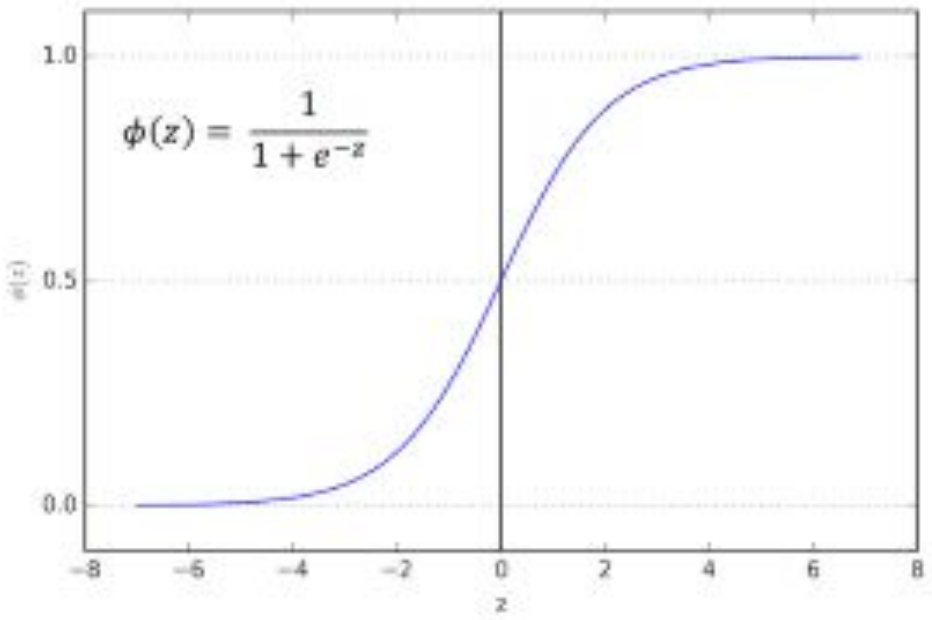
Logistic Regression

Steps

- **Calculate the linear function of the predictor variables**
- **Pass thru the logistic function**

Other GLMs operate similarly,

but employ different link functions.



Logistic Regression

Calculation of Optimal Coefficients

- **Calculation does not have a direct closed form solution (while least square linear regression does)**
- **Iterative fitting procedure is needed**
- **Successive "guesses" at the right coefficients are incrementally improved**

Logistic Regression - Classification

$$y \in \{0,1\} \quad y = \begin{cases} 0: \text{ negative class (benign tumor)} \\ 1: \text{ positive class (malignant tumor)} \end{cases}$$

Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

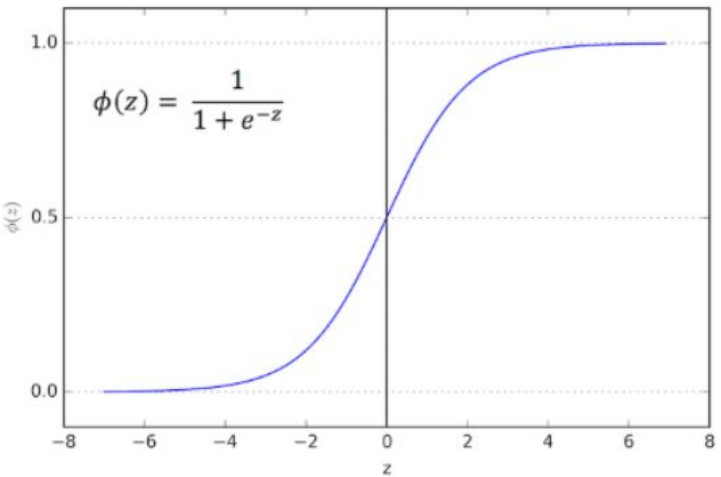
If $h_\theta(x) < 0.5$, predict "y = 0"

Logistic Regression Model
Hypothesis Presentation

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



Logistic Regression
Interpretation of the Output of the Hypothesis

$h_\theta(x)$ = estimated probability that y = 1 on input x

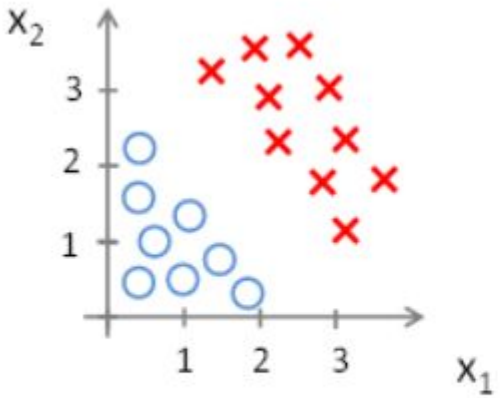
Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ Tell patient that 70% chance of tumor being malignant

probability that y = 1, given x, parameterized by θ

$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$
$$P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)$$

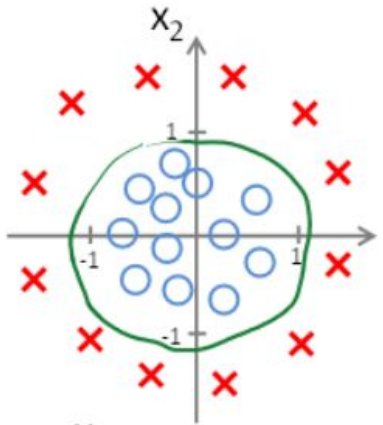
Logistic Regression
Decision Boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

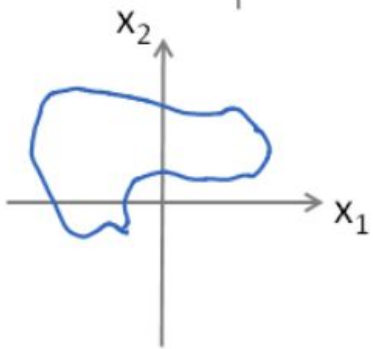
Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

Non-Linear Decision Boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Logistic Regression Solution
Calculate Parameters

Available: Training set with m examples

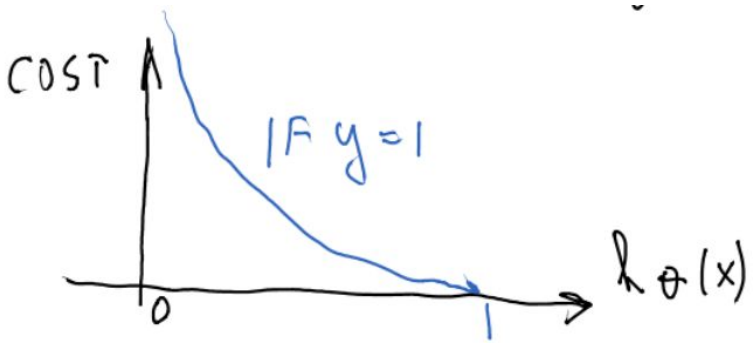
$$\{(x^1,y^1),(x^2,y^2),\dots(x^m,y^m)\}$$

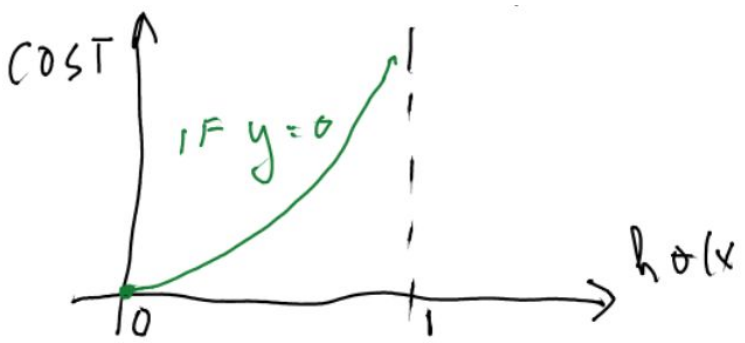
$$x=[x_0 \;\; x_1 \;\; \dots \;\; x_n]^T \quad x_0=1, \;\; y\in\{0,1\}$$

$$h_\theta(x)=\frac{1}{1+e^{-\theta^T x}} \quad : \; choose/calculate \;\; \theta$$

Logistic Regression Solution
Cost Function

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$





Logistic Regression Solution
Cost Function

Full Cost Function has to account for all cases

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Derivations:

- http://people.csail.mit.edu/jrennie/writing/lr.pdf
- http://juangabrielgomila.com/en/logistic-regression-derivation/

Logistic Regression Solution
Cost Function

Gradient Descent

- An iterative method for calculating values of θ based on previous values of θ

while
{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

} until desired level of error is reached

we want

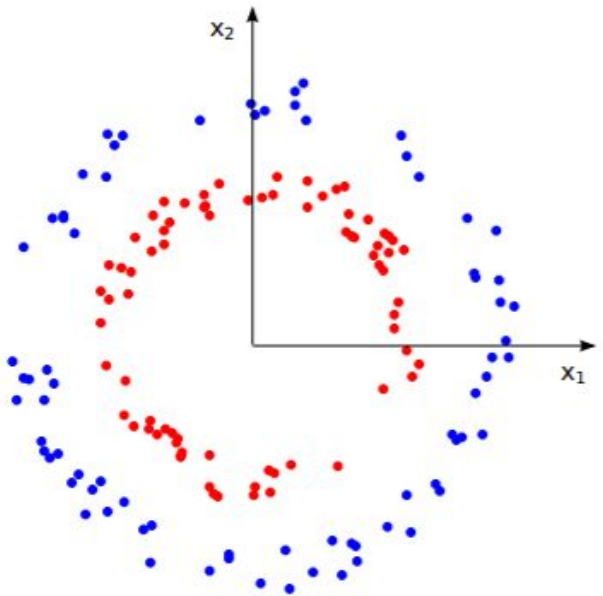
$$\min_\theta J(\theta)$$

while
{

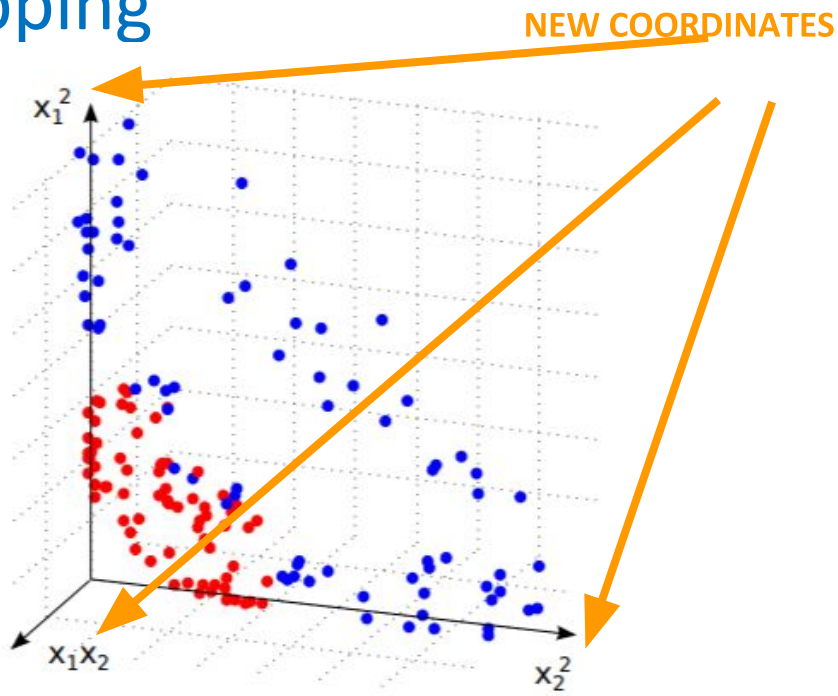
$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

} until desired level of error is reached, or after preset number of iterations

Linearly (Non?)-separable Data
High-Dimensional Data Mapping



(e) Data for binary classification



(f) Same data mapped to higher dimension

Paisley, http://www.columbia.edu/~jwp2128/Teaching/W4721/Spring2017/slides/lecture_2-21-17.pdf

Logistic Regression
Practical Implementation

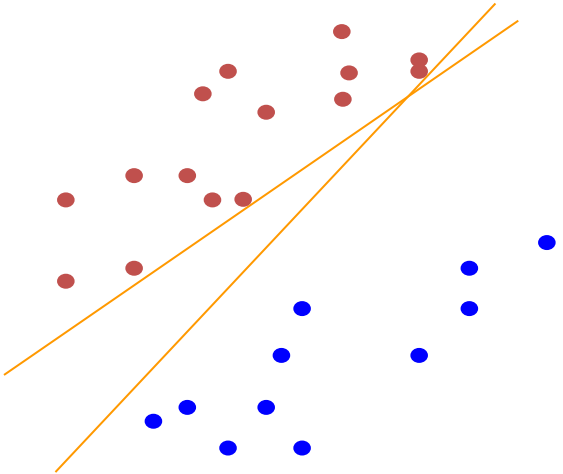
Python Code:

- **https://github.com/jdwittenauer/ipython-notebooks/blob/master/notebooks/ml/ML-Exercise2.ipynb**

Algorithm Survey

Support Vector Machine

Maximum Margin Classifiers
Motivation

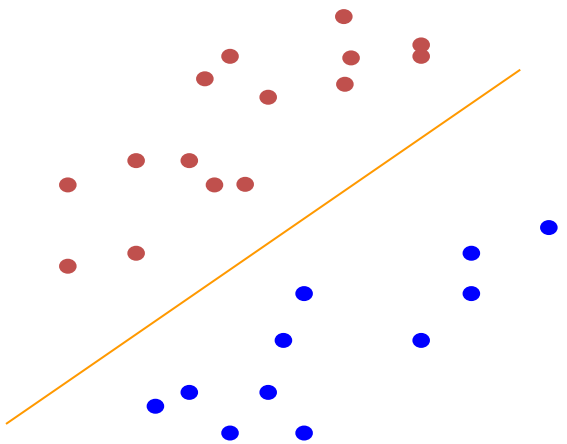


Setting: Linear classification, two linearly separable classes.

Perceptron:

- **Selects some hyperplane(s) separating the classes**
- **not optimal, new data points may fall on the wrong side**

Maximum Margin Classifiers
A Better Solution

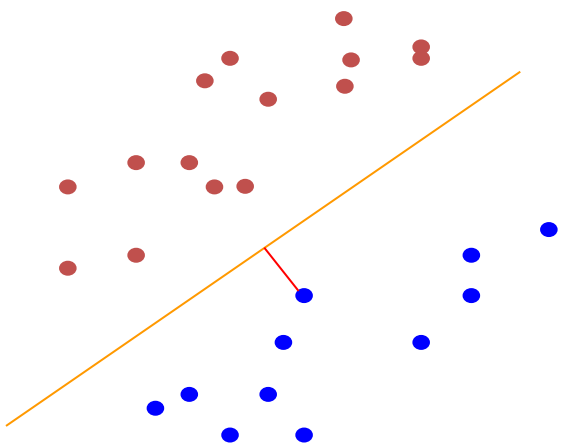


Maximum margin concept:

- **Choose a plane such that its distance to the closest point in each class is maximized. This distance is called the margin.**

Where to place the separating hyperplane?

Maximum Margin Classifiers



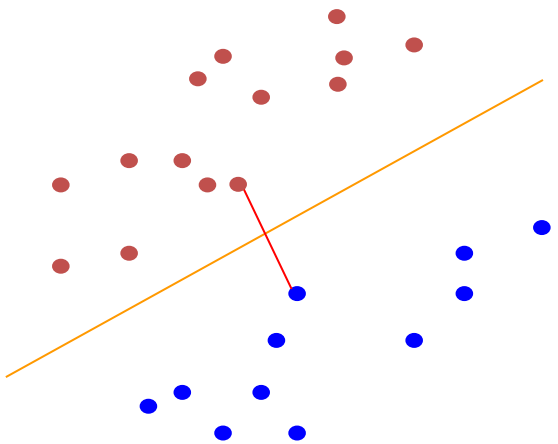
Margin Definition

The margin of a classifying hyperplane H is the shortest distance between the plane and any point in either set.

When we maximize this margin, H is "exactly in the middle" of the two.

How do we find this H?

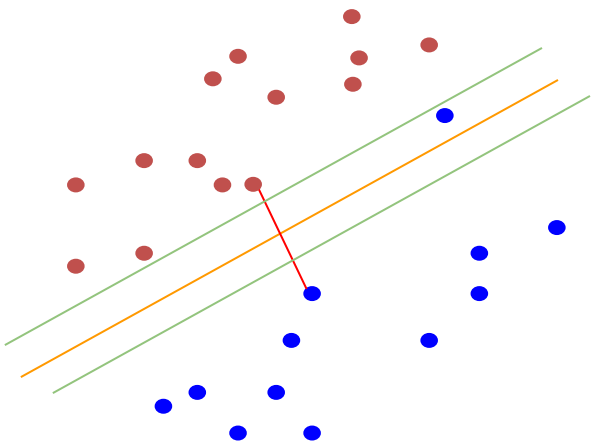
Maximum Margin Classifiers
Support Vector Machine (SVM)



Intuition is right:

- **Find the closest two points from the two classes.**
- **Connect them with a line and put a perpendicular hyperplane in the middle.**

Maximum Margin Classifiers
Support Vector Machine

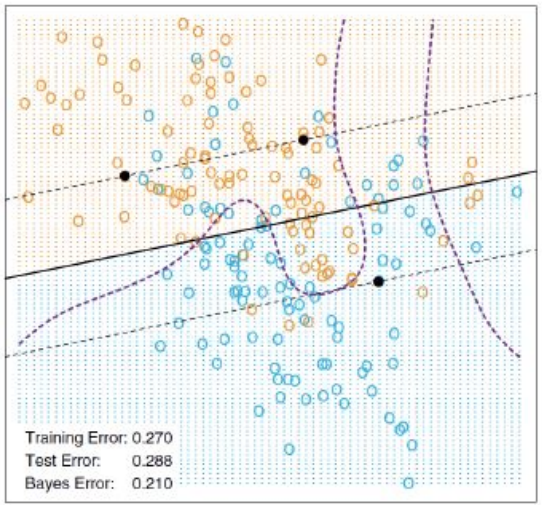


Powerful concept.

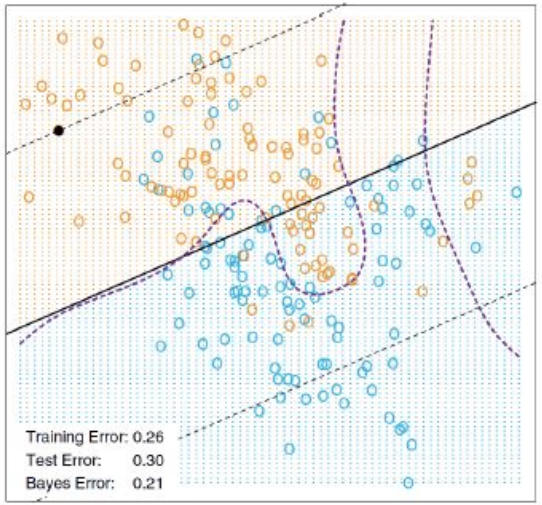
Extensions:

- **Primal problem and Dual Problem**
- **Soft-margin SVM**
- **Slack variables**

Maximum Margin Classifiers
Support Vector Machine



$\lambda = 100000$

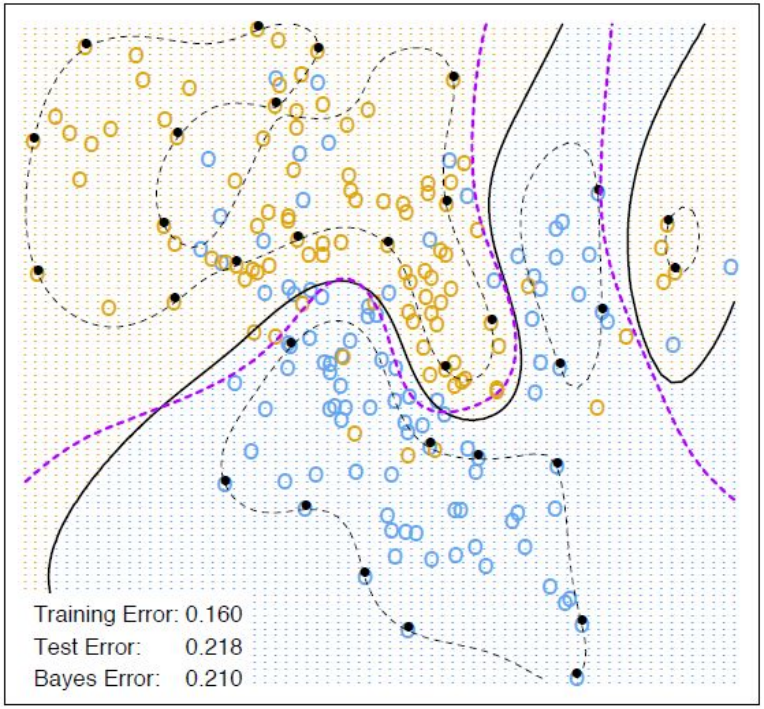


$\lambda = 0.01$

Example: Linear classifier does not work.

Paisley - http://www.columbia.edu/~jwp2128/Teaching/W4721/Spring2017/slides/lecture_2-23-17.pdf

Maximum Margin Classifiers
Support Vector Machine - Kernelized



SVM Kernel Tricks -> Nonlinear decision boundary

Black solid line

- **SVM decision boundary**

Purple line

- **A Bayes classifier**

Python code:
https://github.com/jdwittenauer/ipython-notebooks/blob/master/notebooks/ml/ML-Exercise6.ipynb

Support Vector Machine - Kernel Trick

Many machine learning algorithms can be written in terms of dot products between examples.

$$\boldsymbol{w}^\top \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\top \boldsymbol{x}^{(i)}$$

Replace x by a feature of x $\phi(\boldsymbol{x})$

$$k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}^{(i)})$$

Make predictions with $f(\boldsymbol{x}) = b + \sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$

Support Vector Machine - Kernel Trick
Non-Linear to Linear

- The kernel-based function is exactly equivalent to preprocessing the data by applying $\phi(\boldsymbol{x})$ to all inputs, then learning a linear model in the new transformed space.
- Kernel trick allows us to learn models that are nonlinear as a function of x using convex optimization techniques that are guaranteed to converge efficiently. Because we consider ϕ fixed and optimize only $\boldsymbol{\alpha}$.The algorithm views the decision function as linear in a different space.
- Second, the kernel function *k* often admits an implementation that is significantly more computational efficient than naively constructing two $\phi(\boldsymbol{x})$ vectors and explicitly taking their dot product.

Algorithms

Multi-Class Classification: Softmax

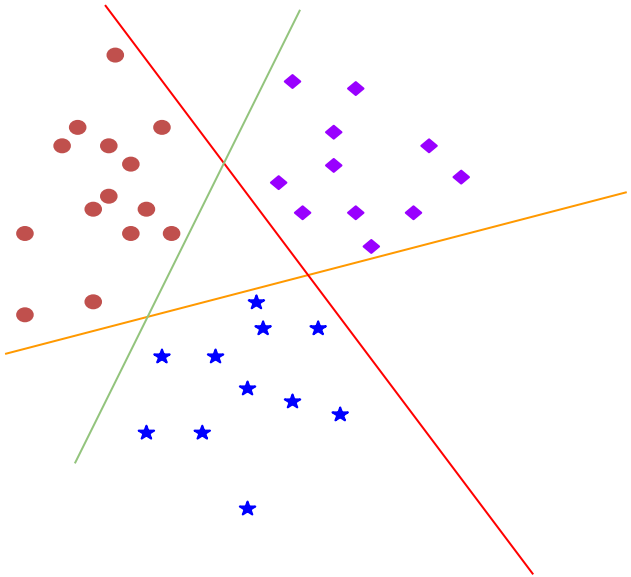
Multi-Class Classification

Binary classification requires a single separating hyper-surface or a single discriminating function to predict the two classes from input data.

For n classes, n - 1 discriminate functions are needed

- **In practice, it is simpler to use n discriminate functions, and train n one-vs-all classifiers.**

Multi-Class Classification
One vs. All Classification



Multi-Class Classification
Extension of Binary Classification

Binary classification: Logistic sigmoid function $P(y = 1 | \mathbf{x})$

$$P(y=1|\boldsymbol{x}) = \frac{1}{1+e^{-z}}$$

Multi-class classification $P(y = i | \mathbf{x})$**: Represent probability distribution over n different classes**

$$softmax: \quad P(y=i|\boldsymbol{x}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$log\ \ softmax: \quad z_i - \log \sum_j e^{z_j}$$

Book 6.2.2.3: Softmax Units for Multinomial Output Distribution

Multi-Class Classification

Softmax

The output of the softmax function can be used to represent a categorical distribution.

Maximize $P(y = i|\mathbf{x})$**: Represent probability distribution over n different classes.**

$$\phi(\mathbf{a}) = \text{softmax}(\mathbf{a}) \Leftrightarrow [\phi(\mathbf{a})]_i = e^{a_i} / \sum_j e^{a_j}$$

$$\text{such that } \sum_i [\phi(\mathbf{a})]_i = 1 \text{ and } [\phi(\mathbf{a})]_i > 0$$

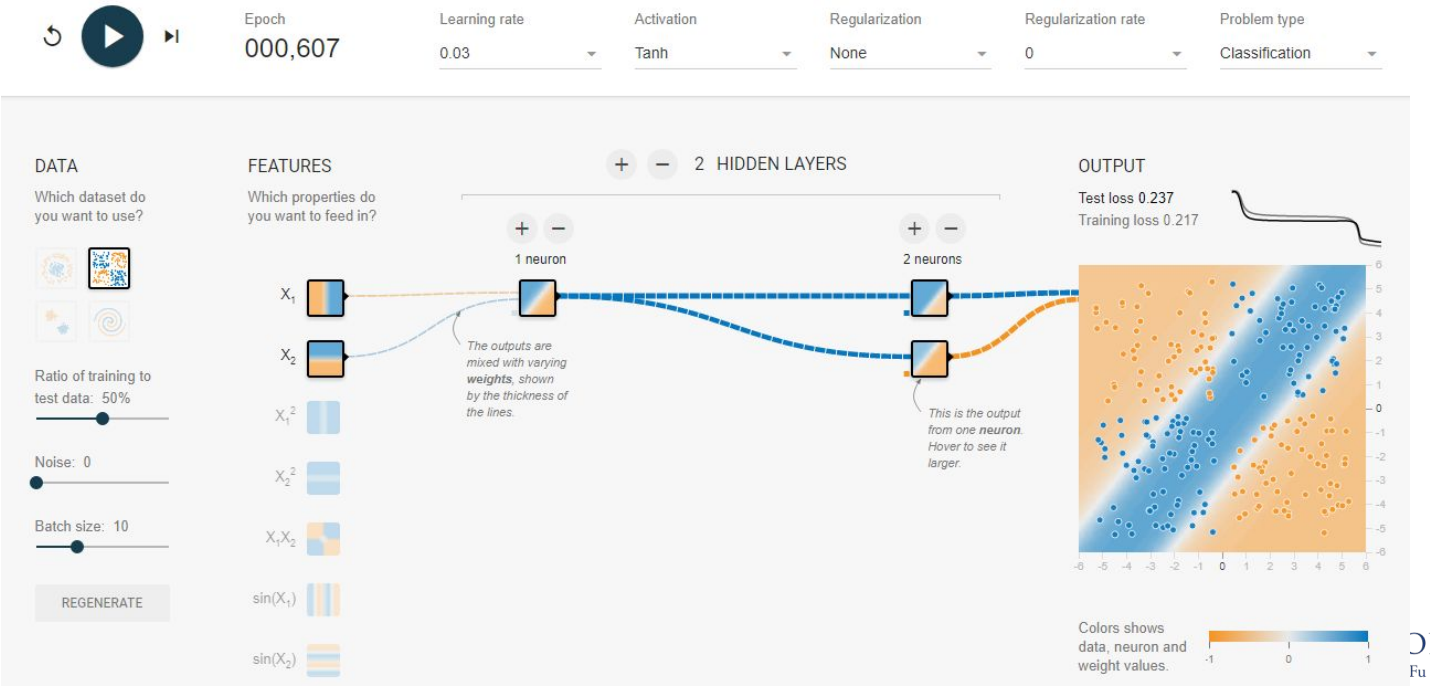
Reading: http://cs231n.github.io/linear-classify/#softmax

Universal Approximation Theorem Demonstration

Any function can be approximated by a Neural Network with enough many nodes

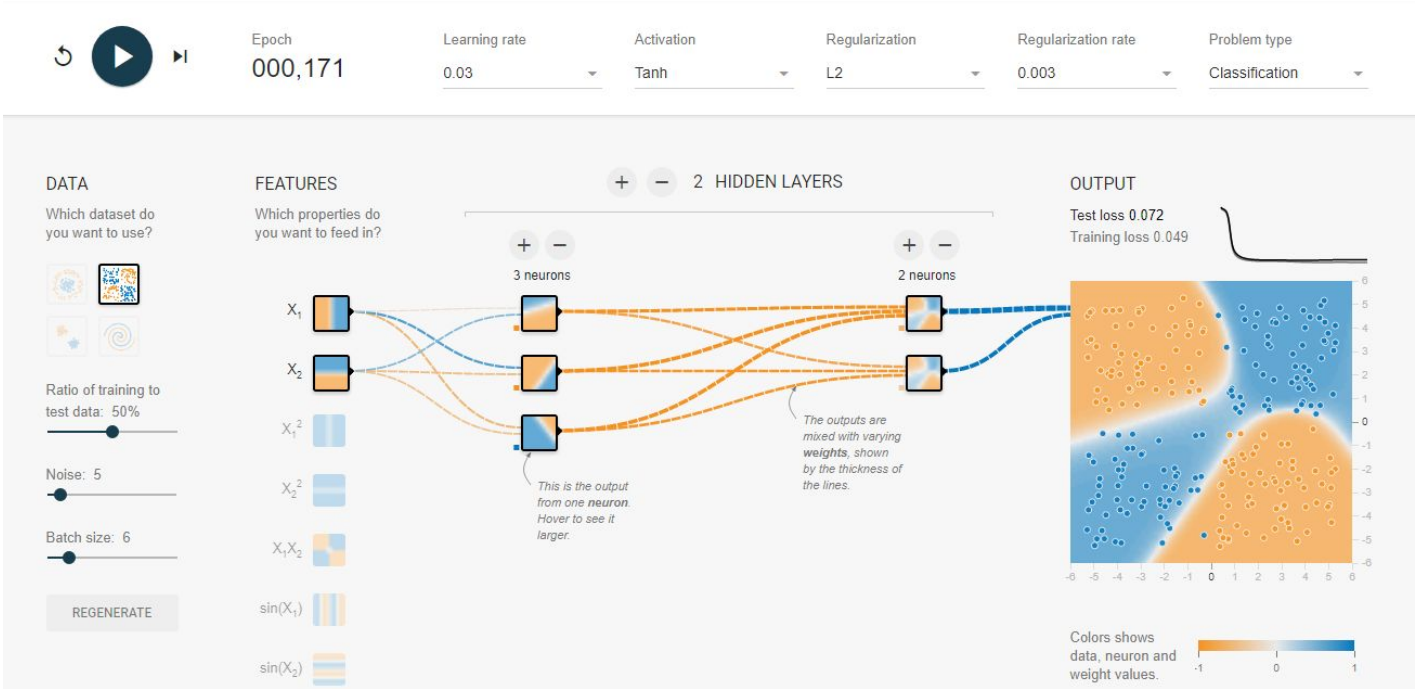
Classification With Neural Networks
...one neuron in a hidden layer...

TensorFlow PlayGround [Setup](): with one neuron in a hidden layer



Classification With Neural Networks
...more neurons in a hidden layer...

TensorFlow PlayGround [Setup](): with three neurons in a hidden layer



Classification With Neural Networks
Universal Approximation Theorem

Deeper networks are more efficient: deep rectified net can represent functions that would require an exponential number of hidden units in a shallow neural network [Montufar et al. 2014].

Deep networks using many hidden layers with rectified units are good at approximating functions which can be composed from simpler functions.

- **Many real tasks (functions) are well represented by a combination of simpler functions (visual in particular).**

Backup Slides

Various