Metrics

- Classification
  - Accuracy: true_positive + true_negative
  - Recall/Precision/F. F1 is balance between recall and precision
  - NegativeLogLikelyhood
  - Area Under ROC Curve(AUC) for binary classification

    he AUC for a classifier with no power, essentially random guessing, is 0.5, because the curve follows the diagonal. The AUC for that mythical being, the perfect classifier, is 1.0. Most classifiers have AUCs that fall somewhere between these two values.
  - Confusion Matrix
- Regression
  - Mean Absolute Error
  - Mean Square Error. It gives more punishment for large error
  - R^2 (R Sqaure)

Model

- Loss Function

  need to be non-negative

  0 = perfect prediction
- No Free Lunch Theorem

  > No single model that works best!
  >
  > Model makes assumptions!
- Model Selection
  - Cross-validation
  - Bayesian model selection

    $P(m|D) = \frac{P(D|m_0)}{P(D|m_1)} * \frac{P(m_0)}{P(m_1)}$ (Usually same prior)
- Bias-Variance Tradeoff
  - Bias: error from mean

    - Variance: spread around center
- Bayesian Information Criterion (BIC)

  $BIC = DOF(\theta)logN - 2logP(D|\theta)$: penalise high DOF model, mitigated by its likelyhood

Linear Regression

- Assumptions:
  1. Linearity: data is linear
  2. Independence of error: data is IDD sampled
  3. Residue(error) is Gaussian
  4. Equal variance of errors
- Loss Function:
  - Based on Gaussian Error
  - MSE
- Optimization (Solution):
  - Derivation of close form solution $(X^T X)^{-1} X^T y$
- Gaussian MLE always biased: $E[\sigma^2] = \frac{n-1}{n}\sigma^2$ Smaller than the true error. Because we measure from sample
- Expectation of Error = bias * bias + variance
- Regularization
  - More regularization -> high bias term -> more bias
  - Less regularization -> more overfitting -> more variance

KNN

- Idea: choose class based on K nearst neighbor
- Caveats:
  - Need to normalise vector
- Model Complexity:
  - K too small: overfitting
  - K too high: underfit

Logistic Regression

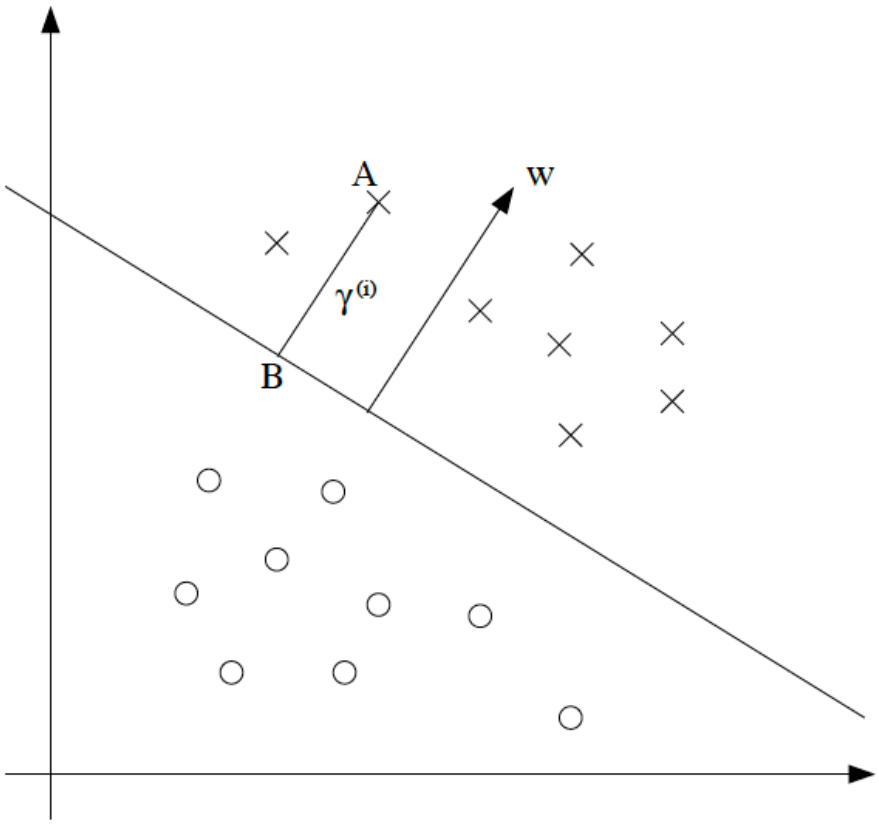
- Idea: $p(y=1|x) = \frac{1}{1+e^{-w^T x}} y \in \{0, 1\}$
- Why not use Linear Regression for Classification?
  - Ans: Classification does not care about the magnitude of loss, outlier will hur linear regression for classification badly.
- Derivation
  - odds of two class $\frac{P(y=1|x)}{P(y=0|x)} = e^{w^T x}$
  - then we have $\frac{p}{1-p} = e^{w^T x}$
  - solve for p
- Soft-max interpretation
  - $\frac{e^{G(x,y)}}{e^{G(x)}+e^0}$
- Loss function:
  - LIkelyhood function: MLE of every data
    $\sum log[y_i P(y=1|x_i) + (1-y_i)P(y_i=0|x_i)]$
- Optimization:
  - Gradient: $-\sum(y_i - p(y_i|x_i, w_i))x_i$
  - Gradient Ascent
    - SGD or Batch or Whole Data

Perceptron

- Idea: $y = sign(w^T x) y \in \{-1, 1\}$
- Loss function:
  - 1 loss if incorrect, 0 otherwise
  - or $L = max(0, -y_i w^T x_i)$
- Optimization:
  - Gradient:
    - $\nabla L = 0$ if $y_i w^T x_i > 0$, else $-y_i x_i$
  - Weight Update:
    - $w^{i+1} = w^i + \eta(y_i - \hat{y}_i)x_i$, ($y - \hat{y}_i$ is either 2 or -2)
- Online-Learning vs. Batch Learning
  - Assumptions behind `batch learning`:
    - **one optimal hypothesis fits all of the data**
    - data are IID
    - Advantage:
      - Guarentee convergency
  - Online Learning:

- No train-test data split. Infinite data steam
- **No consistency in hypothesis**
- Advantage
  - Handle large data
  - Faster for lots of data

SVM

- Idea: find the best plane that separate data by maximise the margin to each of example
- Derivation (highly recommend reading: Stanford CS229 Notes)
  - Functional Margin: $\hat{\gamma}_i = y_i(w^T x_i + b)$
  - Geometric Margin (norm of vector AB), and it satisfy following function $\gamma\frac{\dot{W}}{||w||} = A - B$. Also since B is on the decision boundary, $w^T B = 0$, we can derive geo margin $\gamma_i = y_i[(\frac{w}{||w|}|)^T x_i + \frac{b}{||w||}]$ We can see that if $||w|| = 1$, then $\gamma = \hat{\gamma}$, which implies $\frac{\hat{\gamma}}{\gamma} = ||w||$
  - 

  - Therefore, our objective is to:

    > max γ, s.t. $y_i(w^T x_i + b) \geq \gamma, \forall i$ and $||w|| = 1$

    Then, since $\frac{\hat{\gamma}}{\gamma} = ||w||$, we can rewrite objective function to:

    > max $\frac{\hat{\gamma}}{||w||}$, s.t. $y_i(w^T x_i + b) \geq \hat{\gamma}, \forall i$

Finally, since maximise max $\frac{\hat{\gamma}}{||w||}$ is equivalent to min $\frac{1}{2}||w||^2$, and $\hat{\gamma}$ can be arbitrarily scaled, we set $\hat{\gamma} = 1$, then we have

> max $\frac{1}{2}||w||^2$, s.t. $y_i(w^T x_i + b) \geq 1, \forall i$

- Loss function
  - Hinge Loss: $max(0, 1 - y \cdot wx)$

SVM Duality

- Dual vs Primal max (Please refer to Andrew Ng CS229 notes)

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i^T \mathbf{x}_j)$$

such that $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

- Predictions for new examples:

$$\mathbf{x}^T \cdot \mathbf{w} = \mathbf{x}^T \cdot \sum_{i=1}^{N} [\alpha_i y_i \mathbf{x}_i] = \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}^T \mathbf{x}_i)$$

  - Note $w = \sum [a_i y_i x_i]$ and only some of a_i are non-zero, which are the support vector
- Slack Variable
  - Add regulariser term $min_w 1/2||w||^2 + C \sum \epsilon_i$ s.t $(w^T x_i) y_i + \epsilon_i \geq 1$
  - Larger C means more penalty on slacking the margin
    - Better fit to data. More overfitting.
- New Prediction for test data x_j is given by $x_j^T \cdot w = x_j^T \sum [a_i y_i x_i] = \sum a_i y_i (x_j^T x_i)$

Kernel

- How to use kernel?
  1. Feature mapping: $\phi(x)$ to higher dimension to make data linearly separable. e.g $\phi(x) = [x_{(1)}^2, \sqrt{2} x_{(1)} x_{(2)}, x_{(2)}^2]$

     In primal form, we need to learn w for each feature, which could be very large

     While in dual form, we don't have w
  2. We can use $\phi(x)$ instead of x in our computation in order to learn non-linear boundary.

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j))$$

the only increase for the size of the dataset

ere is no **w!**

3. Then we can define Kernel $K(x, x') = \phi(x)^T \phi(x')$.

   Why we want to define it as kernel? Because we can often have simpler solution to Kernel without actually computing the ϕ in a possibly very high dimension

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}')^2 \\ &= (x_{(1)} x'_{(1)} + x_{(2)} x'_{(2)})^2 \\ &= (x^2_{(1)} x'^2_{(1)} + x^2_{(2)} x'^2_{(2)} + 2 x_{(1)} x_{(2)} x'_{(1)} x'_{(2)}) \\ &= (x^2_{(1)}, x^2_{(2)}, \sqrt{2} x_{(1)} x_{(2)}) \cdot (x'^2_{(1)}, x'^2_{(2)}, \sqrt{2} x'_{(1)} x'_{(2)}) \\ &= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}') \end{aligned}$$

For example above, we **don't** need to actually compute the inner product of $\phi(x)$, which is $(x_1^2 x'^2_{(1)} + \ldots\ldots + 2 x_{(1)} x_{(2)} x'_{(1)} x'_{(2)})$, All we need to compute is $(x \cdot x')^2$

- Kernel as similarity Function

$$\alpha K(\mathbf{x}, \mathbf{x}') = \alpha \phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = \alpha ||\phi(\mathbf{x})|| ||\phi(\mathbf{x}')|| \cos(\theta)$$

same for all x'

- What qualify a Kernel?
  - You can decompose it into $\phi(x)^T \phi(x')$
  - Or it satisfy Mercer's Theorem : kernel matrix is symmetric positive semi-definite
    - Kernel Matrix $\mathbf{K}_{ij} = \mathbf{K}_{ji}$
    - $\mathbf{x}^T K \mathbf{x} \geq 0, \forall \mathbf{x} \in R^m$
- Kernel formation

$$K(\mathbf{x}, \mathbf{x}') = cK_1(\mathbf{x}, \mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})K_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(K_1(\mathbf{x}, \mathbf{x}')\right)$$
$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') + K_2(\mathbf{x}, \mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}')K_2(\mathbf{x}, \mathbf{x}')$$

- Gaussian Kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

  Larger σ means the support vector has greater influence area

- Summary
  - Good
    - Arbitrarily high dimension
    - Extension to other data types
    - Non-linearity
  - -Bad
    - Choose kernel is not easy
    - Can not handle large data

Decision Tree

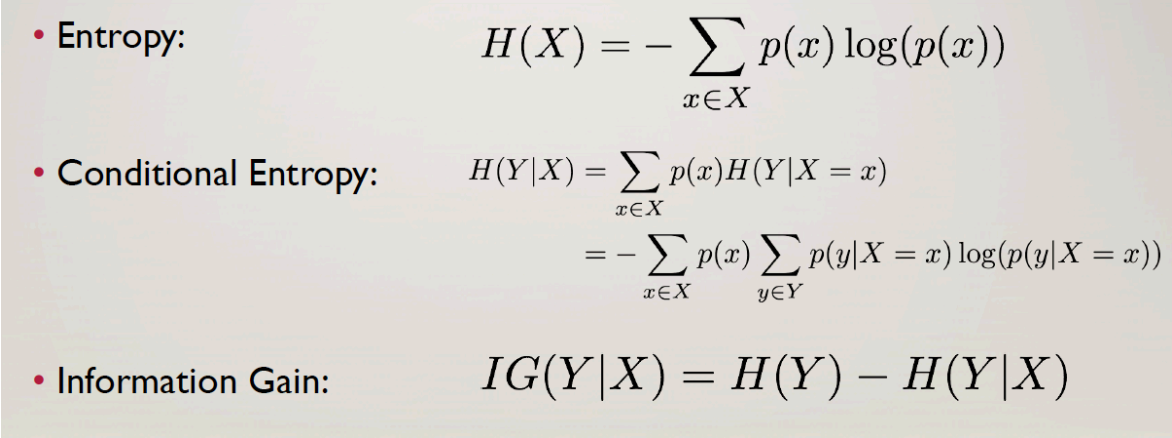
- ID3 Algorithm

```
function buildDecisionTree(data, labels):
    if all labels the same:
        return leaf node for that label
    else:
        let f be best feature for splitting (needs to be computed)
        left = buildDecisionTree(data with f=0, labels with f=0)
        right = buildDecisionTree(data with f=1, labels with f=1)
        return Tree(f, left, right)
```

Does this always terminate?

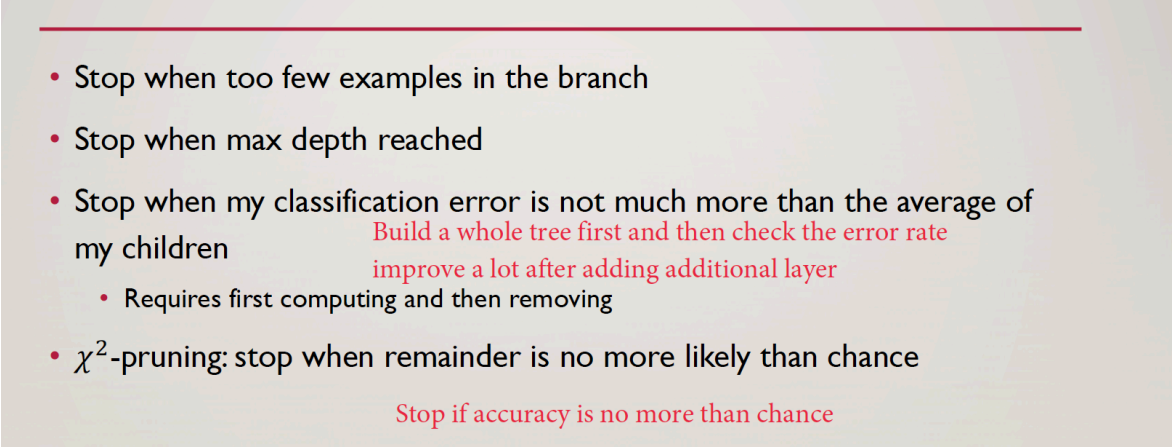
    - Terminating when:
        - All data has same label
        - Unseen example or feature value
        - No further splits possible (run out of splitting feature or feature value for data are the same, but somehow label are not)

- IG and Entropy

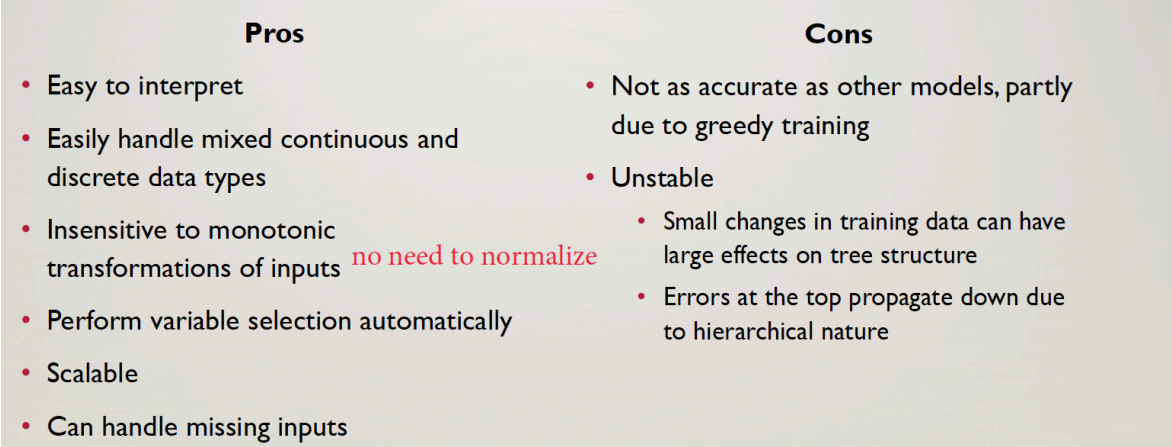


- Overfitting and Underfitting
    - Overfitting: complete tree remembers every data
    - Underfitting: one level tree returns majority label

- Tree Pruning



- Summary



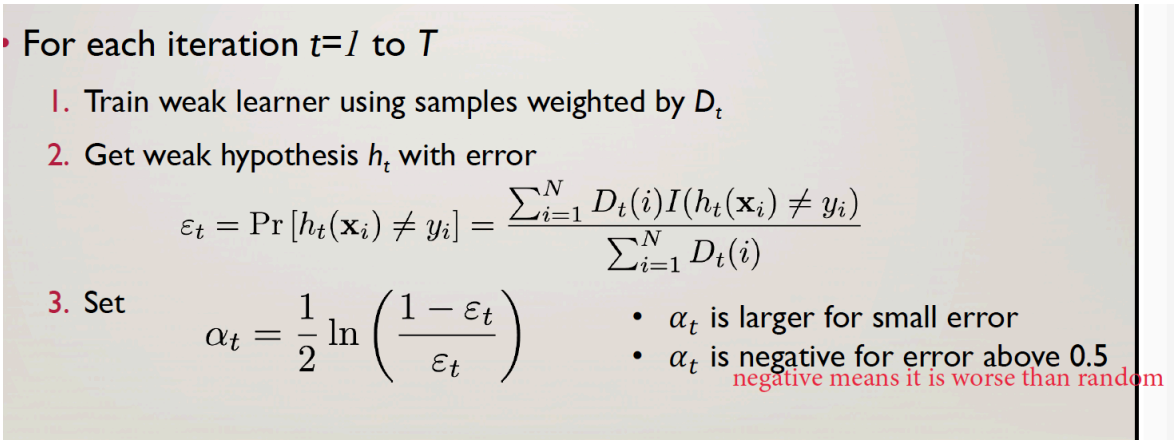
Ensemble Method

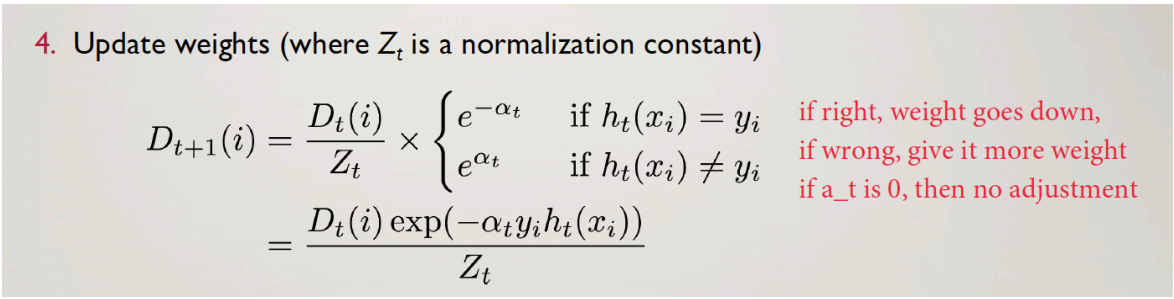
Random Forrest

- Data Sampling: With dataset of size N, create M different samples of size N by resampling wit replacement
  - Bootstrapping and Bagging
- Feature Sampling: random subset of feature at each node

AdaBoost

- Idea: weighted combined decision for weak learners
- Algorithm:





  - Note:
    1. Loss function is different for different classifier
    2. a_m increases as error rate decreases, giving this classifier more weight in the final product
- Why AdaBoos t is not prune to over-fittting
  1. Implicit L1 regularization through stopping at a finite number of weak learners
  2. AdaBoost continues to maximize the margin, leading to better generalization (this yields the similarities with SVMs)
- AdaBoost is able to find better margin even though it training error goes down fast.
  - Other practical Advantages
    - Not prune to over-fitting

    - Fast
    - Easy to implement
    - No parameter tunning
    - Not specific to any weak learner
    - Well-motivated by learning theory
    - Can identify outliers

Diversity and Weighted Experts

- Idea: Diversity in classifiers, datasets
    - Data diversity: feature/instance bagging

Neural Net

- Two layer NN classification
    - $y_k(x, w) = \sigma(\sum_j^M w_{kj}[h(\sum_i^D w_{ji}x_i) + w_{j0}] + w_{k0})$

      Where D is number of input;

      M is number of hidden layer neurones

      K is number of classes

      σ Is sigmoid function

      h Is a non-linear activation function
- Loss function:
    - Cross Entropy for classification: Cross entropy is the number of bits we'll need if we encode symbols from yy using the *wrong* tool ŷ

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

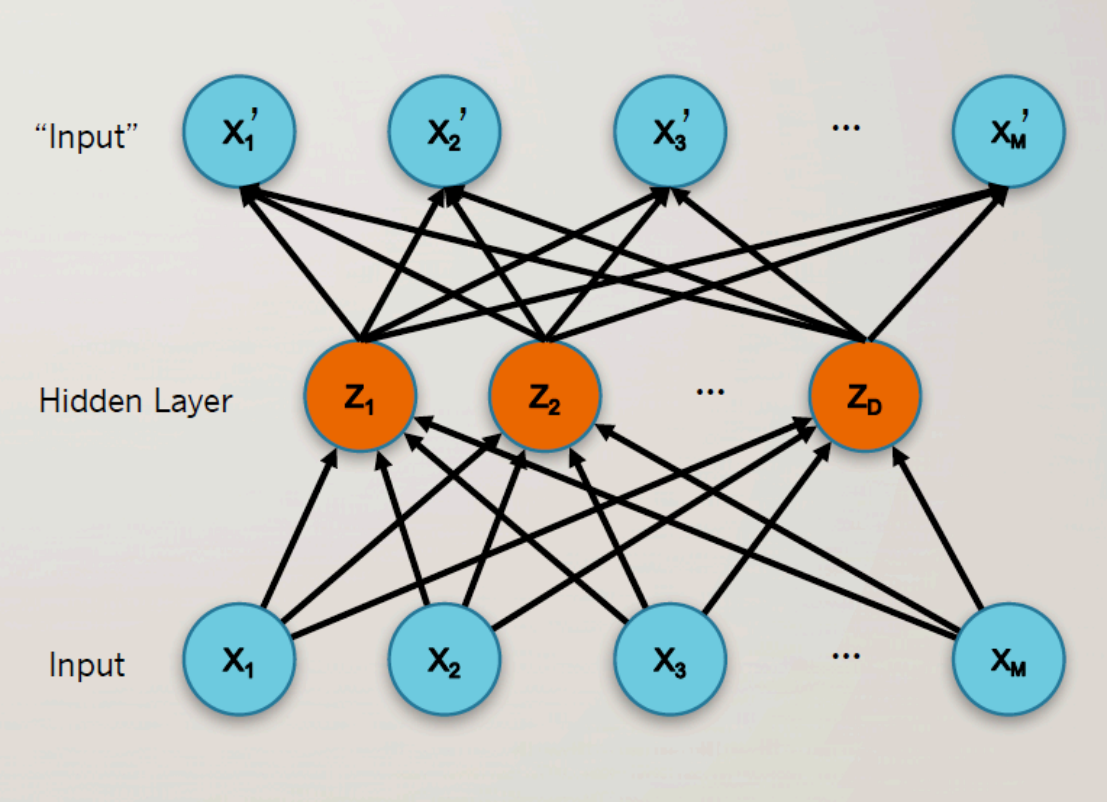
        - Sidenotes: KL divergence is just the difference between cross and entropy
    - OLS
- Optimisation:
    - Gradient descent
    - Back propagation: Chain rule of derivation through computation graph.

Problems with NN:

1. Required Large Labelled Data otherwise easy to overfit.
2. Non-convex objective -> easy stuck in local optima
3. "Vanishing Gradient": as gradient propagate back, gradient become even smaller

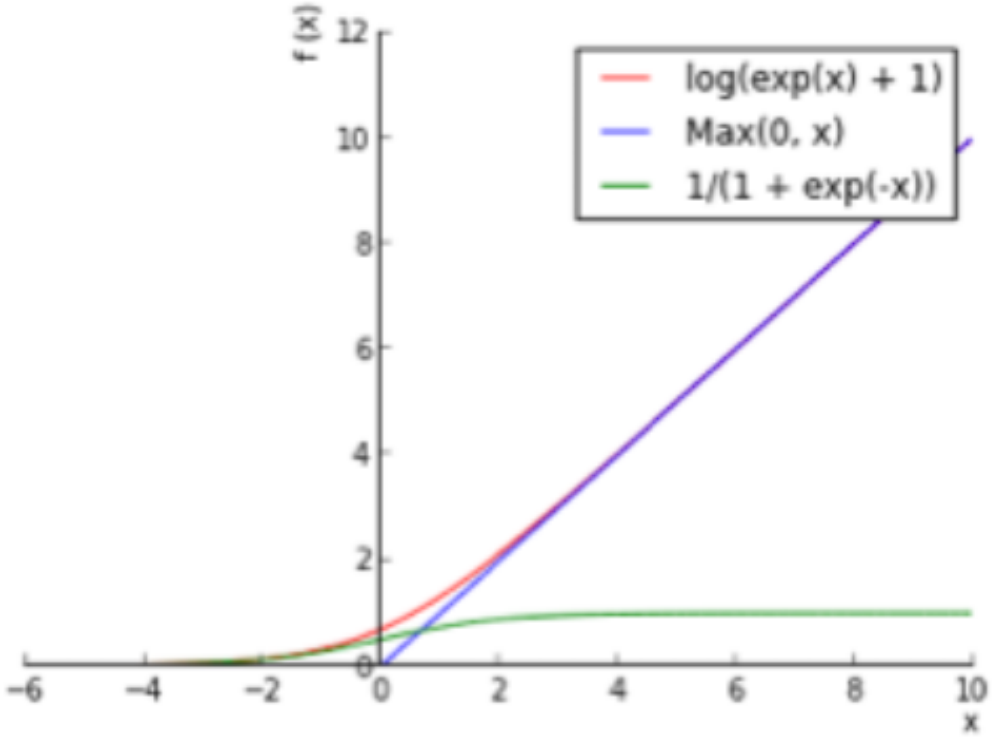
How To Solve?

- Idea1: Back-propragate + SGD -> not always work
- Idea 2: Supervised Pre-training
  - Iteratively add hidden layer. Train current hidden layer with previous hidden layer fixed
  - Then train once together
- Idea 3: Unsupervised Pre-training via Auto-Encoder
  - Auto-encoder: learn input structure by training to encoding input.
  - Better than idea 2 because it is trying to perform a different task: learning the structure in the data which can resulting in a better starting point



Non-linear Activation Functions

- Sigmoid: $\frac{1}{1+e^{-u}}$ [0, 1]
- Tanh: [-1, 1]
- ReLU: $max(0, w * x + b)$. Cap at zero. Faster, no gradient saturation problem like Sigmoid or Tanh for large values (nearly flat curve at the very left or very right on plot)
  - Variations
    1. Leaky ReLU: if $x > 0, f(x) = x$, else $f(x) = ax(0 < a < 1)$
    2. Smoothed ReLU: $log(exp(x) + 1)$. (Notice $log(exp(x)) = x$))
  -



Techniques to Avoid Overfitting

1. Regularization: L1 or L2 on W
2. Early Stopping: Stop training when held-out accuracy no longer improves
3. **Dropout**: Randomly drop neural and their connections in training process; in testing, use full networks but with weights scaled by the probability that they were present during training.

   But why Dropout works?

   Answer:

   1. Dropout is similar to making random perturbations in the input -> reduce variance
   2. Prevents neuron co-adaptation. Neurons take values rely on other neurons for correct prediction
   3. Works like ensemble training.

Other Topics

Transfer Learning

Pretrain ConvNet with a large dataset and then use it as one of following 3 ways:

1. **ConvNet as fixed feature extractor**. Remove last layer fo ConvNet and use the output of the last hidden layer(firstly pass through ReLu function if in original ConvNet there is ReLU). Then use the output as features to feed SVM or SoftMax or other ML models

2. **Fine-Tune the on new dataset**. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features .
3. **Pretrained models**. Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning.

More to look at http://cs231n.github.io/transfer-learning/

Time Serie

Generalised Additive Model(GAM)

The principle behind GAMs is similar to that of regression, except that instead of summing effects of *individual predictors*, GAMs are a sum of *smooth functions*. Functions allow us to model more complex patterns, and they can be averaged to obtain smoothed curves that are more generalizable.

Why GAM?

- Interpretability
- Non-parametric, avoids the pitfalls of dealing higher order polynomial terms in linear models

Project

- FB Prophet Package
- Tesla vs. General Motor(GM)

Data Science Stack

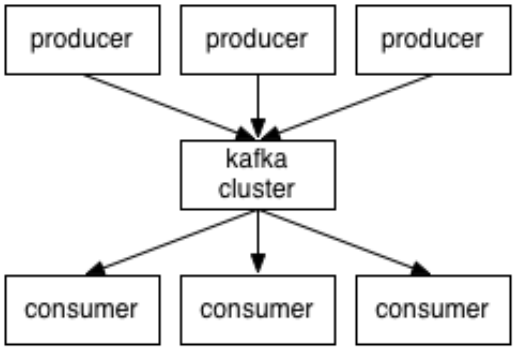
!

Kafka

Key takeaways:

1. Kafka maintains feeds of messages in categories called topics.
2. We'll call processes that publish messages to a Kafka topic producers.
3. We'll call processes that subscribe to topics and process the feed of published messages

consumers..
4. Kafka is run as a cluster comprised of one or more servers each of which is called a broker.



Communication between the clients and the servers is done with a simple, high-performance, language agnostic TCP protocol.

Use Cases:

1. **Messaging**: *Kafka works well as a replacement for a more traditional message broker*. In this domain Kafka is comparable to traditional messaging systems such as ActiveMQ or RabbitMQ
2. **Website Activity Tracking:** The original use case for Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds
3. **Metrics**: Kafka is often used for operational monitoring data, which involves aggregating statistics from distributed applications to produce centralized feeds of operational data
4. **Log Aggregation**
5. **Stream Processing**
6. **Event sourcing** is a style of application design where state changes are logged as a time-ordered sequence of records.
7. **Commit Log**: *Kafka can serve as a kind of external commit-log* for a distributed system. The log helps replicate data between nodes and acts as a re-syncing mechanism for failed nodes to restore their data

Churn User Analyse

Process

1. Understand the problem
    - **Who should be considered as churn user**
    - Is ML the most efficient solution? Would simple heuristic would do like a rule-based system?
    - How interpretable should the system be?
    - How the system is going to be used in future?
    - Scalability?
2. Understand data we get
    - Visualize data to get a quick sense

    - Understand statistical property of data
    - Handle missing data
    - Handle outliers
3. Choose model
    - Interpretability?
    - Time/Space Constrain?
    - Data assumption met for this model?
    - Existing researches/solutions
4. Developing
    - **Baseline system**
    - Feature Engineering
    - Training and Tuning model
5. Deploy
    - Evaluate and improve
    - **Interpret model and build insight to make suggestions to relevant stake-holders**

Clustering

Two parameter overall:

μ_k : cluster center. Defined by mean of examples in that cluster

r_{nk} : whether nth data is in kth cluster

K-Means

1. Select r that minimizes J with fixed μ
2. Select μ that minimizes J with fixed r

Problems:

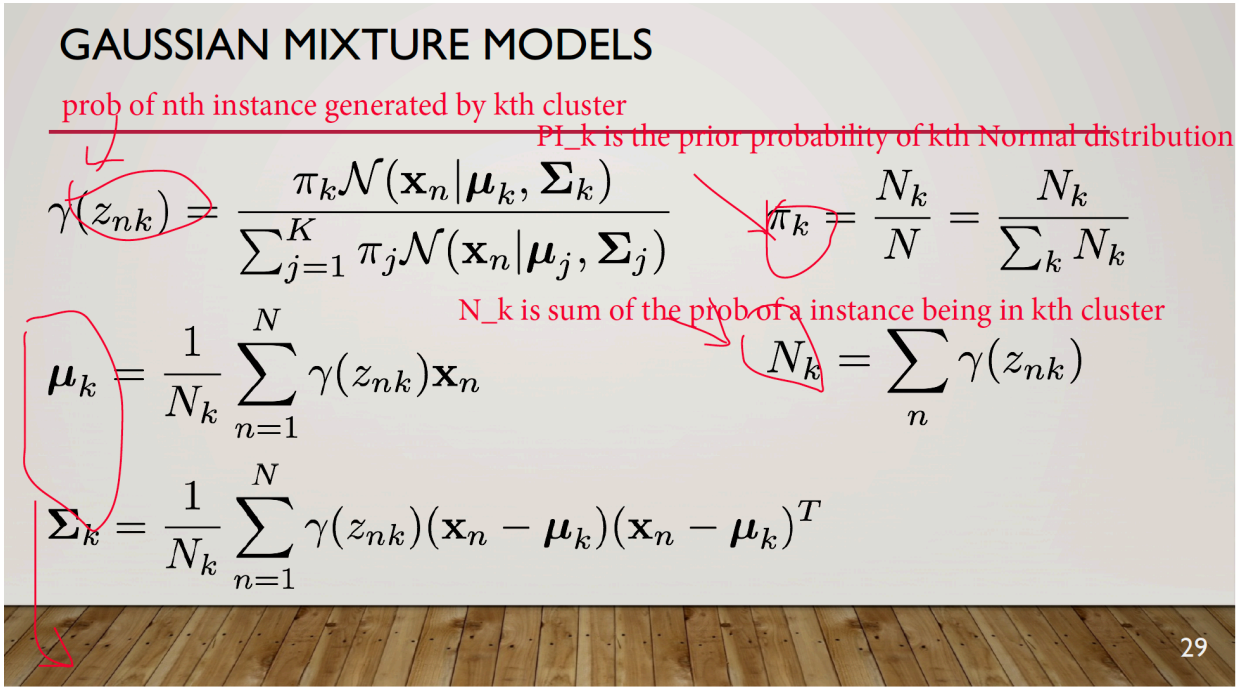
- Slow: O(TKNM), where T is number of iteration, K is number of clusters, N is number of data, M is vector dimension(due to calculate distance of data to μ)
- Hard decision boundary -> Gaussian Mixture Model

GMM

Assume each example is generated by a mixture of Gaussian distributions.

We still have two set of parameters:

1. π_k = prior of this Gaussian, μ_k and σ_k (mean and covariance of Gaussian)
2. r_{nk}

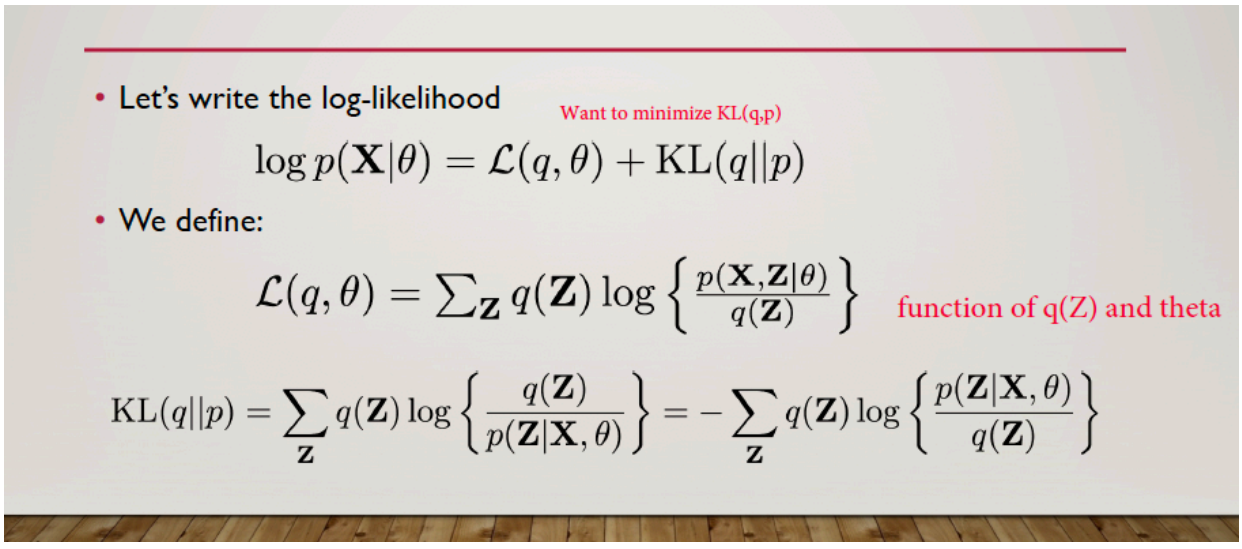


new parameter for Gaussian

Problem:

- Slower than K-Mean: more iteration to converge and more expensive in each iteration
- Mode-Collapse: one data is isolated and become its own Gaussian
- Very non-convex likelyhood

EM



Note:

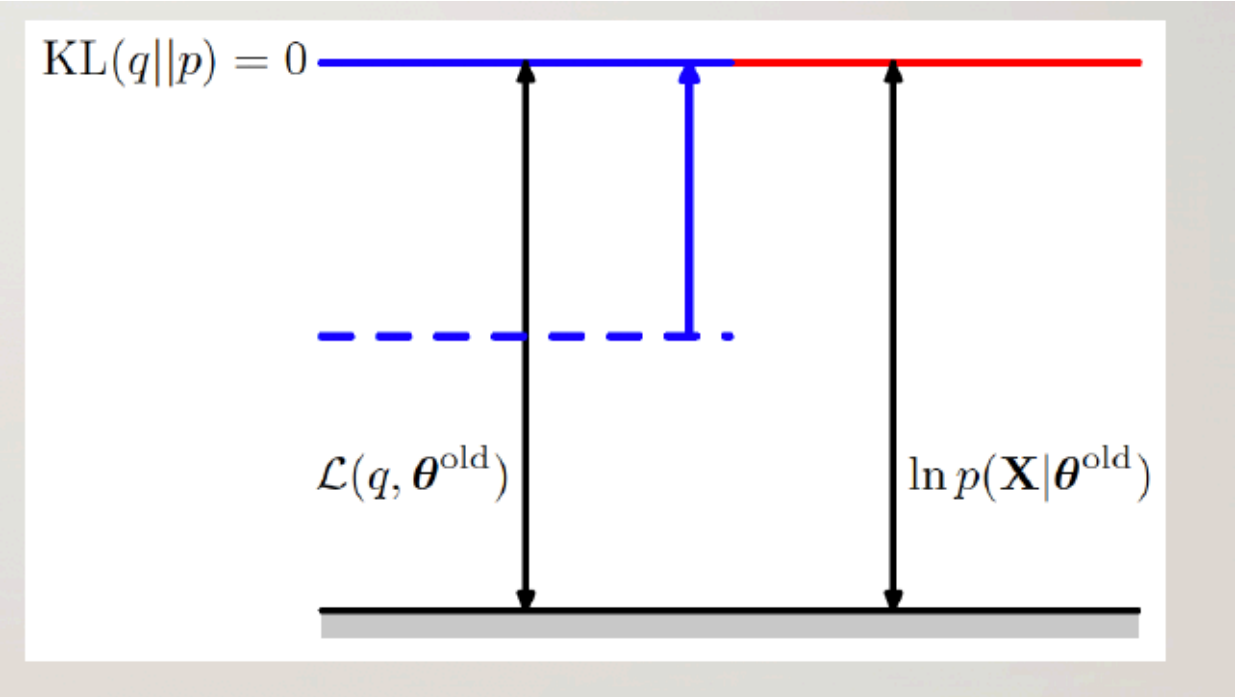
- $\mathcal{L}(q,\theta)$ Is a function of q and θ.
- $KL(q||p)$ Measures the distribution difference in $q(Z)$ and $
- This equation is able to be proved

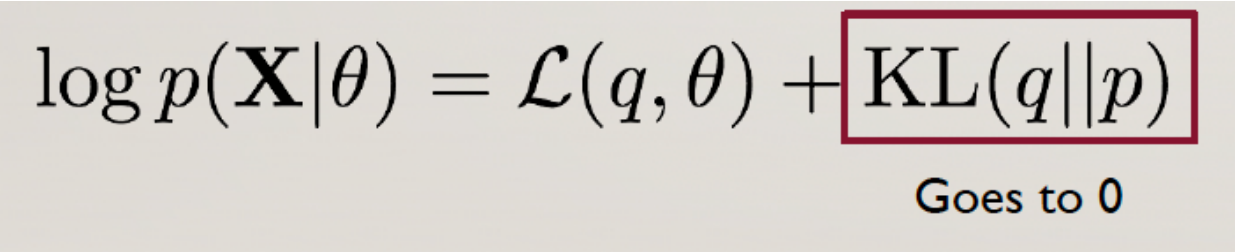
E-step

In E step, we fix θ, and let it be θ^{old} .

We will try to maximise the lower bound of $\log p(X|\theta)$, which is $\mathcal{L}$.

Since likelihood of $p(X|\theta)$ is only depend on θ, it is a constant in this step. Keeping $\log p(X|\theta)$ Constant, if we minimise $KL(q||p)$ by changing $q(Z)$, $\mathcal{L}(q,\theta)$ has to increase. Therefore, the likelihood of our approximate distribution $q(Z)$ will increase when $KL(q||p)$ is minimised when $q(Z) = p(Z|X,\theta^{old})$.

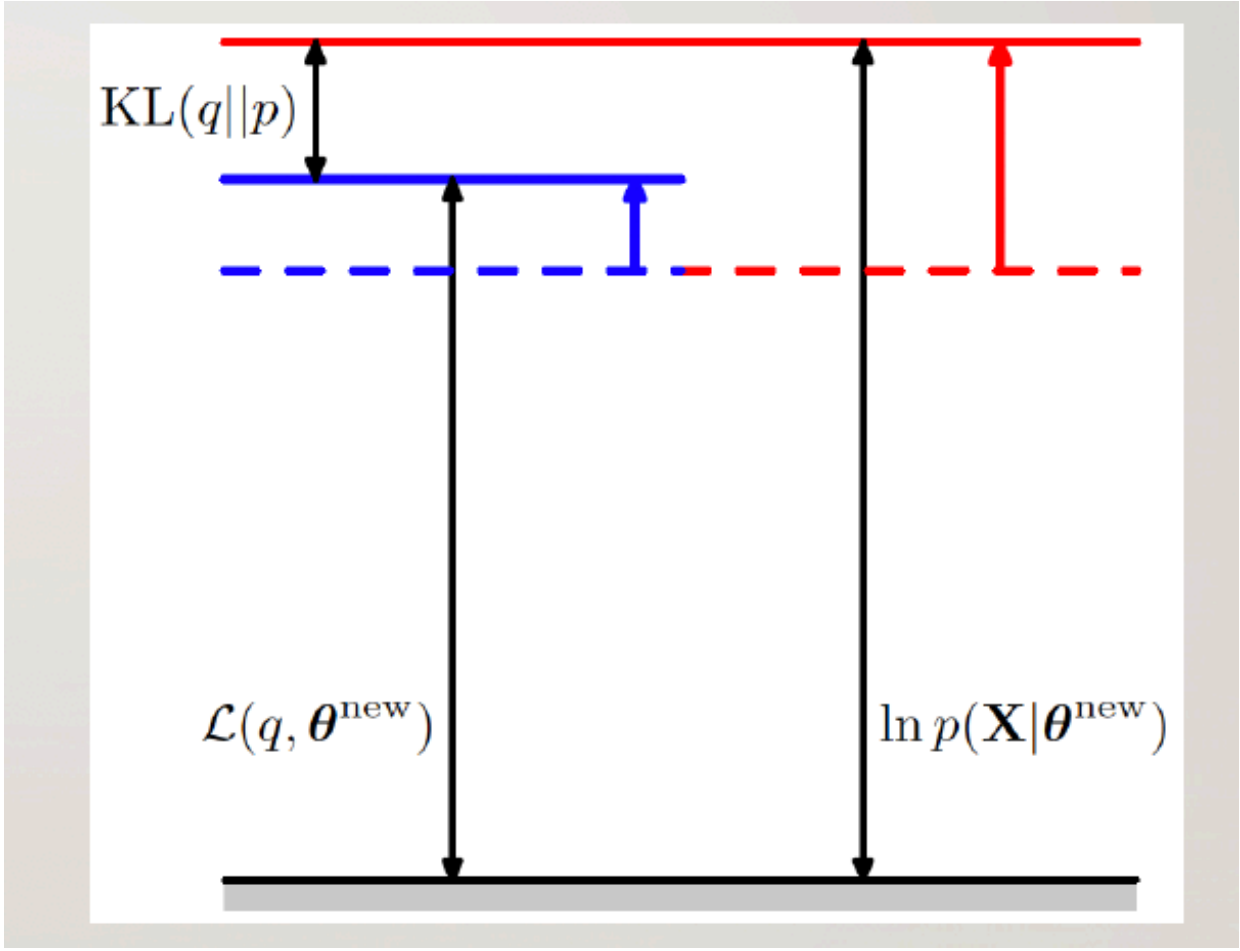




M-step

In M step, we fix $q(Z)$.

We will still try to maximise the lower bound of $\log p(X|\theta)$, which is $\mathcal{L}$. But this time, $\log p(X|\theta)$ is not a constant.



$$\log p(\mathbf{X}|\theta) = \boxed{\mathcal{L}(q,\theta)} + \boxed{\mathrm{KL}(q||p)}$$

Increases — Can only increase

M-step's Q-function

Background: Entropy and Cross-Entropy

Entropy: $H(p) = -\sum_i p(x_i)\log(p(x_i))$

Cross-entropy: $H(p) = -\sum_i p(x_i)\log(q(x_i))$

Cross-entropy measures the number of bits (or nats) is needed to identify an event from a set of possible events if the coding scheme is based on distribution q instead of the true one, p

So after E step, we made $q(Z) = p(Z|X,\theta)$, substitute this in $\mathcal{L}$, we get

$$\mathcal{L}(q,\theta) = \sum_{\boldsymbol{Z}} p\left(\boldsymbol{Z}|\boldsymbol{X},\theta^{old}\right)\log(p(\boldsymbol{X},\boldsymbol{Z}|\theta)) - \sum_{\boldsymbol{Z}} p\left(\boldsymbol{Z}|\boldsymbol{X},\theta^{old}\right)\log\left(p\left(\boldsymbol{Z}|\boldsymbol{X},\theta^{old}\right)\right)$$

Whereby first item is negative cross entropy between $p(X,Z|\theta)$ and $p(Z|X,\theta^{old})$

Second term is constant

- Rewrite the function as

$$\mathcal{L}(q,\theta) = \mathcal{Q}\left(\theta,\theta^{old}\right) + \text{constant}$$

where

$$\mathcal{Q}\left(\theta,\theta^{old}\right) = \sum_{\boldsymbol{Z}} p\left(\boldsymbol{Z}|\boldsymbol{X},\theta^{old}\right)\log(p(\boldsymbol{X},\boldsymbol{Z}|\theta))$$

Q is the expectation of the complete-data log-likelihood evaluated for some θ given our current belief θ^{old}

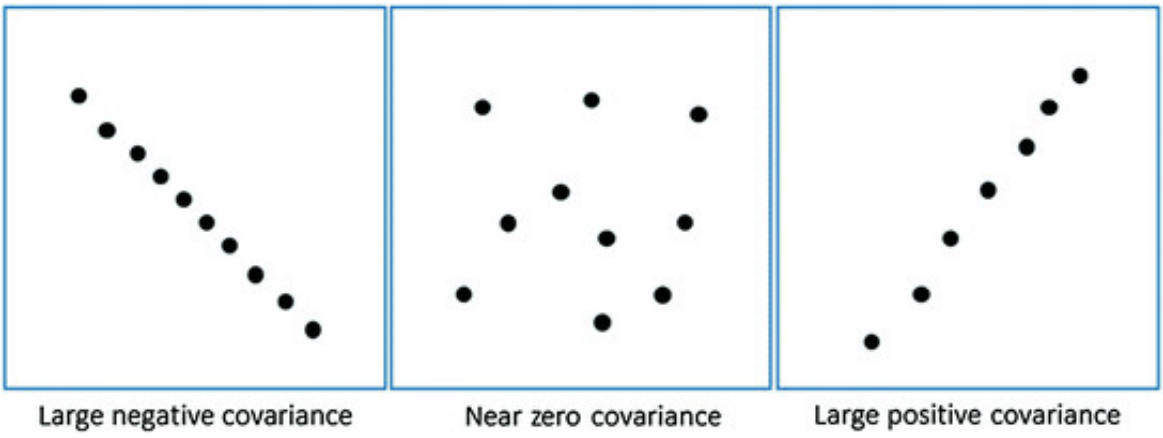
Dimension Reduction

Background

Covariance

It is a measure of the extent to which corresponding elements from two sets of ordered data move in the same direction. Formula is shown above denoted by *cov(x,y)* as the covariance of *x* and *y*.

$$cov(x,y) = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{N}$$



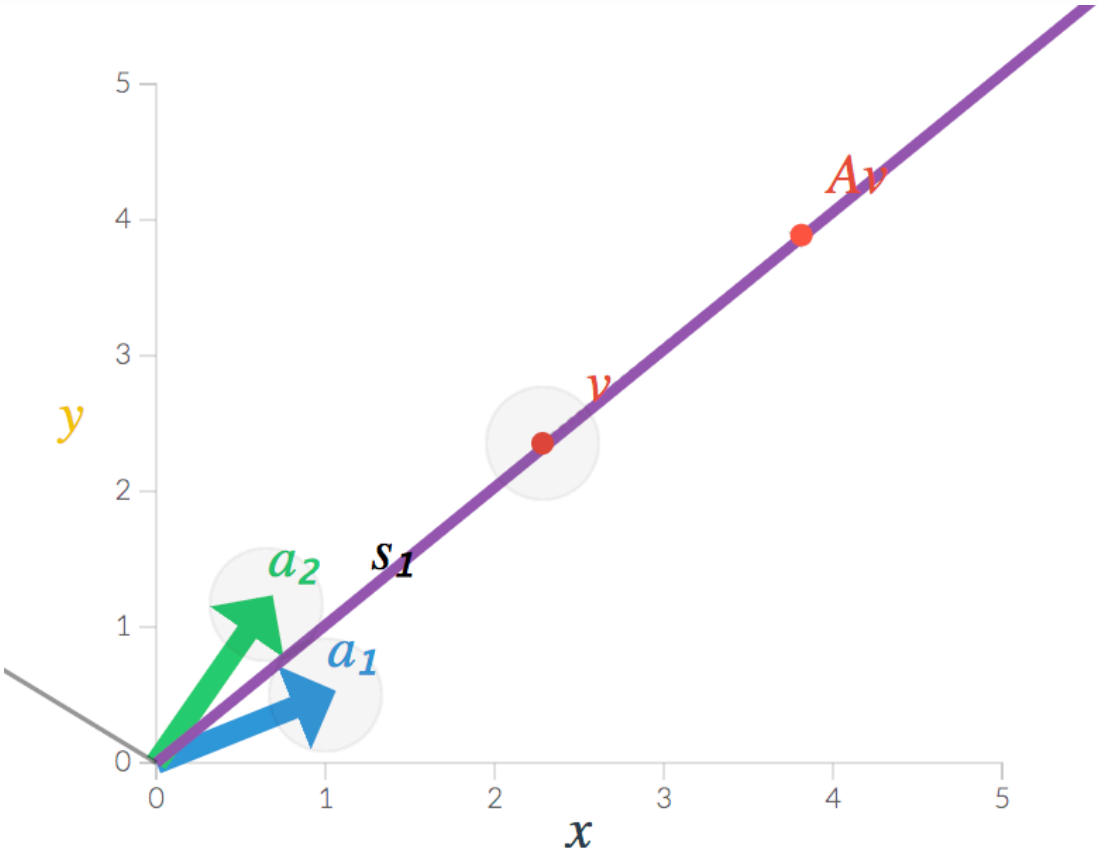
Eigen Value and Eigen Vector

http://setosa.io/ev/eigenvectors-and-eigenvalues/

Assume A is matrix and v is a vector. Av Is essentially transform v by applying transformation A.

Then if $Av = \lambda v$ for some scalar value λ, it means transformation of v by A is only changing magnitude of v (stretching or shrinking v while maintaining its direction).

Given a fix A, we can find the its eigen vectors (by selecting different λ) .



PCA

Intuitions

- Firstly, What is the goal of PCA ?

  find some low independent dimension of projected data, such that the error from reconstruction of the original data error by projected data is minimised.

- Then How to minimise reconstruction error?

  Reconstruction error is minimised when variance of projected data on each dimension is maximised. Therefore, we want to transform the original data points such that the covariance matrix of transformed data points is a diagonal matrix (zero covariance and high variance).

- What is Covariance Matrix? And why it should be diagonal

  We want the data to be spread out i.e. it should have high variance along dimensions. Also we want to remove correlated dimensions i.e. covariance among the dimensions should be zero (they should be linearly independent). Therefore, our covariance matrix should have -

  - large numbers as the main diagonal elements.
  - zero values as the off diagonal elements.

  $$[Covariance \quad matrix] \cdot [Eigenvector] = [eigenvalue] \cdot [Eigenvector]$$

$$\begin{bmatrix} V_a & C_{a,b} & C_{a,c} & C_{a,d} & C_{a,e} \\ C_{a,b} & V_b & C_{b,c} & C_{b,d} & C_{b,e} \\ C_{a,c} & C_{b,c} & V_c & C_{c,d} & C_{c,e} \\ C_{a,d} & C_{b,d} & C_{c,d} & V_d & C_{d,e} \\ C_{a,e} & C_{b,e} & C_{c,e} & C_{d,e} & V_e \end{bmatrix}$$

- Finally, how should we ensure that projected data's covariance matrix is diagonal?

  If we find the matrix of eigen vectors of C_x and use that as P (P is used for transforming X to Y, see the image below) , then C_y (covariance of transformed points) will be a diagonal matrix. Hence Y will be the set of new/transformed data points.

$$\begin{aligned} &C_x = covariance\ matrix\ of\ original\ data\ set\ X \\ &C_y = covariance\ matrix\ of\ transformed\ data\ set\ Y \\ &such\ that, \\ &Y = PX \\ &For\ simplicity,\ we\ discard\ the\ mean\ term\ and\ assume\ the \\ &data\ to\ be\ centered.\ i.e.\ X = (X - \bar{X}) \\ &So,\ C_x = \frac{1}{n}XX^T \\ &C_y = \frac{1}{n}YY^T \\ &= \frac{1}{n}(PX)(PX)^T \\ &= \frac{1}{n}PXX^TP^T \\ &= P(\frac{1}{n}XX^T)P^T \\ &= PC_xP^T \end{aligned}$$

Process

1. Calculate the covariance matrix *X* of data points.
2. Calculate eigen vectors and corresponding eigen values.
3. Sort the eigen vectors according to their eigen values in decreasing order.
4. Choose first k eigen vectors and that will be the new k dimensions.
5. Transform the original n dimensional data points into k dimensions.