**k-Nearest Neighbor Classifier(Instance based learning)**

* **특징**
* 주어진 test sample xt에 대해서, 가장 가까운 k개의 neighbor training samples [(xn1, yn1), …, (xnk, ynk)]을 찾아서 다수를 차지하는 class의 label을 할당.
* 언제 사용할 지
* Vector형태의 Data
* Attribute의 수가 20보다 작은 경우(차원이 커질 수록 거리 계산에 computational cost가 늘어나므로)
* training data의 양이 충분한 경우
* 장점
* Training이 굉장히 빠름(training data를 가지고 있다가 test 시점에 거리 계산만 해주면 되므로)
* 복잡한 형태의 boundary(target function) 생성 가능
* information loss가 없음
* 단점
* query time이 느림
* 관련 없는 attribute나 outlier에 강하지 않음(일반적으로 smoothing을 통해 outlier의 effect를 감소시키는데 k-NN에서는 smoothing과 같은 information loss가 없으므로)
* Voronoi Diagram
* k- nearest neighbor classifier는 모든 training data를 저장하고 있으므로 storage에 대한 요구량이 높다
* 따라서 Voronoi Diagram을 통해 performance에 영향을 주지 않고 storage 요구량을 줄일 수 있음
* Class 사이의 Decision Boundary만 따로 그려 저장
* 단 training 시간은 증가(training때 Decision Boundary 그려야 하므로), storage 사용 감소, test time은 감소
* Distance Metric
* Euclidian distance=
* 특정 dimension에 더 비중을 더 줄 수 있음(특정 attributes가 더 영향을 많이 주도록 할 수 있음)
* Standardization
* 한 쪽 feature(attribute, dimension)에 영향을 많이 받지 않도록 하기 위해서 정규화 해주는 방법
* raw feature value를 z-score로 변환
* = i번 째 sample의 j번 feature
* = 의 평균(j번 feature에 대한 평균)
* = 의 표준편차(j번 feature에 대한 표준편차)
* Efficient Searching
* High dimension인 경우 검색을 위해서 training data를 미리 구조화를 시켜놓는 방법이 가능(KD Tree)
* Choose Dimension
* Choose pivot(median)
* Split data, repeat
* 위의 과정을 통해 training data의 partition을 나누면, test data가 들어왔을 때 해당 partition에서만 search를 하면 됨
* 주의할 점은 partition의 경계에서 test sample이 나타나게 되면 다른 partition에서 nearest neighbor가 나타난다는 문제
* Choosing k(k를 어떻게 고를 것인지, 일반적으로 홀수)
* K값과 어떤 Distance Metric을 사용하느냐에 따라서 성능이 달라짐
* N fold cross validation을 이용하여 cross validation error(average error)를 최소화 하는 k를 고름
* 여러 k값에 대해서 validation해봄
* training sample의 수가 무한대로 가고 k가 적절하게 큰 값을 가짐에 따라 k-NN분류기는 Bayes Classifier(주어진 data로 가장 좋은 성능 얻을 수 있음)만큼 좋은 성능을 보여줌

Clustering

* 특징
* Unsupervised learning
* Data는 필요하지만 Label은 필요하지 않음(Label이 없는 Data 처리), Cluster의 수도 알려지지 않음
* Pattern을 발견
* 찾고 있는 내용을 모를 때 유용
* 어떤 Similarity를 사용하는지에 따라 Clustering 결과가 달라짐.
* Idea
* similar instance를 grouping
* 어떤 기준으로 Similar를 판단할지(옵션1: small Euclidean distance)
* Clustering 결과는 Clustering할 point간의 어떤 Similarity를 사용하는지에 결정적으로 의존
* 종류
* Partition Algorithms
* K-means
* Mixture of Gaussian
* Spectral Clustering
* Hierarchical Algorithms
* Bottom up: Agglomerative Clustering
* Top down: Divisive Clustering
* K-means Clustering
* Iterative Clustering Algorithm(K random point를 잘 골라야함)

1. 처음에 K개의 point를 random하게 고름(cluster의 center로써)
2. data point들을 가까운 cluster center의 cluster로 할당
3. 각 cluster에 할당된 point들의 평균으로 cluster center를 변경
4. data point들의 cluster로의 할당이 더 이상 바뀌지 않을 때까지 2,3을 반복

* 특징
* 유한한 반복 횟수를 통해 수렴하도록 보장
* 각 iteration마다 걸리는 시간

1. O(KN) = data point들을 가까운 cluster center로 할당
2. O(N) = 각 cluster에 할당된 point들의 평균으로 cluster center를 변경

* K-means는 번갈아 최적화를 수행하며, 각 단계는 Objective를 감소시키므로 수렴이 보장됨
* K-means algorithm은 initial mean을 찾아주어야 하는 직관적인 판단이 필요
  1. 따라서 initial mean에 따라 Clustering 결과가 다를 수 있다.
  2. global optimum이 아닌 local optimum에 수렴할 수 있다.
  3. 이를 해결하기 위해 variance-based split/merge, initialization heuristics
* Agglomerative Clustering
* Bottom-up 방식

1. 처음에 각 input data point 사이의 distance matrix 계산
2. 각 data point들을 개별 cluster로 시작
3. 가장 가까운 Cluster 2개를 병합하여 새로운 Cluster로 만듦
4. distance matrix를 업데이트
5. 하나의 Cluster가 남을 때까지 3, 4을 반복

* 작은 Cluster에서 더 큰 Cluster를 점진적으로 구축
* 하나의 Cluster가 아니라 Dendrogram으로 표현되는 Cluster Family를 생성
* Dendrogram에서 각 Clustering 단계 사이의 공백이 큰 곳에서 잘라주는 것이 좋음
* 두 Cluster 사이의 distance를 계산하는 것이 Key Operation
* 가장 가까운 Cluster를 고르는 방법
* Closest pair(Single-link Clustering)
  + Ci와 Cj안의 가장 가까운 Sample들 간의 거리로 Distance Matrix Update
  + 타원형이 아닌 모양 처리 가능(장점)
  + noise 및 outlier에 강력하지 않음(단점)
  + 길고 길쭉한 cluster 생성(단점)
* Farthest pair(Complete-link Clustering)
  + Ci와 Cj안의 가장 먼 Sample들 간의 거리로 Distance Matrix Update
  + 동일한 직경의 균형 잡힌 Cluster(장점)
  + noise에 덜 민감(장점)
  + 큰 Cluster를 쪼개려고 하는 성향이 있음(단점)
  + 모든 Cluster가 유사한 크기를 가지게 하려는 성향이 있음[작은 Cluster가 큰 Cluster에 병합됨](단점)
* Average of all pairs
  + Ci와 Cj안의 모든 Sample들 간의 거리의 평균으로 Distance Matrix Update
  + Noise와 Outlier에 가장 강함(장점)
  + 원형 모양의 Cluster를 가지려는 경향을 보임(단점)
* 위의 방법 중 어떤 방법을 사용하냐에 따라 결과 달라짐
* Divisive Hierarchical Clustering
* Top-down 방식
  1. 모든 data point로 구성된 단일 Cluster로 시작
  2. Cluster를 유사하지 않은 두 개의 Cluster로 분할

< 각 Cluster의 SSE를 계산하고 SSE가 Largest Value를 가지는 Cluster를 골라 두 개의 분할()

* 1. 원하는 수의 Cluster를 얻을 때까지 2를 반복적으로 진행
* Monothetic Divisive Method
  + 한 번에 하나의 변수/차원을 사용하여 군집을 분할
* Polythetic Divisive Method
  + 모든 변수를 함께 기반으로 하여 군집을 분할
* 모든 Inter Cluster Distance 계산 방법을 사용할 수 있음
* 계산 집약적이며 Agglomerative Method보다 덜 사용됨
* Hierarchical Clustering의 특징
* Cluster의 개수를 처음에 임의로 가정하지 않아도 됨
* 적절한 수준에서 Dendrogram을 절단하여 바람직한 수의 Cluster를 얻을 수 있음
* 계층적 Clustering은 의미 있는 분류법에 해당
* 병합/분할할 Cluster를 결정하는데 Distance Matrix 사용
* 모든 Data Point간에 Distance를 계산해야 하므로 Quadratic
* Large Dataset에 적합하지 않음
* K-Means Clustering
* Two steps in Alternate
* Assignment Step
  + 각 data point를 가장 가까운 Cluster로 할당
* Refitting Step
  + 각 Cluster의 Center를 Cluster로 할당된 data point들의 무게중심(평균)으로 이동
* 목적함수
* Objective
* Alternate
  + 를 고정하고 를 Optimize

* + 를 고정하고 를 Optimize

* 목적함수
* rnk=> xn은 특정 cluster k에 대해서만 1이 되고 나머지에서는 0이 됨
* Objective: Minimize J
* Alternate(EM Algorithm)
  + Expectation : 를 고정하고 를 minimize하는 구함
  + Maximization: 를 고정하고 를 minimize하는 구함
* 사용 이유
* Assignment가 변경될 때(E step)마다 할당된 Cluster Center에서 Data point까지의 제곱 거리의 합이 감소(Object function J값 감소시킴)
* Cluster Center가 변경될 때(M step)마다 현재 할당된 Cluster Center로부터 Data point까지의 제곱 거리의 합이 감소(Object function J값 감소시킴)
* Local Minima
* K-means가 local minima에 고정되는 것을 막을 수 있는 방법은 없다.
  + Starting center point들을 여러 k값으로 random하게 하여 여러 번 clustering을 시도해보고 그 중 가장 최적의 Solution을 채택하는 방법을 시도할 수 있음
  + 근처에 있는 두 개의 Cluster를 병합하거나, 큰 Cluster를 두 개로 쪼개는 방법을 시도할 수 있음,
* Global minima(maximum) 또는 Local minima(Maximum)으로 수렴
* Soft k-means
* 해당 지점이 양 쪽의 Cluster로부터 동시에 영향 받는 경우 Clustering의 성능 향상
* 하나의 Cluster는 특정 data point에 p1의 responsibility를 가지고, 다른 Cluster는 그 data point에 p2의 responsibility를 가짐
* M step 에서 Cluster가 data에 대해 더 많은 정보를 사용할 수 있게 해줌
* Generative View of Clustering
* Clustering 결과가 잘 되었는지 판단할 때
* Clustering 결과로 얻게 된 Generative Model()로부터 Data 생성
  + Data가 Generative Model에 의해 생성되었다고 상상
* 얼마나 Data를 관찰된 Data와 유사하게 생성하냐에 따라 Clustering 결과의 품질 판단 가능
* Mixture of Gaussians Generative Model
* 먼저 “Mixing Proportion”이라는 확률로 k Gaussian중 하나를 선택
* 선택된 Gaussian으로부터 Random point를 생성
* 관찰한 정확한 Data를 생성할 확률은 0이지만 Parameter()를 조정하여 확률 밀도를 최대화(관찰된 Data와 유사한 Data를 생성할 수 있도록) 할 수 있음
* Gaussian Mixture Model
* Proportion, 평균, 분산을 변수로 잡아서 최적화 시킴
* E step
  + 각 Gaussian이 각 data point를 생성할 확률을 계산(proportion 조정)
* M step
  + 각 Gaussian의 parameter(평균/분산)를 적절히 조정하여 현재 담당하는 Data를 생성할 확률을 최대화
* Spectral Clustering
* 덩어리 형태가 아닌 Data는 k-means로 구해내기 어려운데 이러한 Data는 Spectral Clustering을 통해 구할 수 있음
* point들을(data를) graph의 link를 통해 grouping
* data를 graph로 간주하고 graph에서 약한 고리를 끊어 냄으로써 data clustering
* Graph를 생성하는 방법
* A fully connected Graph
* K-nearest neighbor graph: 각 노드는 K-nearest neighbors와 connected
* Object사이의 유사도 계산을 위해 일반적으로 Gaussian Kernel을 이용
* Distance가 클수록 Similarity 작아짐
* 절차

1. Similarity(affinity) matrix 계산,
2. Degree Matrix 계산,
3. Laplacian Matrix 계산 L=D-W
4. Laplacian Matrix의 Eigen Vector 계산
5. 작은 Eigenvalue에 해당하는 Eigen Vector값 이용해서 Data Clustering
   * Graph의 연결고리가 약한 부분을 끊어내야 하므로 관계의 밀도가 낮은(작은 Eigen Value에 대응하는) Eigen Vector 이용

* 특징
* Data가 Eigen Vector에 의해 reordering됨
* 저차원의 Data를 고차원으로 Mapping
* N개의 data가 각각이 x개의 벡터의 원소 값으로 표현되어 분류

Dimensionality Reduction

* 특징
* 고차원의 Raw Data는 다루기 어렵기 때문에 Data를 저차원으로 만들어 다루기 쉽게 만듦
* Feature Extraction: Noise, 의미가 없는 부분을 제거하여 중요한 부분(Feature Vector: 밝기, 회전 등의 변화에 강인)만을 취하도록 함
* d dimension의 Data를 정보의 손실을 최소화하도록 하며 k(<d) dimension의 Data Point로 바꿈
* Feature Extraction/Dimensionality Reduction
* d차원의 좌표축을 k차원의 좌표축으로 바꿈
* 다 변수 data set을 해석하기 쉬운 new configuration으로 회전
* 목적
  + data를 simple하게 만들기 위해
  + 변수들 사이의 관계를 살피기 위해
  + data에 존재하는 pattern을 살피기 위해
* 순서

1. Recombination(Rotation) : d -> d’ (분산이 큰 방향으로)
2. Top K개 선택: d’ -> k

* PCA(Principal Component Analysis)
* 목적
* 분산을 가장 잘 보존하는 K 차원 projection을 찾는 것
* 순서

1. original data의 공분산 행렬 와 mean vector 계산
2. 공분산 행렬의 eigenvector와 eigenvalue 계산
   * eigenvector는 0벡터가 될 수 없음
   * eigenvalue는 0이 될 수 있음
3. Top K eigen vectors 선택
   * eigenvalue의 내림차순에 따라 정렬
4. 해당 eigen vector들에 의해 확장된 subspace에 data point들 투영

* 특징
* 공분산행렬의 eigenvector는 새로운 좌표 시스템을 정의
  + eigenvalue가 가장 큰 eigen vector는 데이터들간의 가장 큰 variation을 capture
  + eigenvalue가 가장 작은 eigen vector는 가장 작은 variation 가짐
* top few(k) eigenvector들을 이용하여 data를 압축 가능
  + linear subspace 선택에 해당
  + 해당 eigenvector들을 주성분(principal components)라고 함
* Covariance(공분산
* 평균 주변의 점 집합의 “spread” 측정
* 각 차원이 평균에서 서로 얼마나 다른지 측정
* 공분산은 두 차원 사이에서 측정됨
* 공분산은 두 차원 사이에 관계가 있는지 확인()
* 한 차원의 공분산은 분산()
* Positive Covariance: 한 차원이 증가할 때 다른 차원도 증가
* Negative Covariance: 한 차원이 증가할 때, 다른 차원은 감소
* 간의 상관관계가 적음
* 은 데이터 세트의 원래 분산을 가능한 한 많이 설명
* 은 남아있는 분산을 가능한 한 많이 설명
* Eigenface
* Face Image는 extremely 고차원
* 많은 차원 벡터 중 극히 소수만이 valid face image임
* k개의 eigenvalue가 전체의 95%(90%)를 만족하도록 k 선택
* 원본 Face Image를 가지고 weighted sum을 통해 임의의 얼굴을 합성하는 것보다 PCA 분석을 통해 eigenvector들의 weighted sum을 구하면 소수(k개)의 eigen vector로 임의의 얼굴이 합성 가능
* Linear Discriminant Analysis(LDA)
* 사용
* PCA는 SCATTER가 최대가 되는 방향의 eigenvector 찾으므로 data의 class가 섞이게 되어 class information을 고려하지 않음
* 따라서 CLASS가 구별되면서 차원을 축소하는 LDA 사용
* 특징
* LDA는 (project된 between scatter)/(project된 within scatter) 최대화
* 즉, between scatter(class끼리의 분산)는 최대화 하고, within scatter(class내부의 분산)는 최소화
* 공식
* <
* Fisherface
* Class 정보를 이용해서 서로 다른 Class를 잘 분리하고 같은 Class끼리는 잘 응집이 되도록 차원이 축소되므로 더 좋은 성능

Artificial Neural Network(인공신경망)

* 특징
* overfitting이 큰 단점 중에 하나
* Two layer : Input Layer와 Output Layer만 있는 경우
* XOR과 같이 Non Linear한 경우 2 Layer로 표현할 수 없으므로 Hidden layer 필요
* ANN training이란 training data를 이용해서 w값을 결정하는 것
* 구성

1. Input Layer의 Node 숫자 결정

* 실험을 통해 domain knowledge 이용

1. Output Layer의 Node 숫자 결정

* 실험을 통해 domain knowledge 이용

1. Hidden Layer의 Node 숫자 결정

* 실험을 통해 domain knowledge 이용

1. training algorithm을 이용해서 weight 찾아냄.

* Back propagation algorithm 이용(Supervised Learning)
* Back propagation algorithm
* w를 업데이트함에 있어 Output Layer에서 멀어질 수록 더 많은 노드와 derivative가 포함되고 sigmoid의 derivative는 0~0.25의 값을 가지고 이것들이 점점 곱해지므로
* 따라서 Output Layer에서 멀어질 수록(Backpropagate이 되면서) w의 업데이트 량이 점점 줄어듦
* Activation 함수를 미분 가능하도록 Sigmoid를 사용할 수 있으며 Sigmoid함수는 미분 후에도 자기 자신이 식에 이용이 됨(f’(x)=f(x)(1-f(x))
* Deep Neural Network(DNN)
* 정의
* # of hidden layers>=2이면 DNN
* # of hidden layers<=1이면 Shallow Neural Network(간단한 decision boundary 구할 때)
* hidden layer를 많이 쌓을 수록 보다 복잡한 decision boundary 구현 가능하지만 vanishing gradient, overfitting 등의 문제 있음
* DNN Training
* Vanishing gradient problem(non-linear activation’s problem)
  + Output layer에서 멀어질 수록 w의 업데이트되는 양이 점점 적어진다는 문제
  + 완화하기 위해 activation 함수 및 네트워크 구조를 수정하는 작업
  + 완화하기 위해(ReLU, Layer-wise training)
* 많은 labeled Data를 필요로 한다는 문제
  + Data를 수집하는 것은 time consuming 하고 expensive
* Overfitting 문제
  + training data가 충분하지 않거나 학습 방법이 잘못된 경우 training data에만 과도하게 fit될 수 있음
* Local Minima에 빠질 수 있다는 문제
  + 어떤 값을 초기값으로 하냐에 따라 결과가 달라질 수 있음
  + 충분한 training data에서도 local minima에 빠질 수 있음
  + 완화하기 위해 random하게 여러 초기값에서 진행하여 그 중에 가장 적합한 것 선택
  + 데이터와 computation power를 증가시켜 완화

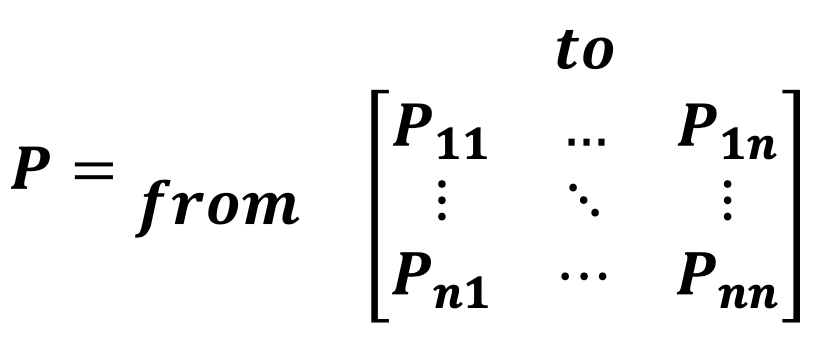
Deep Neural Network

* Deep Learning
* multi level layer를 이용하여 학습
* Input -> Feature Extraction + Classification -> Output
* 필요한 Feature 또한 빠르게 학습하여 Manually Designed Feature보다 쉽게 적용할 수 있으며 좋은 성능을 보임.
* Unsupervised and Supervised
* 많은 양의 training data 활용
* 2012년 이후부터 DNN이 큰 Data Set들에 대해 좋은 성능
* GPU, Big Data, Better Learning Algorithms
* 2006년 이후부터 좋은 알고리즘들이 소개되어 소규모 Data에 대해 좋은 성능
* Fat+Short 보다는 Thin+Tall Network(Deep)가 더 좋은 성능을 보임
* 특징
* Modularization
* Network 구조가 모듈화 되어 작업이 처리되므로 더 좋은 성능
* 특정 Class에 대한 sample이 적은 경우에도 data로부터 자동으로 학습하여 Module화를 통해 잘 학습할 수 있음
* Data가 많아질 수록 점점 더 성능이 좋아짐.
* Mapping 작업과 Feature 추출을 자동으로 수행
* Hand-Crafted에 비해 성능이 좋음
* 기법
* Rectified Linear Unit (ReLU)
* 계산이 빠름(Linear 함수이므로)
* 0 이하의 값을 가지는 Node를 없애도록하여 Layer가 thin한 구조를 가져 학습이 더 빨라짐
* 역전파가 되면서 Update 양이 작아지는 Vanishing Gradient Problem 완화시킴
* Drop Out(Training 시에)
* Gradient를 계산하기 전에 각 Layer의 Node들을 p\*100%의 확률로 drop을 하게 도어 Network를 Thin하게 만듦
* 새로운 Network를 training에 사용(training시에만 사용)
* 어떤 Node들이 drop되면서 이전에는 사용되지 않았던 새로운 feature가 사용될 수 있도록 하여(특정 feature가 다른 Node에 방해받지 않도록 하여) 성능 향상
* 여러가지 다른 버전의 Model을 이용하게 되는 것이므로 Model들의 융합을 통해 최종적으로 좋은 결과를 얻게 된다는 앙상블 개념
* Testing시에 dropout이 안 된 상태로 test해야 하므로 weight은 반드시 (1-p)와 곱해줘야 함.
* Overfitting 문제를 완화시켜 줌
* Convolutional Neural Networks
* Terminology
* Feature Map: 특정 Layer에서의 Data값(Layer의 Node들의 집합)
* Activation Map: Activation을 적용한 단계
* 둘은 같은 의미로도 쓰임
* Locally Connected Layer
* 주변의 인접한 Node들 만의 연관관계를 보는 것
* Spatial Locality에 의해 학습해야 할 양이 줄어듦
* weight sharing: 같은 parameter를 서로 다른 위치에 걸쳐 사용(학습할 parameter의 수가 줄어듦)
* <-> Fully Connected Layer(고차원의 Data를 처리하기에는 연산 량이 너무 많고 많은 양의 데이터가 필요하며, Overfitting 발생할 확률이 높음)
* Convolution, Correlation
* Convolution: 시간 순으로 들어오므로 Filter의 반전이 필요(Signal Processing)
* Correlation: Template Matching에 사용
* Filter가 상하좌우 대칭인 경우 둘의 연산이 같아짐
* training중에 일부 위치에서 유용한 feature의 Detector는 test시의 모든 위치에서 유용, 따라서 Convolution 연산을 수행
* Convolution 연산을 통해 feature를 알아냄
* Pooling
* Convolution 연산 후에 Pooling을 진행
* 이미지의 크기를 줄여주는 작업으로 Channel 단위로 Pooling 진행
* 서로 다른 위치에서의 Filter Response들을 Pooling함으로써 Feature의 exact Spatial location에 대한 강인함을 얻음
  + 특정 Neuron이 Activation 되어 있다는 것은 해당 영역에서 관찰하고자 하는 Feature가 존재한다는 것을 의미함
* Max Pooling: Activation이 될 수 있는 가능성이 가장 높은 값을 선택하는 것
* 뒤쪽에 Fully Connected가 있음
* 특징
* Filter를 통해 중요한 정보를 추출
* Local Connectivity
* Weight Sharing
* Pooling
* Fully Convolutional Network
* Fully Connected가 없는 것

Ensemble Learning

* 특징
* 여러 개의 집합적인 Model들을 통합하여 최종적으로 더 나은 결정을 내리는 것으로, 각 모델은 주어진 문제에 learning process를 적용하여 얻음
* 정확성 향상을 목표로 학습된 여러 모델을 집계
* Classification, Regression, Clustering에 사용 가능
* 각 모델을 결합하는 것은 복잡도를 증가시킴
* 연산량의 증가로 시간의 증가
* Accuracy는 증가
* Occam’s Razor는 decision boundary가 간단해야 된다는 의미이지 Model Complexity랑은 다름. 앙상블의 Decision Boundary는 Simple해질 수 있기 때문에 Occam’s Razor를 위반하는 것은 아님.
* 종류
* Homogeneous: 모든 모델을 하나의 induction Algorithm을 이용해 구하는 것
* Heterogeneous: 각 모델마다 서로 다른 종류의 induction Algorithm을 이용해 구하는 것
* 프로세스
* 각 모델의 Generation -> pruning(Optional) -> 모델들을 Integration
* Generation
* Data Manipulation
* 서로 다른 Model을 얻기 위하여 Training Set을 변형
* Manipulating Input Features
  + Input들의 Attribute중 Subset을 골라 해당 Subset에 대해 전체 Input을 학습시켜 각각의 Model을 만들어 냄
* Sub-sampling from the training set
  + Input들의 Subset을 골라 각각의 Subset별로 학습시켜 각각의 모델을 만듦
* Modeling Process Manipulation
* 서로 다른 Model을 얻기 위하여, Induction Algorithm을 바꾸거나, Model의 Parameter를 바꿈
* Manipulating Parameter Set
  + 같은 Algorithm에 Parameter만 다르게 하여 학습시킴
* Manipulating Induction Algorithm
  + 적용하는 Algorithm을 서로 다르게 하여 학습
  + 절차적 방식에 차이를 만들어 학습
* Combine
* 만들어진 Classifier(Model)들은 정확할 수록 좋고, 각 Classifier들끼리 서로 어느 정도 분산(차이, 오류)이 있어야 함
* 하나의 Classifier 내에서는 오류가 작을 수록 좋고 분산이 작을 수록 좋음
* 종류
* Bagging
  + Homogeneous
  + Bootstrapping
    - Data에서 Subset을 random하게(weight x) Sampling, Sampling시 중복 허용
    - Sample을 선택할 때 Weight을 두지 않음(Sample을 Random하게 선택)
  + Training: d개 튜플의 set D에서, 각 iteration i마다, training set Di를 sampling
    - 각 Classifier Mi는 각 training set Di를 통해 학습됨
  + Classification: 각 Classifier Mi가 unknown sample X에 대한 Class Prediction을 내면, bagged Classifier M\*이 vote 수를 Count하고 가장 많은 표를 얻은 Class로 assign
  + Prediction: 주어진 test tuple에 대한 예측의 평균 값을 취하는 방법 또는 각각의 Class별 Score를 Prediction한 값을 받아서 Score값들을 합쳐 가장 큰 Score가 어떤 Class에서 나오는지 보는 방법
  + 특징
    - single classifier보다 성능 좋음
    - noisy data에 대해 성능 향상은 없을 수 있지만 강인함
    - prediction의 정확도를 향상시킴
    - training data의 작은 변화에 따른 decision boundary의 변화가 큰 민감한(unstable한) classifier type을 필요로 함
    - Decision tree는 Unstable한 Classifier이므로 ensemble하기 좋음(random forest)
    - Sample을 random하게 선택(Sample을 선택할 때 weight을 두지 않음)
* Boosting
  + Homogeneous
  + 각 training sample에 weight 부여
  + k 개의 Classifier가 순차적으로 학습됨
  + Classifier Mi가 학습 된 이후에, Mi가 오류를 내는 Sample들에 대해 weight을 update하여, 오류가 난 Sample이 더 많이 sampling 되도록 하여, Mi+1이 해당 오류가 난 Sample에 더 집중할 수 있도록 함
  + training data set을 전체를 사용하는 것이 아니고 training data set에서 resampling을 하는데 앞의 단계에서 오류가 나온 Sample들을 더 많이 포함해서 다음 단계의 Classifier 학습
  + 즉, 이전 Classifier에서 오류가 난 Sample은 다음 Classifier의 학습 데이터에 포함될 가능성이 큼
  + 최종 Classifier M\*은 각 Classifier에 weight을 할당하여 모두 합침
  + 특징
    - 약한 Classifier들의 Weighted Voting을 통해 강한 Classifier만들어 냄
    - 더 좋은 성능을 냈던 classifier가 더 많은 weight을 얻게 됨
    - 뒤로 갈 수록 분류하기 어려운 데이터에 대해 학습하게 됨
  + Bagging과의 차이
    - Model이 Sequentially 만들어 짐
    - prediction은 Weighted Sum/Vote를 통해 Combined 됨
    - Boosting은 더 좋은 Accuracy를 얻을 수 있지만 오류가 났던 Data들에 Overfitting될 수 있다는 문제(Outlier에 취약)
* Random Forest
  + Homogeneous
  + Bagging에 속함
  + Individual Decision Tree로부터 나옴
  + Bagging의 특징인 Diversity가 보장됨(트리 생성과정에서 원래 feature[attribute]의 subset을 Split마다 random하게 골라)
  + Classification: 각 tree는 vote를 하고 그 중 가장 많은 것으로 Class 할당
  + Regression: 모든 tree에 의해 만들어진 prediction값들의 평균
  + 두 가지 방법
    - Forest RI(Random Input Selection): 각 노드에서 attribute들을 random하게 골라 split의 후보로 함
    - Forest RC(Random Linear Combination): 존재하는 attribute들의 Linear Combination을 통해 새로운 attribute들 만들어 냄(각 classifier의 correlation 감소시킴)
  + AdaBoost와 비교 했을 때 Error와 Outlier에 강인
  + split시 선택되는 속성의 수에 민감하지 않고 Bagging이나 Boosting보다 빠름
* Ensemble Learning via negative correlation learning
  + Homogeneous
* Heterogeneous ensembles
* Model Selection
* problem에 적절한 Algorithm을 고르거나
* Algorithm에 적합하게 Data를 변형시킴
* 대부분의 경우, 좋은 Algorithm은 정확도가 좋음
* 특수한 경우 좋은 Algorithm은 결정을 내리는 과정이 설명가능해야함
* Decision Tree, Bayesian network
* Statistical Validation
* Mixture of Experts(Bagging)
* vote 결과나, score를 결합
* Stacking(Bagging)
* classifier들의 결정을 취합하여 function을 적용해 최종 판단
* Cascading(Boosting)
* 이전 결정이 충분히 신뢰할 수 없다면 다음 Classifier를 결합시킴
* Classifier들을 순차적으로 적용시킴

Reinforcement Learning

* 특징
* Supervisor가 없음, reward Signal만 있음
* trial-and-error learning
* Feedback이 Delay됨
* data가 상호 의존적이므로 시간 순서가 중요함
* Agent의 Action이 뒤에 받게 되는 Data(Subsequent Data)에 영향을 미침
* Rewards
* Reward Rt는 Scalar Feedback Signal
* agent가 특정 time step에 얼마나 잘 했는지를 나타냄
* agent는 누적 reward를 최대화 해야함
* Sequential Decision Making
* 특징
* 연속적인 결정을 만들어나가는 과정으로 total future reward를 최대화 해야함
* Reward는 Delay 됨
* 당장의 Reward는 작더라도 장기적인 관점의 Reward를 얻는 것이 좋을 수 있음(optimal 해야함)
* Reinforcement Learning
* 초기에 environment가 알려져 있지 않음
* agent는 environment와 interact
* agent는 policy를 향상시킴
* Planning
* 초기에 environment의 model이 알려져 있음
* agent는 model의 정보를 이용해 최적의 행동패턴 계산(interaction 없음)
* agent는 policy를 향상시킴
* Major Components of an RL Agent
* Policy
* Agent의 행동 양식
* state와 action 사이의 mapping관계 정의
* Deterministic Policy
  + State가 정해지면 Action이 바로 결정
* Stochastic Policy
  + State s에서 Action a가 나올 수 있는 확률
* Value Function
* Agent의 state와 action이 얼마나 좋은지 나타냄
* State Value Function
  + future reward에 대한 예측 값
  + state가 얼마나 좋은지 나타냄
  + = discount rate(0~1), 1에 가까울 수록 장기적인 reward까지 생각
* Model
* Agent의 environment 표현 방식
* environment가 다음에 무엇을 할지 예측
* P: 다음 상태를 예측
  + 현재 state가 s이고, action이 a일 때, 다음 state가 s[‘일 확률
* R: 다음(immediate) reward를 예측
  + 현재 state가 s이고, action이 a일 때, 얻을 수 있는 다음 reward
* Exploration and Exploitation
* Exploration
* environment에 대한 더 많은 정보를 찾아내는 것
* Exploitation
* reward를 최대화하기 위해 알려진 정보를 이용하는 것
* Markov Decision Processes(MDP)
* Reinforcement Learning을 위한 environment을 묘사
* 완전한 Environment 정보를 관찰 가능(즉, current state가 process를 완전히 설명)
* 대부분의 RL 문제가 MDP로 formalize될 수 있음
* Markov Property
* 현재가 주어지면, 미래는 과거에 independent(즉 미래는 현재에 의해서 결정됨)
* current state가 모든 과거의 정보를 포함(따라서 current state가 알려지면 과거의 정보는 버려도 됨)
* State Transition Matrix
* State Transition Probability
* State Transition Matrix
* 
* 각 row의 합은 1이 되어야 함
* Markov Process(Markov Chain)
* Memoryless random process(미래의 state가 과거의 state에 의존하지 않음)
* Tuple <S, P>
* S: set of states
* P: state transition matrix
* Markov Reward Process
* Markov Chain with Values
* Tuple <S, P, R, >
* R: reward function
* : discount factor (0~1)
  + 미래 보상에 대한 현재 관점에서의 가치
  + 0에 가까울 수록 당장의 reward
  + 1에 가까울 수록 장기적인 관점의 reward
* Return
* time-step t로부터의 total discounted reward
* Value function
* state s의 장기적인 관점의 value
* state s로부터 시작해서 얻을 수 있는 expected return
* optimal value function을 찾는 것이 reinforcement learning의 목표
* Bellman Equation
* current state value= current reward + next state value
* 텍스트, 안테나이(가) 표시된 사진

  자동 생성된 설명
* n states의 computational complexity는 O(n3)
* Markov Decision Process
* Markov Reward Process에 Decision(action)을 추가한 것
* Tuple <S, A, P, R, >
* A: finite set of actions
* P:
* R:
* Policies
* current state s에서 action a가 일어날 확률
* policy는 agent의 행동 양식 정의
* MDP policy는 Current State에 의존(과거에 의존하지 않음)
* Policy는 time independent
* state-value function
* state s에서 시작해서 policy 를 따를 때 expected return
* 현재 state에서 주어진 policy를 따르는 것이 얼마나 좋은지
* action-value function
* state s로부터 시작해서 action a를 취하고, policy 를 따를 때 expected return
* 현재 state에서 주어진 policy와, 관련된 action을 취했을 때 얼마나 좋은지
* =
* Optimal Value Function
* optimal state-value function은 모든 policies를 통해 얻을 수 있는 maximum value function
* optimal action-value function은 모든 policies를 통해 얻을 수 있는 maximum action value function
* optimal value function은 MDP안에서 best possible performance를 특정
* optimal value function을 알 때 solved 되었다고 함
* Optimal Policy
* 다른 모든 Policy 보다 좋거나 성능 동일한 Optimal Policy 이 있음
* 모든 optimal policy는 optimal value function achieve
* 모든 optimal policy는 optimal value function achieve
* Optimal Policy는 action value function을 최대가 되도록 하는 policy 선택
  + 텍스트이(가) 표시된 사진

    자동 생성된 설명
  + 텍스트이(가) 표시된 사진

    자동 생성된 설명

텍스트, 손목시계이(가) 표시된 사진

자동 생성된 설명

* + 텍스트, 손목시계이(가) 표시된 사진

    자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* 어떤 MDP던, deterministic optimal policy가 항상 있음
* Bellman Optimality Equation은 Non-linear
* 일반 적으로 closed form solution이 없음
* iterative solution method들이 있음