## Probability Basics and Linear Algebra

- Marginal probability $$P(A) = \sum_b P(A, B=b)$$
- Conditional probability $$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

$$P(A,B) = P(A \mid B)P(B)$$

- **I**ndependence: P(A, B) = P(A)P(B)

Equivalently: P(A | B) = P(A) or P(B | A) = P(B)

- Mutual independence:
- Every $I \subset \{1,2,\dots,n\}$ and $J \subset \{1,2,\dots,n\}$ have

$$P\left(\bigcap_{i\in I} A_i \cap \bigcap_{j\in J} A_j\right) = P\left(\bigcap_{i\in I} A_i\right) P\left(\bigcap_{j\in J} A_j\right)$$

- Conditional independence:
- $P(A \wedge B \mid C) = P(A \mid C)\, P(B \mid C)$
- Equivalently: $P(A \mid B \wedge C) = P(A \mid C)$ or $P(B \mid A \wedge C) = P(B \mid C)$
  - Independence does not imply conditional independence, or vice versa
- Expectation
  - Discrete: $E[X] = \sum_x x \ \ p(x)$
  - Continuous: $E[X] = \int_x x \ \ p(x)dx$
  - Axioms:
    - E[aX+b] = aE[X] + b
    - E[h(X)] = $\sum_x h(x) \ \ p(x)$
    - $E[h(X)] = \int_x h(x)p(x)dx$
- Variance
  - Var(X) = $E[X^2] - (E[X])^2$
  - Standard deviation, $\sigma = \sqrt{Var(X)}$
  - Axioms:
    - Var(aX+b) = $a^2 \ \ Var(X)$
- Distribution:
  - Bernoulli(Binary variables):

$$P(x) = p^x(1-p)^{1-x}$$
$$E[X] = 1(p) + 0(1-p) = p$$
$$Var(X) = \ p(1-p)$$

  - Binomial(No. of successful outcomes in n Bernoulli trials)

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x}$$
$$E[X] = np$$
$$Var[X] = np(1-p)$$

- **Likelihood function**(L(D;Theta))

$$L(D;\theta) = P(\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n; \theta) = \prod_{i=1}^{n} P(\underline{x}_i;\theta)$$

by independence

  - Basically take product of probabilities of n independent samples x_1, x_2, ..., x_n
- **Maximum Likelihood function(MLE)**
  - **<u>Steps to obtain MLE:</u>**
    - First obtain likelihood function L(D;Theta)
    - Obtain log-likelihood function(take natural log, ln)
    - Convert to summation instead by log property
    - Take derivative and equate to 0 and resolve for parameter

$$l(D;\theta) = \log L(D;\theta) = \log \prod_{i=1}^{n} P(\underline{x}_i;\theta) = \sum_{i=1}^{n} \log P(\underline{x}_i;\theta)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \ l(D;\theta)$$

## K-Nearest Neighbours

- <u>Assumption: Close inputs have similar labels</u>
- Use distance metric and take majority vote idea for <u>classification</u>, take average of k labels for <u>regression</u>
- Have to calculate distance between current and all other data in training set
- KNN is a non-parametric algorithm
- **Problem:**
  - kNN training is fast, but prediction is very slow due to computation of distance metric(esp for data with high dimensionality)
- **Higher values of k:** smoother decision boundary(less complex functions)
- **Lower values of k:** More expressive and complex functions
- Finding right balance: **Find k** that minimizes the **validation error**(set aside some of the data to test on after training)
- **Distance metrics:**
  - Manhattan distance (L1-norm)
  - Euclidean distance (L2-norm)
  - Minkowski distance (L-p norm)

## Decision Trees

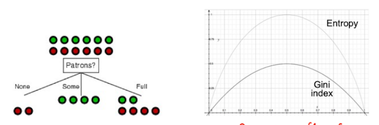
- **Entropy:** $H(X) = -\sum p(x)\log_2 p(x)$
- $$Gain(S,A) = \underline{Entropy(S)} - \sum_{a\in values(A)} \frac{|S_{A=a}|}{|S|} Entropy(S_{A=a})$$
- **Note:** Calculate new entropy after splitting, then use this new entropy to calculate info gain / gini
- $$Gini(X) = 1 - \sum p(x)^2$$

original

$$Gain(S,A) = Gini(S) - \sum_{a\in values(A)} \frac{|S_{A=a}|}{|S|} Gini(S_{A=a})$$

- Example:

Comparing information gain to Gini gain



$$IG = 1.0 - \left[\frac{2}{12}0\right] - \left[\frac{4}{12}0\right] - \left[\frac{6}{12}\underline{0.919}\right] = 0.541$$

$$-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}$$
$$= 0.91830 \text{ (5sf)}$$
$$= 0.918 \text{ (3sf)}$$

$$GG = \underline{0.5} - \left[\frac{2}{12}0\right] - \left[\frac{4}{12}0\right] - \left[\frac{6}{12}\underline{0.444}\right] = 0.278$$

19

$$1-\left(\tfrac{1}{2}\right)^2 - \left(\tfrac{1}{2}\right)^2$$

$$1 - \left[\left(1 - \left(\tfrac{2}{6}\right)^2 - \left(\tfrac{4}{6}\right)^2\right]\right.$$
$$= 0.44444 \text{ (5sf)}$$
$$= 0.444 \text{ (3sf)}$$

- **Pruning**
  - To avoid overfitting
  - <u>Postpruning</u>: Use validation set to evaluate utility of pruning nodes from tree
  - <u>Prepruning</u>: Use stopping criterion(threshold on feature score), before continuing to build tree
- Comparison with kNN:
  - More complex, good for discrete attributes
  - kNN better for continuous attributes

## Naive Bayes

- <u>Bayes rule for probabilistic classifier:</u>

$$P(y \mid \underline{x}) = \frac{P(\underline{x}, y)}{P(\underline{x})} = \frac{P(\underline{x} \mid y)P(y)}{P(\underline{x})}$$

Bayes rule

$$= \frac{P(\underline{x} \mid y)P(y)}{P(\underline{x} \mid y=1)P(y=1) + P(\underline{x} \mid y=-1)P(y=-1)}$$

Law of total probability

$$\propto P(\underline{x} \mid y)P(y)$$

Denominator is not important for Classification

Regression Classification

$$\underset{y}{\operatorname{argmax}} \frac{P(x|y)P(y)}{P(x)} = \underset{y}{\operatorname{argmax}} P(x|y)P(y)$$

- <u>Assumption: Attributes in feature vector are **conditionally independent** given the label</u>

$$P(\underline{x}|y) = \prod_{j=1}^{d} P(x_j|y)$$

- If the attributes are discrete, model $P(x_j|y)$ as a **Multinomial(1,q)** distribution
  - Each attribute $x_j$ can have values in {1,…,k}
  - For each possible y value and each attribute $x_j$ we have k parameters:

    $P(x_j = a \mid y = b)$
- P(y=1) is also a parameter
- Question: If <u>y is binary</u>, how many parameters are there?
- 2dk + 1 (or 2d(k-1)+1 if we're clever) x is a d-dimensional vector
- If the attributes are real-valued, typically model $P(x_j|y)$ as **Normal** distribution

+1 because of the case P(y=1)

- <u>Prior: P(y) where y is the label</u>
- <u>Conditional: P(x|y)</u>
- **<u>Laplace Smoothing:</u>**
  - **<u>Numerator += 1, Denominator += len(labels)</u>**

## Linear Classifiers

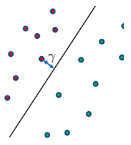
- $\hat{h}(x) = \underline{\text{sign}}\left(\hat{w}^\top x + \hat{b}\right) \in \{-1, 1\}$
- >=0: 1, <0: -1
- Note that weight vector is always normal to hyperplane(decision boundary)
- **Linearly separable:** there exists (w, b) such that h_{w,b} is correct on all training data (zero training loss)
- **<u>Perceptron</u>**
  - Assumption that data is linearly separable
  - For linearly separable data, perceptron makes a finite number of updates and is guaranteed to find a line that separates data
  - Perceptron learning algorithm:

```
Perceptron Algorithm:
1. Initialize $w_0 = 0$, $b_0 = 0$, $m = 0$
2. For $t = 1, 2, \dots, n, 1, 2, \dots, n, \dots$ (until no more mistakes)
3.   If $\text{sign}(w_m^\top x_t + b_m) \neq y_t$ (mistake)
4.     Update $w_{m+1} = w_m + \underline{y_t} x_t$ (update weights)
5.     Update $b_{m+1} = b_m + y_t$
6.     $m \leftarrow m + 1$
```

$y_t \in [-1, 1]$

  - Geometric margin:

$\gamma =$ distance of closest point to the separator



- Data scale

$$r = \max_{t \le n} \|x_t\|$$ largest norm of data

**Theorem:** Perceptron makes at most $\frac{r^2+1}{\gamma^2}$ updates and the final $(w_m, b_m)$ is correct on the data set

  - r = max value of x in x-axis (furthest x-value data point)

## Support Vector Machine(SVM)

- **Exam trick:** Choose two closest points from opposite classes together and then draw 2 perpendicular lines to this connecting line and pass each of these points. If any of the points from either of the classes comes inside the region bounded by these two lines, then that point might have to be a support vector as well and you will have to repeat this case by considering that point, just to make sure. If none of the points lie inside the region, then that means those 2 points are the only support vectors with that region as margin and a middle parallel line would be the decision boundary.
- Geometric margin:

$$\gamma = \max_{(w,b):\|w\|=1} \underbrace{\min_{1\le i\le n} y_i\left(w^\top x_i + b\right)}_{\text{distance of closest point to separator}}$$

- Optimisation problem for SVM:

Minimize $\|w\|^2$ (squared norm)

subject to $y_i\left(w^\top x_i + b\right) \ge 1, \ \forall i: \ 1 \le i \le n$

- The name "*Support Vector Machine*" stems from the fact that w* is supported by (i.e., is the linear span of) the examples that are exactly at a distance 1 / ||w*|| from the separating hyperplane.

- **Soft-margin SVM**
  - $$\text{Minimize}(1/2)\|w\|^2 + C\sum_{i=1}^{n} \overbrace{\max\{1 - y_i\left(w^\top x_i + b\right), 0\}}$$

where max(1-..., 0) is the hinge loss, it quantifies how bad mistakes made are
  - Regarding C:
- It defines a trade-off between the <u>loss on the data</u> and the norm ||w||
- For separable data, taking <u>C → ∞</u> equivalent to "hard margin" SVM from earlier

$\xi_i$ have to be small (less flexibility)

- The term $||w||^2$ is called a **regularizer** (squared Euclidean norm)

## Logistic Regression

- Presence of noisy labels
- Intuitively, we

want a model where $P_w(Y=1|X=x)$ is larger for larger $w^\top x + b$
- Logistic Model:

$$P_{w,b}(Y=1|X=x) = \frac{1}{1 + e^{-(w^\top x + b)}}$$

- So generally:

$$P_{w,b}(Y=y|X=x) = \frac{1}{1 + e^{-y(w^\top x + b)}}$$

### Training: Logistic Regression

& Cheatsheet

- Explicitly: denote $\sigma(x) = \frac{1}{1+e^{-x}}$
- Simple derivative: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

$$\nabla l_{Y|X}(w,b;S) = \nabla \sum_{i=1}^{n} \ln\left(\sigma\left(y_i(w^\top x_i + b)\right)\right)$$

$$= \sum_{i=1}^{n} \frac{1}{\sigma(y_i(w^\top x_i + b))} \sigma\left(y_i(w^\top x_i + b)\right)\left(1 - \sigma\left(y_i(w^\top x_i + b)\right)\right) y_i[x_i, 1]^\top$$

$$= \sum_{i=1}^{n} \left(1 - \sigma\left(y_i(w^\top x_i + b)\right)\right) y_i[x_i, 1]^\top$$

learning rate

gradient ascent: iterate $(w,b) \leftarrow (w,b) + \epsilon \nabla l_{Y|X}(w,b;S)$

## Logistic Regression

- Assumption: Linear relationship between the independent variables and the log odds

## Kernel Methods

- Idea: Want to map the data points into a higher dimensional space and learn a linear separator in that space because there does not exist a linear separator in current dimension
  - However, problem is that such a mapping is computationally expensive and requires a lot of memory esp if dimension mapped to is very large
- **First idea for SVM:**
  - Lagrangian dual form expression

### The Dual Form of the SVM Optimization Problem

$$\hat{w} = \underset{w:\|w\|=1}{\operatorname{argmax}} \min_{1\le i\le n} y_i \left(w^\top x_i\right)$$

- Recall: The SVM **classifier** is the unique solution to a **quadratic program**:

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i\left(w^\top x_i\right) \ge 1,\ \forall i:\ 1\le i\le n \end{array}}$$

- We can re-express this in Lagrangian dual form:

$$\hat{w} = \textstyle\sum_{i=1}^n \alpha_i y_i x_i$$

where $\alpha_1,\dots,\alpha_n$ are solutions to:

$$\boxed{\begin{array}{ll} \text{Maximize} & \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} & \alpha_i \ge 0,\ \forall i:\ 1\le i\le n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{array}}$$

Side note: the training points $(x_i,y_i)$ with non-zero $\alpha_i$ values are called the **support vectors**.

- Then, in higher dims mapping

$$\text{Maximize} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)^\top \phi(x_j)}$$

- But computing inner product is too slow, so we can use kernel K(x_i, x_j)

$$K(x_i,x_j) = \phi(x_i)^\top \phi(x_j)$$

- And the classifier also uses an inner product (no need to comput

$$h_{\hat{w}}(x) = \operatorname{sign}\left(\hat{w}^\top \phi(x)\right) = \operatorname{sign}\left(\left(\sum_{i=1}^n \alpha_i y_i \phi(x_i)\right)^\top \phi(x)\right) = \operatorname{sign}\left(\sum_{i=1}^n \alpha_i y_i \boldsymbol{\phi(x_i)^\top \phi(}\right.$$

- Replace all of these with $K(x_i,x) = \phi(x_i)^\top \phi(x)$

### Kernel SVM

- Training time:

Solve for $\alpha_1,\dots,\alpha_n$:

$$\boxed{\begin{array}{ll} \text{Maximize} & \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n y_i y_j \alpha_i \alpha_j \boldsymbol{K(x_i,x_j)} \\ \text{subject to} & \alpha_i \ge 0,\ \forall i:\ 1\le i\le n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{array}}$$

- Test time:
- Classify a new point x with

$$\hat{h}(x) = \operatorname{sign}\left(\sum_{i=1}^n \alpha_i y_i \boldsymbol{K(x_i,x)}\right)$$

## Kernel Methods

### Kernels

- More Examples:
  - Polynomial kernel: $K(\underline{u},\underline{v}) = (\underline{u}^\top \underline{v} + 1)^p$
    - Implicitly computes an inner product in $\sim d^p$ dimensions
  - Gaussian kernel:

$$K(\underline{u},\underline{v}) = e^{-\frac{\|\underline{u}-\underline{v}\|^2}{2\sigma^2}}$$

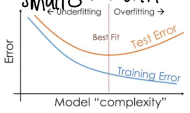
    - Implicitly computes an inner product in **infinite** dimensions
    - Remark: This is the most popular kernel in practice

### Example: Soft-SVM with Gaussian Kernel

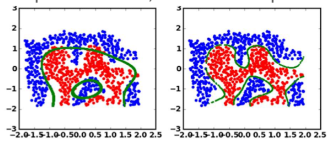
- A common choice is to use the Gaussian kernel

$$K(\underline{u},\underline{v}) = e^{-\frac{\|\underline{u}-\underline{v}\|^2}{2\sigma^2}}$$

large σ : underfit
small σ : overfit



- σ called the **bandwidth**
- It controls smoothness of the boundary
- Gives a notion of model complexity:
- large σ → simple boundaries, small σ → complex boundaries


large σ small σ

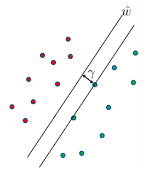
### What about overfitting?

- Margin for kernel SVM:

*Cheatsheet this

geometric margin of classifier w:

$$\gamma = \min_{1\le i\le n} y_i \left(\frac{w^\top}{\|w\|}\phi(x_i)\right)$$



Recall: The $\hat{w}$ solution of SVM primal problem satisfies $\|\hat{w}\| = \frac{1}{\gamma}$

For kernel SVM, this means

$$\frac{1}{\gamma^2} = \|\hat{w}\|^2 = \hat{w}^\top \hat{w} = \sum_{i=1}^n\sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i,x_j)$$

Solve for $\gamma$ to compute the margin

So we can also calculate the margin without computing $\phi$, using $K$

- We need to be careful that the implicit high-dimensional representation doesn't lead to overfitting
- If the solution has large margin, it can avoid overfitting
- Recall that SVM is designed to maximize the margin, so it is well-suited to this type of guarantee

## Gradient Descent

- Assumption: Function is differentiable(normally loss function)
- Optimization method to find the local minimal of a differentiable function (not necessarily global minimum)

### How to decide whether a function is convex?

- In graph of convex function f, the line connecting two points must lie above the function:

**
$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y) \text{ for all } 0 \le \alpha \le 1$$

- Practical test for convexity: a twice (second derivative) differentiable function f of a variable x is convex on an interval if an only if for any x in the interval: $f''(x) \ge 0$

- Sum of convex functions is convex; max of convex functions is convex

### Gradient Descent Algorithm

Given a differential function $f$:

- Pick an initial point $x^0$
- Iterate until convergence

$$x^{t+1} = x^t - \alpha \nabla f(x^t)$$

- If $\|x^{t+1} - x^t\| \le \epsilon$, then stop

- How to set step size/learning rate(alpha)?
  - Observe training loss behaviour (hyperparameter tuning)
- Run GD algorithm with randomized starting points for a few iterations and pick the best(least training loss)
  - Because initial point matters; Might have a chance an initial point might cause GD algorithm to terminate at global min

### Second-order Optimization

- The Hessian matrix is a $n \times n$ matrix and is defined as

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

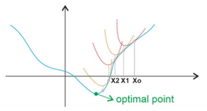
- Algorithm: update parameter following (Newton's Method)

$$x^{t+1} = x^t - [H(x^t)]^{-1} \nabla f(x^t)$$

inverse of Hessian matrix; step size not req, because determined by second order derivative

### Newton's Method

- Pros: converge very fast when the Hessian captures local geometry accurately; no need to tune step sizes
- Cons: Hessian is expensive to compute; Easy to diverge when x is far way from the optimum or the second derivative is close to zero


X2 X1 Xo
optimal point