

# Summer of Science Report

## Machine Learning & Neural Networks

Tezan Sahu

May-July 2018

## Abstract

MACHINE LEARNING, as defined by Wikipedia, is a field of computer science that often uses statistical techniques to give computers the ability to *learn* with data, without being explicitly programmed.

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning “signal” or “feedback” available to a learning system:

- **Supervised Learning:** These algorithms consist of a target variable which is to be predicted from a given set of independent variables. Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of such algorithms include: **Regression, Decision Tree, Random Forest, Logistic Regression** etc.
- **Unsupervised Learning:** In these algorithms, we do not have any target variable to predict. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Such algorithms play a crucial role in discovering patterns and hidden sequences in data. Examples of such algorithms include: **SVM, K-Means, LSH, Mixture of Gaussian Model, PCA, ICA** etc.

Other Machine Learning Task categories include:

- \* **Semi-Supervised Learning:** Examples: Collaborative Filtering, etc.
- \* **Reinforcement Learning**
- \* **Online Learning**

NEURAL NETWORKS are *state-of-the-art* tools that aim to solve complex problems in almost every field using statistical tools, linear algebra and calculus by trying to mimic the human brain. We will look at the different types of Neural Nets, their structures, working and some advanced optimization algorithms.

# Contents

<b>1</b>	<b>Regression</b>	<b>2</b>
1.1	Linear and Polynomial Regression . . . . .	2
1.2	Multiple Regression . . . . .	3
1.3	Gradient Descent Algorithm . . . . .	4
1.4	Overfitting and Regularization . . . . .	4
1.4.1	L2 Norm and Ridge Regression . . . . .	5
1.4.2	L1 Norm and LASSO Regression . . . . .	6
1.5	Bias-Variance Tradeoff . . . . .	7
1.6	Kernel Regression . . . . .	8
<b>2</b>	<b>Classification</b>	<b>9</b>
2.1	Logistic Regression . . . . .	9
2.2	Likelihood Function and Gradient Ascent . . . . .	10
2.3	Decision Trees . . . . .	11
2.3.1	Choosing the Best Splitting Feature . . . . .	11
2.3.2	Preventing Overfitting in a Decision Tree . . . . .	12
2.4	Ensemble of Decision Trees . . . . .	13
2.4.1	AdaBoost Algorithm . . . . .	13
2.5	Stochastic Gradient Ascent . . . . .	14
2.6	Support Vector Machines(SVM) . . . . .	14
2.6.1	Kernels . . . . .	15
<b>3</b>	<b>Clustering and Retrieval</b>	<b>16</b>
3.1	Document Representation . . . . .	16
3.1.1	Bag-of-Words Representation . . . . .	17
3.1.2	TF-IDF Representation . . . . .	17
3.2	Distance Metrics . . . . .	18
3.3	K-Nearest Neighbors Algorithm . . . . .	18
3.4	KD Trees . . . . .	19
3.5	Locality Sensitive Hashing . . . . .	21
3.6	K-Means Clustering . . . . .	21

3.7	Gaussian Mixture Models . . . . .	22
3.7.1	EM Algorithm . . . . .	23
3.8	Mixed Membership Modeling . . . . .	24
3.8.1	Latent Dirichlet Allocation . . . . .	24
3.8.2	Gibbs Sampling . . . . .	25
3.9	Hierarchical Clustering . . . . .	26
3.9.1	Divisive Clustering . . . . .	26
3.9.2	Agglomerative Clustering . . . . .	26
<b>4</b>	<b>PCA and ICA</b>	<b>27</b>
4.1	PCA . . . . .	27
4.2	ICA . . . . .	29
<b>5</b>	<b>Recommender Systems</b>	<b>30</b>
5.1	Content-Based Filtering . . . . .	30
5.2	Collaborative Filtering . . . . .	31
<b>6</b>	<b>Neural Networks and Deep Learning</b>	<b>33</b>
6.1	Neuron Activation and Back-Propagation . . . . .	34
6.2	Types of Neural Networks . . . . .	34
6.3	Advanced Optimization Algorithms . . . . .	36
<b>A</b>	<b>Some Platforms &amp; Libraries to Accomplish Machine Learning Tasks (using Python)</b>	<b>38</b>
<b>B</b>	<b>Deep Spectral Clustering</b>	<b>39</b>

# List of Figures

1.1	Examples of Linear and Polynomial Regression . . . . .	3
1.2	Plot of Training and Generalization Error . . . . .	5
1.3	Coefficient Paths . . . . .	6
1.4	Bias and Variance plotted against Model Complexity . . . . .	7
2.1	Sigmoid Function . . . . .	10
2.2	A decision tree to classify a loan application as safe or risky .	11
2.3	Stochastic Gradient Ascent vs Batch Gradient Ascent . . . . .	14
2.4	Decision Boundary using Gaussian Kernel in SVM . . . . .	15
3.1	KNN with $k = 10$ . . . . .	19
3.2	Diagram of a KD Tree representation . . . . .	20
3.3	Iterations of a K-Means Clustering Algorithm . . . . .	22
3.4	LDA Algorithm using Gibbs Sampling . . . . .	25
3.5	Types of Hierarchial Clustering: Agglomerative and Divisive .	26
4.1	PCA on a 2-dimensional dataset . . . . .	28
4.2	ICA applied to mixture of signals . . . . .	29
5.1	Types of Recommender Systems . . . . .	31
6.1	Basic 3-Layered Neural Network . . . . .	33
6.2	Some commonly used Activation Functions in Neural Networks	35
B.1	Spectral Clustering (top row) outperforms $k$ -Means Clustering (bottom row) on these datasets . . . . .	39

# Chapter 1

## Regression

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable(s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. It is a *Supervised* learning technique, since we are given the value of the independent variable for training our model.

### 1.1 Linear and Polynomial Regression

Linear Regression establishes a relationship between dependent variable  $y$  and independent variables  $x$  using a best fit straight line (also known as regression line). Here, our hypothesis is as follows:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

where we need to estimate the best values of our parameters  $\theta_0$  and  $\theta_1$  in order to best fit the training data.

Now, we define a **cost function** that measures how far our hypothesis is from the actual observed values:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Our objective throughout this process would be to minimize this cost function (here, it is known as **Residual Sum of Squares**). We do this through a process called *Gradient Descent* which would be discussed later in this chapter.

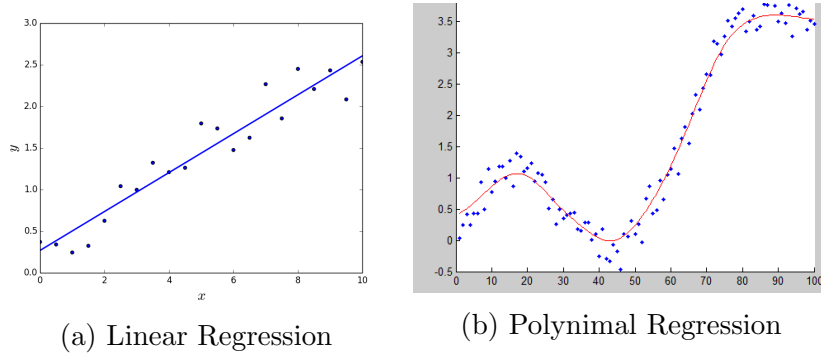


Figure 1.1: Examples of Linear and Polynomial Regression

Often,  $y$  does not just vary linearly with  $x$ , but could be a complex function involving higher degree polynomials in  $x$ . To solve this purpose, we introduce **Polynomial Regression** where our hypothesis is:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_d x^d$$

for some degree  $d$ . Again we minimize the Residual Sum of Squares to try and obtain an optimum value of all the parameters. Fitting a higher-degree polynomial yields lower training error, but can lead to **overfitting**.

## 1.2 Multiple Regression

Multiple Regression refers to Linear Regression with multiple dependent features included. Our hypothesis takes the form:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x_2 + \dots + \theta_n x_n$$

where  $n$  is the number of features included. Using matrix multiplication, this can be viewed as:

$$h_{\theta}(x) = (\theta_0 \ \theta_1 \ \theta_2 \ \dots \ \theta_n) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \theta^T x$$

Geometrically, we try to fit the best possible *hyperplane* to predict the value of the target variable using the  $n$  features. Again, the cost function  $J$  to be minimized is given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

## 1.3 Gradient Descent Algorithm

We wish to choose  $\theta$  such that  $J$  is minimized. Since our function is convex (approx.), we could try to reach the minima by going along the path of maximum decrease in value, i.e, along its negative gradient. To achieve this, we use the Gradient Descent Algorithm.

---

Repeat until convergence  $\{ \theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \}$  for  $j = \{1, 2, 3, \dots, n\}$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 = (h_{\theta}(x) - y) x_j$$

Thus,  $\theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m \frac{1}{m} (y^{(i)} - h_{\theta}(x^{(i)})) x_j$ . Here,  $\alpha$  is the **learning rate**.

Using matrix notation, we can write the algorithm as:

Repeat until convergence  $\{ \Theta \leftarrow \Theta + \alpha \frac{1}{m} X^T (Y - X\Theta) \}$  for  $j = \{1, 2, 3, \dots, n\}$

By representing it through matrices, we get a closed form solution:

$$\Theta = (X^T X)^{-1} X^T Y$$

---

## 1.4 Overfitting and Regularization

To understand overfitting, we need to understand two terms:

1. **Training Error (TE)**: This is the error that our model commits on the *training dataset*. Training Error goes down with increase in model complexity as our model tends to fit the training data very accurately.
2. **Generalization Error (GE)**: It is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data. GE first tends to decrease and then later increase with increase in model complexity. This can be estimated by evaluating our model on a **validation set** using **cross-validation**.

Consider 2 models A and B. Model A is set to overfit the data compared to B when  $TE_A < TE_B$  but  $GE_A > GE_B$ . Polynomial Regression with very high degree polynomials and Multiple Regression with too many features are prone to overfitting.



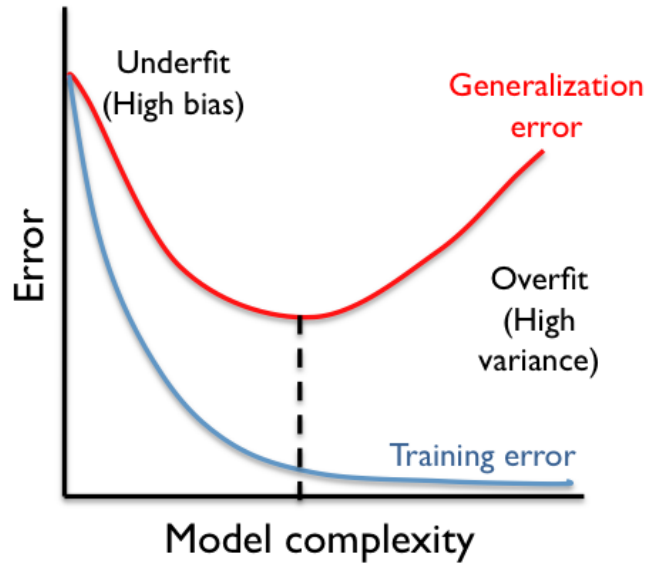


Figure 1.2: Plot of Training and Generalization Error

**Regularization** is a technique that discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. We will now look at two ways to regularize our regression models.

### 1.4.1 L2 Norm and Ridge Regression

L2 Norm of the feature vector  $\theta$  is given by  $\sum_{i=1}^n \theta_i^2$ . This can be added to our cost function  $J$  in order to penalize large weights. The resulting model is known as **Ridge Regression** Model and  $\lambda$  is known as the **L2-Penalty** (extent to which regularization is applied).

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

for  $j = \{1, 2, \dots, n\}$  (We do not penalize the intercept term). As  $\lambda$  increases, coefficients shrink towards 0.

Gradient Descent Algorithm changes to:

---

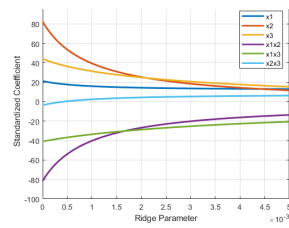
While not converged:

$$\theta_0 \leftarrow \theta_0 + \alpha \sum_{i=1}^m \frac{1}{m} (y^{(i)} - h_{\theta}(x^{(i)})) x_0$$

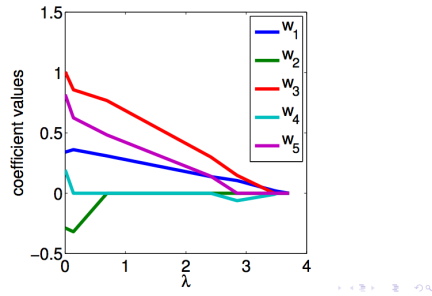
$$\theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m \frac{1}{m} (y^{(i)} - h_{\theta}(x^{(i)})) x_j + \frac{\lambda}{m} \theta_j$$

The closed form solution is given by:

$$\Theta = (X^T X + \lambda I)^{-1} X^T Y$$



(a) Ridge Regression Coefficient Paths



(b) LASSO Regression Coefficient Paths

Figure 1.3: Coefficient Paths

### 1.4.2 L1 Norm and LASSO Regression

L1 Norm of the feature vector  $\theta$  is given by  $\sum_{i=1}^n |\theta_i|$ . Implementation of L1 norm with  $J$  is known as **LASSO Regression (Least Absolute Shrinkage and Selection Operator)**. Along with regularization, it helps in feature selection as well. As  $\lambda$  (**L1 Penalty**) increases, some of the coefficients (corresponding to less important features) become exactly 0, and hence, eliminate the feature from further consideration.

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n |\theta_j|$$

#### Coordinate Descent Algorithm

Coordinate descent is based on the idea that the minimization of a multivariable function can be achieved by minimizing it along one direction at a time. Since  $J(\theta)$  in LASSO Regression is not differentiable at all points, we use this algorithm instead of Gradient Descent (where we require partial derivatives). Here too, we do not regularize the intercept.

Without going into the mathematical derivations of this *soft thresholding*, we directly state its results.

$$\theta_0 = \rho_0$$

$$\theta_j = \begin{cases} \rho_j + \frac{\lambda}{2}, & \rho_j < -\frac{\lambda}{2} \\ 0, & \rho_j \in [-\frac{\lambda}{2}, \frac{\lambda}{2}] \\ \rho_j - \frac{\lambda}{2}, & \rho_j > \frac{\lambda}{2} \end{cases}$$

where  $\rho_j = \sum_{i=1}^m x_j^{(i)}(y^{(i)} - h_{\theta}(x^{(i)})) + \theta_j x_j^{(i)}$

## 1.5 Bias-Variance Tradeoff

- **Bias:** The bias is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- **Variance:** The variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. The bias-variance tradeoff is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set. This is because increasing the bias will decrease the variance whereas increasing the variance will decrease the bias. The following graph explains this tradeoff:

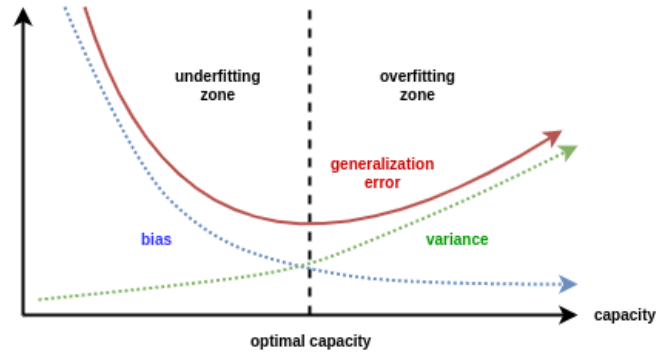


Figure 1.4: Bias and Variance plotted against Model Complexity

## 1.6 Kernel Regression

Kernel regression is a non-parametric technique in statistics to estimate the conditional expectation of a random variable. The objective is to find a non-linear relation between a pair of random variables  $x$  and  $y$ . An example of such a technique is the **k-Nearest Neighbors Regression**. Here, we try to estimate the value of an observation using the  $k$ -nearest observations to our query observation using a certain distance metric. Instead of a global fit, we try to fit the function locally.

# Chapter 2

## Classification

Classification is another form of *Supervised Learning Task* which is used to either predict categorical class labels or classify data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. Classification tasks could be *binary* or *multiclass* classification problems. There are several models used to perform such tasks. Some of them are explained below.

### 2.1 Logistic Regression

Logistic Regression is part of a larger class of algorithms known as **Generalized Linear Models**. It is used to predict a binary outcome (0 or 1), given a set of independent variables. It can be thought of as a special case of linear regression when the outcome variable is categorical, where we are using *log* of odds as dependent variable.

Here, our hypothesis  $h_{\theta}(x)$  lies between 0 and 1, indicative of the probability (or confidence) of its prediction. Suitable thresholding results in categorical output. The hypothesis  $h_{\theta}(x)$  represents  $P(y = 1|x, \theta)$ .

We want  $0 \leq h_{\theta}(x) \leq 1$ , so we use  $h_{\theta}(x) = g(\theta^T x)$  where  $g(x)$  is the **sigmoid** or **logistic function**.

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

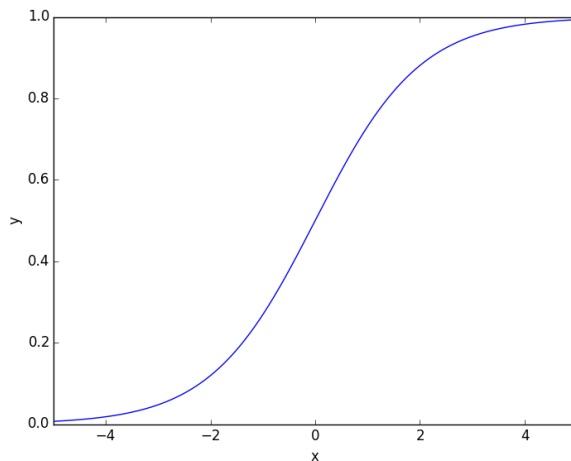


Figure 2.1: Sigmoid Function

Here again, we need to choose the parameters  $\theta$  such that the generalization error of our model is minimum. To achieve this, we define the cost function as the **log likelihood function** and try to use some optimization techniques on it.

For a multiclass classification problem with  $k$  classes, we could use  $k$  different logistic regression models and predict the output to be the class which has the highest probability or confidence. This is called **One vs All Classification**.

## 2.2 Likelihood Function and Gradient Ascent

The quality metric that we use to assess our classification model is *likelihood*, and try to maximize it. The likelihood function is defined as:

$$l(\theta) = \prod_{i=1}^m P(y_i|x_i, \theta) = \prod_{i=1}^m h_{\theta}(x_i)$$

Since it is easier to work with sums as compared to products, we take the *log* of this function and then define our **log-likelihood** function  $ll(\theta)$ .

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

where the first term accounts for the *true* targets while the second term accounts for the *false* targets. Now, we try to maximize this log-likelihood function by applying **Gradient Ascent** algorithm.

---

While not converged:

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

where  $\alpha$  is the *learning rate*.

---

## 2.3 Decision Trees

Decision Tree Algorithm works for both categorical<sup>1</sup> and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter in input variables.

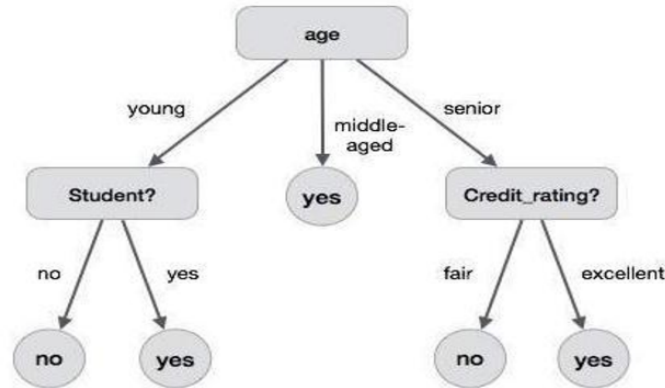


Figure 2.2: A decision tree to classify a loan application as safe or risky

### 2.3.1 Choosing the Best Splitting Feature

At every node of the Decision Tree, we need to *make a decision* to split on some feature and continue doing this until our training data is classified. We use a *recursive greedy algorithm* to select the best splitting feature.

---

<sup>1</sup>For categorical variables, we use the **one-hot encoding** method to represent them

1. At the node under consideration, we calculate the error produced by a **majority-class classifier**<sup>2</sup> (call it MCC-Error)
2. Now, we consider splitting the node based on each feature. For continuous inputs, we need to split for all possible threshold values.
3. After each feature splitting, we calculate the MCC-Error for each child node and calculate the reduction in error from parent to children nodes.
4. The feature which results in the maximum reduction in the MCC-Error is chosen to be the best splitting feature at that node.
5. Now we apply this process recursively until a stopping condition is met.

Instead of MCC-Error, we could use other metrics like **entropy** or **gini-index** to train our decision tree.

### 2.3.2 Preventing Overfitting in a Decision Tree

Extremely complex and deep decision trees are likely to overfit the training data. We can use *early stopping* and *pruning* to overcome this based on **Principle of Occam's Razor**<sup>3</sup>.

Some **Early-Stopping Conditions**:

- \* Limit the depth of tree using a **maxdepth** hyperparameter (by Cross-Validation)
- \* Use classification error to limit tree depth (If no split improves classification error beyond the set threshold, stop splitting)
- \* If number of data points in a node falls below a certain value, stop splitting

**Pruning** is used to simplify a complex tree after learning algorithm terminates. The complexity is measured by the *number of leaf nodes* in the tree. The cost of a tree  $C(T)$  is now given by:

$$C(T) = Error(T) + \lambda(\#leaves)$$

---

<sup>2</sup>Such a classifier predicts the target corresponding to the maximum number of observations under consideration for all the observations.

<sup>3</sup>“Among competing hypotheses, the one with fewest assumptions should be selected”



where  $\lambda$  is *tuning parameter*. The algorithm goes as:

1. Consider a split  $S$  in the tree  $T$
2. Calculate  $C(T)$  of the split  $S$
3. “Undo” the split on  $T_{smaller}$  and calculate  $C(T_{smaller})$
4. If  $C(T_{smaller}) \leq C(T)$ , prune the node.
5. Continue steps 1-4 for all the splits in  $T$

## 2.4 Ensemble of Decision Trees

In 1988, *Kearns* and *Valiant* asked “Can a set of weak learners be combined to create a stronger learner?”. In 1990, *Schapire* came up with an answer “Yes!” and a solution, known as **boosting**. We could come up with an **ensemble of shallow decision tree learners** which are easier to train and consider their predictions, **weighted by their relevance** to build a strong classifier.

### 2.4.1 AdaBoost Algorithm

The AdaBoost Algorithm trains an ensemble of classifiers. To train tree  $f_{t+1}$ , we place higher weights( $\alpha$ ) on misclassified examples in tree  $f_t$  and learn the tree. We weight each tree ( $w$ ) based on how much we trust it and then make prediction based on sum of the weighted predictions of the ensemble.

---

1. Start with unweighted data with  $\alpha_j = 1$
2. For  $t = 1, 2, \dots, T$ :
  - (a) Learn  $f_t(x)$  based on weights  $\alpha_j$ .
  - (b) Compute  $\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - E(\alpha, \hat{y})}{E(\alpha, \hat{y})} \right)$
  - (c) Recompute  $\alpha_j$

$$\alpha_j \leftarrow \begin{cases} \alpha_j e^{-\hat{w}_t} & f_t(x_j) = y_j \\ \alpha_j e^{\hat{w}_t} & f_t(x_j) \neq y_j \end{cases}$$

- (d) Normalize  $\alpha_j$ :  $\alpha_j \leftarrow \frac{\alpha_j}{\sum_{i=1}^N \alpha_i}$

3.  $\hat{y}_{final} = \text{sign} \left( \sum_{t=1}^T \hat{w}_t f_t(x) \right)$

## 2.5 Stochastic Gradient Ascent

For large dataset, with billions of observations, computation of the gradient is computationally very expensive. In stochastic gradient ascent (or descent), the true gradient of  $J(\theta)$  is approximated by a gradient at a single training example. This speeds up the computation exponentially and helps the algorithm converge much faster.

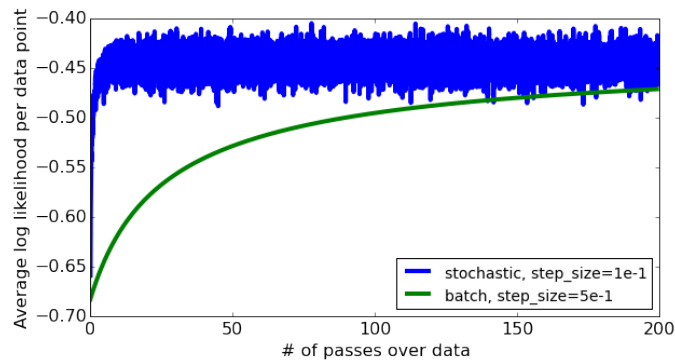


Figure 2.3: Stochastic Gradient Ascent vs Batch Gradient Ascent

But due to the random noise and approximation, the SG algorithm does not exactly converge to the maxima/minima, but oscillates around it. Thus, we need to average over the predictions in the last few iterations to get a good estimate of the actual maxima/minima.

## 2.6 Support Vector Machines(SVM)

SVM is a supervised learning model where we plot each data item as a point in  $n$ -dimensional space (where  $n$  is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well. Maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **margin** and hence, SVM is also called a *large margin classifier*. SVMs can handle both linear as well as non-linear decision boundaries with the help of *kernels*.

### 2.6.1 Kernels

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

Kernels can be different types, for example **linear**, **nonlinear**, **polynomial**, **radial basis function (RBF)** and **sigmoid**. The most used type of kernel function is RBF because it has localized and finite response along the entire x-axis.

Some kernels and their uses:

- **Polynomial Kernel:** It is popular in image-processing. For a  $d$ -degree polynomial,

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j + 1)^d$$

- **Gaussian Kernel:** It is general-purpose kernel, used when there is no prior knowledge about the data.

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

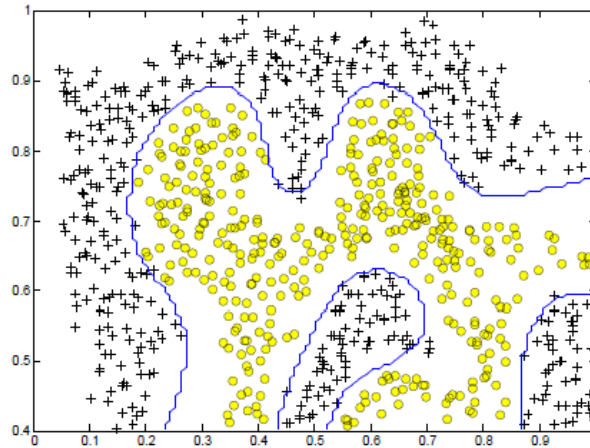


Figure 2.4: Decision Boundary using Gaussian Kernel in SVM

## Chapter 3

# Clustering and Retrieval

**Clustering** is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. Broadly speaking, clustering can be divided into two subgroups :

- **Hard Clustering:** In hard clustering, each data point either belongs to a cluster completely or not.
- **Soft Clustering:** In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned.

**Information Retrieval** is about quickly finding materials in a large collection of unstructured data. The central problem of retrieval is ranking elements of the collection according to relevance for a user query.

Both of these tasks come under the category of *Unsupervised Learning*.

### 3.1 Document Representation

Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers. Thus, we must chose a proper way to represent our documents. A popular and simple method of feature extraction with text data is called the *bag-of-words* model of text.

### 3.1.1 Bag-of-Words Representation

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a “bag” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document. This is done by calculating the *frequency* of occurrence of words in the documents and representing them as vectors. Two documents are said to be **similar if the dot product of their vector representations is large**.

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content”. Example: *the, and, his*, etc.

### 3.1.2 TF-IDF Representation

One approach to overcome the above problem is to rescale the frequency of words by how often they appear in the corpus of documents such that the words occurring frequently across the corpus are penalized heavily. This method is called **Term Frequency-Inverse Document Frequency**.

- **Term Frequency:** Measures how frequently a term occurs in a document.

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

- **Inverse Document Frequency:** Measures how rare the word is across documents

$$IDF(t) = \log_e \left( \frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \right)$$

The TF-IDF score for a word in a document is calculated by multiplying these two terms. Here too, similar documents have higher vector dot products.

## 3.2 Distance Metrics

Measuring similarity or distance between two data points is fundamental to many machine learning algorithms. The similarity is subjective and is highly dependent on the domain and application. Hence, there are several distance metrics developed to cater to different domain specific applications. They include:

- **Minkowski Distance:** For points  $X = (x_1, x_2, \dots, x_n)$  and  $Y = (y_1, y_2, \dots, y_n)$ , it is defined by

$$\left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- **Euclidean Distance:** When  $p = 2$  in Minkowski Distance, it is known as Euclidean Distance. This is the most commonly used distance metric.
- **Manhattan Distance:** Here, distance between two points is the sum of the absolute differences of their Cartesian coordinates, ie,  $p = 1$  in the Minkowski Distance.
- **Chebyshev Distance:** Substituting  $p \rightarrow \infty$  yield this distance. It is used in *warehouse logistics*.
- **Mahalanobis Distance:** It is a measure of the distance between a point  $P$  and a distribution  $D$ , taking into account the covariance of the dataset. Distance of observation  $\mathbf{x}^T$  from set of observations with mean  $\boldsymbol{\mu}^T$  and covariance matrix  $S$  is given by:

$$D(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T S^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

## 3.3 K-Nearest Neighbors Algorithm

K-Nearest Neighbors (KNN) is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. We can implement a KNN model by following the below steps:

1. Initialise the value of  $k$  (by observing and plottig the data points suitably)
2. Calculate the distance between test data and each row of training data (Use the appropriate distance metric for your purpose)

3. Sort the calculated distances in ascending order based on distance values
4. Get the most frequent class of the top  $k$  rows from the sorted array
5. Return this value as the predicted class

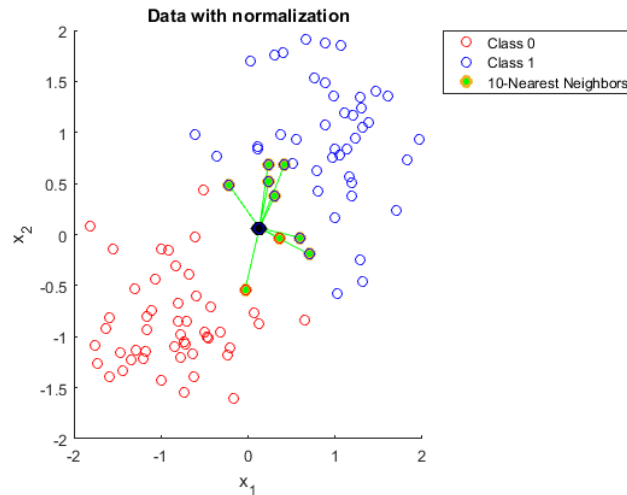


Figure 3.1: KNN with  $k = 10$

The choice of optimum value for  $k$  could be instrumental in preventing overfitting or underfitting. The complexity of KNN algorithm is  $O(N \log k)$  per query. To scale up, we need to prune this search using efficient algorithms discussed below.

## 3.4 KD Trees

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree. We keep recursing this way till a stopping condition is met.

We keep one additional piece of info at each node, which is **the (tight) bounds of points at or below node**. We also use heuristics to make splitting decisions:

- Which dimension to split on? (“Widest” or “Alternate”)
- Which value to split at? (“Median” or “Centre point of the box”)
- When to stop? (“Reach minimum number of points” or “box hits minimum width”)

For a query point, we follow the following steps to look for the nearest neighbour:

1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited
4. Update distance bound when new nearest neighbor is found
5. Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

Under some assumptions on distribution of points, we get complexity of  $O(\log N)$ , but it performs poorly when number of features (dimensions  $d$ ) increases to a large number.

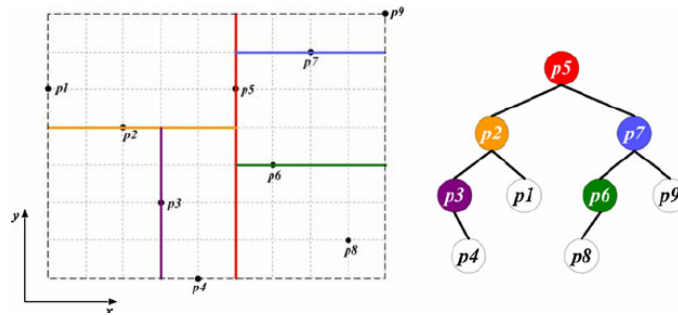


Figure 3.2: Diagram of a KD Tree representation

There are tons of variants of k-d trees which may prune more than allowed but can guarantee that if we return a neighbor at distance  $r$ , then there is no neighbor closer than  $\frac{r}{\alpha}$  for some  $\alpha$ .



### 3.5 Locality Sensitive Hashing

Since KD Trees perform poorly on data with large number of dimensions, we explore another method that efficiently deals with this problem through *approximate neighbor finding*. Locality sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same “bins” with high probability (the number of buckets being much smaller than the universe of possible input items).

This algorithm is implemented as follows:

1. Draw  $h$  random hyperplanes
2. Compute “score” for each point under each line and translate to binary index
3. Use  $h$ -bit binary vector per data point as bin index
4. Create hash table
5. For each query point  $x$ , search  $bin(x)$ , then neighboring bins until time limit or good quality neighbor.

We can improve the search results by building  $h$  hash tables by throwing  $h$  random hyperplanes per hash table and following the steps above and searching only **bins 1 bit off from query**. As  $h$  increases, the probability that two actual nearest neighbors are not in bin 1 bit off reduces exponentially and leads to almost perfect results.

### 3.6 K-Means Clustering

K-means clustering is a type of unsupervised learning. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable  $k$ . It works iteratively to assign each data point to one of  $k$  groups based on the features that are provided. Data points are clustered based on feature similarity.

The algorithm starts with initial estimates for the  $k$  centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

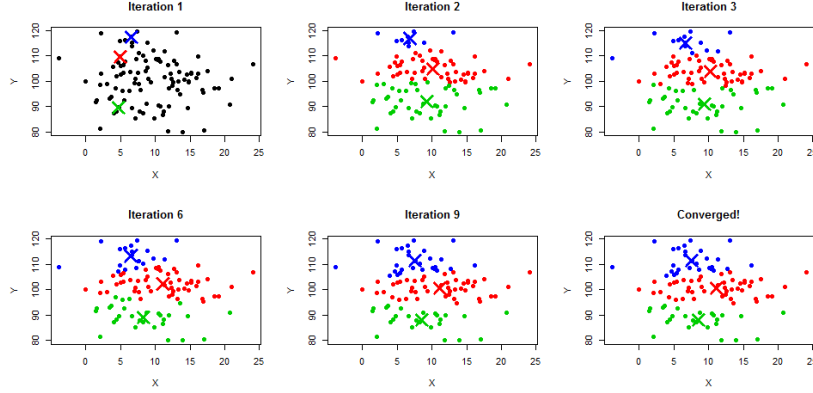


Figure 3.3: Iterations of a K-Means Clustering Algorithm

- **Data assignment step:** If  $c_i$  is the collection of centroids in set  $C$ , then each data point  $x$  is assigned to a cluster based on

$$\arg \min_{c_i \in C} \text{dist}(c_i, x)^2$$

- **Centroid update step:** Now the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster. If  $c_i$  is the centroid of cluster  $S_i$ ,

$$c_i \leftarrow \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

The algorithm iterates between these steps until a stopping criteria is met. It is guaranteed to converge at least to a local optimum.

### 3.7 Gaussian Mixture Models

A mixture model is a probabilistic model for representing the presence of subpopulations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs. A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

Each mixture component represents a unique cluster specified by a *weight*( $\pi_k$ ), *mean*( $\mu_k$ ) and *variance*( $\sigma_k$ ).  $\pi_k$  denotes the initial probability of being from cluster  $k$  while  $N(x_i|\mu_k, \sigma_k)$  denotes how likely is the observed value  $x_i$  under the cluster assignment  $k$ .

### 3.7.1 EM Algorithm

Before discussing the EM Algorithm, we would bring in the concept of responsibility that a cluster  $k$  takes for an observation  $i$ , given  $\pi_k, \mu_k$  and  $\sigma_k$ :

$$r_{ik} = \frac{\pi_k N(x_i|\mu_k, \sigma_k)}{\sum_{j=1}^K \pi_j N(x_i|\mu_j, \sigma_j)}$$

The **Expectation-Maximization Algorithm** is an iterative algorithm comprizing of 2 steps:

1. **E-Step:** *Estimate* cluster responsibilities given current parameter estimates (using the above formula)
2. **M-Step:** *Maximize* the likelihood over parameters given current responsibilities, ie, recalculate  $\pi_k, \mu_k$  and  $\sigma_k$  using the calculated responsibilities.

$$\begin{aligned}\pi_k &\leftarrow \frac{\sum_{i=1}^N r_{ik}}{N} \\ \mu_k &\leftarrow \frac{\sum_{i=1}^N r_{ik} x_i}{\sum_{i=1}^N r_{ik}} \\ \sigma_k &\leftarrow \frac{\sum_{i=1}^N r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^N r_{ik}}\end{aligned}$$

The EM Algorithm is a *Coordinate Descent* algorithm and converges to a local mode. The  $k$  centroids could be initialized using the resulting centroids of running a  $k$ -means clustering on the dataset to obtain better results. The  $k$ -means hard assignments could be retrieved from the EM method by setting the responsibilities to strictly 1 or 0 based on closest centroid to the data point.

## 3.8 Mixed Membership Modeling

Mixed membership models have emerged over the past 20 years as a flexible cluster-like modeling tool for unsupervised analyses of high-dimensional multivariate data. Here, one assumes that every observation partially belongs to all clusters, according to an individual membership vector.

The general mixed membership model relies on four levels of assumptions: **population**, **subject**, **latent variable** and **sampling scheme**. Population level assumptions describe the general structure of the population that is common to all subjects. Subject level assumptions specify the distribution of observed responses given individual membership scores. Membership scores are usually unknown and hence can also be viewed as latent variables. The next assumption specifies whether the membership scores are treated as unknown fixed quantities or as random quantities in the model. Finally, the last level of assumptions specifies the number of distinct observed characteristics and the number of replications for each characteristic.

Such models are important for document classification, where a document could belong to several topics (according to the ideas that it presents).

### 3.8.1 Latent Dirichlet Allocation

LDA represents documents as mixtures of topics that split out words with certain probabilities. It assumes that documents are produced in the following fashion: when writing each document, we:

- Decide on the number of words  $N$  the document will have
- Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of  $K$  topics)
- Generate each word  $w_i$  in the document by:
  - First picking a topic (according to the multinomial distribution that has been sampled above)
  - Using the topic to generate the word itself (according to the topic's multinomial distribution)

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

### 3.8.2 Gibbs Sampling

Gibbs Sampling is one member of a family of algorithms from the *Markov Chain Monte Carlo* (MCMC) framework. It is based on sampling from conditional distributions of the variables of the posterior. To sample  $\mathbf{x}$  from a joint distribution  $p(\mathbf{x}) = p(x_1, \dots, x_m)$  using Gibb's Sampling, one would perform the following:

1. Randomly initialize each  $x_i$ .

2. For  $t = 1, \dots, T$ :

$$\begin{aligned} x_1^{t+1} &\sim p(x_1 | x_2^t, x_3^t, \dots, x_m^t) \\ x_2^{t+1} &\sim p(x_2 | x_1^t, x_3^t, \dots, x_m^t) \\ &\vdots \\ x_m^{t+1} &\sim p(x_m | x_1^t, x_2^t, \dots, x_{m-1}^t) \end{aligned}$$

This procedure is repeated a number of times until the samples begin to converge to what would be sampled from the true distribution. In the

```

Input: words  $\mathbf{w} \in$  documents  $\mathbf{d}$ 
Output: topic assignments  $\mathbf{z}$  and counts  $n_{d,k}$ ,  $n_{k,w}$ , and  $n_k$ 
begin
  randomly initialize  $\mathbf{z}$  and increment counters
  foreach iteration do
    for  $i = 0 \rightarrow N - 1$  do
       $word \leftarrow w[i]$ 
       $topic \leftarrow z[i]$ 
       $n_{d,topic} -= 1$ ;  $n_{word,topic} -= 1$ ;  $n_{topic} -= 1$ 
      for  $k = 0 \rightarrow K - 1$  do
         $p(z = k | \cdot) = (n_{d,k} + \alpha_k) \frac{n_{k,w} + \beta_w}{n_k + \beta \times W}$ 
      end
       $topic \leftarrow \text{sample from } p(z | \cdot)$ 
       $z[i] \leftarrow topic$ 
       $n_{d,topic} += 1$ ;  $n_{word,topic} += 1$ ;  $n_{topic} += 1$ 
    end
  end
  return  $\mathbf{z}$ ,  $n_{d,k}$ ,  $n_{k,w}$ ,  $n_k$ 
end

```

Figure 3.4: LDA Algorithm using Gibbs Sampling

algorithm,  $n_{d,k}$  represents number of words assigned to topic  $k$  in document  $d$ ;  $n_{k,w}$  represents the number of times word  $w$  is assigned to topic  $k$ ;  $n_k$  represents total number of times any word is assigned to topic  $k$ .

## 3.9 Hierarchical Clustering

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. The results of hierarchical clustering can be shown using **dendrograms**. In order to decide which clusters should be combined or where a cluster should be split, a measure of dissimilarity between sets of observations is required. This is achieved by use of an appropriate *metric* and a *linkage criterion*. The choice of an appropriate metric will influence the shape of the clusters while linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations.

### 3.9.1 Divisive Clustering

This is a “top down” approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy. Divisive clustering with an exhaustive search is  $O(2^n)$ , but it is common to use faster heuristics to choose splits, such as k-means.

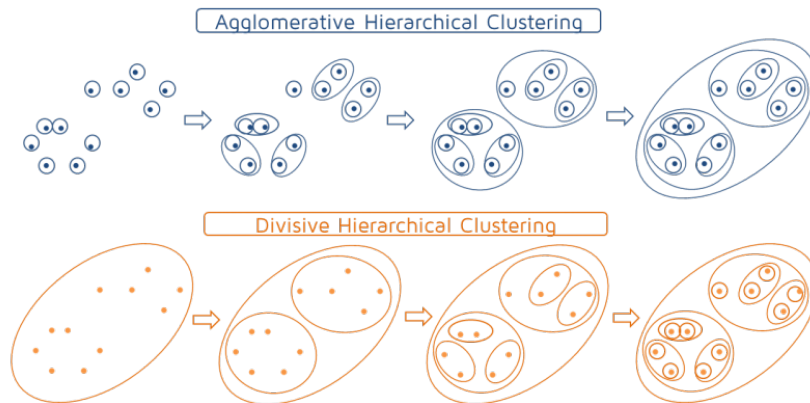


Figure 3.5: Types of Hierarchical Clustering: Agglomerative and Divisive

### 3.9.2 Agglomerative Clustering

This is a “bottom up” approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. The algorithm has a time complexity of  $O(n^3)$ , which makes it slow even for medium datasets. However, for some special cases, optimal efficient agglomerative methods are known. Cutting the dendrogram at a given height will give a partitioning clustering at a selected precision.

# Chapter 4

## PCA and ICA

Measurements often do not reflect the very thing intended to be measured. They are corrupted by random noise, and may also be redundant (example: one feature measuring speed in km/h and another in m/s). This is quite common in real-world datasets with a large number of features. Often, measurements can not be made in isolation, but reflect the combination of many distinct sources. PCA or Principal Component Analysis solves the problem of redundancy by *Dimensionality reduction* whereas ICA or Independent Component Analysis addresses a more general class of *blind source separation* problems.

### 4.1 PCA

**Principal Component Analysis** is a great tool that helps to automatically detect correlations between different features in a dataset. Given a dataset  $\{x^{(i)} | i = 1, 2, 3, \dots, m\}$ , PCA helps to reduce this  $n$ -dimensional dataset ( $n \ll m$ ) to a certain  $k$ -dimensional dataset where  $k < n$ . Here, the features of the  $k$ -dimensional dataset may not hold *physical* significance, but it nearly captures all the variance that the data possesses.

Prior to applying the PCA algorithm, the data must be pre-processed as follows:

1. Let  $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$
2.  $x^{(i)} \leftarrow x^{(i)} - \mu$
3. Let  $\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)})^2$
4.  $x_j^{(i)} \leftarrow \frac{x_j^{(i)}}{\sigma_j}$

Consider an example of a 2-dimensional dataset where the 2 features are highly correlated. After the above normalization, we wish to find a direction  $u$  such that majority of the variance of the data lies along  $u$ . Thus, we need to maximize the sum of squares of lengths of the projections on unit-vector  $u$ , which could be found by  $x^T u$ . Thus we have to maximize:

$$\frac{1}{m} \sum_{i=0}^m (x^{(i)T} u)^2 = u^T \left( \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \right) u$$

subject to  $\|u\|_2 = 1$ .

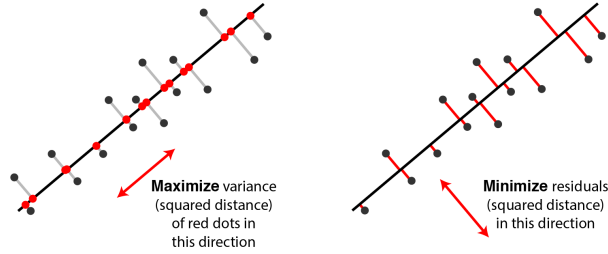


Figure 4.1: PCA on a 2-dimensional dataset

This boils down to finding the **principal eigenvector** of  $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T}$ , which is just the **covariance matrix**. More generally, if we wish to project our data into a  $k$ -dimensional subspace ( $k < n$ ), we should choose  $u_1, u_2, \dots, u_k$  to be the top  $k$  eigenvectors of  $\Sigma$ . These form an orthogonal basis to represent  $y^{(i)}$ .

$$y^{(i)} = \begin{pmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{pmatrix} \in \mathbb{R}^k$$

PCA is used often to obtain a visual representation of the data in 2D or 3D. It is also used to store large volumes of data in smaller spaces without losing much information. They **eigenfaces algorithm** used in face recognition, implements PCA to recognize faces based on most important features.



## 4.2 ICA

**Independent Component Analysis**, like PCA is a method to represent data using a different basis. A motivating example is to consider the *cocktail party problem*. Here,  $n$  speakers are speaking simultaneously at a party. We have  $n$  different microphones placed at different distances from the speakers that record the sound of the speakers. Using these recordings, we wish to generate the speech of each individual speaker.

To formalize this, we have some data  $s \in \mathbb{R}^n$  produced by  $n$  different sources. We observe  $x = As$ , where  $A$  is an unknown **mixing matrix**. Our goal is to recover the sources  $s^{(i)}$  given  $\{x^{(i)} | i = 1, 2, 3, \dots, m\}$ .

The ICA algorithm involves the following steps:

1. Subtract off the mean of data in each dimension
2. *Whiten* the data by calculating the eigenvectors of the covariance of the data
3. Identify final rotation matrix that optimizes statistical independence

The above steps involve serious linear algebra including **Singular Value Decomposition (SVD)**. The *scipy* library of python contains a function called **FastICA** used to perform ICA on a given dataset and yield the individual components.

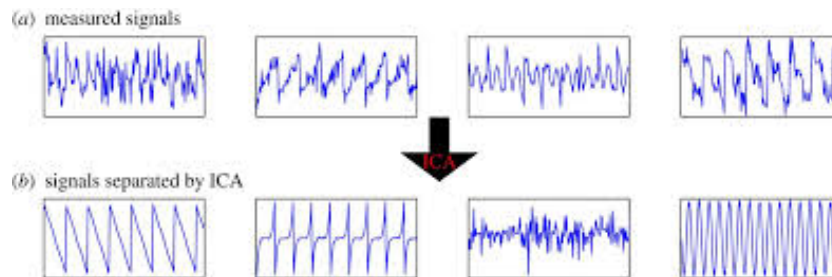


Figure 4.2: ICA applied to mixture of signals

ICA finds applications in signal processing, text mining, image denoising, etc.

# Chapter 5

## Recommender Systems

A recommender system is a subclass of information filtering system that seeks to predict the “rating” a user would give to an item. Recommendation engines discover data patterns in the data set by learning consumers’ choices and produce the outcomes that co-relate to their needs and interests. They typically produce a list of recommendations in one of two ways – through *content-based filtering* or through *collaborative filtering*. Of late, *hybrid methods* have provided much better results than using the above methods individually.

### 5.1 Content-Based Filtering

A content-based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

The features of items are stored as vectors and mapped with features of users in order to obtain user-item similarity. The top matched pairs are given as recommendations. The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) which are used in information retrieval systems, are also used for content-based filtering mechanisms. Similarity is then calculated using the **cosine similarity** parameter for the n-dimensional vectors.

However, content-based recommenders have their own limitations. They are not good at capturing inter-dependencies or complex behaviors.

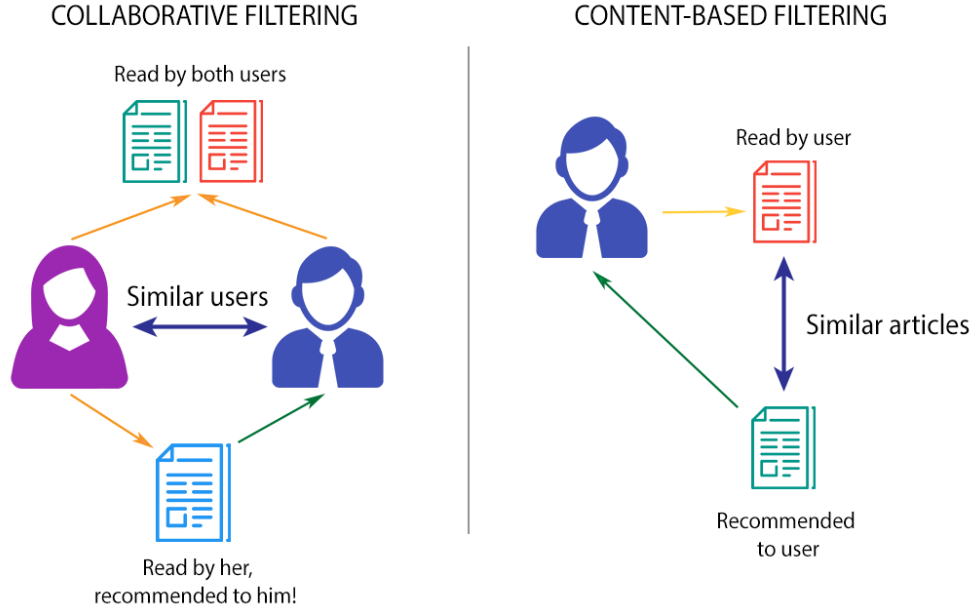


Figure 5.1: Types of Recommender Systems

## 5.2 Collaborative Filtering

Collaborative Filtering algorithm considers “user behaviour” for recommending items. They exploit behaviour of other users and items in terms of transaction history, ratings, selection and purchase information. Other users behaviour and preferences over the items are used to recommend items to the new users. In this case, features of the items are not known. This method takes into account the fact that every user may not have rated all the movies.

The problem can be formulated as follows:

- $r(i, j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
- $y^{(i,j)}$  = rating by user  $j$  on movie  $i$
- $\theta^{(j)}$  = parameter vector for user  $j$
- $x^{(i)}$  = feature vector for movie  $i$
- For user  $j$ , movie  $i$ , predicted rating:  $(\theta^{(j)})^T(x^{(i)})$
- $m^{(j)}$  = number of movies rated by user  $j$

The cost functions is given by:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 +$$

$$\frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

where  $\lambda$  is the regularization parameter.

Given  $x^{(1)}, \dots, x^{(n_m)}$  (movie ratings), we can predict  $\theta^{(1)}, \dots, \theta^{(n_u)}$  (using Linear Regression). Similarly, given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , we can calculate  $x^{(1)}, \dots, x^{(n_m)}$  and iterate through these steps.

Therefore, we first initialize  $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$  to small random values and minimize the cost function  $J$  using Gradient descent (or Advanced Optimization Algorithms).

$$x_k^{(i)} \leftarrow x_k^{(i)} - \alpha \left( \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} \leftarrow \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

Since the solving of this system of equation of systems can also be represented as a product of two **low-rank matrices**  $X$  and  $\Theta$  ( $\hat{Y} = X\Theta^T$ ) where:

$$X = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(n_m)})^T \end{pmatrix} \Theta = \begin{pmatrix} (\theta^{(1)})^T \\ (\theta^{(2)})^T \\ \vdots \\ (\theta^{(n_u)})^T \end{pmatrix}$$

The problem boils down to estimate the best values of these matrices and the process is also called **Low-Rank Matrix Factorization**.

## Chapter 6

# Neural Networks and Deep Learning

Artificial neural networks (ANNs) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected nodes or **perceptrons** which loosely model the neurons in a biological brain. The original goal of the ANN approach was to solve problems in the same way that a human brain would. Now, they are being used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

The simplest neural network consists of 3 layers: *input*, *hidden* and *output* layers. The connections between these layers shown have associated **weights**

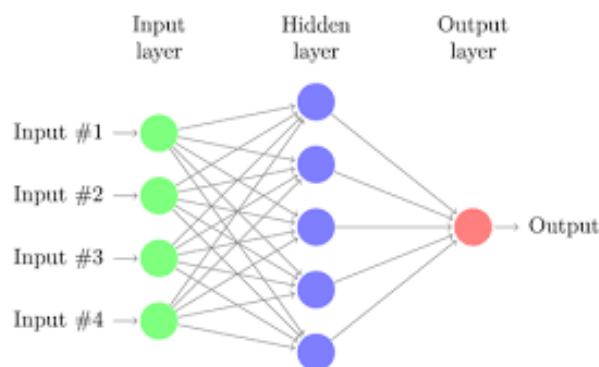


Figure 6.1: Basic 3-Layered Neural Network

and an overall **bias**. These interconnections, along with the use of **sigmoid**,

**ReLU** and other *activation functions* help to model a great deal of non-linearity between different features in the dataset.

Introducing multiple hidden layers makes the network “deep” and can be used to learn extremely complex non-linear relationships between features to predict the final output and produce a staggering accuracy.

Without going into the nitty-gritties of the mathematics involved in various neural networks, we will discuss some of the important theoretical aspects of these beautiful Deep Learning tools.

## 6.1 Neuron Activation and Back-Propagation

In neural networks, the activation function of a node defines the output of that node given a set of inputs. Nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes.

Backpropagation (or **backprop**) is a method used to calculate a gradient that is needed in the calculation of the weights to be used in the network. To calculate the gradient of a scalar  $z$  w.r.t some ancestor  $x$  in a graph, we start by observing that  $\frac{dz}{dz} = 1$ . We then compute the gradient of each parent of  $z$  by multiplying the gradient with the *Jacobian*<sup>1</sup> of the operation that produced  $z$ . We continue multiplying Jacobians, travelling *backwards* until we reach  $x$ .

## 6.2 Types of Neural Networks

There are several kinds of artificial neural networks. These type of networks are implemented based on the mathematical operations and a set of parameters required to determine the output.

- **Convolutional Neural Networks:** In a convolutional layer each neuron is only connected to a few nearby local neurons in the previous layer, and the same set of weights is used to connect to them (to look for similar features spacially). This is followed by *pooling layer* which

---

<sup>1</sup>the Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function. If the matrix is a square matrix, both the matrix and its determinant are referred to as the Jacobian





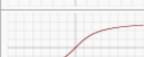

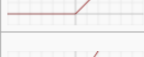


Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figure 6.2: Some commonly used Activation Functions in Neural Networks

combine the outputs of neuron clusters at one layer into a single neuron in the next layer. After some convolutional and pooling layers, some fully connected layers are added before concluding outputs.

- **Restricted Boltzmann Machine:** RBMs are shallow neural nets that learn to reconstruct data by themselves in an unsupervised fashion. They have two layers - *visible* and *hidden* layers. RBM training has 2 phases: the Forward Pass and the Backward Reconstruction. They are useful for Collaborative Filtering, dimensionality reduction, classification, etc.
- **Recurrent Neural Networks:** In RNNs, a recurrent neuron stores the state of a previous input and combines with the current input thereby preserving some relationship of the current input with the previous input. This can be achieved using **Long-Short Term Memory (LSTM)** Model and is used for speech recognition, text analytics, image captioning, translations, etc.

- Other major types include **Recursive Neural Tensor Nets** (RNTN), **Deep Belief Nets** (DBN), **Autoencoders**, etc.

## 6.3 Advanced Optimization Algorithms

With a massive number of parameters to work with, simple Stochastic Gradient Descent approaches fail to perform effectively on several forms of neural networks. Following are some advanced optimization algorithms that work with adaptive learning mechanisms to efficiently help a neural network learn from a dataset.

- **AdaGrad**: This method adapts the learning rate to the parameters by scaling them inversely to the *square root of the sum of all the historical squared gradients*. It performs smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features. AdaGrad performs well on *sparse data*.
- **Nesterov Momentum**: Momentum is an update method where the parameter vector will build up *velocity* in any direction that has consistent gradient. The core idea of Nesterov Momentum is that if we are about to compute the gradient, we can treat the future approximate position as a “lookahead” and compute the gradient at the lookahead position instead of at the old position.
- **RMSProp**: This method modifies AdaGrad to perform better in non-convex settings by using an *exponentially decaying average* to discard the history from extreme past so as to converge rapidly after finding a convex bowl.
- **Adam**: It is seen as the variant of RMSProp and Momentum (with some distinctions). Momentum is incorporated directly as an estimate of the first-order moments of the gradients. Adam also contains a *bias-correction term* to make the convergence faster.



# References

1. Theoretical Guidance on Machine Learning Algorithms and Techniques  
<http://www.analyticsvidhya.com>  
<http://www.datascience.com>
2. Stanford CS 229 Machine Learning Course Material  
<http://cs229.stanford.edu/materials.html>  
<http://cs229.stanford.edu/notes>
3. Machine Learning Specialization by University of Washington on Coursera  
<https://www.coursera.org/learn/ml-foundations>  
<https://www.coursera.org/learn/ml-regression>  
<https://www.coursera.org/learn/ml-classification>  
<https://www.coursera.org/learn/ml-clustering-and-retrieval>
4. Machine Learning and Deep Learning Blogs  
<https://iamtrask.github.io/>  
<http://colah.github.io/>
5. Cognitive Class.ai ML0120EN Course - Deep Learning with TensorFlow
6. Research Papers on Several Topics:
  - William M. Darling. *A Theoretical and Practical Implementation Tutorial on Topic Modeling and Gibbs Sampling*, 2011
  - Jonathon Shlens. *A Tutorial on Independent Component Analysis*, 2014
  - Jonathon Shlens. *A Tutorial on Principal Component Analysis*, 2014
  - Damien Francois, Vincent Wertz, and Michel Verleysen. *Choosing the Metric: A Simple Model Approach*, 2011
  - Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, Yuval Kluger. *SpectralNet: Spectral Clustering using Deep Neural Networks*, 2018
  - Marc T. Law, Raquel Urtasun, Richard S. Zemel. *Deep Spectral Clustering Learning*, 2017

# Appendix A

## Some Platforms & Libraries to Accomplish Machine Learning Tasks (using Python)

Following are the Python Libraries and Platforms that I used throughout this Summer for performing Machine Learning and Deep Learning tasks:

- **Scikit-learn:** It is a python package that contains simple and efficient tools for data mining and data analysis, built on `numpy`, `SciPy` and `matplotlib`. It provides easy-to-use functions to perform Regression, Classification, Clustering, Dimensionality Reduction, Model Selection, Text Analytics, etc. on datasets.
- **GraphLab Create:** It is a Python library, backed by a C++ engine, for quickly building large-scale, high-performance data products. It offers state-of-the-art machine learning algorithms including deep learning, boosted trees, and factorization machines. It uses a data structure called `SFrame` and offers a wide range of functions to implement ML algorithms on the data.
- **TensorFlow:** It is an open source software library for high performance numerical computation. It can be used for Machine Learning tasks, but is more often used for Deep Learning tasks as it allows users to create and manipulate the architecture of their Neural Nets with great ease and facilitates learning for all kinds of networks. The data is represented and manipulated as *tensors* and its flow is stored in a *computational graph*. Several companies such as Google, Nvidia, Uber, Snapchat, etc. have used TensorFlow to create great custom applications suiting their requirements.

# Appendix B

## Deep Spectral Clustering

I am including this in the Appendix due to the fact that this clustering algorithm has fascinated me a lot. Also, I am trying to implement this model with some variations on relatively large datasets.

Spectral Clustering techniques make use of the **spectrum (eigenvalues)** of the similarity matrix<sup>1</sup> of the data to perform dimensionality reduction before clustering in fewer dimensions. The general approach to spectral clustering is to use a standard clustering method on relevant eigenvectors of a Laplacian matrix of the similarity matrix. The eigenvectors that are relevant are the ones that correspond to smallest several eigenvalues of the Laplacian except for the smallest eigenvalue which will have a value of 0.

Deep Spectral Clustering uses the power of Autoencoders and other types of Neural Networks to efficiently compute eigenvectors and use SGD to cluster huge and complex looking datasets. Scalability is the major issue that is addressed by Deep Spectral Clustering over simple Spectral Clustering.

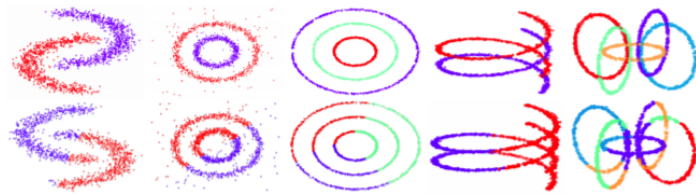


Figure B.1: Spectral Clustering (top row) outperforms  $k$ -Means Clustering (bottom row) on these datasets

---

<sup>1</sup>A similarity matrix is a matrix of scores which express the similarity between two data points