Classification

Linear Separation Training examples S_n are *linearly separable* if there exists a parameter vector $\widehat{\theta}$ and offset parameter $\widehat{\theta}_0$ such that $y^{(i)}\left(\widehat{\theta}\cdot x^{(i)}+\widehat{\theta}_0\right)>0$ for all $i=1,\ldots,n$.

Perceptron Algorithm

- Will find one classifier if data are linear separable
- Initialization $\theta=0$
- For $t=1,\ldots,T$, $i=1,\ldots,n$
- If $y^{(i)}\left(\theta\cdot x^{(i)}+\theta_0\right)\leq 0$

$$\theta=\theta+y^{(i)}x^{(i)}$$

$$\theta_0=\theta_0+y^{(i)}$$

Convergence of Perceptron

- There exists θ^* such that $\frac{y^{(i)}\left(\theta^*\cdot x^{(i)}\right)}{||x^{(i)}||}\geq\gamma$ for all $i=1,\ldots,n$ for some $\gamma>0$.
- All examples are bounded $||x^{(i)}||\leq R$, $i=1,\ldots,n$.

Then the number k of updates made by the perceptron algorithm is bounded by $\frac{R^2}{\gamma^2}$.

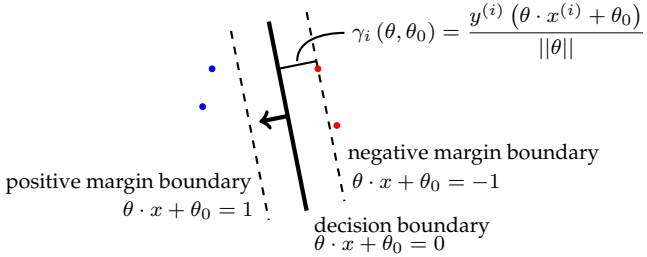
Hinge Loss

Distance to a plane $d=\dfrac{|\theta\cdot x+\theta_0|}{||\theta||}$

Hinge Loss

$$\text{Loss}_h(z)=\begin{cases}0 & \text{if } z\geq 1\\ 1-z & \text{if } z<1\end{cases}$$

with $z=y^{(i)}\left(\theta\cdot x^{(i)}+\theta_0\right)$.



Distance from Decision to Margin is $\frac{1}{\theta}$ after scaling (k = 1)

Objective Function

$$J(\theta,\theta_0)=\frac{1}{n}\sum_{i=1}^{n}\left[\text{Loss}_h\left(y^{(i)}\left(\theta\cdot x^{(i)}+\theta_0\right)\right)+\frac{\lambda}{2}\|\theta\|^2\right]$$

λ is the regularization factor.

Derivative

- if Loss >0 $\Delta J=-y^iX^{(i)}$
- if Loss $=0$ $\Delta J=0$

Recommender Systems

$$Y_{n*m}$$

n users, m movies. Key is to borrow preferences from other users and measure how much a user is close to others

K-Nearest Neighbor Method Does not allow to identify hidden structure in the data

$$\widehat{Y}_{ai}=\frac{\sum\limits_{b\in\text{KNN}(a)}\text{sim}(a,b)Y_{bi}}{\sum\limits_{b\in\text{KNN}(a)}\text{sim}(a,b)}$$

Collaborative Filtering with Matrix Factorization

$$Y_{n*m}=U_{n*d}V^T_{d*m}$$

$$J(X)=\sum_{(a,i)\in D}\frac{1}{2}\left(Y_{ai}-[UV^\intercal]_{ai}\right)^2+\frac{\lambda}{2}\left(\sum_{a,k}U_{ak}^2+\sum_{i,k}V_{ik}^2\right).$$

To find the solution, we fix (initialize) U (or V) and minimize the objective with respect to V (or U). We plug-in the result back to the objective and minimize it with respect to U (or V). We repeat this alternating process until there is no change in the objective function.

- For the case $k=1$. Then, $U_{a1}=u_a$ and $V_{i1}=v_i$. If we initialize u_a to some values, then we have to optimize the function

$$\sum_{(a,i)\in D}\frac{1}{2}\left(Y_{ai}-u_av_i\right)^2+\frac{\lambda}{2}\sum_i v_i^2.$$

Kernel

Inner products between high dim vectors are cheap. Use kernel but not explicitly the high dim vector representation

$$K\left(x,x'\right)=\phi(x)\cdot\phi(x')$$

Kernel Perceptron

$$\theta=\sum_{j=1}^{n}\alpha_jy^{(j)}\phi\left(x^{(j)}\right)$$

$$\theta\cdot\phi\left(x^{(i)}\right)=\sum_{j=1}^{n}\alpha_jy^{(j)}\underbrace{\phi\left(x^{(j)}\right)\cdot\phi\left(x^{(i)}\right)}_{K\left(x^{(j)},x^{(i)}\right)}$$

- For $t=1,\ldots,T$, $i=1,\ldots,n$
- If $y^{(i)}\sum\limits_{j=1}^{n}\alpha_jy^{(j)}K\left(x^{(j)},x^{(i)}\right)\leq 0$

$$\alpha_j=\alpha_j+1$$

Kernel Composition Rules

1. $K(x,x')=1$ is a kernel function.($\phi(x)=1$)
2. Let $f:\mathbb{R}^d\to\mathbb{R}$ and $K(x,x')$ is a kernel. Then so is $\widetilde{K}(x,x')=f(x)K(x,x')f(x')$. ($\phi(x)=f(x)\phi(x)$)
3. If $K_1(x,x')$ and $K_2(x,x')$ are kernels, then $K(x,x')=K_1(x,x')+K_2(x,x')$ is a kernel. $\phi(x)=[\phi_1(x),\phi_2(x)]^T$
4. If $K_1(x,x')$ and $K_2(x,x')$ are kernels, then $K(x,x')=K_1(x,x')K_2(x,x')$ is a kernel.

Radial Basis Kernel

$$K(x,x')=e^{-\frac{1}{2}||x-x'||^2}$$

Neural Network

Motivation: learn classifier and feature representation at the same time to improve performance.

Overcapacity: Large models tend to be easier to learn because their units need to be adjusted so that they are, collectively sufficient to solve the task

Backpropagation

- **Model Setup**

$$Loss=C(a^L)$$

$$a_j^l=f(\sum_k w_{jk}^la_k^{l-1}+b_j^l)$$

$$z_j^l=\sum_k w_{jk}^la_k^{l-1}+b_j^l$$

  - b_j^l is the bias of the j-th neuron in the l-th layer
  - a_j^l is the activation of j-th neuron in the l-th layer
  - w_{jk}^l is the weight for the connection from the k-th neuron in the (l-1)-th layer to the j-th neuron in the l-th layer

- **Chain** define $\delta^l=\dfrac{\partial C}{\partial z_j^l}$

$$\delta^L=\frac{\partial C}{\partial a_j^L}f'(z_j^L)$$

$$\delta^l=\sum_k w_{kj}^{l+1}\delta_k^{l+1}f'(z_j^l)$$

$$\frac{\partial C}{\partial w_{jk}^l}=a_k^{l-1}\delta_j^l$$

$$\frac{\partial C}{\partial b_j^l}=\delta_j^l$$

Unsupervised Learning

Clustering: Input

- Set of feature vectors $S_n = \{x^{(i)} | i = 1, \ldots, n\}$
- The number of clusters K

Clustering: Output

- A partition of indices $\{1, \ldots, n\}$ into K sets, $C_1, \ldots, C_K$
- *Representatives* in each of the K partition sets, given as $z_1, \ldots, z_K$

Cost We can calculate the total cost by summing the cost of each cluster:

$$\text{Cost}(C_1, \ldots, C_K) = \sum_{j=1}^{K} \text{Cost}(C_j)$$

Similarity Measure

- We use the Euclidean distance between the elements of a cluster and its representative to calculate the cost for each cluster. Then, the total cost is

$$\text{Cost}(C_1, ..., C_K, z_1, ..., z_K) = \sum_{j=1}^{K} \sum_{i \in C_j} ||x^{(i)} - z_j||^2$$

- Cosine Distance: invariant to magnitude of the vectors

$$dist(X^{(i)}, X^{(j)}) = \frac{X^{(i)} . X^{(j)}}{||X^{(i)}|| * ||X^{(i)}||}$$

K-Mean Algorithm Maximum Likelihood Estimate

1. Randomly select $z_1, \ldots, z_K$.
2. Iterate:
   (a) Given $z_1, \ldots, z_K$, assign each data point $x^{(i)}$ to the closest z_j so that

$$\text{Cost}(z_1, \ldots, z_K) = \sum_{i=1}^{n} \min_{j=1,\ldots,K} ||x^{(i)} - z_j||^2.$$

   (b) Given $C_1, \ldots, C_K$, find the best representatives $z_1, \ldots, z_K$, i.e., find $z_1, \ldots, z_K$ such that

$$z_j = \underset{z}{\text{argmin}} \sum_{i \in C_j} ||x^{(i)} - z||^2 = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}.$$

   - Local Solution (May run into troubles when initial rep are close to each other)
   - Not robust to outliers
   - Does not scale well with large dim: With increasing number of dims, a distance-based similarity measure converges to a constant value between any given examples
   - **Computational complexity**: O(ndK)

K-Medoids Algorithm

1. Randomly select $\{z_1, \ldots, z_K\} \subseteq \{x_1, \ldots, x_n\}$.
2. Iterate:
   (a) Given $z_1, \ldots, z_K$, assign each $x^{(i)}$ to the closest z_j so that

$$\text{Cost}(z_1, \ldots, z_K) = \sum_{i=1}^{n} \min_{j=1,\ldots,K} \text{dist}\left(x^{(i)}, z_j\right)$$

   (b) Given $C_j \in \{C_1, \ldots, C_K\}$, find the best representative $z_j \in \{x_1, \ldots, x_n\}$ such that

$$\sum_{x^{(i)} \in C_j} \text{dist}\left(x^{(i)}, z_j\right)$$

   is minimal.

- **Computational complexity**: $O(ndK) + O(n^2 dK)$

Generative Model

- **Generative vs. Discriminative Models** *Generative models* work by explicitly modeling the probability distribution of each of the individual classes in the training data. *Discriminative models* learn explicit decision boundary between classes.

Multinomial Generative Model

$$\mathbb{P}(w|\theta) = \theta_w, \quad \theta_w \geq 0, \sum_{w \in W} \theta_w = 1.$$

Then, the probability of generating the document D is

$$\mathbb{P}(D|\theta) = \prod_{i=1}^{n} \theta_{w_i} = \prod_{w \in W} \theta_w^{\text{count}(w)}.$$

$$\widehat{\theta}_w = \frac{\text{count}(w)}{\sum_{w' \in W} \text{count}(w')}.$$

Prediction Consider using a multinomial generative model M for the task of binary classification consisting of two classes: $+$ (positive class) and $-$ (negative class).

- θ^+: parameter for the positive class
- θ^-: parameter for the negative class

Suppose that we classify a new document D to belong to the positive class if and only if

$$\log \frac{\mathbb{P}(D|\theta^+)}{\mathbb{P}(D|\theta^-)} \geq 0.$$

The generative classifier is equivalent to a linear classifier:

$$\log \frac{\mathbb{P}(D|\theta^+)}{\mathbb{P}(D|\theta^-)} = \sum_{w \in W} \text{count}(w) \log \frac{\theta_w^+}{\theta_w^-} = \sum_{w \in W} \text{count}(w)\, \theta'_w.$$

Baye

$$\mathbb{P}(y = +|D) = \frac{\mathbb{P}(D|\theta^+)\, \mathbb{P}(y = +)}{\mathbb{P}(D)}.$$

$$\log \frac{\mathbb{P}(y = +|D)}{\mathbb{P}(y = -|D)} = \sum_{w \in W} \text{count}(w)\, \theta'_w + \theta'_0$$

where $\theta'_w = \log \frac{\theta_w^+}{\theta_w^-}$ and $\theta'_0 = \log \frac{\mathbb{P}(y = +)}{\mathbb{P}(y = -)}$.

Generative Gaussian $X \in \mathbb{R}^d$ with mean $\mu \in \mathbb{R}^d$ and the same standard deviation σ (uncorrelated)

$$f_X(\mathbf{x}|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \mu||^2).$$

$$\widehat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)}$$

$$\widehat{\sigma}^2 = \frac{1}{nd} \sum_{i=1}^{n} ||\mathbf{x}^{(i)} - \mu||^2$$

EM Algorithm

Gaussian Mixture Models

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{K} p_j \mathcal{N}\left(\mathbf{x}; \mu^{(j)}, \sigma_j^2 \mathbf{I}\right).$$

For the training set

$$S_n = \left\{\mathbf{x}^{(i)}, i = 1, \ldots, n\right\},$$

the likelihood is

$$\mathbb{P}(S_n|\theta) = \prod_{i=1}^{n} \sum_{j=1}^{K} p_j \mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2 \mathbf{I}\right).$$

Observed Case Consider the case of hard clustering, i.e., a point either belongs to a cluster or not. Let

$$\delta(j|i) = \begin{cases} 1, & \mathbf{x}^{(i)} \text{ is assigned to } j \\ 0, & \text{otherwise.} \end{cases}$$

Also, let $\widehat{n}_j = \sum_{i=1}^{n} \delta(j|i)$ denote the number of points belonging to cluster j. Maximizing the likelihood gives

$$\widehat{p}_j = \frac{\widehat{n}_j}{n}$$

$$\widehat{\mu}^{(j)} = \frac{1}{\widehat{n}_j} \sum_{i=1}^{n} \delta(j|i)\, \mathbf{x}^{(i)}$$

$$\widehat{\sigma}_j^2 = \frac{1}{\widehat{n}_j d} \sum_{i=1}^{n} \delta(j|i)\, ||\mathbf{x}^{(i)} - \mu^{(j)}||^2.$$

The EM Algorithm Instead of hard clustering, the data can actually be generated from different clusters with different probabilities. We have soft clustering. We can maximize the likelihood through the EM algorithm.

Randomly initialize θ: $\mu^{(1)}, \ldots, \mu^{(K)}, \sigma_1^2, \ldots, \sigma_K^2, p_1, \ldots, p_K$.

1. **E-step:**

$$p(j|i) = \frac{p_j \mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2 \mathbf{I}\right)}{p(\mathbf{x}|\theta)}$$

where $p(\mathbf{x}|\theta) = \sum_{j=1}^{K} p_j \mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2 \mathbf{I}\right)$

2. **M-step:**

$$\widehat{n}_j = \sum_{i=1}^{n} p(j|i)$$

$$\widehat{p}_j = \frac{\widehat{n}_j}{n}$$

$$\widehat{\mu}^{(j)} = \frac{1}{\widehat{n}_j} \sum_{i=1}^{n} p(j|i)\, \mathbf{x}^{(i)}$$

$$\widehat{\sigma}_j^2 = \frac{1}{\widehat{n}_j d} \sum_{i=1}^{n} p(j|i) ||\mathbf{x}^{(i)} - \mu^{(j)}||^2.$$

Reinforcement Learning

Goal: Learn a good policy with none or limited supervision

MDP

- **Transition** $T(s, a, s') = \mathbb{P}(s'|s, a)$
- **Reward** $R(s, a, s')$. The reward of starting at s, taking action a and ending up at s'

Utility

- **Infinite Utility** $U[s_0, s_1, \dots] = \sum_{k=0}^{\infty} \gamma^k R(s_k)$.
- An finite step utility function can depend on steps left (I.E. very few steps left may lead to risk taking behavior)
- Inifnite discounted utility make the utility depend on current step only
- Infinite discunted utility converge to $\frac{R_{min}}{1-\gamma}$
- **Policy** $\pi(s)$ that assign an action π to the state s
- **Optimal Policy** $\pi_s^\star = argmax_{a_s} \mathbb{E}[U(a_s)]$

Bellman Equations

- **Value Function** $V(s)$ expected reward starting from state s and acting optimally ever since

$$V(s) = \max_a Q(s, a) = Q(s, \pi^\star(s))$$

- **Q Function** $Q(s, a)$ expected reward strating at s, acting with acting a and then acting optimally aftewrwards

$$Q(s, a) = \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V(s'))$$

- **Value Iteration** $V(s) = \max_a \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V(s')$

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

- For the iterations, can plug in the final value to the rhs and solve for value
- **Q-value Iteration**

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$

- Value Iteration will finally converge as long as $\gamma < 1$ (speed for one iterationL $O(|S|^2|A|)$)

$$V(s) - V_k(s) = \gamma(V(s) - v_{k-1}(s))$$

Q-Value Iteration for RL

1. Initialization: $Q(s, a) = 0\ \forall s, a$
2. Iterate until convergence:
   (a) Collect sample: $s, a, s', R(s, a, s')$
   (b) Update:

$$Q_{i+1}(s, a) \leftarrow \alpha \left[R(s, a, s') + \gamma \max_{a'} Q_i(s', a')\right] + (1 - \alpha)\, Q_i(s, a)$$
$$= Q_i(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a)\right]$$

Exploration

- Exploitation (Optimal) $1 - \epsilon$
- Exploitation (Random) ϵ should decay as get data

NLP

- Difficulty arises from ambiguity intrinsic in natural languages
- **Approaches**
  1. Symbolic: Structured rules. Encode all required info into an elaborate knowledge representation
  2. Statistical: Infer language properties from large language samples.
- **Word Embedding**
  1. cosine similarity between words are used for a much sparse encoding
  2. Bag of words approach sums up all the word embeddings to encode hence does not capture sequence

Recurrent Network

Motivation: Model the sequence. How history is mapped to a feature vector (encoding) is alos part of the learning process

Difference with FF

- Inputs received at each layer.
- Layer number depend on lenght of sentence
- parameters of each layer are shared

Basic RNN

$$s_t = tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$$

- Gradient can vanish or explode since the sequences are long and we apply transformation repeatedly. Can resolve by introducing gate

Simple Gated RNN

$$g_t = \text{sigmoid}\ (W^{g,s} s_{t-1} + W^{g,x} x_t)$$
$$s_t = (1 - g_t) \odot s_{t-1} + g_t \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$$

LSTM

$$f_t = \text{sigmoid}\left(W^{f,h} h_{t-1} + W^{f,x} x_t\right) \quad \text{forget gate}$$
$$i_t = \text{sigmoid}\left(W^{i,h} h_{t-1} + W^{i,x} x_t\right) \quad \text{input gate}$$
$$o_t = \text{sigmoid}\left(W^{o,h} h_{t-1} + W^{o,x} x_t\right) \quad \text{output gate}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{c,h} h_{t-1} + W^{c,x} x_t) \quad \text{memory cell}$$
$$h_t = o_t \odot \tanh(c_t) \quad \text{visible state}$$

Markov Model (First-Order)

- Consist of an UNK symbol for any word, ¡beg¿, ¡end¿
- **Bigram Model** $\prod_{i=1} \mathbb{P}(w_i|w_{i-1})$
- ML estimation: $\widehat{\mathbb{P}}(w'|w) = \frac{count(w', w)}{\sum_{w_i} count(w, w_i)}$

Markov Model to FNN

$$z_k = \sum_j x_j W_{jk} + W_{0,k}, k \in \mathbb{R}$$

$$P_k = P(w_i = k|w_{i-1}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

- Advantages: Fewer number of parameters, easily to control complexity by introducing hidden layers
- Triagram: $2K^2 + K$ vs K^3

Markove Model to RNN At each step

$$s_t = tanh(W^{s,s} s_{t-1} + W^{s,w} x_t)$$

$$p_t = softmax(W^0 s_t)$$

or more complext like LSTM

$$p_t = softmax(W^0 h_t)$$

CNN

Goal: Extend local learning globally

convolution stride: layer l is obtained by slid (convolute) across the image at layer l-1 with a small kernel

$$(f * g)(t) \equiv \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau.$$

$$(f * g)[n] \equiv \sum_{m=-\infty}^{m=+\infty} f[m] g[n - m].$$

Pool: Further dim reduction. Can help to identify whether a feature is there withou knowing where it is

- Induce translation invariance at cost of spatial resolution
- Stried help to reduce the size

python

Distribution

- scipy.stats.binom(n=n, p=p)
- scipy.stats.poisson(mu=mu)
- scipy.stats.norm()
- scipy.stats.chi(df=df)

Distribution Function

- $1 - \alpha$ = rv.cdf(q_α)
- q_α = rv.ppf($1 - \alpha$)
- α = rv.sf(q_α)
- q_α = rv.isf(α)

Hypothesis Test

- **Fisher exact test** scipy.stats.fisher_exact(np.array([[x_{test}, x_{cont}], [n_{test}, n_{cont}]]), 'less') with less in alternative hypothesis
- **One sample t test** scipy.stats.ttest_1samp(array, popmean=H_0, alternative='greater')
- If nul rejected by Bonferroni, then it is also rejected by HB. Power of HB is greater than or equal to that of Bonferroni
- If nul rejected by HB then it is always rejected by BH
- Restriction on FWER is more strict
- **Multi Test Correction** statsmodels.stats.multitest.multipletests(pvals, alpha=0.05, method='holm')
- statsmodels.api.qqplot(x,line='s')
- **Likelihood**: scipy.stats.poisson.pmf(gamma_data['count'], gamma_data['seconds']*lamb).prod(axis=0)
- Lambda_observed = -2*np.log(likelihood_H0(lambda_hat_H0)/likelihood_H1(lambdas_hat_H1))
- scipy.stats.kstest